USN | | | | | | | | | | |

Internal Assessment Test I – November 2024

Sub:	Software Engineering and Project Management					Sub Code:	BCS501	Branch:	ISE
Date:	07-11-2024	Duration:	90 min's	Max Marks:	50	Sem/Sec:	V / A, B & C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a. Write the IEEE's definition of Software Engineering? Briefly explain the nature of software? Scheme: a: 2+3M Solutions: IEEE Definition: The IEEE [IEEE93a] has developed a more comprehensive definition when it states: Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).Instructions (computer programs) that when executed provide desired function and performance,data structures that enable the programs to adequately manipulate information,documents that describe the operation and use of the programs. Explain:	5 5	CO1	L1

1.1 The Nature of Software

'NT
both a
a vehicle
; a product.

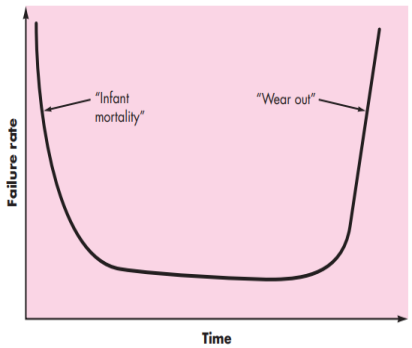
Today, software takes on a dual role. It is a product, and at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or more broadly, by a network of computers that are accessible by local hardware. Whether it resides within a mobile phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation derived from data acquired from dozens of independent sources. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).

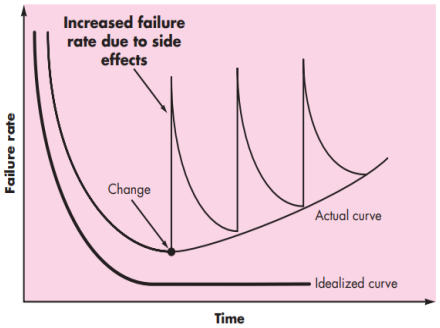
1.1.1 Defining Software

Today, most professionals and many members of the public at large feel that they understand software. But do they?

A textbook description of software might take the following form:

> Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information, and (3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.





b. What are the software myths governing the development of software? What is a well-engineered software?
Scheme:3+2M
Solution:

Software Myths

Software myths—erroneous beliefs about software and the process that is used to build it—can be traced to the earliest days of computing. Myths have a number of attributes that make them insidious. For instance, they appear to be reasonable statements of fact (sometimes containing elements of truth), they have an intuitive feel, and they are often promulgated by experienced practitioners who "know the score."

Management myths. Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

Myth: *We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?*

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Myth: *If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian horde" concept).*

Reality: Software development is not a mechanistic process like manufacturing. In the words of Brooks [Bro95]: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner.

Customer myths. A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.

Myth: *A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.*

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived

Practitioner's myths. Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture. During the early days, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: *Once we write the program and get it to work, our job is done.*

Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: *Until I get the program "running" I have no way of assessing its quality.*

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—*the technical review.* Software reviews (described in Chapter 15) are a "quality filter" that have been found to be more effective than testing for finding ce classes of software defects.

2	a. Compare waterfall model with other similar models. Scheme:3+2M	5 5	CO1	L2

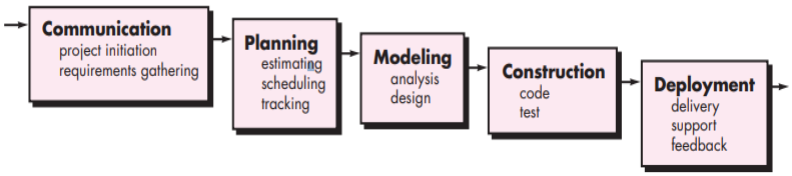
2.3.1 The Waterfall Model

There are times when the requirements for a problem are well understood—when work flows from **communication** through **deployment** in a reasonably linear fashion. This situation is sometimes encountered when well-defined adaptations or enhancements to an existing system must be made (e.g., an adaptation to accounting software that has been mandated because of changes to government regulations). It may also occur in a limited number of new development efforts, but only when requirements are well defined and reasonably stable.

The *waterfall model,* sometimes called the *classic life cycle*, suggests a systematic, sequential approach[6] to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software (Figure 2.3).

A variation in the representation of the waterfall model is called the *V-model*. Represented in Figure 2.4, the V-model [Buc99] depicts the relationship of quality

FIGURE 2.3 The waterfall model



V-model

An extension of the waterfall model that emphasizes testing at each stage, making it more adaptable to changes than the traditional waterfall model.

