

SOFTWARE ENGINEERING

Chapter 4

Software Project Management

Detailed Study Notes with MCQs and Broad Questions

For Engineering Students

Contents

4.1 The Management Spectrum – The People, The Product, The Process & The Project

4.2 Project Scheduling – Basic Concepts, People and Effort, Effort Distribution,
Defining a Task, Task Network & Scheduling

4.3 Risk Management – Reactive vs Proactive Strategies, Software Risks,
Identification, Projection, Refinement, Monitoring & Management

4.4 Change Management – SCM Scenario, SCM Repository & Process

4.5 Formal Methods, Cleanroom Software Development & Management Approach

Multiple Choice Questions (15)

Answer Key

Broad Questions (10)

4.1 The Management Spectrum – People, Product, Process & Project

Introduction to Software Project Management

Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development and support of computer software. It is the discipline of **planning, organising, monitoring and controlling** software projects so that they are delivered on time, within budget and with the required quality. Effective management focuses on four important elements, popularly called the **4 Ps: People, Product, Process and Project**, in that order. The order is important: people are the most valuable resource, the product defines what is to be built, the process provides the road map, and the project is the whole undertaking.

P	Focus	Key Question
People	The stakeholders, team leaders and team organisation	Who will do the work and how will they be organised?
Product	Scope and decomposition of the problem	What is to be built?
Process	Framework activities and process model	How will the work be done?
Project	Planning, monitoring and control	How will we deliver successfully?

A. The People

The 'people factor' is so important that the Software Engineering Institute developed the **People Capability Maturity Model (People-CMM)**, which states that every organisation needs to continually improve its ability to attract, develop, motivate, organise and retain the workforce needed to accomplish its strategic business objectives. Key practice areas include staffing, communication and coordination, work environment, performance management, training, compensation, competency analysis, career development, workgroup development, team/culture development, and others.

The Stakeholders

Stakeholder	Role
Senior managers	Define business issues that often have significant influence on the project.
Project (technical) managers	Plan, motivate, organise and control the practitioners who do the software work.
Practitioners	Deliver the technical skills necessary to engineer a product or application.
Customers	Specify the requirements for the software to be engineered; other stakeholders who have a peripheral interest in the outcome.
End users	Interact with the software once it is released for production use.

Team Leaders - the MOI Model (Jerry Weinberg)

Element	Meaning
Motivation	The ability to encourage (by 'push or pull') technical people to produce to their best ability.
Organization	The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
Ideas or innovation	The ability to encourage people to create and feel creative, even when they must work within bounds established for a particular software product.

Characteristics of an effective project manager (Edgemon): **Problem solving**, **managerial identity** (control and take charge), **achievement** (reward initiative), and **influence and team building** (read people, understand verbal and non-verbal signals, stay under pressure).

The Software Team - Organisational Paradigms (Mantei)

Paradigm	Description	Best Suited For
Democratic decentralised (DD)	No permanent leader; task coordinators are appointed for short durations and then replaced; decisions by group consensus; horizontal communication.	Difficult problems, modular, long-lived projects
Controlled decentralised (CD)	A defined leader coordinates specific tasks and secondary leaders are responsible for subtasks; problem solving remains a group activity; communication is horizontal and vertical.	Mid-sized problems
Controlled centralised (CC)	Top-level problem solving and internal team coordination are managed by the team leader; communication is vertical.	Simple, well-understood, short-duration projects

Factors Affecting Team Structure

- Difficulty of the problem to be solved.
- Size of the resultant program(s) in lines of code or function points.
- The time that the team will stay together (team lifetime).
- The degree to which the problem can be modularised.
- The required quality and reliability of the system.
- The rigidity of the delivery date.
- The degree of sociability (communication) required for the project.

Coordination and Communication Issues

Projects fail because of the scale of the project (large, complex), uncertainty (requirements change), interoperability, and the inability to communicate. Coordination is achieved through **formal, impersonal approaches** (documents, milestones, memos, project bulletins, review meetings, change requests, error tracking reports) and **formal, interpersonal procedures** (quality assurance activities such as reviews and design inspections) and **informal, interpersonal procedures** (group meetings and collocation), **electronic communication**, and **interpersonal networking**.

The Agile Team

Agile teams are small, self-organising, cross-functional teams. They have trust, high skill levels, mutual respect and a common focus, with team members making local decisions. They are suitable when requirements are volatile.

B. The Product

Before a project can be planned, **product objectives and scope** should be established, alternative solutions should be considered, and technical and management constraints should be identified. Without this information, it is impossible to define reasonable estimates of cost, an effective assessment of risk, a realistic breakdown of project tasks, or a manageable project schedule.

Software Scope

- **Context:** How does the software to be built fit into a larger system, product or business context, and what constraints are imposed as a result?
- **Information objectives:** What customer-visible data objects are produced as output? What data objects are required for input?
- **Function and performance:** What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be **unambiguous and understandable** at both managerial and technical levels.

Problem Decomposition

Problem decomposition (partitioning) is applied in two areas: (1) the functionality and content that must be delivered, and (2) the process that will be used to deliver it. The problem is divided into smaller, more manageable parts - a divide-and-conquer strategy. Decomposition continues until all functions or problem content can be estimated with reasonable accuracy.

C. The Process

A software process provides the framework from which a comprehensive plan for software development can be established. A small number of framework activities are applicable to all software projects, and different task sets enable the activities to be adapted to project characteristics and team requirements. Umbrella activities such as software quality assurance, configuration management and measurement overlay the process model.

