ITC 504 SOFTWARE ENGINEERING

Risks

- "Tomorrow problems are today's risk."
- "Risk" refers to a situation that could result in a loss or jeopardies the project's progress but has not yet occurred.
- These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.
- There are various classifications of risks like:

Classification of Risks: 1

- Reactive risk: the software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a fire-fighting mode. When this fails, “crisis management” takes over and the project is in real jeopardy
- Proactive Risk: A proactive strategy begins long before technical work is initiated. Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk. The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner

Classification of Risks: 2

- 1. Known risks: Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
- 2. Predictable risks: Those risks that are hypothesized from previous project experience (e.g., past turnover)
- 3. Unpredictable risks: Those risks that can and do occur, but are extremely tough to identify in advance.

Classification of Risks: 3

- Project risks are those that have an impact on the project's schedule or resources.
- Product risks affect the quality or performance of the product being developed.
- Business risks are risks to the corporation developing or licensing the software.

Classification of Risks: 4

- 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.

Classification of Risks: 5

Requirements Volatility: Changes in project requirements can lead to scope creep, increased development time, and added costs. Managing changes effectively is crucial to mitigate this risk.

Technical Risks:

- Technological Obsolescence: The technology stack chosen for a project may become outdated during development, leading to compatibility and maintenance issues.
- Complexity: Overly complex software designs can lead to difficulties in implementation, testing, and maintenance.

Schedule Risks:

- Unrealistic Deadlines: Setting overly aggressive deadlines can result in rushed development, which may lead to errors and increased technical debt.
- Resource Constraints: Inadequate staffing or resources can cause delays in development.

Classification of Risks: 5

Quality Risks:

- **Bugs and Defects:** Poor code quality can result in software defects, leading to increased maintenance efforts and potential security vulnerabilities.
- **Security Vulnerabilities:** Failing to address security concerns can result in data breaches and other security incidents.

Integration Risks:

- **Third-Party Dependencies:** Reliance on third-party libraries or services can introduce risks if those dependencies change or become unavailable.
- **Interoperability Issues:** Integration challenges between different software components or systems can lead to functionality gaps.

Classification of Risks: 5

Communication Risks:

- **Miscommunication:** Poor communication among team members or with stakeholders can result in misunderstandings and misaligned expectations.
- **Cultural and Language Differences:** In global development teams, differences in culture and language can lead to communication breakdowns.

Resource Risks:

- **Staff Turnover:** Losing key team members during a project can disrupt progress and impact knowledge transfer.
- **Resource Constraints:** Limited budget or access to necessary tools can hinder development efforts.

Classification of Risks: 5

Legal and Compliance Risks:

- **Intellectual Property Issues:** Failure to respect intellectual property rights can lead to legal disputes.
- **Regulatory Compliance:** Non-compliance with industry regulations or data protection laws can result in penalties.