Prototype model

Ideal for projects with unclear or changing requirements, while the waterfall model is better for projects with well-defined requirements.

Spiral model

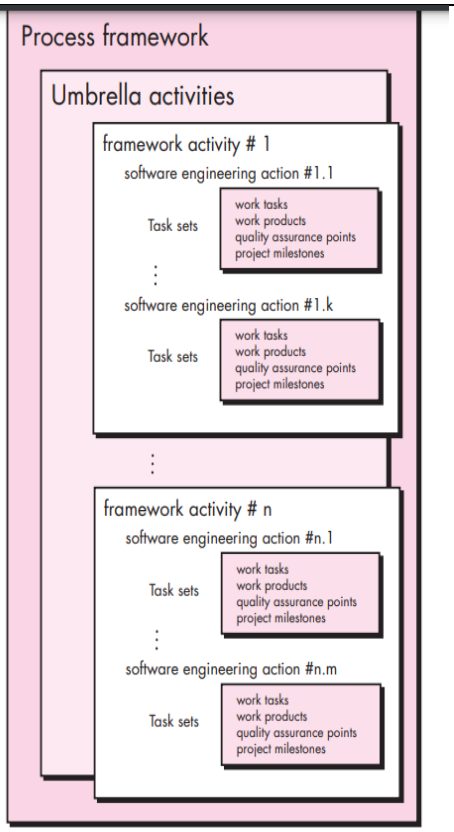
A flexible methodology that emphasizes risk analysis and management, making it suitable for large, complex, and long-term projects.

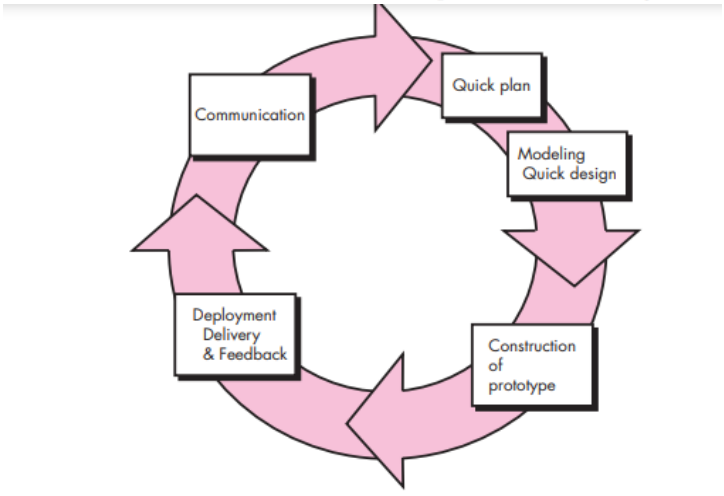
b. Define task, activity, and milestone. Write a note on software components and software applications.

Scheme:3+2M

A task focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

An activity strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.

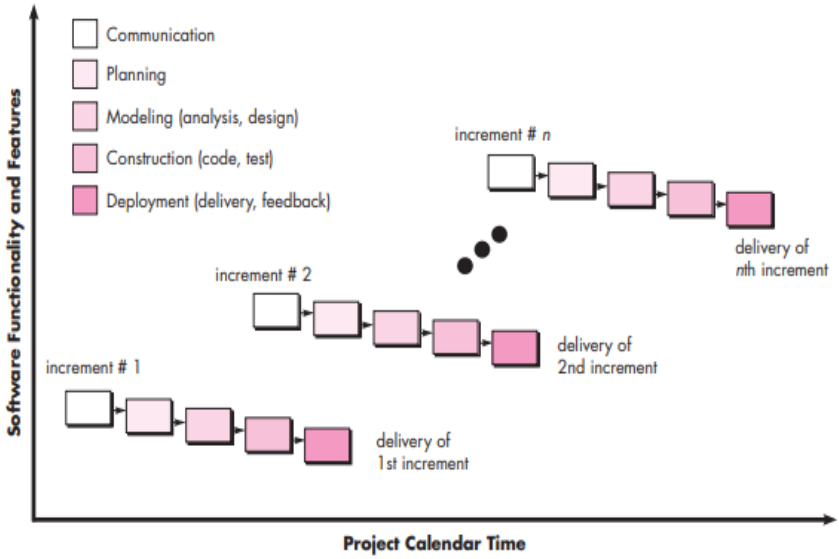


3	a. Explain the prototype and incremental development process with a neat diagram. **Prototyping.** Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a *prototyping paradigm* may offer the best approach. Although prototyping can be used as a stand-alone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models noted in this chapter. Regardless of the manner in which it is applied, the prototyping paradigm assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.  	5 5	CO1	L3