Steps for Choosing the Process

- 1 Decide on the **process model** that is most appropriate for the customers and developers, the characteristics of the product, and the project environment.
- 2 Define a **preliminary project plan** based on the framework activities.
- 3 Perform **process decomposition**: the framework activity is decomposed into a set of tasks, each with a work product and schedule. Example: the communication activity might involve customer meeting, requirements gathering, preparation of a statement of scope and review.

Project Characteristic	Likely Process Model
Well-defined requirements, small project	Waterfall
Need early partial delivery	Incremental
Unclear requirements	Prototype
High risk, large project	Spiral
Volatile requirements, small team	Agile (Scrum / XP)

D. The Project

To avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a commonsense approach for planning, monitoring and controlling the project.

Common Reasons for Project Failure (Reel)

- Software people do not understand their customer's needs.
- The product scope is poorly defined.
- Changes are managed poorly.
- The chosen technology changes.
- Business needs change or are ill-defined.
- Deadlines are unrealistic.
- Users are resistant.
- Sponsorship is lost or never properly obtained.
- The project team lacks people with appropriate skills.
- Managers and practitioners avoid best practices and lessons learned.

The W5HH Principle (Barry Boehm)

Boehm suggests an approach that addresses project objectives, milestones and schedule, responsibilities, management and technical approaches, and required resources using a series of questions called the **W5HH principle**.

Question	Purpose
Why is the system being developed?	Do all stakeholders agree on the business reasons? Is it worth spending the money?
What will be done?	Identify the functionality to be built.
When will it be accomplished?	Establish milestones and a schedule.
Who is responsible for each function?	Define the role and responsibility of each team member.

Question	Purpose
Where are they organisationally located?	Customers, users and other stakeholders also have responsibilities.
How will the job be done technically and managerially?	Define the management and technical strategy.
How much of each resource is needed?	Develop estimates of effort, tools and people.

Critical Practices (Airlie Software Council)

- Formal risk management.
- Empirical cost and schedule estimation.
- Metrics-based project management.
- Earned value tracking.
- Defect tracking against quality targets.
- People-aware program management.

4.2 Project Scheduling

A. Basic Concepts

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is a mapping of tasks, people and time. Fred Brooks asked, 'How does a large software project get to be one year late?' and the answer was: 'One day at a time!' Scheduling aims to prevent such small slips.

Reasons Why Software Is Delivered Late (Reel)

- An unrealistic deadline established by someone outside the software development group.
- Changing customer requirements that are not reflected in schedule changes.
- An honest underestimate of the amount of effort and/or the number of resources that will be required.
- Predictable and/or unpredictable risks that were not considered when the project commenced.
- Technical difficulties that could not have been foreseen in advance.
- Human difficulties that could not have been foreseen in advance.
- Miscommunication among project staff that results in delays.
- A failure by project management to recognise that the project is falling behind schedule and a lack of action to correct the problem.

Basic Principles of Project Scheduling

Principle	Explanation
Compartmentalisation	The project must be compartmentalised into a number of manageable activities and tasks (product and process are decomposed).
Interdependency	The interdependency of each compartmentalised activity or task must be determined; some tasks occur in sequence and others in parallel.
Time allocation	Each task must be allocated some number of work units (person-days or person-weeks), a start date and an end date.
Effort validation	Every project has a defined number of people on the team; the project manager must ensure that no more than the allocated number of people are scheduled at any given time.
Defined responsibilities	Every task that is scheduled should be assigned to a specific team member.
Defined outcomes	Every task should have a defined outcome (work product, usually a deliverable or part of one).
Defined milestones	Every task or group of tasks should be associated with a project milestone, accomplished when one or more work products has been reviewed for quality and approved.

B. Relationship Between People and Effort

In small software projects, a single person can analyse requirements, perform design, generate code and conduct tests. As the size of a project increases, more people must become involved. A common myth is that if a project is behind schedule, adding people will catch up. The reverse is often true: **adding people late in a project often makes it later (Brooks' Law)**. This happens because people who are added must learn the system, and the people who teach them are the same people who were doing the work; and as the number of people grows, the **communication paths** increase. If there are N people, the number of possible communication paths is $N(N - 1)/2$. For example, 4 people have 6 paths, 8 people have 28 paths, and 10 people have 45 paths.

Team Size (N)	Communication Paths $N(N-1)/2$
3	3
5	10
8	28
10	45
20	190

Putnam-Norden-Rayleigh (PNR) Curve

The relationship between the number of delivered lines of code (L), effort (E) and development time (t) is given by the **software equation** (Putnam): $L = P \times E^{1/3} \times t^{4/3}$, where P is a productivity parameter reflecting process maturity and management practices, team skills and the complexity of the software. Rearranging, $E = L^3 / (P^3 t^4)$. The key insight: a small **extension of the delivery date** can reduce effort dramatically, while compressing the schedule increases effort sharply. The PNR curve shows that effort is a function of time (rises, peaks and falls) and that there is an **impossible region** where the schedule is compressed beyond about 75 percent of the nominal time.

Concept	Meaning
Brooks' Law	Adding manpower to a late software project makes it later.
Optimal time	The project duration at which effort is minimum.
Impossible region	Duration less than about 75 percent of nominal; the project cannot be completed.
Rayleigh curve	Staffing build-up, peak and tail-off over the project life.

C. Effort Distribution

A recommended distribution of effort across the software process is often referred to as the **40-20-40 rule**: 40 percent of effort on front-end analysis and design, 20 percent on coding, and 40 percent on back-end testing. This is only a guideline; the characteristics of each project dictate the distribution.

Activity	Typical Effort (Pressman)
Planning (customer communication and planning)	About 2-3 percent
Requirements analysis	10-25 percent
Design	20-25 percent

Activity	Typical Effort (Pressman)
Coding	15-20 percent
Testing and debugging	30-40 percent

- Work expended on **design** can be correlated to the quality of the finished product; well-designed software needs less coding and testing.
- Coding is a small part (15-20 percent) compared with the rest of the activities.
- Testing and debugging effort is highest; error removal gets more expensive as the project proceeds.

