

SOFTWARE ENGINEERING

Chapter 1

Overview of Software Engineering & the Software Development Process

Detailed Study Notes with MCQs and Broad Questions

For Engineering Students

Contents

1.1 The Evolving Role of Software & Changing Nature of Software

1.2 Software Engineering – A Layered Technology Approach

1.3 A Process Framework & Software Project Tracking & Control

1.4 The Capability Maturity Model Integration (CMMI)

1.5 Process Patterns, Process Assessment, Personal & Team Process Models & Process Technology Theories

1.6 Process Models – Waterfall, Incremental, RAD, Prototype, Spiral

Multiple Choice Questions (15)

Answer Key

Broad Questions (10)

1.1 The Evolving Role of Software & Changing Nature of Software

Introduction to Software

Software is a collection of instructions (programs) that, when executed, provide desired features, functions and performance; data structures that enable the programs to adequately manipulate information; and documentation that describes the operation and use of the programs. Software is *logical* rather than physical, so it is **developed or engineered, not manufactured** in the classical sense.

Software can be viewed in two roles: as a **product** and as a **vehicle for delivering a product**. As a product, it delivers the computing potential of the hardware. As a vehicle, it controls the computer (operating system), supports communication (networks), and helps create other programs (software tools and environments).

Characteristics of Software

- **Software is developed, not manufactured:** Once developed, producing copies costs almost nothing; the major cost lies in design and development.
- **Software does not wear out:** Hardware follows a 'bathtub curve' (high failure early, low in mid-life, high at end). Software does not suffer from environmental wear, but it *deteriorates* because of changes made during maintenance, which introduce new defects.
- **Most software is custom-built:** Although the industry is moving toward component-based assembly, most software is still built to order rather than assembled from off-the-shelf parts.
- **Software is complex and intangible:** It cannot be touched or seen; its behaviour is understood only through execution and documentation.
- **Software is easily changed but changes are risky:** Flexibility is a strength, yet uncontrolled change is a major source of failure.

Feature	Hardware	Software
Nature	Physical, tangible	Logical, intangible
Failure pattern	Bathtub curve (wear and tear)	Idealized curve; deteriorates through change
Production	Manufactured (cost in each unit)	Developed (cost mostly in design)
Spare parts	Replacement parts available	No spare parts; defects must be corrected
Construction	Mostly assembled from components	Mostly custom built

Evolving Role of Software

The role of software has undergone dramatic change over the past five decades. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of input and output options have led to more

sophisticated and complex systems. Today software is the driving force behind business decisions, a basis for modern scientific investigation, a key factor in differentiating modern products and services, and is embedded in nearly every system (transportation, medicine, telecommunications, military, industrial processes, entertainment, and office products).

Phases in the Evolution of Software

Era	Approximate Period	Major Features
Early era	1950s - 1960s	Batch orientation, limited distribution, custom software, hardware-driven; programs written by the same person who used them.
Second era	Mid-1960s - 1970s	Multiuser, real-time systems, databases, software as a product, 'software crisis' emerges; maintenance becomes costly.
Third era	Mid-1970s - 1980s	Distributed systems, microprocessors, personal computers, low-cost hardware, consumer impact.
Fourth era	1990s - 2000	Object-oriented technology, expert systems, GUIs, networks, parallel computing, AI.
Present era	2000 onwards	Internet, web and mobile apps, cloud, big data, AI/ML, IoT, open-source, agile and DevOps culture.

The Software Crisis

During the 1960s-70s large projects regularly ran over budget, missed schedules, produced unreliable systems and were hard to maintain. This situation, called the **software crisis**, led to the 1968 NATO conference where the term '**software engineering**' was formally introduced. Its causes included:

- Rapid growth in size and complexity of software.
- Poor project planning, estimation and management.
- Lack of formal methods and discipline in development.
- Inadequate requirements gathering and changing customer needs.
- Insufficient testing and poor-quality documentation.

Changing Nature of Software

Software today falls into several broad categories. Each category has its own style of development and challenges.

