Software Risk, Configuration Management

Q. Explain Risk Identification

Risk Identification is the **process of recognizing potential risks** that can impact the success of a software project. It is the first step in **Risk Management**, where risks are identified, analyzed, and addressed before they become major issues.

Importance of Risk Identification

✓ Helps in **early detection** of possible project failures.
✓ Reduces **costs and delays** caused by unexpected issues.
✓ Ensures **better project planning and management**.
✓ Improves **software quality and reliability**.

Types of Risks in Software Engineering

Risk Category	Description	Example
Project Risks	Risks related to **budget, schedule, and resources**.	**Project delay** due to developer shortage.
Technical Risks	Risks due to **technology limitations** or software complexity.	**Integration issues** between two systems.
Business Risks	Risks affecting the **business value** of the software.	**Low customer demand** for the product.
Operational Risks	Risks due to **process failures or security issues**.	**Data breach** in an e-commerce website.
External Risks	Risks due to **market changes, legal, or environmental factors**.	**Government regulations** requiring changes in software.

Steps for Risk Identification

Step 1: Brainstorming Sessions

Conduct discussions with the **development team, stakeholders, and clients** to list potential risks.

✓ **Example:** The team realizes that **third-party APIs** used in the project may become unavailable.

Step 2: Checklist Analysis

Use a **predefined checklist** of common risks to check for possible threats.

✓ **Example:** A checklist includes risks like **budget overruns, unclear requirements, or security vulnerabilities**.

Step 3: SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats)

Identify risks based on **internal weaknesses and external threats**.

✓ **Example:** A startup may have **strong innovation but limited funding**, making financial risks a concern.

Step 4: Past Project Analysis

Review **previous similar projects** to identify risks that occurred before.

✓ **Example:** A software company finds that **previous projects faced delays due to scope creep** (requirements increasing over time).

Step 5: Expert Judgment

Consult **experienced project managers and technical experts** to predict risks.

✓ **Example:** A cybersecurity expert warns about potential **data leakage** in the project.

Risk Identification Techniques

Technique	Description	Example
Delphi Method	Collects expert opinions anonymously to identify risks.	Experts predict that **server failures** may affect performance.
Checklist Method	Uses a predefined list of common risks.	**Budget overrun** is a listed risk.
Root Cause Analysis	Identifies the main causes of potential risks.	If **frequent code changes** cause defects, stricter version control is needed.
Assumption Analysis	Evaluates assumptions made during project planning.	Assumption: The **client will provide data on time** (risk: they might delay).

Q. What are the different types of risk

In software engineering, risks are **uncertain events** that can negatively impact a project's success. These risks can be related to **project management, technology, business, and operations**.

Major Types of Risks

Risk Type	Description	Example
Project Risks	Risks related to **budget, schedule, and resources**.	Developer shortage leads to **project delays**.
Technical Risks	Risks due to **technology limitations, software complexity, or integration issues**.	A new framework **fails to integrate** with existing software.
Business Risks	Risks affecting the **business value** of the software.	Customers may **not adopt** the new software.
Operational Risks	Risks due to **process failures, human errors, or security issues**.	A **data breach** in an e-commerce system.
External Risks	Risks due to **market changes, regulations, or environmental factors**.	Government **introduces new laws** requiring software modifications.

Detailed Explanation of Each Risk Type

1. Project Risks

Definition: Risks that affect project management, such as **budget overruns, missed deadlines, or resource unavailability**.

Examples:
✓ The **client frequently changes requirements**, causing delays.
✓ The **team lacks skilled developers**, slowing down progress.
✓ The **budget runs out before project completion**.

Impact:
✓ Project delays
✓ Increased costs
✓ Reduced software quality

---

2. Technical Risks

Definition: Risks caused by **technological issues**, such as software defects, performance problems, or system failures.

Examples:
✓ A **new programming language** used in the project turns out to be unstable.
✓ The **software cannot handle a large number of users** as expected.
✓ **Integration issues** arise when connecting third-party APIs.

Impact:
✓ Software crashes
✓ Security vulnerabilities
✓ Poor system performance

---

3. Business Risks

Definition: Risks affecting the business value or profitability of the software.

Examples:
✓ Customers **prefer competitor products** instead of the developed software.
✓ The software does not **generate expected revenue**.
✓ The client **cancels the project** midway.

Impact:
✓ Wasted resources
✓ Financial loss
✓ Failure to meet business goals

---

4. Operational Risks

Definition: Risks caused by **human errors, process failures, or security threats**.

Examples:
✓ **Human error** leads to accidental deletion of customer data.
✓ A hacker **exploits a security flaw**, causing a data breach.
✓ **Software updates fail**, making the system unstable.

Impact:
✓ Legal issues due to data breaches
✓ Customer dissatisfaction
✓ System downtime

---

5. External Risks

Definition: Risks caused by **factors beyond the company’s control**, such as legal changes, market shifts, or environmental disasters.

Examples:
✓ The government **introduces new regulations**, requiring software updates.
✓ A competitor launches a **better version of the product** before release.
✓ A **pandemic disrupts** the development process.

Impact:
✓ Unexpected software modifications
✓ Increased development costs
✓ Loss of market share

Q. Explain Risk Assessment

Risk Assessment is the **process of analyzing and evaluating identified risks** to understand their potential impact on a software project. It helps project managers **prioritize risks** and decide on mitigation strategies.

Steps in Risk Assessment

Step	**Description**	**Example**
1. Risk Identification	List all potential risks that may impact the project.	Possible risk: **Server crashes due to high traffic**.
2. Risk Analysis	Determine the **likelihood** and **impact** of each risk.	High impact: **Security breach in the application**.
3. Risk Prioritization	Rank risks based on their severity and probability.	Prioritize **risks that can delay project completion**.
4. Risk Mitigation Planning	Develop strategies to reduce the impact of risks.	Implement **data backups** to prevent data loss.
5. Risk Monitoring	Continuously track risks throughout the project lifecycle.	Use **risk logs** to monitor progress.

Risk Analysis Techniques

1. Qualitative Risk Analysis

Definition: Assesses risks based on **subjective judgment**, without numerical data.

Example: Using a **Risk Matrix** to classify risks as **Low, Medium, or High**.

Likelihood / Impact	**Low**	**Medium**	**High**
Low Probability	Minor issue	Manageable	Acceptable only with mitigation
High Probability	Manageable	Needs immediate action	Critical! Must be addressed ASAP

---

2. Quantitative Risk Analysis

Definition: Uses **numerical data and mathematical models** to calculate risk impact.

Example: Expected Monetary Value (EMV) Calculation
EMV = **Probability of Risk × Estimated Cost of Impact**

Example Calculation:

- **Risk:** Server failure
- **Probability:** 30% (**0.3**)
- **Impact Cost:** ₹50,000

EMV = 0.3 × ₹50,000 = ₹15,000
This means the organization should allocate ₹15,000 for risk mitigation.