D. Defining a Task for the Software Project

A **task set** is a collection of software engineering work tasks, milestones, work products and quality assurance filters that must be accomplished to complete a particular project. The task set must provide enough discipline to achieve high software quality, but not burden the project team with unnecessary work.

Types of Software Projects

Project Type	Description
Concept development	Initiated to explore some new business concept or application of some new technology.
New application development	Undertaken as a consequence of a specific customer request.
Application enhancement	Occur when existing software undergoes major modifications to function, performance or interfaces visible to the end user.
Application maintenance	Correct, adapt or extend existing software in ways that may not be immediately obvious to the end user.
Reengineering	Undertaken with the objective of rebuilding an existing (legacy) system in whole or in part.

Degree of Rigor

Degree of Rigor	Description
Casual	All framework activities are applied, but only a minimum task set is required; documentation is minimal.
Structured	Framework activities and related tasks appropriate to the project type are applied; quality assurance and umbrella activities are applied.
Strict	The full process is applied for a project with a high-quality requirement; umbrella activities are robust; complete documentation.
Quick reaction	Framework activities are applied, but because of an emergency only those tasks essential to maintain good quality are applied; 'back-filling' occurs later.

Selecting Software Engineering Tasks - Example

Consider the **concept development** project type. The typical task set is: (1.1) concept scoping - determine overall project scope; (1.2) preliminary concept planning - establish the ability to undertake the work implied by scope; (1.3) technology risk assessment - evaluate the risks associated with the technology; (1.4) proof of concept - demonstrate the feasibility of the technology; (1.5) concept implementation - implement the concept representation in a manner that can be reviewed by the customer; (1.6) customer reaction - solicit customer feedback on the concept.

E. Defining a Task Network and Scheduling the Project

An individual task and subtask have interdependencies based on their sequence. In addition, when more than one person is involved, development activities and tasks can occur concurrently. A **task network** (activity network) is a graphic representation of the task flow for a project; it depicts the tasks, their dependencies and the order in which they occur. A task network is useful for determining **concurrent tasks** and for identifying the **critical path**.

Scheduling Methods and Charts

Method	Description
PERT (Program Evaluation and Review Technique)	Network-based, probabilistic; uses optimistic (a), most likely (m) and pessimistic (b) time estimates. Expected time $t_e = (a + 4m + b) / 6$.
CPM (Critical Path Method)	Network-based, deterministic; one time estimate per task; identifies the critical path.
Timeline (Gantt) chart	Bar chart with tasks listed on the vertical axis and calendar dates on the horizontal axis; shows task start, finish, duration and overlap.
Tracking table	A project table listing tasks, planned and actual start and end dates, effort and notes.

Critical Path

The **critical path** is the longest path through the task network from start to finish; it determines the minimum project duration. Tasks on the critical path have **zero slack (float)**: any delay in them delays the whole project. $\text{Slack} = \text{LS} - \text{ES} = \text{LF} - \text{EF}$ (latest minus earliest times).

Terms Used in CPM / PERT

Term	Meaning
ES / EF	Earliest start / earliest finish time of a task; $\text{EF} = \text{ES} + \text{duration}$.
LS / LF	Latest start / latest finish time without delaying the project; $\text{LS} = \text{LF} - \text{duration}$.
Slack (float)	Amount of time a task can be delayed without delaying the project.
Milestone	A point in the schedule that marks completion of a significant deliverable.
Critical task	A task with zero slack.

Worked Example

A project has six tasks. Durations are in weeks.

Task	Duration	Predecessor	ES	EF	LS	LF	Slack
A	3	-	0	3	0	3	0
B	4	A	3	7	3	7	0
C	2	A	3	5	7	9	4
D	5	B	7	12	7	12	0
E	3	C	5	8	9	12	4
F	2	D, E	12	14	12	14	0

Path A-B-D-F = 3 + 4 + 5 + 2 = 14 weeks; path A-C-E-F = 3 + 2 + 3 + 2 = 10 weeks. The **critical path is A-B-D-F** and the project duration is **14 weeks**; tasks C and E have 4 weeks of slack.

Tracking the Schedule

- Conduct periodic project status meetings in which each team member reports progress and problems.
- Evaluate the results of all reviews conducted throughout the software engineering process.
- Determine whether formal project milestones have been accomplished by the scheduled date.
- Compare the actual start date to the planned start date for each project task.
- Meet informally with practitioners to obtain their subjective assessment of progress.
- Use earned value analysis to assess progress quantitatively.

4.3 Risk Management

Introduction

A **risk** is an uncertain event or condition that, if it occurs, has a negative effect on the project's objectives. Risk involves two characteristics: **uncertainty** (the risk may or may not happen; there are no 100 percent probable risks) and **loss** (if the risk becomes a reality, unwanted consequences or losses will occur). **Risk management** is the process of identifying, analysing, prioritising and controlling risks. Robert Charette's maxim: 'Your problem is not that you have risks; it is that you do not manage them.'

A. Reactive vs Proactive Risk Strategies

Aspect	Reactive Strategy	Proactive Strategy
Approach	'Don't worry, I'll think of something'; the team does nothing about risks until something goes wrong.	Begins long before technical work starts; risks are identified, assessed and prioritised in advance.
Action	Fire-fighting; the team flies into action to correct the problem rapidly.	A plan is developed to avoid risks or to manage them if they occur.
Timing	After the risk becomes a problem.	Before the project starts and throughout.
Outcome	Project is in jeopardy if the problems are serious; crisis management.	Controlled, predictable project; reduced impact.
Cost	Often higher, because corrective action is late.	Lower, because mitigation is planned.
Example	The key developer leaves and the team scrambles to find a replacement.	Cross-train developers and document knowledge so that the departure of one has limited impact.