Category	Description	Examples
System software	Collection of programs written to service other programs; heavy interaction with hardware, concurrent operation, complex data structures.	Operating systems, compilers, device drivers, editors, file-management utilities
Application software	Stand-alone programs that solve a specific business or technical need.	Payroll, point-of-sale, transaction processing
Engineering / Scientific software	Number-crunching algorithms; moving toward real-time and system-software features.	CAD, simulation, astronomy, molecular biology

Category	Description	Examples
Embedded software	Resides inside a product or system; implements limited and esoteric functions or control capabilities.	Microwave oven keypad, automobile fuel control, digital watch
Product-line software	Designed for use by many customers; targets mass consumer market or limited vertical markets.	Word processors, spreadsheets, multimedia, inventory control
Web applications (WebApps)	Network-centric software; from simple hypertext pages to complex computing functions integrated with corporate databases.	E-commerce sites, social networking, online banking
Artificial Intelligence software	Uses non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.	Expert systems, pattern recognition, neural networks, robotics, game playing

New Challenges and Trends

- **Open-world computing:** Ubiquitous, wireless computing where devices connect and exchange data automatically.
- **Netsourcing:** The web as a computing engine as well as content provider (software as a service).
- **Open source:** Software distributed with source code so that customers can modify and improve it.
- **Legacy software:** Old systems that are critical to business but evolve poorly; they must be adapted to new environments, enhanced for new needs, extended for interoperability and re-architected.

Software Myths

Myths are misleading beliefs that cause serious problems for managers and practitioners.

Type	Myth	Reality
Management	We already have a book of standards, so the team will follow them.	Standards are rarely used or up to date; practices must be adopted and maintained.
Management	If we fall behind schedule, add more programmers.	Adding people late makes a project later (Brooks' Law).
Customer	A general statement of objectives is enough to start coding.	Poor up-front definition is a major cause of failure; detailed requirements are needed.
Customer	Requirements change, but software is flexible.	Late changes cost far more; impact grows as the project proceeds.
Practitioner	Once the program works, our job is done.	60-80% of the effort is spent after delivery (maintenance).
Practitioner	Quality can be assessed only when the program is running.	Formal technical reviews can find defects much earlier.

1.2 Software Engineering – A Layered Technology Approach

Definition of Software Engineering

The **IEEE definition**: 'Software Engineering is (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; and (2) the study of approaches as in (1).'

Fritz Bauer defined it as: 'the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.'

In short: Software engineering = engineering discipline + software. It aims to produce **high-quality software, on time, within budget**, that satisfies user requirements and is maintainable.

Need for Software Engineering

- To manage the **large size** and **complexity** of modern software.
- To control **cost** and meet **schedules**.
- To improve **quality**, reliability and security.
- To support **scalability** and **maintainability** (software evolves).
- To manage **teamwork** and communication among many developers.
- To reduce the effects of **changing requirements**.

Software Engineering is a Layered Technology

Software engineering is organised as a stack of layers. Each layer rests on the one below it, and the foundation is an organisational commitment to **quality**.

Layer	Meaning	Explanation
4. Tools	Automated or semi-automated support for process and methods.	When tools are integrated so that information created by one tool can be used by another, a system for software development support called CASE (Computer-Aided Software Engineering) is established.
3. Methods	Technical 'how-to's for building software.	Include communication, requirements analysis, design modelling, program construction, testing and support. Rely on modelling activities and descriptive techniques.
2. Process	The foundation that holds the layers together.	Defines a framework of key process areas; forms the basis for management control of projects, and establishes the context in which technical methods are applied, work products produced, milestones established, quality ensured and change managed.
1. Quality focus	The bedrock.	Any engineering approach must rest on an organisational commitment to quality - total quality management, Six Sigma and similar philosophies foster continuous process improvement.

Explanation of Each Layer

- **A Quality Focus:** Culture of continuous improvement that leads to more mature approaches to software engineering.
- **Process:** The glue that holds technology layers together and enables rational and timely development. It defines the sequence in which methods are applied and the deliverables required.
- **Methods:** Provide the technical questions of 'how to' build software, covering a broad array of tasks from requirement gathering to maintenance.
- **Tools:** Provide automated or semi-automated support for the process and the methods, such as compilers, debuggers, version control, modelling tools and test tools.

Software Engineering Principles (Hooker's Seven Principles)

- 1 **The Reason It All Exists:** A software system exists to provide value to its users.
- 2 **KISS (Keep It Simple, Stupid):** All design should be as simple as possible, but no simpler.
- 3 **Maintain the Vision:** A clear vision is essential to the project's success.
- 4 **What You Produce, Others Will Consume:** Always specify, design and implement knowing someone else will have to understand and maintain it.
- 5 **Be Open to the Future:** Systems with a long lifetime have more value; design for change.
- 6 **Plan Ahead for Reuse:** Reuse saves time and effort.
- 7 **Think!** Placing clear, complete thought before action produces better results.