Risk Prioritization Methods

1. Risk Exposure Formula

✓ Risk Exposure = **Probability × Loss Due to Risk**
✓ Helps determine **which risks need urgent attention**.

2. Failure Mode and Effects Analysis (FMEA)

✓ **Assigns a Risk Priority Number (RPN):**
✓ RPN = **Severity × Occurrence × Detection**
✓ Risks with the **highest RPN** need immediate action.

Example: Risk Assessment in a Software Project

Scenario: Developing an Online Banking Application

Risk	**Likelihood (L)**	**Impact (I)**	**Risk Score (L × I)**	**Mitigation Strategy**
Security Breach	High (5)	Critical (10)	**50**	Implement **strong encryption**

Risk	Likelihood (L)	Impact (I)	Risk Score (L × I)	Mitigation Strategy
Server Crash	Medium (3)	High (8)	**24**	Use **load balancing**
Developer Leaves Project	Low (2)	Medium (5)	**10**	Keep **backup resources**

✓ **Security Breach is the highest priority risk (Score = 50)**, so it requires **immediate action**.

Q. Explain Risk Planning & Projection

Risk Planning and Projection are **key activities** in risk management that help **identify, analyze, and prepare for potential risks** in a software project.

Risk Planning

Definition:

Risk Planning is the **process of creating strategies** to **avoid, minimize, or handle risks** that could negatively impact a software project. It helps teams proactively manage risks before they become major issues.

Steps in Risk Planning:

Step	Description	Example
1. Identify Risks	Recognize potential risks in the project.	Server failure, budget overrun, security breach.
2. Assess Risk Impact	Determine how serious each risk is.	Security breach → High impact.
3. Develop Risk Mitigation Strategies	Plan ways to **reduce or eliminate risks**.	Use **data encryption** for security threats.
4. Assign Responsibilities	Assign **team members** to handle risks.	Security team monitors cyber threats.
5. Monitor & Review	Continuously check risks and update plans.	Weekly **risk review meetings**.

Why is Risk Planning Important?
✓ Ensures **project stability and smooth execution**.

✓ **Reduces financial losses** by handling risks early.
✓ Improves **team preparedness** for unexpected issues.

Risk Projection (Risk Forecasting)

Definition:

Risk Projection (also called **Risk Forecasting**) involves **analyzing risks to predict their probability, impact, and possible consequences**.

Goal: Estimate **which risks are most likely to occur** and how they will affect the project.

Steps in Risk Projection:

Step	Description	Example
1. Estimate Probability of Risk	Assess **how likely** a risk is to occur.	**Server failure: 40% probability**
2. Analyze Impact of Risk	Determine how much damage the risk can cause.	**Security breach → Critical impact**
3. Calculate Risk Exposure	Use formula: **Risk Exposure = Probability × Loss**	**(40% × ₹1,00,000) = ₹40,000**
4. Prioritize Risks	Rank risks based on **probability & impact**.	High-priority risks need **immediate action**.

Example: Risk Planning & Projection in a Software Project

Scenario: Developing a Mobile Banking App

Risk	Probability (%)	Impact (₹ Loss)	Risk Exposure (₹)	Mitigation Plan
Security Breach	50%	₹2,00,000	₹1,00,000	Use **strong encryption, firewalls**
Server Crash	30%	₹1,50,000	₹45,000	Implement **load balancing**
Key Developer Leaves	20%	₹50,000	₹10,000	Keep **backup developers**

Security Breach is the highest priority risk because of its high probability and impact (₹1,00,000 exposure).

Risk Mitigation Strategies

Strategy	Description	Example
Risk Avoidance	Change project plans to remove risk.	Use **stable technologies** instead of experimental ones.
Risk Reduction	Minimize risk probability or impact.	Implement **automated testing** to reduce software bugs.
Risk Transfer	Shift risk responsibility to third parties.	Buy **cybersecurity insurance** to cover losses.
Risk Acceptance	Accept risk if mitigation is too costly.	Some **minor UI bugs are accepted** if fixing them is expensive.

Q. Explain RMMM

RMMM is a structured **risk management strategy** used in software projects to **identify, assess, mitigate, and monitor risks** throughout the project lifecycle. It ensures that risks do not cause project failures, delays, or cost overruns.

What is RMMM?

RMMM stands for:

Term	Description
Risk Mitigation	Developing strategies to **reduce or eliminate risks**.
Risk Monitoring	Tracking and **observing risks continuously** throughout the project.
Risk Management	Implementing corrective actions when risks occur and **adjusting strategies**.

Goal: To ensure software projects are **protected from potential risks** and can **handle uncertainties efficiently**.

Components of RMMM

1. Risk Mitigation (Prevention Strategies)

✓ **What it does:** Reduces the **probability** and **impact** of risks.
✓ **How it works:** Implementing proactive measures to handle identified risks before they become problems.

Example: If there’s a **risk of data loss**, implement **automated backups**.

Risk	Mitigation Strategy
Security Breach	Use **firewalls & encryption**
Server Crash	Implement **load balancing**
Key Developer Leaves	Maintain **documentation & backup resources**

---

2. Risk Monitoring (Tracking Risks)

✓ **What it does:** Observes risks throughout the project lifecycle to detect **early warning signs**.
✓ **How it works:** Uses **risk logs, reports, and meetings** to track risks.

Example: If a key developer is absent frequently, it might indicate they **plan to leave**, requiring proactive resource planning.

Risk	Monitoring Method
Security Breach	Monitor **logs for suspicious activity**
Server Crash	Track **server performance metrics**
Developer Leaves	Conduct **regular team check-ins**

---

3. Risk Management (Taking Action)

✓ **What it does:** Implements **corrective actions** when a risk occurs.
✓ **How it works:** Uses predefined **contingency plans** to minimize damage.

Example: If a server crashes, the **backup server** is activated immediately to **reduce downtime**.

Risk	Management Strategy
Security Breach	Implement **emergency security patches**
Server Crash	Use **backup servers**
Budget Overrun	Adjust **project scope & prioritize tasks**

Example: RMMM in a Software Project

Scenario: Developing an E-commerce Website

Risk	Probability (%)	Impact (₹ Loss)	Mitigation Plan	Monitoring Plan	Management Plan
Cyber Attack	40%	₹2,00,000	Install **firewalls & encryption**	Monitor **server logs**	Activate **emergency security response**
Server Failure	30%	₹1,50,000	Implement **load balancing**	Monitor **CPU usage**	Switch to **backup server**
Budget Overrun	25%	₹1,00,000	Use **cost estimation tools**	Track **budget in real-time**	**Reallocate resources**

Cyber Attacks are the highest priority risk because of **high impact and probability**.

Benefits of RMMM

✓ **Prevents project failure** by addressing risks early.
✓ **Reduces financial loss** by handling risks efficiently.
✓ **Improves project stability** through continuous monitoring.
✓ **Enhances decision-making** with structured risk plans.