B. Software Risks

Characteristics

- **Uncertainty:** the event that characterises the risk may or may not happen.
- **Loss:** if the risk becomes reality, unwanted consequences or losses will occur.

Categories of Risk by Nature

Category	Description	Examples
Project risks	Threaten the project plan; if they become real, the schedule will slip and costs will increase. Identify potential budgetary, schedule, personnel, resource, stakeholder and requirements problems.	Staff turnover, unrealistic schedule, customer changes

Category	Description	Examples
Technical risks	Threaten the quality and timeliness of the software to be produced. Identify potential design, implementation, interface, verification and maintenance problems.	Unproven technology, ambiguous specification, technical obsolescence, leading-edge technology
Business risks	Threaten the viability of the software to be built and often jeopardise the project or the product.	See sub-types below

Business Risk Type	Meaning
Market risk	Building an excellent product that no one really wants.
Strategic risk	Building a product that no longer fits into the overall business strategy.
Sales risk	Building a product that the sales force does not understand how to sell.
Management risk	Losing the support of senior management due to a change in focus or personnel.
Budget risk	Losing budgetary or personnel commitment.

Categories of Risk by Predictability (Charette)

Type	Description
Known risks	Can be uncovered after careful evaluation of the project plan, business and technical environment (e.g., unrealistic delivery date, lack of documented requirements, poor development environment).
Predictable risks	Extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).
Unpredictable risks	Can and do occur, but are extremely difficult to identify in advance.

C. Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

Generic vs Product-Specific Risks

- **Generic risks** are a potential threat to every software project.
- **Product-specific risks** can be identified only by those with a clear understanding of the technology, the people and the environment that is specific to the software that is to be built.

Risk Item Checklist

Risk Category	Questions / Factors
Product size	Risks associated with the overall size of the software to be built or modified.
Business impact	Risks associated with constraints imposed by management or the marketplace.

Risk Category	Questions / Factors
Stakeholder characteristics	Risks associated with the sophistication of the stakeholders and the developer's ability to communicate with them in a timely manner.
Process definition	Risks associated with the degree to which the software process has been defined and is followed by the development organisation.
Development environment	Risks associated with the availability and quality of the tools used to build the product.
Technology to be built	Risks associated with the complexity of the system to be built and the 'newness' of the technology that is packaged by the system.
Staff size and experience	Risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk Components and Drivers (US Air Force)

Component	Meaning
Performance risk	The degree of uncertainty that the product will meet its requirements and be fit for its intended use.
Cost risk	The degree of uncertainty that the project budget will be maintained.
Support risk	The degree of uncertainty that the resultant software will be easy to correct, adapt and enhance.
Schedule risk	The degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into four categories: **negligible, marginal, critical and catastrophic**.

D. Risk Projection (Risk Estimation)

Risk projection attempts to rate each risk in two ways: the **likelihood (probability)** that the risk is real, and the **consequences** of the problems associated with the risk, should it occur. The activities are: (1) establish a scale that reflects the perceived likelihood of a risk; (2) delineate the consequences of the risk; (3) estimate the impact of the risk on the project and the product; (4) assess the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing a Risk Table

A risk table provides a simple technique for risk projection. The team lists all risks, categorises each, estimates its probability and assesses its impact on a scale of 1 to 4. The table is sorted by probability and impact; high-probability, high-impact risks are at the top. A **cut-off line** is drawn: risks above the line are managed; risks below are re-evaluated.

Risk	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	Plan A
Larger number of users than planned	PS	30%	3	Plan B

Risk	Category	Probability	Impact	RMMM
Less reuse than planned	PS	70%	2	Plan C
End users resist system	BU	40%	3	Plan D
Delivery deadline will be tightened	BU	50%	2	Plan E
Funding will be lost	CU	40%	1	Plan F
Staff inexperienced	ST	30%	2	Plan G
Staff turnover will be high	ST	60%	2	Plan H

Category codes: PS = project size, BU = business impact, CU = customer characteristics, ST = staff size and experience. *Impact values:* 1 = catastrophic, 2 = critical, 3 = marginal, 4 = negligible (as used in Pressman's example table).

Assessing Risk Impact

- The **nature** of the risk indicates the problems that are likely if it occurs.
- The **scope** of a risk combines the severity (how serious was it?) with its overall distribution (how much of the project will be affected or how many stakeholders are harmed?).
- The **timing** of a risk considers when and for how long the impact will be felt.

Risk Exposure (RE)

RE = P x C, where P is the probability of occurrence of a risk and C is the cost to the project should the risk occur.

Example: Risk identified: only 70 percent of the software components scheduled for reuse will be integrated into the application; the remaining functionality will have to be custom developed. Total of 60 reusable components, each about 100 LOC, cost 14 per LOC. Probability P = 80 percent. If 30 percent of the 60 components (18 components) must be developed, that is $18 \times 100 = 1,800$ LOC, so $C = 1,800 \times 14 = 25,200$. Then **RE = 0.80 x 25,200 = 20,160**. The total risk exposure for all risks can be computed to help adjust the project cost estimate.

E. Risk Refinement

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor and manage. One way to do this is to represent the risk in **condition-transition-consequence (CTC)** format: *Given that (condition), then there is concern that (consequence).*

Example: General risk: 'Overall resources not sufficient.' Refined: *Given that* all reusable software components must conform to specific design standards and that some do not conform, *then* there is concern that (1) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, *resulting in* the need to custom engineer the remaining 30 percent of components. Other sub-conditions: (2) the reuse components may have no interface problems; (3) the staff may have lack of training.