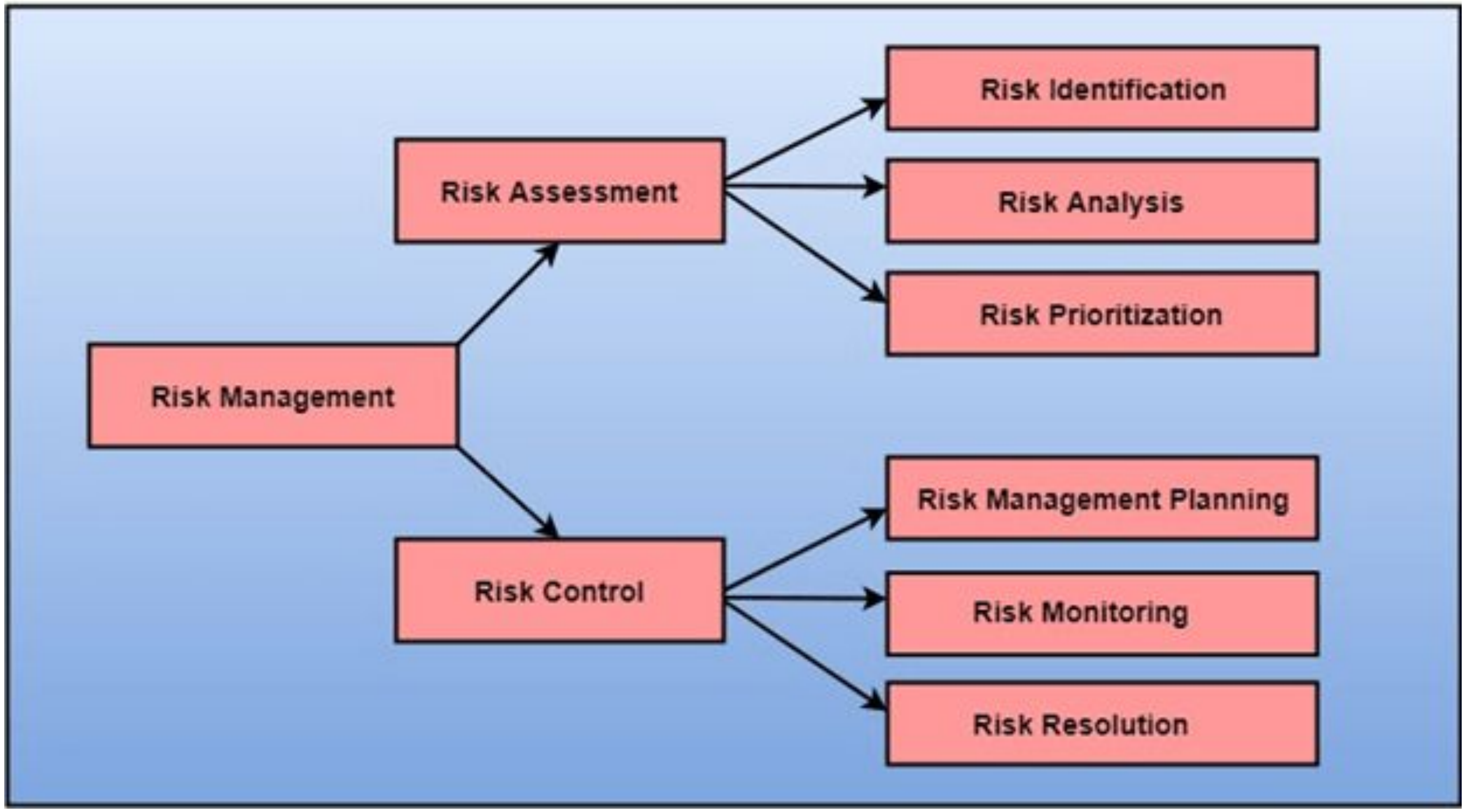
Scalability and Performance Risks: Inadequate planning for scalability and performance can lead to system bottlenecks and poor user experiences as user loads grow.

Maintenance and Technical Debt: Neglecting software maintenance can accumulate technical debt, making future development and enhancements more challenging and costly.

To manage these risks effectively, software engineering teams typically engage in risk assessment and mitigation activities, such as risk identification, risk analysis, risk prioritization, and the development of risk mitigation plans. It's essential to continuously monitor and adapt these plans throughout the software development lifecycle to reduce the impact of potential risks.

Risks

Risk Management Activities



Risk Management

Risk management in software engineering is the process of identifying, analyzing, prioritizing, and mitigating risks that can potentially impact the success of a software project. Effective risk management helps software development teams anticipate and address potential problems before they become critical issues, ultimately leading to a more successful and predictable project outcome. Key steps involved in risk management in software engineering:

Risk Identification:

Risk Identification: The first step is to identify potential risks that could affect the project. This involves brainstorming sessions, review of project documentation, and past project experiences.

Risk Categorization: Risks can be categorized into various types, such as technical, schedule, cost, quality, and external risks.

Risk Management

Risk Analysis:

- **Risk Assessment:** Once identified, risks are assessed in terms of their probability (likelihood of occurrence) and impact (severity of consequences).
- **Risk Prioritization:** Risks are then prioritized based on their significance. High-impact and high-probability risks are given the highest priority.

Risk Mitigation Planning:

- **Risk Mitigation Strategies:** For each identified risk, the team develops specific strategies to mitigate or reduce its impact. Strategies may include preventive actions to reduce the likelihood of occurrence or contingency plans to address the consequences if the risk materializes.
- **Allocating Resources:** Resources (budget, time, personnel) are allocated for risk mitigation activities.

Risk Management

Risk Monitoring and Control:

- **Regular Monitoring**: The project team monitors the identified risks throughout the software development lifecycle. This involves tracking risk indicators and assessing whether the risks are evolving as expected.
- **Risk Response Execution:** If a risk materializes or changes in severity, the predefined mitigation strategies are executed.

Documentation and Communication:

- **Risk Register:** All identified risks and their associated details (probability, impact, mitigation plans) are documented in a risk register.
- **Communication:** Risk information is communicated to relevant stakeholders, including team members, project managers, and clients, to ensure everyone is aware of potential project challenges.