Q. Explain Software Configuration Management with its benefits and disciplines

Software Configuration Management (SCM) is a **systematic process** of handling **changes in software** to ensure that it remains **consistent, reliable, and traceable** throughout its lifecycle.

Goal: To **control and track changes** in software development, avoiding conflicts and ensuring smooth collaboration.

Why is SCM Important?

✓ Prevents **version conflicts** when multiple developers work on the same project.
✓ Ensures **consistency** by keeping track of **code changes, documents, and configurations**.
✓ Helps in **error tracking and rollback**, allowing developers to revert to a previous stable version.
✓ Supports **team collaboration**, especially in large-scale software projects.

Key Activities in SCM

1. Configuration Identification

✓ Identifying all software components that need to be managed.
✓ Includes **source code, documents, libraries, test cases, databases, and configuration files**.

Example: In a banking app, configuration items include **UI components, database schemas, and authentication modules**.

---

2. Configuration Control

✓ Ensures that **changes are made in a controlled manner** and do not introduce defects.
✓ Uses **version control systems (Git, SVN)** to manage changes.

Example: A developer requests to update the **login system**; the change is reviewed before merging into the main codebase.

---

3. Configuration Status Accounting

✓ Tracks and **records all changes** made to the software.
✓ Generates reports on **who made changes, when, and why**.

Example: A project manager checks logs to see which developer modified the **payment processing code** and when.

---

4. Configuration Auditing & Review

✓ Ensures that the software **meets project requirements and standards**.
✓ Conducts **regular audits** to verify that **approved changes are implemented correctly**.

Example: A **code review session** is conducted before releasing an update for a mobile app.

SCM Process

Step	Description
1. Identify Configuration Items	Define components that need version control.
2. Version Control	Track different versions of files.
3. Change Management	Review and approve changes before implementation.
4. Status Reporting	Record and report all modifications.
5. Auditing & Review	Verify compliance with project requirements.

Software Configuration Management Tools

SCM Tool	Description
Git	Most popular distributed version control system (GitHub, GitLab, Bitbucket).
SVN (Subversion)	Centralized version control system.
Mercurial	Another distributed version control tool.
ClearCase	Enterprise SCM tool from IBM.

Git is widely used because of its flexibility, distributed nature, and strong branching features.

Example: SCM in a Real-World Scenario

Scenario: A team is developing an online shopping website.

SCM Activity	Example
Configuration Identification	Define **product catalog, payment module, user authentication** as configuration items.
Version Control	Use **Git** to manage different versions of the shopping cart code.
Change Control	Developer updates the **checkout system**, and the change is reviewed before merging.
Status Accounting	Maintain logs showing **who modified the order processing system**.
Auditing & Review	Conduct a **code review** before deploying a new feature.

Benefits of SCM

✓ **Prevents accidental loss of code**.
✓ **Improves collaboration** in teams.
✓ **Helps in debugging and rollback**.
✓ **Ensures consistency** across multiple environments.

Q. Explain Software Configuration Management Repositories

A **Software Configuration Management (SCM) Repository** is a **centralized storage system** that holds and manages all the **software artifacts** (such as source code, documents, configurations, and libraries) in a structured way.

Purpose:
✓ To **store, track, and manage** all versions of files.
✓ To **enable collaboration** among developers.
✓ To **maintain consistency** and prevent conflicts in software development.

Types of SCM Repositories

SCM repositories can be categorized based on how they store and manage version control:

Type	Description	Example Tools
Centralized Repository	Stores all files in a single **central server**. Users must be connected to the server to access and update files.	**SVN (Subversion), ClearCase, Perforce**
Distributed Repository	Each developer has a **local copy** of the repository, allowing work without an internet connection. Changes are merged later.	**Git, Mercurial**
Artifact Repository	Stores **compiled code (binaries), libraries, and dependencies** for software deployment.	**JFrog Artifactory, Nexus, AWS CodeArtifact**

Key Features of SCM Repositories

✓ **Version Control** – Tracks changes made to files and allows rollback to previous versions.
✓ **Branching and Merging** – Enables multiple developers to work on different features simultaneously.
✓ **Access Control** – Restricts file access to authorized users.
✓ **Backup and Recovery** – Prevents data loss by keeping backups of all changes.
✓ **Logging and Auditing** – Maintains a history of changes, showing **who changed what and when**.

How SCM Repositories Work?

Example: A development team is working on an online **banking application** using **GitHub (a Git repository)**.

Step	Action
1. Developer Clones Repository	A developer copies the latest project version to their local system.
2. Makes Code Changes	The developer modifies the login feature in the code.
3. Commits Changes	The changes are saved locally with a commit message.
4. Pushes to Repository	The updated code is sent to the central GitHub repository.
5. Team Reviews and Merges	The team reviews the code, merges it into the main branch, and deploys the changes.

This ensures that **all changes are tracked, stored safely, and reviewed before final integration**.

Popular SCM Repository Tools

Tool	Type	Key Features
Git	Distributed	Supports branching, merging, and offline work.
GitHub	Distributed	Cloud-based, integrates with CI/CD tools.
SVN (Subversion)	Centralized	Linear version control, used in enterprises.
Perforce	Centralized	Handles large-scale projects efficiently.
Bitbucket	Distributed	Supports Git and integrates with Jira.
Azure DevOps	Distributed	SCM + Project management.

Git and GitHub are the most widely used SCM repositories in modern development.

Benefits of Using SCM Repositories

✓ **Prevents Data Loss** – Code is stored securely with backups.
✓ **Facilitates Collaboration** – Multiple developers can work on different features.
✓ **Enforces Code Review** – Ensures better code quality before deployment.
✓ **Improves Software Stability** – Helps track and revert bad changes.
✓ **Supports Continuous Integration** – Automates testing and deployment.

Q. Explain Change Control activities in Software Configuration Management

Change Control is a structured process in **Software Configuration Management (SCM)** that ensures all software changes are **systematically requested, evaluated, approved, implemented, and tracked** to maintain software quality and stability.

Purpose of Change Control:
✓ Prevents **unauthorized or untested changes**.
✓ Ensures **changes do not break the system**.
✓ Maintains **software version integrity**.
✓ Tracks **who made changes, when, and why**.

Change Control Activities

The **Change Control Process** consists of **six key activities**:

1. Change Request Initiation

✓ Developers, testers, or customers can **request changes**.
✓ Requests are submitted through a **Change Request Form (CRF)**.
✓ Each request is assigned a **unique ID** for tracking.

Example: A developer requests a change to **improve the payment gateway performance** in an e-commerce app.

---

2. Change Impact Analysis

✓ The Change Control Board (CCB) analyzes the **impact of the change** on the software.
✓ Assesses **cost, time, risks, and dependencies**.
✓ Determines whether the change is **feasible and necessary**.

Example:

- Will the change **affect other modules** like order processing?
- How much **time and cost** is required?