Generic Activities of the Software Engineering Process

Activity	Purpose
Communication	Heavy communication and collaboration with the customer and stakeholders; requirements gathering.
Planning	Creates a 'map' of the project: tasks, risks, resources, work products, and a schedule.
Modelling	Creates models (analysis and design) to better understand requirements and the design that will satisfy them.
Construction	Combines code generation (manual or automated) and the testing needed to uncover errors.
Deployment	Software is delivered to the customer who evaluates it and gives feedback.

1.3 A Process Framework & Software Project Tracking & Control

Software Process

A **software process** is a collection of activities, actions and tasks that are performed when some work product is to be created. An **activity** strives to achieve a broad objective (e.g., communication); an **action** (e.g., architectural design) is a set of tasks that produce a major work product; a **task** focuses on a small, well-defined objective (e.g., conducting a unit test) and produces a tangible outcome.

Process Framework

A **process framework** establishes the foundation for a complete software process by identifying a small number of **framework activities** that are applicable to all software projects regardless of size or complexity. It also includes a set of **umbrella activities** that are applicable across the entire software process.

Components of a Process Framework

- **Framework activities:** Communication, planning, modelling, construction and deployment.
- **Task sets:** Each framework activity has task sets (work tasks, work products, quality assurance points and project milestones) adapted to the project's needs.
- **Umbrella activities:** Span the whole process and are listed below.
- **Process flow:** Describes how activities, actions and tasks are organised with respect to sequence and time.

Process Flow Type	Description
Linear	Executes each of the five activities in sequence, beginning with communication and ending with deployment.
Iterative	Repeats one or more activities before moving to the next; each pass yields a more complete version.
Evolutionary	Executes the activities in a 'circular' manner; each circuit produces a more complete version of the software.
Parallel	Executes one or more activities in parallel with other activities (e.g., modelling of one aspect while construction of another).

Umbrella Activities

- **Software project tracking and control:** Assess progress against the plan and take action to maintain the schedule.
- **Risk management:** Assess risks that may affect the outcome or quality.
- **Software quality assurance (SQA):** Define and conduct activities to ensure software quality.
- **Technical reviews:** Assess work products to uncover and remove errors before they propagate.
- **Measurement:** Define and collect process, project and product measures.

- **Software configuration management (SCM):** Manage the effects of change throughout the process.
- **Reusability management:** Define criteria for work product reuse and establish mechanisms to achieve reusable components.
- **Work product preparation and production:** Create models, documents, logs, forms and lists.

Software Project Tracking and Control

Project tracking is the process of monitoring the actual progress of a project and comparing it with the plan. **Project control** means taking corrective action when the project deviates from the plan. Together they ensure that the project is delivered on time, within budget and with the required quality.

Objectives

- Know the current status of the project at any time.
- Detect deviations from schedule, cost and quality early.
- Take corrective actions and re-plan if needed.
- Provide visibility to management and customers.

Steps in Tracking and Control

- 1 Establish the baseline plan (work breakdown structure, schedule, budget, resources).
- 2 Collect progress data regularly - hours worked, tasks completed, defects found.
- 3 Compare actual progress against the planned values.
- 4 Analyse variances and identify the root causes.
- 5 Take corrective action - re-allocate resources, revise schedule, reduce scope.
- 6 Report status to stakeholders and update the plan.

Common Tracking Techniques

Technique	Description
Periodic status meetings	Regular team meetings to review progress, problems and risks.
Milestone review	Evaluate completion of major project checkpoints (e.g., SRS approved, design completed).
Gantt chart	Bar chart showing tasks against time; shows task durations and overlaps.
PERT / CPM	Network diagrams showing task dependencies; the critical path determines the minimum project duration.
Earned Value Analysis (EVA)	Compares the planned work with work actually completed in terms of cost and schedule.
Formal technical reviews	Walkthroughs and inspections to check quality of work products.
Defect tracking	Logging and following up bugs until closure.