Risk Management

- **Continuous Improvement:**

After each project phase or iteration, lessons learned from risk management are reviewed and used to improve future risk management processes.

Risk management practices should be flexible and adaptable, allowing teams to respond to evolving project conditions and emerging risks.

- **Contingency Planning:**

In cases where high-impact risks cannot be completely mitigated, contingency plans are put in place to define how the project will respond if the risk occurs. This includes predefined actions to minimize the impact and ensure project progress.

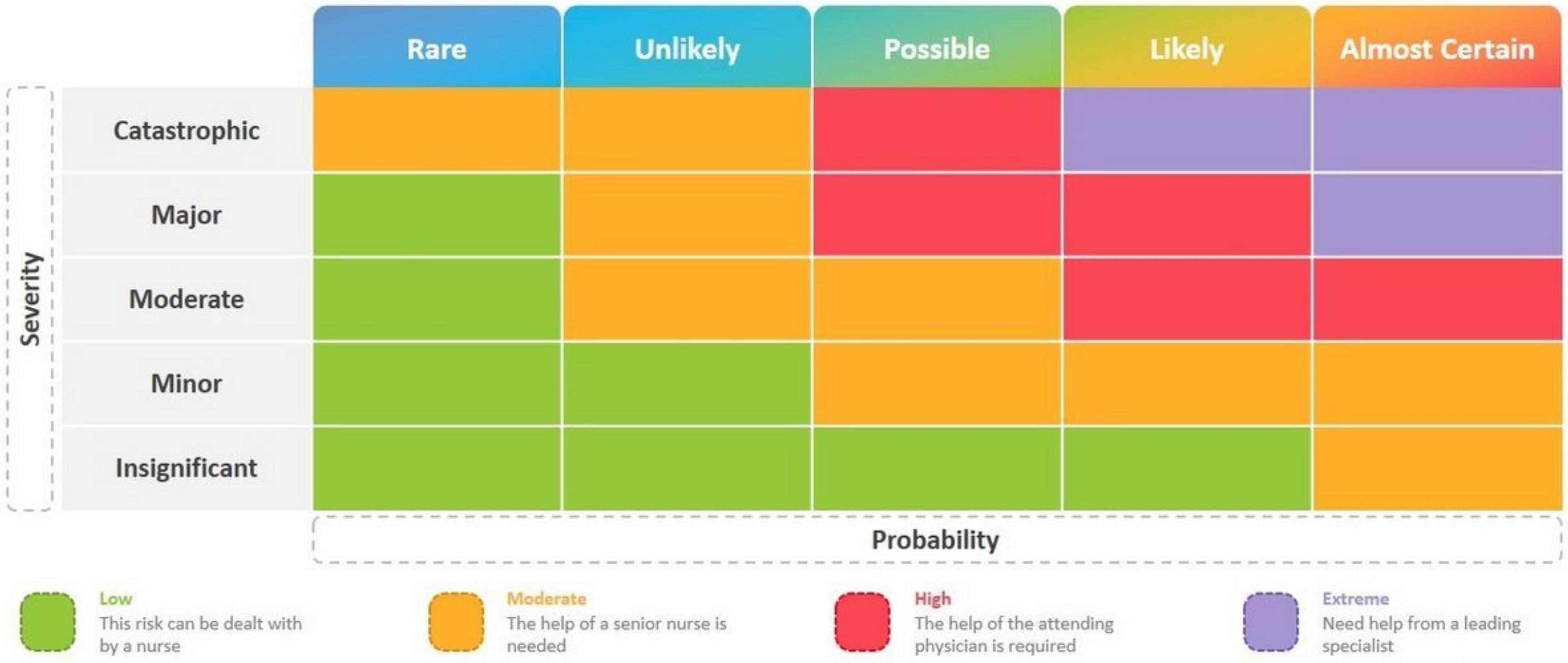
- **Risk Reporting:**

Regular reports are generated to provide project stakeholders with updates on the status of identified risks and the effectiveness of risk mitigation strategies.

Risk Management

- It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:
- The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).
- The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency), or insignificant.

Risks



Severity	Rare	Unlikely	Possible	Likely	Almost Certain
Catastrophic	🟧	🟧	🟥	🟪	🟪
Major	🟩	🟧	🟥	🟥	🟪
Moderate	🟩	🟧	🟧	🟥	🟥
Minor	🟩	🟩	🟧	🟧	🟧
Insignificant	🟩	🟩	🟩	🟩	🟧

Probability

🟩	**Low** This risk can be dealt with by a nurse
🟧	**Moderate** The help of a senior nurse is needed
🟥	**High** The help of the attending physician is required
🟪	**Extreme** Need help from a leading specialist

RMMM

- Risk Mitigation, Monitoring and Management(RMMM) Plan
- In most cases, a risk management approach can be found in the software project plan. This can be broken down into three sections: risk mitigation, monitoring, and management (RMMM). All work is done as part of the risk analysis in this strategy. The project manager typically uses this RMMM plan as part of the overall project plan.

