SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Supportive Processes in Software Engineering

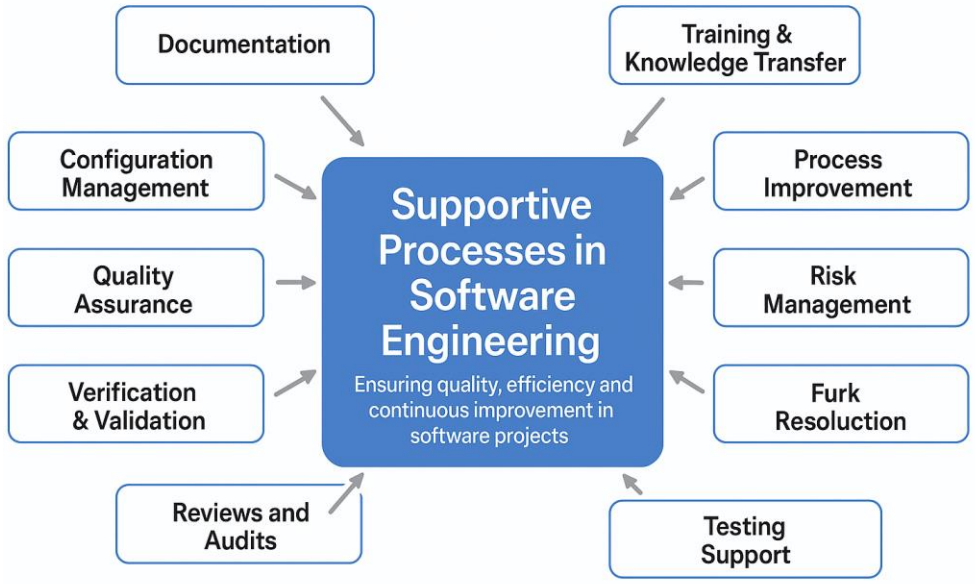
Ensuring quality, efficiency, and continuous improvement in software projects

Content

- Introduction
- Role of Supportive Processes
- Key Supportive Processes
- Configuration Management
- Change Management
- Version Controlling
- Risk Management
- Process Improvement
- Training & Knowledge Transfer

Introduction to Supportive Processes

- In Project management, these include the activities that do not directly create the product but ensure smooth and efficient development.
- Purpose: Support, control, and improve the main development processes.
- Standards : ISO/IEC 12207 and CMMI.



Role of Supportive Processes

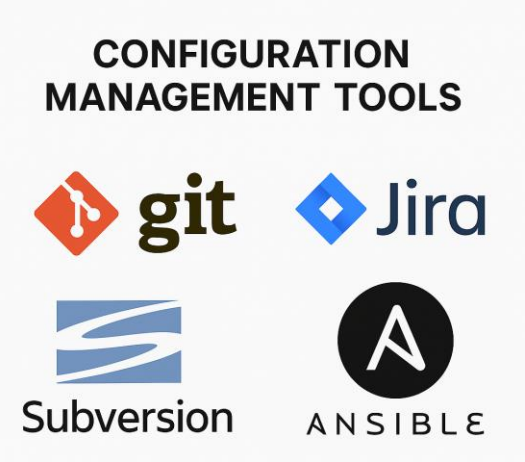
- Maintain quality and consistency
- Manage project artifacts and versions
- Detect and resolve issues early
- Reduce risks and improve team efficiency
- Facilitate communication and documentation

Key Supportive Processes

- Configuration management (CM)
- **Configuration Management (CM)** in software engineering is the process of **systematically controlling, organizing, and tracking changes** to software artifacts and related documentation throughout the project lifecycle.