F. Risk Mitigation, Monitoring and Management (RMMM)

Activity	Purpose	Example (High Staff Turnover)
Risk Mitigation (avoidance)	Steps taken to reduce the probability or impact of the risk before it occurs.	Meet with current staff to determine causes of turnover (poor working conditions, low pay, competitive job market); reduce those causes before the project starts; assume turnover will occur and develop techniques to ensure continuity; organise project teams so that information about each activity is widely dispersed; define documentation standards; conduct peer reviews of all work so more than one person is up to speed; assign a backup staff member for every critical technologist.
Risk Monitoring	Track the project to determine if the risk is becoming more or less likely; watch risk indicators; assess whether mitigation steps are working.	General attitude of team members based on project pressures; the degree to which the team has jelled; interpersonal relationships among team members; potential problems with compensation and benefits; availability of jobs within the company and outside it.
Risk Management (contingency planning)	Actions to be taken when mitigation has failed and the risk has become a reality.	If the critical person leaves, use the backup person; temporarily refocus resources and readjust the schedule; ensure that knowledge transfer has occurred.

The RMMM Plan

The **RMMM plan** documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan. Alternatively, each risk can be documented individually using a **Risk Information Sheet (RIS)**, which contains: risk ID, date, probability, impact, description, refinement/context, mitigation/monitoring, management/contingency/trigger, current status, originator, and assigned person. Risk monitoring is a continuous activity and the risk table is updated regularly.

Principles of Risk Management (SEI)

- **Global perspective:** view software risk within the context of a system and the business problem it is intended to solve.
- **Forward-looking view:** think about the risks that may arise in the future; establish contingency plans.
- **Open communication:** encourage all stakeholders and users to point out risks at any time.
- **Integrated management:** risk management should be integrated into the project management process.
- **Continuous process:** the team must remain vigilant throughout the software process.
- **Shared product vision:** a common vision of the product makes risk identification easier.

- **Teamwork:** pool the skills and knowledge of all stakeholders when risk management activities are conducted.

Steps in the Risk Management Process

- 1 Risk identification.
- 2 Risk analysis (projection and estimation).
- 3 Risk prioritisation (risk table, RE).
- 4 Risk refinement.
- 5 Risk mitigation, monitoring and management planning (RMMM).
- 6 Continuous risk monitoring and control.

4.4 Change Management – Software Configuration Management (SCM)

Introduction

Change is inevitable when computer software is built. Changes occur in requirements, technology, business conditions, budgets and schedules. **Software Configuration Management (SCM)** is a set of activities designed to **identify, control and report changes** as they occur during the software process and after the software is released. It is an umbrella activity applied throughout the software process. The goals of SCM are to maximise productivity by minimising mistakes caused by confusion when coordinating software development.

Software changes arise from: new business or market conditions; new customer needs that demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system; reorganisation or business growth or downsizing; budgetary or scheduling constraints.

Key Terms

Term	Meaning
Software Configuration Item (SCI)	Information that is created as part of the software engineering process: computer programs (source and executable), documentation (technical and user), and data (contained within the program or external to it).
Baseline	A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures (IEEE).
Version	A specific, identifiable instance of a configuration item at a given time.
Variant	A parallel version of an item built for different platform or customer needs.
Release	A version that is made available to customers or users.
Configuration	The set of all SCIs and their versions that define a product at a given time.
Change control board (CCB)	A group that reviews, approves or rejects change requests.

A. The SCM Scenario

A typical SCM operational scenario involves a **project manager** who is in charge of a software group, a **configuration manager** who is in charge of the configuration procedures and policies, the **software engineers** who are responsible for developing and maintaining the software product, and the **customer** who uses the product. The SCM scenario shows how SCM requirements affect each role.

Role	SCM Responsibility
Project manager	Ensures that a software product is developed within a certain time frame. Monitors the progress of development and recognises and reacts to problems. This is done by generating and analysing reports about the status of the software system and by performing reviews on the system.

Role	SCM Responsibility
Configuration manager	Ensures that procedures and policies for creating, changing and testing of code are followed, and makes information about the project accessible. Introduces mechanisms for making changes to the code, and controls access to the baseline, versions and configuration items.
Software engineers	Work to develop and maintain the software. Need tools to maintain control of their work: check out and check in items, merge and branch changes, and avoid conflicts with colleagues. Work in their own workspaces and synchronise with the repository.
Customer	Uses the product and requests changes. Follows the formal procedure for change requests and must be able to learn which change was incorporated in which release.

The scenario describes how the SCM system answers questions such as: What is the status of a change request? Who made a change and why? What other components will be affected by a change? What versions do we need to rebuild release 2.1?

B. The SCM Repository

In the early days of software engineering, software configuration items were maintained as paper documents in file folders. Today, SCIs are maintained in a **project database or repository**. The repository is a set of mechanisms and data structures that allow a software team to manage change in an effective manner. It provides the obvious functions of a modern database management system by ensuring data integrity, sharing and integration. In addition, it provides a hub for the integration of software tools, is central to the flow of the software process and can enforce uniform structure and format for software engineering work products.

Role of the Repository

- **Integrity:** validates entries, ensures consistency, cascades modifications.
- **Information sharing:** shares information among multiple developers and tools; manages and controls multiuser access to data.
- **Tool integration:** establishes a data model that can be accessed by many software engineering tools.
- **Data integration:** database functions allow various SCM tasks to be performed on one or more SCIs.
- **Methodology enforcement:** an E-R model stored in the repository implies a specific paradigm for software engineering.
- **Document standardisation:** defining objects in the database leads directly to a standard approach for the creation of software engineering documents.