Earned Value Analysis - Key Terms

Term	Meaning	Formula
BCWS / PV	Budgeted Cost of Work Scheduled (planned value)	Sum of budgeted effort of tasks scheduled up to date
BCWP / EV	Budgeted Cost of Work Performed (earned value)	Sum of budgeted effort of tasks actually completed
ACWP / AC	Actual Cost of Work Performed	Actual effort spent to date
SPI	Schedule Performance Index	$BCWP / BCWS$ (less than 1 means behind schedule)
CPI	Cost Performance Index	$BCWP / ACWP$ (less than 1 means over budget)
SV	Schedule Variance	$BCWP - BCWS$
CV	Cost Variance	$BCWP - ACWP$

Project Control Mechanisms

- **Change control:** Formal procedure for requesting, evaluating and approving changes.
- **Risk control:** Monitor risks and apply the contingency plan if a risk materialises.
- **Quality control:** Reviews, inspections and testing to maintain defined standards.
- **Configuration control:** Versioning and baselines so that all team members work on correct versions.

1.4 The Capability Maturity Model Integration (CMMI) Technique

Background

The **Capability Maturity Model (CMM)** was developed by the **Software Engineering Institute (SEI)** at Carnegie Mellon University, funded by the U.S. Department of Defense, to assess the process capability of software contractors. **CMMI** (Capability Maturity Model Integration) merged several separate models (the software CMM, systems engineering CMM and integrated product development CMM) into a single integrated framework for process improvement across organisations. It provides a comprehensive set of guidelines of **best practices** to improve an organisation's process capability.

Aims of CMMI

- Provide a measure of an organisation's process maturity.
- Identify strengths and weaknesses of the existing process.
- Offer a roadmap for continuous process improvement.
- Reduce risk in selecting software vendors by providing a standard rating.

Representations of CMMI

Representation	Approach	Rating Result
Staged	Organisation is evaluated as a whole; improvement follows a predefined sequence of maturity levels.	Maturity Level (1 to 5)
Continuous	Each process area is improved independently, chosen by business needs.	Capability Level (0 to 5) per process area

The Five Maturity Levels (Staged Representation)

Level	Name	Characteristics	Key Process Areas (Examples)
1	Initial	Processes are ad hoc, chaotic and unpredictable. Success depends on individual heroics. Schedules and budgets are frequently exceeded.	None defined
2	Managed	Projects are planned, performed, measured and controlled. Basic project management is established; requirements are managed.	Requirements Management, Project Planning, Project Monitoring and Control, Supplier Agreement Management, Measurement and Analysis, Process and Product Quality Assurance, Configuration Management

Level	Name	Characteristics	Key Process Areas (Examples)
3	Defined	Processes are well characterised, documented and understood; organisation-wide standard process tailored for each project.	Requirements Development, Technical Solution, Product Integration, Verification, Validation, Risk Management, Organisational Process Focus/Definition, Training, Integrated Project Management, Decision Analysis
4	Quantitatively Managed	Process and product quality are measured and controlled using statistical and quantitative techniques.	Organisational Process Performance, Quantitative Project Management
5	Optimizing	Continuous process improvement through quantitative feedback and innovative ideas and technologies.	Organisational Innovation and Deployment, Causal Analysis and Resolution

Capability Levels (Continuous Representation)

Level	Name	Meaning
0	Incomplete	Process is not performed or only partially performed.
1	Performed	Process achieves its specific goals but is not institutionalised.
2	Managed	Process is planned, monitored and controlled.
3	Defined	Process follows the organisation's tailored standard process.
4	Quantitatively Managed	Process is controlled using statistical techniques.
5	Optimizing	Process is continuously improved.

Process Areas, Goals and Practices

- **Process area:** A cluster of related practices in an area that, when performed collectively, satisfy a set of goals (22 process areas in CMMI-DEV v1.3).
- **Specific goals (SG) and specific practices (SP):** Apply to a single process area.
- **Generic goals (GG) and generic practices (GP):** Apply to multiple process areas and describe institutionalisation of the process.

Benefits and Limitations

Benefits	Limitations
Improved product quality and customer satisfaction	Implementation is costly and time-consuming
Predictable schedules and costs	Can be heavy for small organisations and start-ups
Reduced rework and defects	Risk of focusing on documentation instead of results

Benefits	Limitations
Recognised standard for appraisal and vendor selection	Needs strong top-management commitment

Poly Notes Hub

1.5 Process Patterns, Process Assessment, Personal & Team Process Models & Process Technology Theories

A. Process Patterns

A **process pattern** describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem occurs, and suggests one or more proven solutions. Patterns provide a consistent method for describing problem solutions within the context of the software process. Patterns can be defined at any level of abstraction - an entire process (e.g., prototyping), a framework activity (e.g., planning) or an action (e.g., requirements gathering).