---

3. Change Approval or Rejection

✓ The CCB **approves or rejects** the change request based on impact analysis.
✓ If rejected, reasons are documented.
✓ If approved, a **Change Implementation Plan** is created.

Example: A UI improvement request is **approved**, but a request to migrate to a new database is **rejected** due to high risk.

4. Change Implementation

✓ Developers **implement the approved changes** in a **separate branch** of the code.
✓ Changes are **tested in a controlled environment** before merging.

Example: The **checkout button issue** is fixed and tested in the **staging environment** before deployment.

5. Change Verification & Testing

✓ The modified software undergoes **testing to ensure stability**.
✓ **Regression Testing** is done to confirm the change does not break other parts.

Example: If the login system is changed, testers check if **password reset and multi-factor authentication** still work properly.

6. Change Deployment & Documentation

✓ After successful testing, the change is **deployed to production**.
✓ The change is **documented for future reference**.
✓ Version control tools (e.g., **Git, SVN**) are used to update the repository.

Example:

- The new **shopping cart feature** is deployed to the **live website**.
- A log entry is created:

"Feature: Improved Cart UI - Version 1.2 - Deployed on 15 March 2025."

Tools Used for Change Control

Tool	Purpose
Git / GitHub	Version control for tracking code changes
JIRA / Trello	Managing change requests & tracking progress
SVN (Subversion)	Centralized version control
Bugzilla / Redmine	Issue tracking & change management

Importance of Change Control

✓ **Prevents software failure** – Ensures changes don't introduce critical bugs.
✓ **Improves software quality** – Changes are properly reviewed & tested.
✓ **Enhances project management** – Avoids delays & cost overruns.
✓ **Provides traceability** – Tracks all changes with logs & version history.

Q. Explain Version Control activities in Software Configuration Management

Version Control is an essential part of **Software Configuration Management (SCM)** that helps track and manage **changes to software code, documents, and configuration files** over time.

Purpose of Version Control:
✓ Keeps track of **changes and updates** in software.
✓ Allows **multiple developers** to work on the same project without conflicts.
✓ Enables easy **rollback to previous versions** if issues occur.
✓ Maintains a **history of modifications** for accountability.

Version Control Activities

Version control consists of **five key activities** to ensure smooth tracking and management of software changes.

1. Create and Initialize a Repository

✓ A **repository** is created to store all software files.
✓ The repository maintains a **history of changes** made to files.
✓ Can be **local (on a developer's machine)** or **remote (on a server like GitHub, GitLab, or Bitbucket)**.

Example: A team developing a **banking app** creates a GitHub repository named BankingAppRepo.

---

2. Check-in (Commit) Changes

✓ Developers make **changes** to code and **commit** them to the repository.
✓ Each commit has a **unique ID (hash)** and a **message describing the change**.

Example:

- Developer **adds a new login feature** and commits:
- git commit -m "Added login authentication feature"

---

3. Check-out (Update) and Retrieve Versions

✓ Developers can **check out** a specific version of the software.
✓ Ensures that everyone is **working with the latest code**.

Example: If a developer wants to work on the signup module, they pull the latest code:

git pull origin main

---

4. Branching and Merging

✓ **Branching** allows multiple developers to work on different features **without affecting the main code**.
✓ **Merging** combines the changes from different branches into the main project.

Example:

- A developer works on a **"dark mode" feature** in a separate branch:
- git checkout -b dark-mode-feature
- Once completed, they **merge** the branch into the main code:
- git merge dark-mode-feature

---

5. Version Tagging and Release Management

✓ **Tagging** is used to mark important versions (e.g., releases like v1.0, v2.0).
✓ Helps in **tracking stable software versions** for deployment.

Example: A **stable version (v1.0)** of an e-commerce website is tagged:

git tag -a v1.0 -m "Initial release of e-commerce website"

git push origin v1.0

Types of Version Control Systems (VCS)

Type	Description	Examples
Local Version Control	Stores all changes on a single developer's machine.	RCS (Revision Control System)
Centralized Version Control (CVCS)	A central server stores all versions. Developers pull and push changes.	SVN (Subversion), Perforce

Type	Description	Examples
Distributed Version Control (DVCS)	Each developer has a full copy of the repository, allowing offline work.	Git, Mercurial

---

Tools Used for Version Control

Tool	Type	Used for
Git	Distributed	Source code version control
GitHub, GitLab, Bitbucket	Cloud-based	Remote repository hosting
SVN (Subversion)	Centralized	Version tracking
Mercurial	Distributed	Similar to Git

Importance of Version Control in SCM

✓ **Prevents data loss** – Every version is saved and can be restored.
✓ **Enhances collaboration** – Multiple developers can work on the same project.
✓ **Improves tracking** – Every change is documented with timestamps & authors.
✓ **Supports rollback** – If a bug is introduced, the software can be reverted to a previous version.

Q. How change control is different from version control

Aspect	Change Control	Version Control
Purpose	Manages and controls changes in software development.	Tracks and manages different versions of files, code, and documents.
Focus	Ensures changes are evaluated, approved, and documented before implementation.	Maintains history, allows collaboration, and enables rollback to previous versions.
Key Concern	How changes affect the project, including risk assessment and approval.	How to store, track, and manage different versions of the code.
Change Request	Developers or stakeholders submit a change request (CR).	Developers modify code and commit changes.

Aspect	**Change Control**	**Version Control**
Impact Analysis	Evaluates feasibility, cost, risk, and impact before approval.	Developers check how new code interacts with existing versions.
Approval Process	Changes must be approved before implementation.	No approval needed for creating a new version, but review processes exist.
Implementation	Approved changes are integrated into the software.	Developers update files, merge branches, and track changes.
Tracking & Documentation	Tracks why, how, and when changes were made.	Tracks what changes were made, who made them, and allows rollbacks.
Example Scenario	A "Dark Mode" feature is requested, evaluated, and approved before work starts.	Developer works on "Dark Mode," commits changes, and merges them.
Tools Used	JIRA, Trello, ServiceNow	Git, GitHub, GitLab, Bitbucket, SVN, Mercurial
Approval Needed?	Yes, changes must be reviewed and approved.	No, developers can commit changes freely.
Scope	Focuses on managing and documenting changes.	Focuses on maintaining different versions of the software.
Rollback Support?	No direct rollback; changes must go through formal process.	Yes, previous versions can be restored at any time.

Q. What is Software Reliability

Software Reliability refers to the ability of a software system to function correctly **without failure** over a specified period under given conditions. It is a measure of how dependable and error-free the software is during operation.

Key Definition:

"Software Reliability is the probability that software will operate without failure for a specified time in a specified environment."

Characteristics of Software Reliability

✓ **Correctness** → The software produces expected and accurate results.
✓ **Consistency** → The software behaves the same way under similar conditions.
✓ **Fault Tolerance** → The software can handle errors and recover from failures.
✓ **Robustness** → The software remains stable under unexpected inputs or conditions.
✓ **Availability** → The software is operational and accessible when needed.