Main Activities in Configuration Management

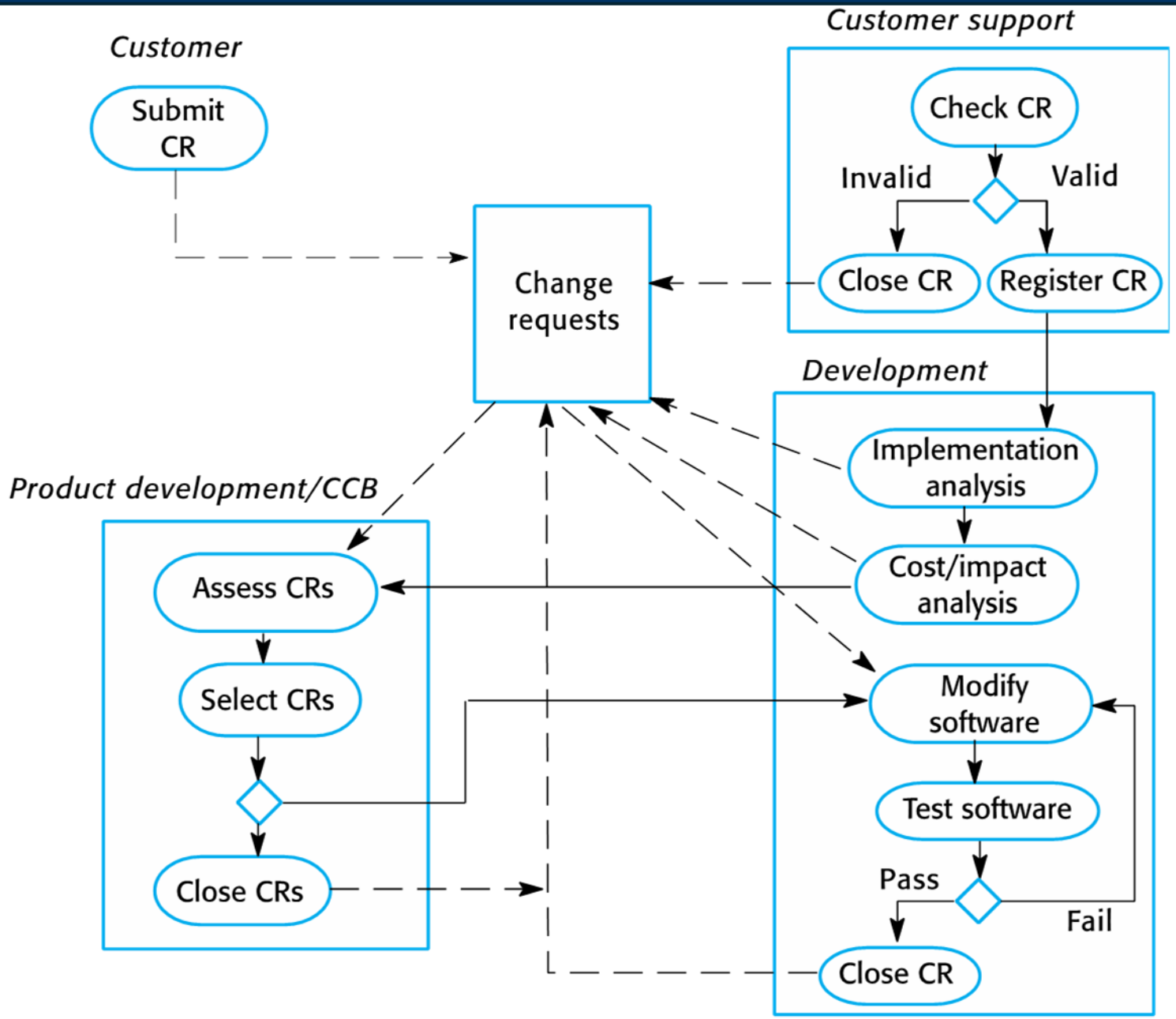
1. **Configuration Identification**
   - Defining which items need to be managed.
   - Example: Source files, requirement docs, design diagrams.
2. **Configuration Control**
   - Managing changes using formal approval processes (e.g., change control boards).
3. **Configuration Status Accounting**
   - Recording and reporting the status of configuration items (current version, changes made, who made them).
4. **Configuration Audits**
   - Ensuring that the product matches its specifications and approved changes.



Change Management

Change Management

- **Change Management** in software engineering is the structured approach for **requesting, evaluating, approving, implementing, and tracking changes** to software systems, project artifacts, or processes.
- It ensures changes are made **in a controlled, traceable, and low-risk manner** so that the project's quality, cost, and schedule are not negatively impacted.



Change management process

Change Management Process Steps

1. **Initiation** – A change request (CR) is submitted by a stakeholder (e.g., new feature, bug fix, requirement update).
2. **Impact Analysis** – Assess the effect on scope, time, cost, quality, and risks.
3. **Approval/Review** – Decision made by a Change Control Board (CCB) or project manager.
4. **Implementation** – Change is developed, tested, and deployed.
5. **Verification** – Confirm that the change works as intended and doesn't break existing functionality.
6. **Closure** – Update records, documentation, and communicate to stakeholders.

Tools for Change Management

- **Jira** – For tracking change requests and workflow.
- **ServiceNow** – For IT change request management.
- **Azure DevOps** – For tracking work items and changes.
- **GitHub Issues** – For managing code-related changes.

Version Controlling

Version Controlling

- Version controlling (or **Version Control**) is the practice of managing changes to files, code, documents, or any digital content over time.
- It allows individuals and teams to track modifications, revert to previous states, and collaborate efficiently.

Types of Version Control Systems (VCS)

- **Local Version Control**
  - Keeps versions on your computer.
  - Example: RCS (Revision Control System).
- **Centralized Version Control (CVCS)**
  - One central server stores all versions.
  - Clients check out and commit changes.
  - Example: **SVN (Subversion)**, **CVS**.
- **Distributed Version Control (DVCS)**
  - Every user has a full copy of the repository.
  - Changes can be shared peer-to-peer.
  - Example: **Git**, **Mercurial**.

Benefits of Version Control

- History tracking
- Collaboration
- Backup
- Branching and Merging
- Revertibility

Git Workflow

- **Working Directory** → Where you edit files.
- **Staging Area** → Files marked for commit.
- **Repository** → Permanent record of commits.
- Commands to Remember:
- git init → Initialize a repository
- git add file.txt → Add file to staging
- git commit -m "message" → Save changes
- git log → View history
- git push / git pull → Share updates

Risk Management

What is Risk Management?