Content of the Repository

Category	Examples
Business information	Business rules, business functions, organisation structure, information architecture.

Category	Examples
Business process models	Process definitions and process flow diagrams.
Software engineering work products	Requirements, models, design documents, user manuals.
Source code and executables	Programs and builds.
Test information	Test plans, test cases and test results.
Project management information	Project plans, schedules, estimates, risk tables.
Tools and references	Standards, templates, tool configurations.

Features of the Repository

Feature	Description
Versioning	As a project progresses, many versions of individual work products will be created. The repository must save all of these versions to enable effective management of product releases and to permit developers to go back to previous versions.
Dependency tracking and change management	The repository manages a wide variety of relationships among the data elements stored in it, including relationships between enterprise entities and processes, among the parts of an application design, between design components and the enterprise information architecture.
Requirements tracing	The ability to track all the design and construction components and deliverables that result from a specific requirement specification (forward tracing) and the requirement that spawned any given work product (backward tracing).
Configuration management	Keeps track of a series of configurations representing specific project milestones or production releases.
Audit trails	Establish additional information about when, why and by whom changes are made; information about changes can be entered as annotations attached to specific objects.

C. The SCM Process

The SCM process defines a series of tasks that have four primary objectives: (1) to identify all items that collectively define the software configuration; (2) to manage changes to one or more of these items; (3) to facilitate the construction of different versions of an application; and (4) to ensure that software quality is maintained as the configuration evolves over time. The process addresses these questions through five SCM tasks.

Task	Description
1. Identification of objects in the software configuration	Each object (basic object: a unit of information created during analysis, design, code or test; aggregate object: a collection of basic objects) is given a unique name , description , a list of resources and a realisation. Evolution graphs describe the change history.

Task	Description
2. Version control	Combines procedures and tools to manage different versions of configuration objects created during the software process. A version control system implements or is directly integrated with four major capabilities: a project database (repository), a version management capability, a make facility and an issues tracking (bug tracking) capability.
3. Change control	Combines human procedures and automated tools to provide a mechanism for the control of change.
4. Configuration auditing	Ensures that the change has been properly implemented. Done through formal technical reviews and the software configuration audit.
5. Status reporting (configuration status accounting)	Answers: What happened? Who did it? When did it happen? What else will be affected? Each time a change is approved, a CSA entry is made.

Change Control Process

For a large software engineering project, uncontrolled change rapidly leads to chaos. Change control combines human procedures and automated tools to provide a mechanism for the control of change. Steps:

- 1 A **change request** is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and projected cost of the change.
- 2 The results of the evaluation are presented as a **change report**, used by a **change control authority (CCA)** - a person or group that makes a final decision on the status and priority of the change.
- 3 An **engineering change order (ECO)** is generated for each approved change, describing the change to be made, the constraints and the criteria for review and audit.
- 4 The object to be changed is **checked out** of the project database; the change is made and appropriate SQA activities are applied.
- 5 The object is **checked in** to the database and appropriate version control mechanisms are used to create the next version.
- 6 Appropriate synchronisation controls (access control and synchronisation control) are applied: **access control** governs which software engineers have authority to access and modify a particular configuration object; **synchronisation control** helps to ensure that parallel changes performed by two different people do not overwrite one another.

The CCB (change control board) is typically composed of representatives from the customer, management, development, testing and SQA. For minor changes, a developer can have authority; for major changes, formal approval is needed.

Configuration Audit

The audit asks: Has the change specified in the ECO been made? Have any additional modifications been incorporated? Has a formal technical review been conducted to assess technical correctness? Has

the software process been followed and software engineering standards properly applied? Has the change been highlighted in the SCI, with the change date and author specified? Have SCM procedures for noting the change, recording it and reporting it been followed? Have all related SCIs been properly updated?

SCM Tools

Tool Type	Examples
Version control systems	Git, Subversion (SVN), Mercurial, Perforce
Build and CI tools	Jenkins, GitHub Actions, GitLab CI
Issue / change tracking	Jira, Bugzilla, GitHub Issues
Repository hosting	GitHub, GitLab, Bitbucket

4.5 Formal Methods, Cleanroom Software Development & Management Approach

A. Formal Methods

Formal methods are a set of mathematically based techniques for the specification, development and verification of software and hardware systems. Using discrete mathematics (set theory, logic, functions, sequences) and formal specification languages, a software engineer can specify the system precisely, prove properties and verify the correctness of a design or program. The object is to reduce the ambiguity, incompleteness and inconsistency that arise from natural-language specifications; as Pressman notes, **formal methods provide a basis for defect avoidance** and are essential for **safety-critical** systems (avionics, medical devices, nuclear control, railway signalling).

Concepts and Basis

- **Formal specification:** A system's behaviour is described using a language with precisely defined syntax and semantics (mathematical notation).
- **Formal verification (proof of correctness):** Mathematical proof that the implementation satisfies the specification.
- **Refinement:** The specification is gradually transformed into code using proven correctness-preserving steps.
- **Mathematical foundation:** Set theory (collections of objects), logic (predicates, quantifiers) and finite state machines.
- **Data invariants and pre/post-conditions:** An invariant is a condition that is always true; a precondition must hold before an operation; a postcondition must hold after it.

Formal Specification Languages

Language	Description
Z (Zed)	Based on set theory and first-order predicate logic; uses schemas to describe the state and operations.
VDM (Vienna Development Method)	Model-based method using sets, lists, maps and logic; supports refinement to code.
B-Method	Set-theory based; supports specification, design and code generation; used in railway systems.
OCL (Object Constraint Language)	Used with UML to specify constraints on models.
Algebraic specification (Larch, OBJ)	Describes the system via abstract data types and equations.
Petri nets, CSP, CCS	Used to model concurrency and communication.