Factors Affecting Software Reliability

Factor	Impact on Reliability
Software Complexity	More complex software has a higher chance of defects.
Quality of Code	Poor coding practices lead to more errors and failures.
Testing & Debugging	Comprehensive testing increases reliability.
Hardware Failures	Software depends on hardware, and failures affect its performance.
Operating Environment	Changes in OS, network, or user behavior can impact reliability.

Software Reliability Metrics

To measure software reliability, several metrics are used:

Metric	Description
Mean Time Between Failures (MTBF)	The average time between software failures. Higher MTBF = More reliable software.
Mean Time To Failure (MTTF)	The average time a system operates before a failure. Used for non-repairable systems.
Mean Time To Repair (MTTR)	The average time required to fix a software failure. Lower MTTR = Faster recovery.
Failure Rate	The number of failures per unit time. Lower failure rate = More reliable software.

Example: If software runs for 1000 hours and fails 5 times,
MTBF = **1000 hours / 5 failures** = **200 hours per failure** (higher is better).

Improving Software Reliability

✓ **Use Software Engineering Best Practices** → Follow proper design, coding, and testing methods.
✓ **Perform Extensive Testing** → Unit testing, integration testing, and stress testing help find errors early.
✓ **Fault Tolerance Mechanisms** → Implement recovery and backup strategies.
✓ **Regular Maintenance & Updates** → Fix bugs and enhance performance over time.
✓ **Use Formal Methods** → Mathematical verification techniques ensure correctness.

Importance of Software Reliability

✓ Ensures **user satisfaction** by reducing failures.
✓ Improves **business reputation** by delivering dependable software.
✓ Reduces **maintenance costs** by minimizing errors.
✓ Enhances **system security** by preventing unexpected crashes.

Q. What is Software Safety

Software Safety refers to the process of ensuring that software **does not cause hazards or contribute to failures** that could lead to harm, damage, or catastrophic consequences. It is a subset of **software reliability** but focuses specifically on **preventing risks** in critical systems.

Key Definition:

"Software Safety ensures that software operates in a way that prevents accidents, hazards, or failures that could cause harm to people, property, or the environment."

Example:

- In an **airplane's autopilot system**, software safety ensures that the system does not malfunction and cause a crash.
- In a **medical device like a pacemaker**, software safety prevents incorrect signals that could harm a patient.

Importance of Software Safety

✓ **Prevents Accidents** → Ensures safe operation in critical applications.
✓ **Protects Human Lives** → Used in life-critical systems like healthcare, aerospace, and nuclear power plants.
✓ **Minimizes Financial Losses** → Reduces the cost of system failures and lawsuits.
✓ **Ensures Compliance** → Meets safety standards and regulations (ISO 26262 for automotive, DO-178C for aviation).

Software Safety vs. Software Reliability

Aspect	Software Safety	Software Reliability
Focus	Prevents hazards and failures that cause harm.	Ensures correct operation over time.
Goal	Avoid accidents even if software fails.	Reduce failures and increase uptime.
Examples	Preventing a **self-driving car** from crashing.	Ensuring **an app runs without crashing**.
Scope	Applies to **critical systems** (aerospace, healthcare).	Applies to **all software systems**.

Key Difference: A system can be **reliable** but not necessarily **safe** (e.g., a banking app that never crashes but has a security flaw that leaks user data).

Software Safety Techniques

✓ **1. Hazard Analysis** → Identifies potential risks before development.
✓ **2. Fault Tolerance** → Ensures the system can recover from failures.
✓ **3. Redundancy & Backup Systems** → Uses duplicate systems for safety.
✓ **4. Formal Methods** → Uses mathematical verification to prove software correctness.
✓ **5. Defensive Programming** → Writes code that can handle unexpected errors.

Example:

- In **nuclear power plants**, software safety ensures that an emergency shutdown occurs if a sensor detects overheating.

Software Safety Standards

Industry	Safety Standard
Automotive	ISO 26262 (Functional Safety)
Aerospace	DO-178C (Software Considerations in Airborne Systems)
Medical Devices	IEC 62304 (Software Lifecycle for Medical Devices)
Industrial Control	IEC 61508 (Functional Safety of Electrical Systems)

Example: A **self-driving car's** software must meet **ISO 26262** safety standards to avoid accidents.

Q. What is Software Quality

Software Quality refers to the ability of software to meet **functional, performance, security, and user expectations** while being **reliable, maintainable, and free from defects**.

Key Definition:

"Software Quality is the degree to which software satisfies stated and implied requirements, ensuring reliability, usability, efficiency, and maintainability."

Example:

- A **banking app** should be **secure, user-friendly, and bug-free**.
- A **video streaming app** should provide **high performance and smooth playback**.

Characteristics of Software Quality

✓ **Correctness** → The software must function as expected without errors.
✓ **Reliability** → The software should operate without failure for a specified time.
✓ **Efficiency** → The software should use system resources optimally.
✓ **Usability** → The software should be easy to use and understand.
✓ **Maintainability** → The software should be easy to modify and update.
✓ **Portability** → The software should run on different platforms without issues.
✓ **Security** → The software should be protected from threats and vulnerabilities.

Example: A **secure payment gateway** should **prevent fraud**, a **gaming app** should be **responsive and fast**, and an **e-commerce website** should be **user-friendly and reliable**.

Software Quality Factors (McCall’s Model)

1. Product Operation Factors: Focuses on how the software performs in real-world use.

- **Correctness, Reliability, Efficiency, Usability, Integrity (Security)**

2. Product Revision Factors: Focuses on how easy it is to modify the software.

- **Maintainability, Flexibility, Testability**

3. Product Transition Factors: Focuses on adaptability to different environments.

- **Portability, Reusability, Interoperability**

Example:

- A **mobile app** should work on **both Android & iOS** (Portability).
- An **ERP system** should easily integrate with other systems (Interoperability).

How is Software Quality Measured?

Software Quality Metrics:

Metric	Description
Defect Density	Number of defects per 1000 lines of code (lower is better).
Mean Time Between Failures (MTBF)	The average time a system runs before failure.
Code Coverage	Percentage of code covered by tests (higher is better).
Customer Satisfaction	User feedback and experience with the software.
Performance Metrics	Response time, load time, and efficiency of the system.

Example: High-quality software has low defect density, high MTBF, and high customer satisfaction.

How to Improve Software Quality?

✓ **Use Software Engineering Best Practices** → Follow structured design and coding guidelines.
✓ **Perform Code Reviews** → Identify defects early through peer review.
✓ **Follow Testing Strategies** → Unit testing, integration testing, and security testing.
✓ **Implement Continuous Integration (CI/CD)** → Automate testing and deployment.
✓ **Maintain Proper Documentation** → Ensure clarity in requirements, design, and updates.

Example: Companies like **Google and Microsoft** use **automated testing & CI/CD pipelines** to improve software quality.