2.3.2 Incremental Process Models

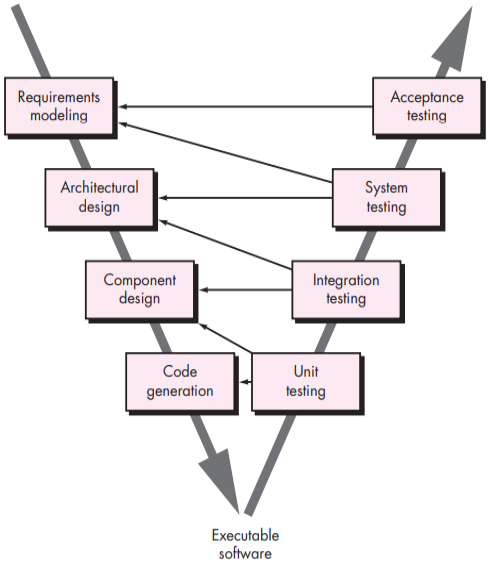
There are many situations in which initial software requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. In addition, there may be a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases. In such cases, you can choose a process model that is designed to produce the software in increments.

The *incremental* model combines elements of linear and parallel process flows discussed in Section 2.1. Referring to Figure 2.5, the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software [McD93] in a manner that is similar to the increments produced by an evolutionary process flow (Section 2.3.3).



b. Mention the benefits of these models compared to the V model using suitable diagrams. Point out at least two differences between them.

The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work. The waterfall model is the oldest paradigm for software engineering. However, over the past three decades, criticism of this process model has caused even ardent supporters to question its efficacy [Han95]. Among the problems that are sometimes encountered when the waterfall model is applied are: 1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects. 3. The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.



4	a. What is requirement validation? Explain different types of checks carried out during the process. **Validation.** The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification[5] to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product. The primary requirements validation mechanism is the technical review (Chapter 15). The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies (a major problem when large products or systems are engineered), conflicting requirements, or unrealistic (unachievable) requirements. b. Explain the process of requirement negotiation with an example. **NEGOTIATING REQUIREMENTS** In an ideal requirements engineering context, the inception, elicitation, and elaboration tasks determine customer requirements in sufficient detail to proceed to subsequent software engineering activities. Unfortunately, this rarely happens. In reality, you may have to enter into a negotiation with one or more stakeholders. In most cases, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time-to-market. The intent of this negotiation is to develop a project plan that meets stakeholder needs while at the	5 5	CO2	L2

same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team.

The best negotiations strive for a "win-win" result.[20] That is, stakeholders win by getting the system or product that satisfies the majority of their needs and you (as a member of the software team) win by working to realistic and achievable budgets and deadlines.

Boehm [Boe98] defines a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined:

1. Identification of the system or subsystem's key stakeholders.
2. Determination of the stakeholders' "win conditions."
3. Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team).

Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to subsequent software engineering activities.

5 a. Define and differentiate functional and non-functional requirements. 5 CO2 L2

5

Difference between Functional & Non Functional Requirements

No.	Functional Requirements	Non-Functional Requirements
1	Help to understand the functions of the system.	Help to understand the system's performance.
2	Mandatory requirements.	Not mandatory requirements.
3	They are easy to define.	They are hard to define.
4	It concentrates on the user's requirement.	It concentrates on the expectation of the user.
5	These requirements are specified by the user.	These requirements are specified by the software developers, architects and technical persons.

b. What is a requirement specification? Explain various ways of writing requirement specifications.

Specification

In the context of computer-based systems (and software), the term specification means different things to different people.

A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.

Some suggest that a "standard template" should be developed and used for a specification, arguing that this leads to requirements that are presented in a consistent and therefore more understandable manner.

However, it is sometimes necessary to remain flexible when a specification is to be developed.

	For large systems, a written document, combining natural language descriptions and graphical models may be the best approach. However, usage scenarios may be all that are required for smaller products or systems that reside within well-understood technical environments.			
6	*Draw the use case diagram, activity diagram and sequence diagram for the following case study:* A Modern Bazar Supermarket sells books and CDs using Online shopping. The customer adds items to the shopping cart. The customer may remove items or go to the checkout to make purchases at any time. The customer receives the purchased items by choosing a payment method. A sales employee at modern bazaar supermarket gets the order and purchase confirmation from the system and sends the electronic order to the warehouse. The warehouse employee updates the order status. The customer may check the order status. Use Case Diagram – 4 Marks Activity Diagram – 3 Marks Sequence Diagram – 3 Marks	10	**CO2**	**L3**