- Some development teams use a Risk Information Sheet(RIS) to document risk. For faster information handling, such as creation, priority sorting, searching, and other analyses, this RIS is controlled by a database system. Risk mitigation and monitoring will begin after the RMMM is documented and the project is launched.

RMMM

- **Risk Mitigation, Monitoring, and Management Plan (RMMM)**
- Project Name: **Online Shopping Website Development**
- **Risk Identification:**
- **Requirement Changes**
  - *Probability*: High
  - *Impact*: High
  - *Description*: Due to evolving client needs, there is a high likelihood of requirement changes during the project.
- **Technical Skill Gaps**
- *Probability*: Medium
- *Impact*: Medium
- *Description*: Some team members lack experience with specific technologies required for the project.

RMMM

- **Third-Party Dependency**
- *Probability*: Low
- *Impact*: High
- *Description*: We rely on a third-party payment gateway, which may have service interruptions.

RMMM

- **Risk Analysis:**
- **Requirement Changes**:
  - *Priority*: High
  - *Mitigation Strategy*: Frequent client communication, robust change management process, and detailed requirement documentation.
  - *Contingency Plan*: Allocate additional time and budget for handling changes.
- **Technical Skill Gaps**:
  - *Priority*: Medium
  - *Mitigation Strategy*: Provide training sessions for team members on required technologies, consider hiring a consultant.
  - *Contingency Plan*: Reallocate tasks among team members, adjust the project schedule.

RMMM

- **Third-Party Dependency**:
- *Priority*: Low
- *Mitigation Strategy*: Identify alternative payment gateways, establish a backup plan in case of service interruptions.
- *Contingency Plan*: Switch to an alternative payment gateway temporarily.

RMMM

- **Risk Monitoring and Control:**
- **Requirement Changes**:
  - Regular client meetings to capture changes.
  - Review requirements document and update as needed.
  - Track scope changes in project management software.
- **Technical Skill Gaps**:
  - Monitor team members' progress in skill development.
  - Conduct periodic assessments to identify skill gaps.
  - Adjust training sessions based on the team's progress.

RMMM

- **Third-Party Dependency**:
- Monitor the performance of the third-party payment gateway.
- Implement automated notifications for service interruptions.
- Maintain contact with alternative payment gateway providers.

RMMM

- **Reporting:**
- **Regular Risk Reports**: Weekly meetings to review the status of identified risks, including their probability, impact, and current mitigation status.
- **Exception Reports**: Immediate reporting for high-impact and high-probability risks that materialize.
- **Continuous Improvement:**
- After each project phase, conduct a review to identify any new risks or changes to existing risks.
- Document lessons learned and update the RMMM plan for future projects.

RIS

- In software engineering, a Risk Information Sheet (RIS) is a document used to capture and communicate detailed information about a specific risk that has been identified in a project. It serves as a tool for documenting essential information about the risk, its potential impact, probability, mitigation strategies, and monitoring procedures.

RIS

- Example of a Risk Information Sheet:
- **Risk ID**: R001
- **Risk Name**: Unavailability of Key Developer
- **Risk Description**: One of the key developers on the project has an unpredictable health condition, which may lead to periods of unavailability. This could disrupt project timelines and tasks dependent on this developer's expertise.
- **Risk Assessment**:
- **Probability**: Medium
- **Impact**: High
- **Overall Risk Score**: 3 (Medium-High)