Software Quality Assurance (SQA)

Software Quality Assurance (SQA) is a systematic approach to **monitoring and improving software quality** throughout the development lifecycle.

Key SQA Activities:

- **Defining Quality Standards** (ISO 9001, CMMI)
- **Quality Audits & Reviews**
- **Testing & Bug Tracking**
- **Process Improvement**

Example: ISO 9001 ensures **standardized quality** for software products.

Q. Explain Mc Calls Quality Factor

McCall’s Quality Model was introduced by **James McCall in 1977** to define **software quality** in terms of **factors, criteria, and metrics**. It is widely used in **software engineering** to evaluate software **maintainability, reliability, and usability**.

McCall defined **11 software quality factors**, grouped into **three main categories**:

1. **Product Operation** – How well the software functions.
2. **Product Revision** – How well the software can be maintained.
3. **Product Transition** – How well the software adapts to new environments.

McCall’s 11 Quality Factors

1. Product Operation Factors (Functionality & Performance)

These factors determine **how well the software performs its tasks**.

Factor	Description
Correctness	The software should meet all specified requirements and provide accurate outputs.
Reliability	The software should function without failures over a specific period.
Efficiency	The software should use **minimum resources** (CPU, memory, time).
Integrity	The software should **protect against unauthorized access** and maintain data security.
Usability	The software should be **user-friendly, easy to learn, and easy to use**.

Example: A **banking app** should be **accurate (correctness)**, run **without crashes (reliability)**, respond **quickly (efficiency)**, keep **data secure (integrity)**, and have an **easy-to-use UI (usability)**.

---

2. Product Revision Factors (Maintainability & Adaptability)

These factors determine **how easily software can be updated or fixed**.

Factor	Description
Maintainability	The software should be easy to **fix bugs and update features**.
Flexibility	The software should be **easily adaptable** to changes in requirements.

Factor	Description
Testability	The software should be easy to **test** for errors and defects.

Example: Google Chrome regularly releases updates with **new features (maintainability)**, supports **new web standards (flexibility)**, and has **automated testing (testability)** before each release.

---

3. Product Transition Factors (Portability & Compatibility)

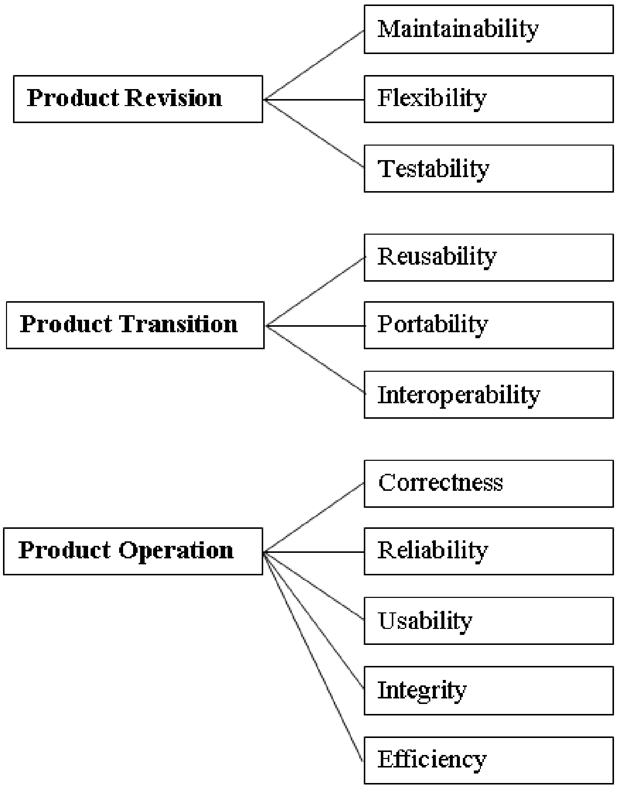
These factors determine **how well the software adapts to different environments**.

Factor	Description
Portability	The software should **work on different operating systems and devices**.
Reusability	Parts of the software should be **usable in other projects**.
Interoperability	The software should be able to **work with other software and systems**.

Example: Microsoft Word runs on **Windows, macOS, and mobile (portability)**, reuses **code from previous versions (reusability)**, and supports **PDF, DOCX, and Google Docs (interoperability)**.

McCall's Quality Model Diagram

The **11 quality factors** can be represented in a diagram:



Importance of McCall’s Quality Factors

✓ Helps **evaluate software quality** from multiple perspectives.
✓ Used in **software testing, development, and maintenance**.
✓ Ensures **customer satisfaction** by focusing on usability, reliability, and efficiency.

Q. What is FTR? It’s important in software development process

A **Formal Technical Review (FTR)** is a **structured and systematic process** used in software development to evaluate the **quality, correctness, and completeness** of software work products (like requirements, design, code, or test cases).

Definition:

"FTR is a peer review process where software engineers analyze the software artifacts to find defects early and improve quality before moving to the next phase."

Importance of FTR in Software Development

FTR plays a crucial role in improving software quality. Below are the **key benefits**:

Benefit	Description
Early Defect Detection	Helps find and fix errors **before coding or testing**, reducing costly later-stage defects.
Improved Software Quality	Ensures the **software meets requirements** and follows coding standards.
Cost Reduction	Fixing defects early in development is **cheaper** than fixing them in later stages.
Better Communication	Encourages **team collaboration** and knowledge sharing among developers, designers, and testers.
Prevention of Design Flaws	Helps identify **logical errors, missing functionalities, and inconsistencies** in the design.
Verification of Requirements	Ensures that requirements are **correctly understood and implemented**.
Standard Compliance	Helps maintain **coding standards, documentation, and best practices**.

Stages of FTR Process

FTR generally follows these steps:

Step	Description
1. Planning	The review leader schedules the meeting, selects participants, and prepares the necessary documents.
2. Preparation	Reviewers study the **software artifact (code, design, or document)** before the meeting.
3. Meeting Conduct	Reviewers discuss **findings, identify defects**, and suggest improvements.
4. Reporting	The team **documents errors**, assigns action items, and prioritizes fixes.
5. Rework	The development team **fixes the identified issues**.
6. Follow-up	A second review may be conducted to ensure **all issues are resolved**.

Types of FTR

There are different types of formal reviews:

Type	Purpose
Walkthrough	The author presents the work product to the team for **feedback and suggestions**.
Technical Review	Experts analyze the technical **correctness and feasibility** of the design/code.
Inspection	A **detailed, formal process** focused on defect detection, following strict guidelines.

Example:

- A **software design review** can reveal **missing requirements or security loopholes**.
- A **code inspection** can find **logic errors, coding standard violations, or inefficiencies**.

Q. Explain different guidelines considered during FTR

A **Formal Technical Review (FTR)** is a structured process where software artifacts (like requirements, design, code, or test cases) are reviewed to ensure **quality, correctness, and compliance** with standards.

To make FTR **effective and productive**, several **guidelines** should be followed. These guidelines help ensure that the review process is systematic, focused, and results in **meaningful improvements** to the software.