Advantages and Disadvantages

Advantages	Disadvantages
Eliminates ambiguity, incompleteness and inconsistency	Requires a strong mathematical background; high training cost
Early defect discovery at specification stage	Time-consuming and expensive to apply to the entire system
Enables mathematical proofs of correctness	Difficult to use for communication with non-technical customers
Very high reliability for safety-critical systems	Hard to specify user interfaces and some non-functional requirements
Reduces testing and rework effort later	Not scalable to very large systems

B. Cleanroom Software Engineering

Cleanroom software engineering is a software development approach that emphasises **defect prevention rather than defect removal**. It was developed by **Harlan Mills** and his colleagues at IBM in the 1980s. The name comes from the 'clean rooms' of the semiconductor industry, where contamination is prevented rather than removed. The approach applies **formal methods** for specification and design, **correctness verification** through mathematical reasoning and team review in place of unit testing, and **statistical usage testing** to certify the reliability of the software. Cleanroom emphasises **doing it right the first time**: programmers do not execute their own code; it is verified by inspection and passed to an independent testing team.

Principles of Cleanroom Development

- **Software development based on formal methods:** box structure specification and design with correctness verification.
- **Incremental implementation under statistical quality control:** each increment is released only after its quality is measured and certified.
- **Statistically based testing:** tests are derived from the expected usage profile of the software.
- **Mathematical correctness verification** in place of unit testing by developers.

The Cleanroom Strategy - Tasks

Task	Description
Increment planning	A project plan using the incremental strategy is developed; the functionality of each increment, its projected size and a cleanroom development schedule are created.
Requirements gathering	A more detailed description of customer-level requirements (for each increment) is developed using conventional techniques.
Box structure specification	A specification method that uses box structures - black box, state box and clear box - to describe the functional specification.

Task	Description
Formal design	Using the box structure approach, cleanroom design is a natural and seamless extension of specification; the specifications (black boxes) are iteratively refined to become architectural and component-level designs (state boxes and clear boxes).
Correctness verification	The cleanroom team conducts a series of rigorous correctness verification activities on the design and then on the code.
Code generation, inspection and verification	The box structure specifications, represented in a specialised language, are translated into the appropriate programming language; walkthroughs or inspections ensure semantic conformance of code and box structures.
Statistical test planning	The projected usage of the software is analysed and a suite of test cases that exercise a 'probability distribution' of usage is planned and designed.
Statistical use testing	Test cases are derived from the probability distribution of usage rather than exhaustive testing.
Certification	Once verification, inspection and usage testing are complete (and all errors corrected), the increment is certified as ready for integration.

Box Structure Specification

Box	Description
Black box	Specifies the externally visible behaviour of the system; maps all stimuli (inputs) and the history of use into responses (outputs). Does not reveal internal state.
State box	Encapsulates state data and services (operations); represents the 'knowledge' of the black box about past history.
Clear box	Contains the transition function of the state box, i.e., the procedural design (structured programming constructs) of the state box.

Cleanroom Testing

Statistical use testing exercises the software by sampling test inputs from the expected usage **probability distribution**. Each use case is assigned a probability of occurrence. Test cases are then generated randomly in proportion to these probabilities, so the most frequently used functions are tested most. The failure data are recorded and fed into a reliability model (such as mean-time-to-failure, MTTF) to **certify** reliability quantitatively. Certification models include the sampling model, the component model and the certification model.

Cleanroom vs Conventional Development

Aspect	Conventional Development	Cleanroom
Philosophy	Defect removal through testing and debugging	Defect prevention through formal specification and verification
Specification	Natural language, informal models	Formal, box-structure specification
Unit testing	Done by programmers	Not done; replaced by correctness verification and inspection

Aspect	Conventional Development	Cleanroom
Testing	Coverage-based, exhaustive attempts	Statistical usage testing by an independent team
Quality measure	Defect counts	Statistical reliability certification (MTTF)
Development	Mostly big-bang or iterative	Incremental with certified increments
Debugging	Developers debug	Developers do not execute code; failures lead to rework through verification

Advantages and Limitations of Cleanroom

Advantages	Limitations
Very low defect rates in delivered software (reported 2-3 errors per KLOC)	Requires specialised training in formal methods
Better quality than conventional approaches; high productivity	Seen as theoretical; difficult to adopt widely
Reliability is measured and certified	Not suitable for volatile or poorly-understood requirements
Reduced testing and maintenance cost	Mathematics-based correctness proofs are tedious for large systems
Suited to safety-critical systems	Programmers' resistance to not executing their own code

C. Management Approach for Formal Methods and Cleanroom Projects

The use of formal methods and cleanroom engineering changes the way a project is managed. The manager must plan for more effort at the early stages (specification and design) and less at the testing stage, because defects are prevented rather than removed. The key management points are:

- **Team organisation:** Cleanroom projects use small, independent teams: a **specification team** (develops and maintains the specification), a **development team** (designs and codes using verification), and a **certification team** (develops statistical tests and certifies). These teams work separately to maintain independence.
- **Incremental pipeline:** Increments are developed in a pipeline: while one increment is being certified, the next is being developed and another specified. The manager tracks each increment through the pipeline.
- **Training and skills:** Developers need training in discrete mathematics, formal specification and verification techniques. The manager must schedule training and use the 'learning curve' in estimation.
- **Effort planning:** Front-loaded effort distribution (more in specification and design; less in testing and debugging). Productivity measured by certified increments.