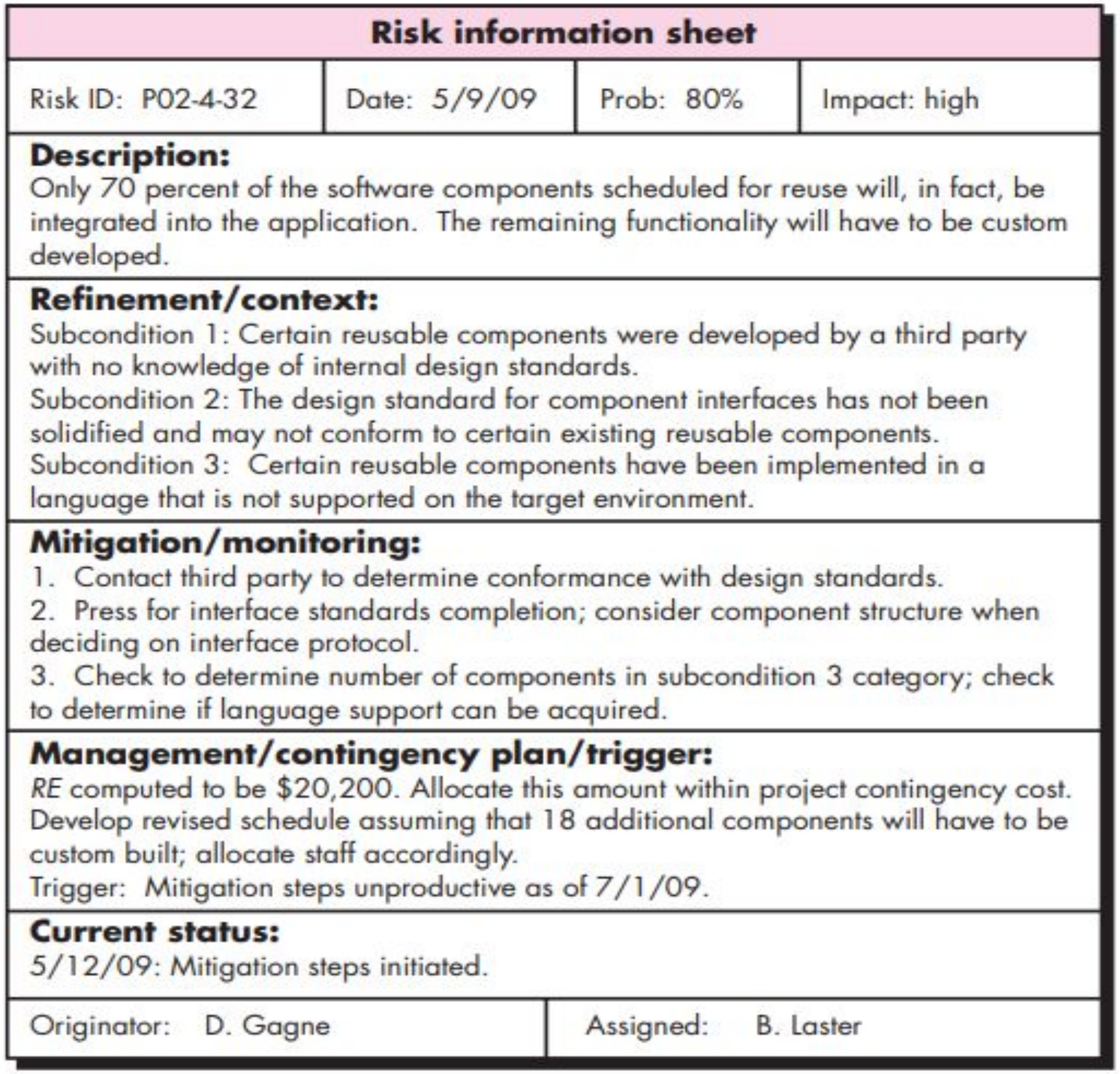
RIS

- **Risk Mitigation**:
- **Mitigation Strategy**: Cross-train other team members in critical areas of the project to reduce reliance on the key developer.
- **Responsible Parties**: Team Lead and Senior Developer
- **Timeline**: Ongoing throughout the project
- **Resources**: Allocate 10% of developer time for cross-training activities.
- **Contingency Plan**:
- **Contingency Strategy**: Hire a temporary developer with expertise in the key areas if the key developer becomes unavailable for an extended period.
- **Responsible Parties**: Project Manager and HR
- **Timeline**: Within two weeks of the key developer's extended absence.

RIS

- **Risk Monitoring and Control**:
- **Monitoring Parameters**: Track the key developer's availability on a weekly basis and maintain a backup plan for critical tasks.
- **Reporting**: Include updates on the key developer's status in the weekly project status reports.
- **Status and Updates**:
- **Current Status**: Active (No unavailability reported yet).
- **Updates**: None at this time.

Risks



Risk information sheet

Risk ID: P02-4-32	Date: 5/9/09	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/09.

Current status:
5/12/09: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

Software Configuration management

- Software Configuration Management (SCM) is a set of practices and processes used in software development to systematically manage and control software artifacts, track changes, and ensure the integrity of software configurations throughout their lifecycle. SCM helps teams collaborate effectively, maintain version history, and streamline the development process.