Key Guidelines for Conducting FTR

Guideline	Description
1. Review the product, not the producer	The goal is to **evaluate the software artifact**, not criticize the person who created it. Keep discussions professional and constructive.
2. Set clear review objectives	Define the **purpose** of the review (e.g., finding defects, verifying requirements, ensuring coding standards). This keeps the meeting focused.
3. Limit the number of participants	Typically, **3 to 5 reviewers** are ideal. Too many participants can make discussions unproductive.
4. Prepare in advance	Reviewers should receive **documents/code** before the meeting and **analyze them beforehand** to save time.
5. Keep meetings short and focused	FTR meetings should **not exceed 1–2 hours** to maintain focus and avoid fatigue. If needed, schedule multiple short sessions.
6. Follow a structured review process	Use a **step-by-step review process** (Planning → Preparation → Review Meeting → Reporting → Follow-up).
7. Identify problems, not solutions	Reviewers should **focus on finding issues**, while the development team will later decide on fixes.
8. Take detailed notes and document findings	Keep a **formal record of issues**, decisions, and action items. This helps track improvements.
9. Prioritize defects	Categorize issues based on severity (Critical, High, Medium, Low) to **address urgent problems first**.
10. Encourage constructive feedback	Avoid blaming individuals; instead, **suggest improvements** and keep discussions positive.
11. Follow up on defect resolution	Ensure that identified issues are **fixed and rechecked** in later reviews.
12. Use checklists	Standard **review checklists** for code, design, and documentation help ensure **consistency and thoroughness**.
13. Include multiple perspectives	Review should involve **developers, testers, designers, and domain experts** for a well-rounded analysis.

Guideline	Description
14. Conduct reviews at the right time	Perform FTRs at **critical points** (before coding starts, before release) to maximize impact.
15. Maintain confidentiality	Discussions should remain **within the team**, ensuring that feedback is used constructively.

Example: Applying FTR Guidelines

Suppose a **software development team** is working on an **online banking application**.

1. Before the FTR, the team **prepares** by reviewing the **SRS document and UI design**.
2. During the **review meeting**, the team **checks for missing security requirements**, incorrect workflows, and usability issues.
3. They **document** their findings and **assign priority levels** to each defect.
4. After the review, the development team **fixes the issues**, and a **follow-up review** is scheduled.

✓ **Result:** The application is improved **before development**, reducing costly errors later.

Q. Explain walkthrough

A **walkthrough** is a **peer review process** where the author of a software artifact (such as a document, design, or code) presents their work to a group of reviewers to **identify defects, gather feedback, and improve quality**.

Definition:

"A walkthrough is an informal process where the author guides a group of peers through a software artifact to gather feedback and detect potential issues."

Unlike **formal technical reviews (FTRs)** or **inspections**, a walkthrough is **less structured**, making it a more flexible way to review work in progress.

Objectives of a Walkthrough

✓ Identify **defects, inconsistencies, or missing requirements** in software artifacts.
✓ Improve **clarity, correctness, and completeness** of documents or code.
✓ Ensure that the work **meets project requirements** and **stakeholder expectations**.
✓ Share knowledge among team members and **encourage collaboration**.

Steps Involved in a Walkthrough

Step	**Description**
1. Preparation	The author prepares the software artifact (document, code, or design) and shares it with the review team.
2. Meeting Setup	The author schedules a **walkthrough session** with reviewers (team members, domain experts, or stakeholders).
3. Author Presentation	The author explains the **goal, functionality, and key details** of the artifact to the reviewers.
4. Discussion & Feedback	Reviewers provide **feedback, ask questions, and suggest improvements**.
5. Documentation of Issues	Any **errors, concerns, or improvement points** are recorded for future action.
6. Follow-up & Fixes	The author incorporates the **suggested changes and improvements**. If necessary, a second walkthrough may be scheduled.

Characteristics of a Walkthrough

✓ **Informal & Flexible** – No strict process or documentation is required.
✓ **Author-driven** – The **author** presents and explains the artifact.
✓ **Collaborative** – Involves **team members, stakeholders, and domain experts**.
✓ **No Strict Roles** – Unlike inspections, there are no formal **review roles (moderator, scribe, etc.)**.
✓ **Not for Approval** – Focuses on **improving quality, not formal approval**.

Example of a Walkthrough

Scenario: Code Walkthrough for a Banking Application

A developer is implementing a **fund transfer module** for an online banking system.

1. The developer schedules a **code walkthrough** with teammates.
2. During the walkthrough, the **developer explains** how the transfer logic works.
3. Peers **ask questions and suggest improvements**, like adding **error handling** for failed transactions.
4. The developer **documents the suggestions** and **modifies the code** accordingly.
5. The updated code is reviewed again before integration into the project.

✓ **Result:** Errors are detected **early**, improving software quality before testing.

Walkthrough vs. Inspection

Aspect	Walkthrough	Inspection
Formality	Informal	Formal
Purpose	Identify **issues & improve clarity**	Detect **defects & ensure compliance**
Driven by	**Author**	Moderator
Roles	No predefined roles	Defined roles (moderator, recorder, inspector)
Documentation	Minimal	Extensive documentation

Q. Explain Software Quality Assurance

Software Quality Assurance (SQA) is a set of **processes, activities, and standards** that ensure software meets **quality requirements** and works as expected before deployment.

It involves **monitoring, improving, and enforcing** quality processes throughout the **software development lifecycle (SDLC)** to **prevent defects** rather than just detecting them.

Objectives of SQA

✓ **Ensure software reliability, functionality, and performance**
✓ **Prevent defects early** rather than just finding them later
✓ **Ensure compliance** with industry standards (ISO 9001, CMMI)
✓ **Improve customer satisfaction** by delivering high-quality software
✓ **Optimize development processes** to reduce costs and time

---

Key Activities in SQA

Activity	Description
1. Process Definition	Establishing **standard procedures and guidelines** for development.
2. Software Reviews & Inspections	Conducting **walkthroughs, formal technical reviews (FTRs), and inspections** to find defects early.
3. Testing	Performing **unit testing, integration testing, system testing, and acceptance testing** to verify software quality.
4. Defect Management	Identifying, recording, and tracking defects to resolution.

Activity	Description
5. Risk Management	Identifying and mitigating **risks** related to quality and performance.
6. Quality Audits	Conducting **internal and external audits** to ensure compliance with standards.
7. Change Management	Managing and controlling software changes to avoid introducing new defects.
8. Metrics and Measurements	Using **software quality metrics** (defect density, reliability, maintainability) to measure performance and improve processes.
9. Compliance with Standards	Ensuring adherence to quality standards like **ISO 9001, IEEE, CMMI**.