Template of a Process Pattern (Ambler)

Element	Description
Pattern name	A meaningful name within the context of the software process.
Forces	Environment in which the pattern is encountered and issues that make the problem visible.
Type	Stage pattern, Task pattern or Phase pattern (see below).
Initial context	Conditions under which the pattern applies - prior activities, entry conditions, existing artefacts.
Problem	The specific problem to be solved by the pattern.
Solution	How to implement the pattern successfully.
Resulting context	Conditions that will result once the pattern has been successfully implemented.
Related patterns	Direct references to other process patterns.
Known uses and examples	Specific instances where the pattern is applicable.

Pattern Type	Description	Example
Stage pattern	Defines a problem associated with a framework activity for the process; includes multiple task patterns.	EstablishingCommunication
Task pattern	Defines a problem associated with a software engineering action or work task.	RequirementsGathering
Phase pattern	Define the sequence of framework activities that occur within the process, even when the flow is iterative.	SpiralModel, Prototyping

B. Process Assessment

Process assessment does not by itself guarantee quality, but it examines whether the organisation's software process is mature and effective, and identifies improvement areas. It is a **diagnostic** technique.

Common Assessment Approaches

Approach	Description
SCAMPI	Standard CMMI Appraisal Method for Process Improvement - provides a five-step process assessment model (initiate, diagnose, establish, act, learn); used with CMMI.
CBA IPI	CMM-Based Appraisal for Internal Process Improvement - diagnostic technique for assessing relative maturity of a software organisation; uses the SEI CMM.
SPICE (ISO/IEC 15504)	Defines a set of requirements for software process assessment; helps organisations develop an objective evaluation of the process.
ISO 9001:2000 for Software	Generic standard applicable to any organisation that wants to improve overall quality of products, systems or services.

Typical Steps in an Assessment

- 1 Plan the assessment and define scope.
- 2 Collect data through questionnaires, interviews and document review.
- 3 Analyse the data and rate against the reference model.
- 4 Report findings, strengths and weaknesses.
- 5 Prepare an action plan for process improvement.

C. Personal and Team Process Models

Software process models can be tailored to the individual (personal) and to groups (team). The two well-known models developed by **Watts Humphrey** at SEI are PSP and TSP.

Personal Software Process (PSP)

PSP emphasises personal measurement of both the work product that is produced and the resultant quality of the work product. It makes the practitioner responsible for project planning and empowers the practitioner to control the quality of all software work products developed.

PSP Activity	Description
Planning	Isolates requirements and develops both size and resource estimates; a defect estimate is made; tasks and schedule are recorded.
High-level design	External specifications for each component are developed; component design is created; prototypes built when uncertainty exists.
High-level design review	Formal verification methods are applied to uncover design errors; metrics are maintained for all important tasks.
Development	Component-level design is refined and reviewed; code is generated, reviewed, compiled and tested.
Postmortem	Using measures and metrics collected, the effectiveness of the process is determined; guides improvement.

Team Software Process (TSP)

The goal of TSP is to build a '**self-directed**' **project team** that organises itself to produce high-quality software. Humphrey's TSP objectives are:

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans (teams of 3 to about 20 engineers).
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behaviour normal and expected.
- Provide improvement guidance to high-maturity organisations.
- Facilitate university teaching of industrial-grade team skills.

TSP defines the following framework activities: launch, high-level design, implementation, integration and test, and postmortem. A key TSP feature is the **launch workshop**, where the team defines roles, goals, and plans, and the use of **scripts, forms and standards** to guide the work. Team roles include team leader, development manager, planning manager, quality/process manager and support manager.

Aspect	PSP	TSP
Focus	Individual engineer	Team of engineers
Main goal	Improve personal skills, estimation and quality	Build self-directed, high-performing teams
Key tools	Time, size and defect logs; personal scripts	Launch workshop, team roles, team plan, weekly meetings
Developed by	Watts Humphrey (SEI)	Watts Humphrey (SEI)
Prerequisite	Basic programming knowledge	PSP training for team members

D. Process Technology (Process Technology Theories)

Process technology refers to the tools and technologies used to describe, model, analyse and automate software processes. A software process can be represented as a **formal model** and then enacted, supported or improved by tools. Process technology theories explain how a software process can be understood, defined and controlled.

Key Ideas

- **Process modelling:** Describing the process using formal notations (workflow diagrams, state machines, process programming languages) so that it is precise and repeatable.
- **Process enactment / automation:** Executing the modelled process with support from tools. A **process-centred software engineering environment (PSEE)** guides developers, assigns tasks and enforces process rules.
- **Process measurement and analysis:** Gathering metrics to evaluate effectiveness and detect problems.