Software Configuration management

- **1. SCM Repositories:** SCM repositories are central locations where software artifacts, such as source code, documentation, binary files, and configuration files, are stored, organized, and managed. There are two primary types of SCM repositories:
- **Version Control Repositories:** These repositories are used to manage source code and related assets. Popular version control systems (VCS) like Git, Subversion (SVN), and Mercurial are commonly used for tracking changes, branching, and merging in these repositories.
- **Artifact Repositories:** These repositories store binary artifacts, libraries, dependencies, and other files necessary for building and deploying software. Examples include Nexus, Artifactory, and Docker Hub for container images.
- SCM repositories provide version history, access control, and audit trails for all changes made to the stored artifacts. Developers can retrieve and commit changes to these repositories, ensuring a centralized and controlled development environment.

Software Configuration management

- **2. SCM Process:** SCM encompasses several key processes to manage software configurations effectively:
- **Configuration Identification:** In this phase, SCM defines and identifies the software configuration items (SCIs). SCIs are the individual elements that make up the software, such as source code files, documentation, libraries, and configuration files. Proper identification ensures that all necessary components are tracked and controlled.
- **Change Control:** Change control processes establish procedures for requesting, reviewing, and approving changes to software configurations. These changes can include bug fixes, feature additions, or updates. Change control ensures that changes are carefully considered and documented, reducing the risk of introducing errors or instability into the software.

Software Configuration management

- **Configuration Baseline:** A configuration baseline is a snapshot of the entire software configuration at a specific point in time. It represents a stable and known state of the software, typically used as a reference for quality assurance, testing, and deployment. Baselines help ensure that specific versions of the software are used for testing and production.
- **Configuration Management Planning:** This involves creating a plan that outlines how SCM will be implemented within the software development process. It defines roles and responsibilities, procedures, tools, and the overall strategy for managing configurations.

Software Configuration management

- **Build and Release Management:** SCM encompasses the processes for creating builds and managing releases. A build is a compiled version of the software that is ready for testing or deployment. Release management involves controlling the distribution and deployment of software to various environments (e.g., development, testing, production).
- **Change Auditing and Reporting:** SCM maintains records of all changes made to the software, who made them, when they were made, and why. This information is crucial for traceability, accountability, and compliance with industry standards and regulations.

Software Configuration management

- **Branching and Merging:** In larger software projects, multiple development teams or individuals may work on different features or bug fixes concurrently. Branching and merging strategies within version control systems allow for isolation of work and later integration, helping teams collaborate efficiently.
- **Backup and Recovery:** SCM systems often include backup and recovery mechanisms to prevent data loss in case of hardware failure or accidental deletion. This ensures the integrity and availability of configuration data.

Software Configuration management

- **Continuous Integration and Continuous Delivery (CI/CD):** SCM plays a critical role in CI/CD pipelines by enabling the automation of build, test, and deployment processes. Changes are automatically integrated, tested, and delivered to various environments, reducing the risk of integration problems and streamlining software delivery.
- Software Configuration Management is a systematic approach to managing and controlling software artifacts throughout their lifecycle. It ensures that changes are well-documented, coordinated, and integrated, resulting in a more organized, stable, and maintainable software development process. Effective SCM practices are essential for software quality, collaboration, and project success.

Software Quality Assurance (SQA)

- Software Quality Assurance (SQA) is a systematic process within software development that focuses on ensuring that software products and processes meet established quality standards and objectives. SQA tasks and plans are essential components of this process:
- **SQA Tasks:** SQA tasks involve a range of activities aimed at monitoring and improving the quality of software throughout its lifecycle. These tasks typically include:
- **Requirements Analysis:** Reviewing and validating the accuracy and completeness of software requirements.
- **Test Planning:** Developing a comprehensive plan for testing the software, including defining test objectives, strategies, and test cases.
- **Code Reviews:** Evaluating the source code for adherence to coding standards, best practices, and identifying defects.

Software Quality Assurance (SQA)

- **Testing:** Executing test cases, including unit testing, integration testing, system testing, and user acceptance testing.
- **Defect Tracking:** Monitoring and documenting defects or issues discovered during testing and development.
- **Process Audits:** Assessing and improving development processes to ensure compliance with quality standards and best practices.
- **Documentation Review:** Verifying that documentation, such as user manuals and technical documentation, is accurate and up to date.
- **Release Management:** Ensuring that software releases are well-planned, tested, and documented.