- **Quality control:** Quality is controlled by verification reviews and by measured reliability (MTTF) rather than by number of bugs removed. Management sets the target reliability and the usage model.
- **Risk management:** The major risks are lack of mathematical skills, customer unfamiliarity with formal notation and resistance to change; these are mitigated by training, pilot projects and gradual adoption.
- **Change management:** Formal specifications are placed under SCM; any change in requirements must be re-specified and re-verified.
- **Where to apply:** Best used for critical components (safety, security, financial) rather than the whole system; a mixed approach is often cost-effective.

Management Area	Conventional Project	Formal Methods / Cleanroom Project
Effort focus	Heavy at testing and debugging (about 40 percent)	Heavy at specification and verification; reduced testing
Team structure	Analysts, programmers, testers in one team	Independent specification, development and certification teams
Progress measure	Lines of code, defects found	Verified and certified increments, MTTF
Major risk	Late discovery of defects	Skills gap, tool support, customer acceptance
Documentation	Varies	Formal specifications and proofs form the core documentation

Multiple Choice Questions

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

Q1. The four elements of the management spectrum (4 Ps) are:

- | | |
|--|--------------------------------------|
| (a) People, Product, Process, Project | (b) Plan, Product, Process, Profit |
| (c) People, Plan, Performance, Project | (d) Product, Price, Place, Promotion |

Q2. In the MOI model of a team leader, 'I' stands for:

- | | |
|-----------------|-------------------------|
| (a) Integrity | (b) Ideas or innovation |
| (c) Information | (d) Interface |

Q3. Which team structure has no permanent leader and uses consensus decisions?

- | | |
|------------------------------|------------------------------|
| (a) Controlled centralised | (b) Controlled decentralised |
| (c) Democratic decentralised | (d) Hierarchical |

Q4. Boehm's W5HH principle is used for:

- | | |
|---------------------|---|
| (a) Risk estimation | (b) Defining project objectives, milestones, responsibilities and resources |
| (c) Testing | (d) Configuration audit |

Q5. Brooks' Law states that:

- | | |
|---|--|
| (a) More people always speed up a project | (b) Adding people to a late project makes it later |
| (c) Testing takes 40 percent of effort | (d) Risk is proportional to size |

Q6. For a team of 8 people, the number of communication paths is:

- | | |
|--------|--------|
| (a) 16 | (b) 28 |
| (c) 36 | (d) 56 |

Q7. According to the 40-20-40 rule, how much effort goes to coding?

- | | |
|----------------|----------------|
| (a) 20 percent | (b) 40 percent |
| (c) 60 percent | (d) 10 percent |

Q8. The longest path through a task network, which determines minimum project duration, is the:

- | | |
|-------------------|-------------------|
| (a) Slack path | (b) Critical path |
| (c) Shortest path | (d) Float path |

Q9. Which strategy begins long before technical work starts and plans to manage risks in advance?

- | | |
|--------------|----------------|
| (a) Reactive | (b) Proactive |
| (c) Passive | (d) Corrective |

Q10. Risk exposure (RE) is calculated as:

- | | |
|-------------|------------------|
| (a) $P + C$ | (b) $P \times C$ |
| (c) P / C | (d) $C - P$ |

Q11. Building an excellent product that no one really wants is a:

- | | |
|--------------------|-----------------|
| (a) Strategic risk | (b) Sales risk |
| (c) Market risk | (d) Budget risk |

Q12. A baseline in SCM is:

- (a) A formally reviewed and agreed work product that can be changed only through formal change control
- (b) A rough draft document
- (c) A bug report
- (d) A user manual

Q13. Which of the following is NOT an SCM task?

- (a) Version control
- (b) Change control
- (c) Configuration auditing
- (d) Cost estimation

Q14. Cleanroom software engineering was developed by:

- (a) Harlan Mills
- (b) Watts Humphrey
- (c) Barry Boehm
- (d) Fred Brooks

Q15. Which of these is a formal specification language?

- (a) HTML
- (b) Z notation
- (c) SQL
- (d) CSS

Answer Key

Q. No.	Answer	Explanation
1	a	People, Product, Process, Project.
2	b	MOI = Motivation, Organization, Ideas or innovation.
3	c	Democratic decentralised teams use consensus.
4	b	W5HH answers Why, What, When, Who, Where, How, How much.
5	b	Adding people late increases communication overhead.
6	b	$N(N-1)/2 = 8 \times 7 / 2 = 28$.
7	a	Coding gets about 20 percent in the 40-20-40 rule.
8	b	The critical path has zero slack.
9	b	Proactive strategies plan ahead.
10	b	RE = probability x cost.
11	c	Market risk.
12	a	A baseline changes only through formal change control.
13	d	Cost estimation is a project planning task, not an SCM task.
14	a	Harlan Mills at IBM.
15	b	Z notation is based on set theory and logic.

Broad Questions

- 1 Explain the management spectrum (4 Ps) in software project management. Describe the people and product factors in detail.
- 2 Explain the team organisational paradigms and the factors that affect team structure. What is the MOI model?
- 3 Explain the process and project elements of the management spectrum. Describe the W5HH principle and critical practices.
- 4 Explain the basic concepts and principles of project scheduling. Why is software delivered late?
- 5 Explain the relationship between people and effort. State Brooks' Law and the Putnam-Norden-Rayleigh curve. Describe the effort distribution (40-20-40 rule).
- 6 What is a task network? Explain defining a task set, degree of rigor, PERT/CPM and the critical path with a suitable example.
- 7 Differentiate between reactive and proactive risk strategies. Explain the categories of software risks with examples.
- 8 Explain risk identification, risk projection (risk table and risk exposure) and risk refinement (CTC format). Describe the RMMM plan.
- 9 Explain the SCM scenario, SCM repository and the SCM process in detail. Describe the change control process.
- 10 What are formal methods? Explain cleanroom software engineering with its strategy, box structures, testing and the management approach.

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