- **Process simulation:** Using models to predict schedule, cost and quality effects of process changes before applying them.
- **Process improvement cycles:** Continuous 'define - measure - analyse - improve' loops (e.g., PDCA, IDEAL model).

Technology	Role in Process
Process modelling tools	Draw and document the process workflow, roles and artefacts.
Workflow / BPM engines	Automate task routing, approvals and notifications.
CASE tools	Provide analysis, design, code and test support integrated into the process.
Version control and CI/CD	Control change, automate build, test and deployment.
Metrics and dashboard tools	Collect and visualise process and product measures.

1.6 Process Models – Waterfall, Incremental, RAD, Prototype, Spiral

Introduction

A **software process model** (also called a software development life cycle, or SDLC, model) is an abstract representation of a process. It prescribes the framework activities, their order and the flow between them. Choosing the right model depends on the type of project, requirement stability, team size, risk, and time constraints.

Generic Phases of the SDLC

- 1 **Requirement analysis** - collect and document what the system must do (SRS).
- 2 **System and software design** - architectural, data, interface and component design.
- 3 **Implementation (coding)** - convert the design into code.
- 4 **Testing** - verify and validate the software; find and fix defects.
- 5 **Deployment** - release the product to the customer.
- 6 **Maintenance** - correct errors, adapt to environment, and enhance.

1. Waterfall Model (Linear Sequential / Classic Life Cycle)

The waterfall model is the oldest paradigm. It suggests a systematic, sequential approach in which each phase must be completed before the next begins, with progress flowing downward like a waterfall. Output of one phase becomes input to the next. Original proposal: Winston Royce (1970).

Phases

- **Communication / Requirements:** Project initiation and requirements gathering.
- **Planning:** Estimating, scheduling and tracking.
- **Modelling:** Analysis and design.
- **Construction:** Code generation and testing.
- **Deployment:** Delivery, support and feedback.

Advantages	Disadvantages
Simple, easy to understand and manage	Real projects rarely follow a strictly sequential flow
Clearly defined stages and milestones	Requirements must be stable and fully known up front
Well-documented at each stage	Working software is available only late in the cycle
Works well for small projects with fixed requirements	High risk and uncertainty; difficult to go back to a previous phase
Easy to assign tasks and measure progress	Customer must be patient; poor for long and complex projects

Best suited for: projects with well-understood requirements, such as porting an existing system, or small maintenance projects with well-defined adaptations.

2. Incremental Process Model

The incremental model combines elements of the waterfall model applied in an iterative fashion. The product is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. The first increment is often a **core product** that addresses basic requirements; supplementary features are added in subsequent increments. Each increment delivers an operational product.

Features

- Requirements are divided into several builds or increments.
- Each increment passes through communication, planning, modelling, construction and deployment.
- Early increments act as prototypes; the customer can evaluate and give feedback.
- Useful when staffing is unavailable for a complete implementation by the deadline - early increments can be built with fewer people.

Advantages	Disadvantages
Working software is generated early	Needs good planning and design
More flexible; changes are less costly	Total cost may be higher than waterfall
Easier to test and debug in smaller iterations	Problems may arise from system architecture if not planned for all increments
Lowers initial delivery cost; risk is spread	Requires a well-defined module interface
Customer can respond to each build	Complete system requirements must still be known at start (broadly)

3. RAD Model (Rapid Application Development)

RAD is an incremental software process model that emphasises a **very short development cycle (typically 60 to 90 days)**. It is a 'high-speed' adaptation of the waterfall model in which rapid development is achieved by using **component-based construction**. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a 'fully functional system' within a very short time. Teams work in parallel, and each team builds one RAD module.

Phases of RAD

Phase	Description
Business modelling	Information flow among business functions is modelled: what information drives the process, who generates it, where it goes, and who processes it.
Data modelling	The information flow is refined into data objects needed to support the business; attributes and relationships are defined.

Phase	Description
Process modelling	Data objects are transformed to achieve the information flow necessary to implement a business function; processing descriptions created for add, modify, delete, retrieve.
Application generation	RAD uses fourth-generation techniques and reuses existing program components or creates reusable ones; automated tools build the software.
Testing and turnover	Since reuse is emphasised, many components are already tested; new components and all interfaces must be tested.