Software Quality Assurance (SQA)

- **SQA Plan:** An SQA plan outlines the strategies, objectives, and activities that will be employed to achieve quality assurance throughout the software development project. It typically includes:
- **Scope and Objectives:** Describing the scope of SQA activities and the quality objectives to be achieved.
- **SQA Team Roles and Responsibilities:** Defining the roles and responsibilities of team members involved in SQA.
- **Testing and Review Procedures:** Detailing the testing methodologies, review processes, and criteria for acceptance.
- **Resources:** Identifying the resources, tools, and infrastructure required for SQA.
- **Schedule:** Outlining the timeline and milestones for SQA activities.

Software Quality Assurance (SQA)

- **Metrics and Measurements:** Describing the metrics and key performance indicators (KPIs) that will be used to assess quality.
- **Risk Management:** Identifying potential risks to quality and mitigation strategies.
- **Documentation:** Specifying the documentation standards and templates to be used for recording SQA activities.

Software Quality Assurance (SQA)

- **2. Software Metrics:**
- Software metrics are quantifiable measurements used to assess various aspects of software development, quality, and performance. These metrics provide valuable insights into the software project's health and progress. Some common software metrics include:
- **Code Coverage:** Measures the percentage of code that is tested by automated test cases, helping evaluate the comprehensiveness of testing.
- **Defect Density:** Calculates the number of defects or issues found per unit of code (e.g., lines of code), indicating code quality.
- **Velocity:** Used in Agile development, velocity measures the amount of work completed by a development team in a specific time period (e.g., a sprint).
- **Lead Time:** Measures the time it takes from the initiation of a software request or user story to its completion.

Software Quality Assurance (SQA)

- **Cycle Time:** Measures the time it takes to complete a specific task or process within the software development lifecycle.
- **Effort Variance:** Compares estimated effort with actual effort expended on a project, highlighting deviations from the plan.
- **Bug Severity and Priority:** Classifies and prioritizes defects based on their severity and impact on the software.
- **Customer Satisfaction:** Collects feedback from users or stakeholders to gauge their satisfaction with the software.
- Effective use of software metrics can help teams identify areas for improvement, make informed decisions, and ensure that software quality objectives are met.

Software Quality Assurance (SQA)

- **3. Software Reliability:**
- Software reliability refers to the ability of a software system to perform its intended functions without failures or errors under specified conditions for a specified period. Key aspects of software reliability include:
- **Availability:** It measures how often the software is operational and accessible to users. High availability implies that the software is rarely down or experiencing outages.
- **Fault Tolerance:** A reliable software system can continue functioning even when certain components or subsystems fail. Fault tolerance mechanisms, such as redundancy, help achieve this.

Software Quality Assurance (SQA)

- **Mean Time Between Failures (MTBF):** MTBF quantifies the average time between failures in the software. Higher MTBF values indicate greater reliability.
- **Mean Time to Recovery (MTTR):** MTTR measures the average time it takes to recover from a failure or downtime. Lower MTTR values indicate faster recovery and, therefore, better reliability.
- **Failure Rate:** This metric represents the rate at which failures occur in the software system over time. A lower failure rate indicates higher reliability.
- **Reliability Testing:** Various testing techniques, such as reliability testing and failure mode analysis, are used to assess and improve software reliability.
- Achieving high software reliability is crucial for mission-critical systems, as it ensures that software behaves predictably and consistently, reducing the risk of disruptions and failures that can lead to significant consequences. Reliability is often a key consideration in industries like aerospace, healthcare, and finance.

Software Quality Assurance (SQA)

- Formal Technical Review (FTR) and Walkthrough are two important techniques used in software engineering to improve the quality of software products through systematic inspection and review processes. These processes involve gathering a group of stakeholders to examine software artifacts, such as code or design documents, to identify defects, ensure compliance with standards, and promote knowledge sharing.

Software Quality Assurance (SQA)

- **1. Formal Technical Review (FTR):**
- Formal Technical Reviews are structured, well-documented review processes that follow a defined set of steps and guidelines. FTRs are typically more rigorous and comprehensive than other review methods, and they are often used to evaluate various artifacts such as code, design documents, and requirements specifications. Here are the key steps involved in an FTR:
- **Planning:** In this initial phase, the review team is assembled, and the objectives, scope, and schedule of the review are defined. The specific document or artifact to be reviewed is also selected.
- **Preparation:** Reviewers are provided with the relevant documents or code to be examined in advance. They are expected to study the material and prepare for the review meeting.