SQA in Software Development Life Cycle (SDLC)

SQA is applied at every phase of SDLC:

SDLC Phase	SQA Activities
Requirement Analysis	Verify that requirements are **clear, complete, and feasible**.
Design	Perform **design reviews** to check system architecture.
Implementation (Coding)	Follow **coding standards** and perform **code reviews**.
Testing	Conduct **unit, integration, system, and acceptance testing**.
Deployment & Maintenance	Perform **user acceptance testing (UAT)** and monitor **post-release defects**.

Software Quality Assurance vs. Software Testing

Aspect	SQA	Software Testing
Scope	Focuses on **process improvements**	Focuses on **finding defects in the software**
Goal	**Prevents defects**	**Detects defects**
When Applied	Throughout **SDLC**	Primarily during **testing phase**
Methods Used	Reviews, audits, testing, standards compliance	Test execution, bug tracking, debugging

Aspect	SQA	Software Testing
Example	**Defining coding standards, conducting reviews**	**Executing test cases, reporting bugs**

Importance of SQA

✓ Reduces **cost and time** by preventing defects early.
✓ Ensures **high-quality** and **reliable** software.
✓ Enhances **customer satisfaction**.
✓ Improves **software maintainability and scalability**.
✓ Ensures **compliance with industry standards**.

Example of SQA in Action

Scenario: Online Shopping Website Development

A company developing an **e-commerce website** follows **SQA practices**:
1. Requirement Review ensures all **functional and security** requirements are defined.
2. Code Reviews detect issues before testing.
3. Automated Testing finds defects early.
4. Performance Testing ensures the website handles high traffic.
5. Quality Audits confirm compliance with standards.
✓ **Result:** Fewer defects, better performance, and a smooth user experience.

Q. What are the different steps to determine the overall consequences of risks

Step Name	Description
1. Identify Risks	**List all potential risks** that may affect the software project. These include **technical risks, project risks, business risks, and external risks**.
2. Categorize Risks	Classify risks into **different categories** (e.g., technical, financial, operational, security, etc.) to analyze them efficiently.
3. Analyze Risk Impact	Evaluate the potential **impact** of each risk on the project. Impact levels can be categorized as **Low, Medium, or High**.
4. Determine Probability	Assess the **likelihood** of each risk occurring (e.g., **Rare, Likely, or Highly Likely**).

Step Name	Description
5. Calculate Risk Exposure	Risk Exposure = **Probability × Impact**. This helps prioritize which risks need urgent attention.
6. Assess Risk Consequences	Identify the **direct and indirect consequences** of each risk, such as project delays, cost overruns, security vulnerabilities, etc.
7. Develop Risk Mitigation Strategies	Plan **preventive and corrective actions** to minimize risk impact. This includes **contingency plans, risk avoidance, and risk acceptance** strategies.
8. Monitor & Control Risks	Continuously **track risks** and update the risk assessment based on new information throughout the project lifecycle.

Q. Discuss different categories of risk that help to define impact values in a risk table

In **software engineering**, risks are uncertainties that can negatively impact a project. To manage them effectively, risks are categorized based on their **impact on cost, schedule, performance, and quality**.

A **risk table** is used to assess risks by defining their **impact values** (e.g., **Low, Medium, or High**) based on their severity.

Major Categories of Risk in Software Engineering

Risk Category	Description	Impact on Project
1. Project Risks	Risks related to **management, budget, resources, and deadlines**.	**Missed deadlines, increased costs, scope creep**.
2. Technical Risks	Risks associated with **technology, tools, and implementation feasibility**.	**Software failures, performance issues, security vulnerabilities**.
3. Business Risks	Risks affecting the **organization's profitability and market position**.	**Loss of customers, financial losses, product failure**.
4. Operational Risks	Risks related to **workflow, internal processes, and team coordination**.	**Inefficiency, miscommunication, resource wastage**.
5. External Risks	Risks arising from **external factors beyond the organization's control**.	**Regulatory changes, competition, economic downturn**.

Risk Category	Description	Impact on Project
6. Security Risks	Risks related to **cyber threats, data breaches, and hacking**.	**Loss of sensitive data, legal penalties, system downtime**.
7. Legal and Compliance Risks	Risks involving **violations of legal, regulatory, or contractual requirements**.	**Legal action, fines, project shutdown**.

How These Categories Define Impact Values in a Risk Table

A **risk table** helps prioritize risks by assigning impact values based on **severity**.

Risk Category	Example Risk	Probability	Impact	Risk Exposure (Probability × Impact)
Technical Risk	New AI algorithm may not work as expected	High	High	Critical
Project Risk	Team members leaving the project	Medium	High	Moderate
Security Risk	System vulnerability leading to a data breach	High	High	Critical
Business Risk	Competitor releases a better product	Medium	High	Moderate
Legal Risk	Non-compliance with GDPR regulations	Low	High	Moderate

✓ **Result**: Risks with **higher impact values** are prioritized and **mitigated first**.

Q. Explain the various steps involved in change control

Step Name	Description
1. Change Request Submission	Any change (feature modification, bug fix, design update) is formally **requested** using a **Change Request (CR) form**.
2. Change Classification	The request is categorized based on **type, priority, and impact** (e.g., critical bug fix, feature enhancement, security update).
3. Change Impact Analysis	The impact of the change on **cost, schedule, performance, and dependencies** is evaluated.

Step Name	Description
4. Change Approval or Rejection	A **Change Control Board (CCB)** reviews the change and decides whether to **approve, modify, or reject** it.
5. Change Planning	If approved, a detailed **implementation plan** (timeline, resources, testing strategy) is created.
6. Change Implementation	The approved change is **coded, integrated, and tested** in a controlled environment.
7. Change Verification & Testing	The change is validated through **unit testing, integration testing, and system testing** to ensure correctness.
8. Change Documentation	All modifications are **documented** in configuration management records for **traceability and future reference**.
9. Change Deployment	The verified change is **rolled out to production** (if applicable) using **version control systems**.
10. Post-Implementation Review	The change is monitored to assess its **effectiveness** and ensure it does not cause new issues.

Q. Explain the various steps involved in version control

Step Name	Description
1. Initialize Repository	A **Version Control System (VCS)** (e.g., Git, SVN) is set up to track changes. A repository (repo) is created to store the project files.
2. Add Files to Repository	Source code and related files are added to the repository for version tracking.
3. Commit Changes	Developers **make modifications** and commit changes with a **message** describing the update.
4. Branching	A new **branch** is created to develop new features, fix bugs, or experiment without affecting the main codebase.
5. Merging	After testing, changes from branches are merged into the **main branch** (e.g., master/main).
6. Conflict Resolution	If two developers modify the same file, a **merge conflict** occurs and must be resolved manually.

Step Name	Description
7. Tagging and Versioning	Tags are used to label **stable versions** (e.g., v1.0, v2.0) for easy identification and release management.
8. Code Review & Testing	Changes are reviewed, tested, and validated before final deployment.
9. Deployment	The latest stable version is released into production.
10. Backup and Rollback	Older versions are stored, and in case of failure, rollback to a previous version is possible.

Q.