Advantages	Disadvantages
Very rapid delivery; reduced development time	Needs sufficient human resources for the multiple teams
Reusability of components	Requires strong commitment of developers and customers to rapid-fire activities
Quick initial reviews and early customer feedback	Not appropriate for systems that cannot be modularised
Encourages customer feedback	Not suited when technical risks are high or new technology is used
Increases productivity with fewer people over a short time	Highly skilled designers and modelling skills are needed

4. Prototype Model

Often a customer defines a set of general objectives for software but does not identify detailed requirements for functions and features. In such cases, the **prototyping paradigm** may offer the best approach. A **prototype** is a working model of the software with limited functionality, built quickly so that users can evaluate it and refine the requirements. The prototype can serve as 'the first system' - often 'thrown away'.

Steps in Prototyping

- 1 **Communication:** Meet the stakeholders, define overall objectives and identify known requirements.
- 2 **Quick plan:** Plan the prototyping iteration quickly.
- 3 **Modelling - quick design:** Focus on those aspects visible to the user (input approaches, output formats).
- 4 **Construction of prototype:** Build the prototype.
- 5 **Deployment, delivery and feedback:** Stakeholders evaluate the prototype and give feedback to refine requirements; iterate until requirements are agreed.

Type of Prototype	Description
Throwaway (rapid)	Built only to understand requirements; discarded afterward, then the real system is built.
Evolutionary	Prototype is refined repeatedly and eventually becomes the final system.
Incremental	Final product built as separate components; prototypes of components integrated.
Extreme (web)	Used for web applications: static pages, then screens with simulated services, then full services.

Advantages	Disadvantages
Users get a feel of the actual system early	Customer may think the prototype is the final product and demand quick delivery
Reduces risk of requirement misunderstanding	Developers may make implementation compromises (inefficient algorithms, poor tools) that become permanent
Missing functionality can be identified early	Poor documentation; excessive time may be spent on prototype
Increases user involvement and satisfaction	Can lead to scope creep
Good for unclear or changing requirements	Can be costly if many iterations are performed

5. Spiral Model

Proposed by **Barry Boehm (1986)**, the spiral model is an **evolutionary, risk-driven** process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. Software is developed in a series of evolutionary releases. During early iterations the release might be a model or prototype; during later iterations, increasingly more complete versions are produced. The model is divided into a number of framework activities (task regions), and the software team performs these activities in a clockwise spiral, starting at the centre. Each pass around the spiral includes risk analysis.

Four Quadrants / Task Regions of Each Loop

Quadrant	Activity	Details
1	Objective setting (Planning)	Determine objectives, alternatives and constraints for this phase.
2	Risk analysis and resolution	Identify and evaluate risks; resolve them via prototyping, simulation, benchmarking, etc.
3	Development and validation (Engineering)	Develop and verify the next-level product using a suitable model.
4	Evaluation and planning of next phase	Customer evaluates the result; plan the next loop of the spiral.

- **Radial dimension:** Cumulative cost incurred to date.

- **Angular dimension:** Progress made in completing each cycle of the spiral.
- The first circuit might lead to a concept development project; later circuits lead to new product development, then product enhancement projects.

Advantages	Disadvantages
Risk handling is explicit; best for high-risk projects	Can be costly; requires considerable risk-assessment expertise
Suitable for large and complex systems	Not suitable for small or low-risk projects
Changes can be accommodated at each loop	Complex to manage; end of project may be hard to define
Customer feedback obtained regularly	Success depends heavily on risk analysis quality
Software evolves as the project progresses	Heavy documentation and process overhead

Comparison of Process Models

Feature	Waterfall	Incremental	RAD	Prototype	Spiral
Requirements	Well known, stable	Mostly known	Well known, modular	Unclear / evolving	Evolving, unclear
Flow	Linear sequential	Iterative increments	Parallel, component-based	Iterative prototypes	Iterative, risk-driven
Risk handling	Poor	Moderate	Moderate	Moderate	Excellent
User involvement	Low (at start/end)	Medium	High	Very high	High
First delivery	Late	Early	Very fast (60-90 days)	Prototype early	After first loop
Cost	Low	Moderate	Moderate to high	Moderate	High
Documentation	Heavy	Medium	Light	Light	Heavy
Best for	Small, fixed-requirement projects	Staged delivery of large products	Time-critical, modular systems	Unclear requirements, UI-heavy	Large, high-risk projects

Choosing a Process Model - Quick Guide

- Requirements clear and stable: **Waterfall**.
- Need early partial delivery: **Incremental**.
- Very short time and modular design: **RAD**.
- Requirements unclear, need user feedback: **Prototype**.
- High risk, large and complex project: **Spiral**.