- Risk Management is the process of identifying, assessing, and controlling potential issues that could negatively impact a software project's success.

- Why is it Important?
  a) Ensures Project Success
  b) Improves Cost Management
  c) Enhances Quality
  d) Supports Timely Delivery
  e) Boosts Team Confidence & Stakeholder Trust

Common Risks in Software Development

- Technical Risks → Technology failure, integration issues.
- Project Management Risks → Poor scheduling, scope creep.
- Resource Risks → Lack of skilled staff, budget shortage.
- Operational Risks → Inefficient processes, communication gaps.
- Security Risks → Data breaches, vulnerabilities.

Examples of Risks in Software Development

- **Technical Risk**
- Example: A project uses a new AI framework, but later the framework becomes incompatible with existing systems.
- Impact: Extra time and cost to re-engineer the solution.

- **Schedule Risk**
- Example: Developers underestimate the time required for testing.
- Impact: Project delivery gets delayed.

- **Resource Risk**
- Example: A key developer leaves the company in the middle of the project.
- Impact: Knowledge gap and slowdown until a replacement is trained.

•Scope Creep Risk

❖Example: The client keeps requesting new features beyond the original agreement.
Impact: Increased workload, higher cost, and missed deadlines.

•Security Risk

•Example: A payment system is built without proper encryption.
•Impact: Data breach leading to financial and reputational damage.

•Operational Risk

•Example: Poor communication between onshore and offshore teams.
•Impact: Misunderstanding of requirements, resulting in defects.

Benefits of Effective Risk Management

- ✅ Reduces the chance of project failure
- ✅ Minimizes financial loss
- ✅ Improves decision-making
- ✅ Increases software reliability
- ✅ Ensures customer satisfaction

Process Improvement

Process Improvement in Software Development

- Process Improvement is the practice of **analyzing, evaluating, and enhancing software development processes** to make them more efficient, reliable, and aligned with organizational goals.
- It focuses on **reducing defects, lowering costs, improving productivity, and delivering higher quality software**.

- **Why Process Improvement is Important**
- **Higher Quality** → Fewer defects, better customer satisfaction.
- **Cost Efficiency** → Less rework and waste.
- **Predictability** → Better project planning and control.
- **Faster Delivery** → Streamlined workflows reduce delays.
- **Continuous Learning** → Teams adapt and improve over time.

Process improvement Models

a) Capability Maturity Model Integration (CMMI)

- Provides maturity levels (from **Initial → Optimizing**).
- Helps organizations measure and improve their processes.

b) ISO 9001 (Quality Management Standard)

- Focuses on **standardized processes and customer satisfaction**.

C) Agile Process Improvement

- Continuous feedback and retrospectives to refine processes.

Training & Knowledge Transfer in Software Development

Training & Knowledge Transfer in Software Development

- Training

→ A planned process of improving employees' technical, managerial, or soft skills through workshops, courses, and practice.

- Knowledge Transfer (KT)

→ The systematic sharing of critical information, skills, and expertise between individuals, teams, or organizations to ensure continuity and efficiency.

Importance in Software Development

a) Skill Development

- Ensures developers are up to date with the latest tools, frameworks, and technologies.

b) Smooth Onboarding

- New team members can quickly understand existing systems, reducing ramp-up time.

c) Risk Reduction

- Prevents knowledge loss when key employees leave the project or organization.

d) Consistency

- Standardizes coding practices, design patterns, and testing approaches.

e) Productivity & Quality

- Well-trained teams deliver better, more reliable software with fewer errors.

Knowledge Transfer Techniques

- **Documentation** → User manuals, design docs, FAQs.
- **Shadowing** → New member observes an experienced one.
- **Reverse Shadowing** → New member performs tasks while expert observes.
- **KT Sessions** → Presentations or walkthroughs of the system.
- **Wikis & Knowledge Repositories** → Centralized place for project knowledge.

Example in Software Development

- A senior developer working on a **banking system module** is leaving the project.
- Risk: Knowledge gap for critical features (e.g., transaction handling).
- Solution: Organize **KT sessions + documentation + code walkthroughs** before handover.
- Outcome: New developer takes over smoothly without project delays.

Methods of Training

- Workshops & Seminars
- Online Courses / e-Learning
- Pair Programming & Mentoring
- Code Reviews
- Case Studies & Simulations

Summary

• **Supportive processes** do not build the product directly but ensure **quality, efficiency, and consistency**.

• They provide **control and improvement** for the main development processes.

• Key supportive processes include:
  • **Configuration Management** – track and control changes to artifacts
  • **Change Management** – structured handling of change requests
  • **Version Control** – manage file/code history and collaboration
  • **Risk Management** – identify and mitigate project risks early
  • **Process Improvement** – refine workflows for quality and efficiency
  • **Training & Knowledge Transfer** – maintain skills and prevent knowledge gaps

Thank You