Software Quality Assurance (SQA)

- **Review Meeting:** This is the core of the FTR process. The review team, including authors and reviewers, gathers to discuss the artifact in detail. The meeting is typically led by a moderator or facilitator who ensures that the process follows a predefined agenda. Reviewers raise issues, ask questions, and suggest improvements. The focus is on identifying defects, ambiguities, inconsistencies, and violations of coding or design standards.
- **Rework:** After the review meeting, the authors of the artifact address the identified issues and make necessary corrections or improvements.
- **Follow-up:** A follow-up review may be conducted to ensure that the identified issues have been resolved satisfactorily. The artifact is re-evaluated to confirm that it now meets the required quality standards.
- **Documenting Results:** The results of the review, including identified issues, their severity, and any actions taken, are documented. This documentation serves as a valuable reference for future development phases.

Software Quality Assurance (SQA)

- **Closure:** Once the artifact has been reviewed, issues addressed, and reviewers are satisfied with the changes, the review is officially closed.
- **Example of FTR:** Suppose a software development team is working on a critical financial application. They have completed the design phase and are preparing to move on to coding. Before coding begins, the team decides to conduct a Formal Technical Review of the design documents. The steps involved might look like this:
- **Planning:** The team selects the design documents for review and forms a review team consisting of experienced developers and system architects. They schedule a review meeting for one week later.
- **Preparation:** Reviewers are given access to the design documents a few days before the review meeting. They are expected to thoroughly study the documents and make notes of any concerns or questions.

Software Quality Assurance (SQA)

- **Review Meeting:** During the meeting, the team goes through the design documents section by section. Reviewers ask questions, discuss potential risks, and point out any design flaws or ambiguities they find.
- **Rework:** The design document authors take notes of the issues raised during the review meeting and make necessary updates to the design.
- **Follow-up:** A follow-up review meeting is scheduled for the next week to verify that the issues have been addressed correctly.
- **Documenting Results:** The results of both the initial review and the follow-up review are documented. Any remaining issues or concerns are noted for further action.
- **Closure:** Once all issues are resolved, the Formal Technical Review of the design documents is closed. The team can proceed with confidence to the coding phase, knowing that the design has been thoroughly reviewed and improved.

Software Quality Assurance (SQA)

- **2. Walkthrough:**
- A Walkthrough is a less formal and more collaborative review process than an FTR. It is often used for educational purposes, knowledge sharing, and early-stage reviews. During a Walkthrough, the author of the document or code presents it to a group of reviewers, explaining the content and seeking their feedback. the key characteristics and steps of a Walkthrough:
- **Presentation:** The author presents the document or code to the reviewers, explaining its purpose, structure, and key components. The goal is to provide context and help reviewers understand the material.
- **Reviewers' Questions:** Reviewers ask questions, seek clarifications, and provide feedback during the presentation. The emphasis is on understanding and improving the content.
- **No Formal Agenda:** Unlike FTR, Walkthroughs may not follow a strict agenda or predefined set of steps. The process is more fluid and interactive.
- **Collaboration:** Walkthroughs encourage collaboration and open discussion among team members. The focus is on sharing knowledge and insights.

Software Quality Assurance (SQA)

- **Example of Walkthrough:** Consider a software development team that is working on a new web application project. Before starting the coding phase, the team decides to conduct a Walkthrough of the project's user interface (UI) design. Here's how it might work:
- **Presentation:** The UI designer presents the design mockups to the team, explaining the layout, color scheme, navigation, and user interactions.
- **Reviewers' Questions:** Developers, testers, and other team members ask questions and seek clarifications. They might inquire about the reasoning behind certain design choices, potential usability issues, or alignment with user requirements.
- **Feedback and Suggestions:** Reviewers provide feedback on the design, offering suggestions for improvements. They might suggest changes to improve user experience or identify design elements that could lead to usability issues.

Software Quality Assurance (SQA)

- **Discussion and Iteration:** The team engages in discussions to explore different design alternatives and reach a consensus on improvements. The UI designer takes notes on the feedback received.
- **Follow-up Actions:** Based on the Walkthrough discussions, the UI designer makes necessary revisions to the design. The updated design is shared with the team for further feedback and validation.
- **Iteration and Finalization:** This iterative process continues until the team is satisfied with the UI design. Once finalized, the design serves as the basis for the development phase.

Software Quality Assurance (SQA)

- Formal Technical Reviews (FTRs) and Walkthroughs are valuable software engineering practices for improving software quality. FTRs are formal, structured, and detailed reviews suitable for critical phases of the software development lifecycle, while Walkthroughs are more informal, interactive, and educational, often used for early-stage reviews and knowledge sharing. The choice between FTR and Walkthrough depends on the specific goals and needs of the review process.