Multiple Choice Questions

Choose the correct option for each question. The answer key is given on the next page.

Q1. Software is best described as:

- (a) A physical product that wears out
- (b) A logical entity that is developed or engineered
- (c) A manufactured hardware component
- (d) A set of spare parts

Q2. Which of the following is NOT a characteristic of software?

- (a) Software does not wear out
- (b) Software is developed, not manufactured
- (c) Software has spare parts
- (d) Most software is custom built

Q3. The term 'software engineering' was first introduced at the conference held in:

- (a) 1968 (NATO)
- (b) 1950 (ACM)
- (c) 1986 (IEEE)
- (d) 1999 (SEI)

Q4. The bedrock (foundation) of the software engineering layered technology is:

- (a) Tools
- (b) Methods
- (c) Process
- (d) Quality focus

Q5. Integrated tools that support software development are called:

- (a) CASE
- (b) CMMI
- (c) SCAMPI
- (d) PSP

Q6. Which is NOT a generic framework activity?

- (a) Communication
- (b) Planning
- (c) Marketing
- (d) Deployment

Q7. Umbrella activities are:

- (a) Applied only during coding
- (b) Applied only during testing
- (c) Applied throughout the software process
- (d) Applied only during maintenance

Q8. In Earned Value Analysis, SPI is calculated as:

- (a) $BCWP / BCWS$
- (b) $BCWS / BCWP$
- (c) $BCWP / ACWP$
- (d) $ACWP / BCWP$

Q9. In CMMI staged representation, Level 3 is called:

- (a) Managed
- (b) Defined
- (c) Quantitatively Managed
- (d) Optimizing

Q10. Which CMMI maturity level focuses on continuous process improvement?

- (a) Level 2
- (b) Level 3
- (c) Level 4
- (d) Level 5

Q11. The Personal Software Process (PSP) was developed by:

- (a) Barry Boehm
- (b) Watts Humphrey
- (c) Winston Royce
- (d) Ivar Jacobson

Q12. Which model emphasises a very short development cycle of 60 to 90 days?

- (a) Waterfall
- (b) Spiral
- (c) RAD
- (d) Prototype

Q13. Which process model is risk-driven and was proposed by Barry Boehm?

- (a) Waterfall
- (b) Spiral
- (c) Incremental
- (d) RAD

Q14. Which model is most suitable when requirements are well understood and stable?

- (a) Prototype
- (b) Spiral
- (c) Waterfall
- (d) RAD

Q15. In the incremental model, the first increment is usually:

- (a) The final product
- (b) The core product
- (c) A throwaway prototype
- (d) The documentation set

Poly Notes Hub

Answer Key

Q. No.	Answer	Explanation
1	b	Software is logical, developed or engineered.
2	c	Software has no spare parts; hardware does.
3	a	NATO conference, 1968.
4	d	Quality focus is the foundation.
5	a	CASE = Computer-Aided Software Engineering.
6	c	Marketing is not a framework activity.
7	c	Umbrella activities span the entire process.
8	a	$SPI = BCWP / BCWS$.
9	b	Level 3 = Defined.
10	d	Level 5 = Optimizing.
11	b	Watts Humphrey developed PSP and TSP.
12	c	RAD targets 60-90 days.
13	b	Spiral model by Boehm (1986).
14	c	Waterfall suits stable requirements.
15	b	The core product is delivered first.

Broad Questions

- 1 Define software and explain its characteristics. Differentiate between hardware and software with a suitable table.
- 2 Explain the evolving role of software. Describe the various categories (changing nature) of software with examples.
- 3 What are software myths? Explain management, customer and practitioner myths with their realities.
- 4 Explain software engineering as a layered technology with a neat diagram. Describe the role of each layer.
- 5 What is a process framework? Explain framework activities and umbrella activities in detail.
- 6 Explain software project tracking and control. Describe the techniques used, including Earned Value Analysis.
- 7 What is CMMI? Explain the five maturity levels of the staged representation with key process areas.
- 8 Explain process patterns and process assessment. Describe PSP and TSP with their activities and differences.
- 9 Explain the Waterfall model with a diagram. State its advantages, disadvantages and situations where it is suitable.
- 10 Compare Incremental, RAD, Prototype and Spiral models. Explain the Spiral model in detail with its advantages and disadvantages.

Tip: In exams, support each answer with neat diagrams, tables and examples to score better.