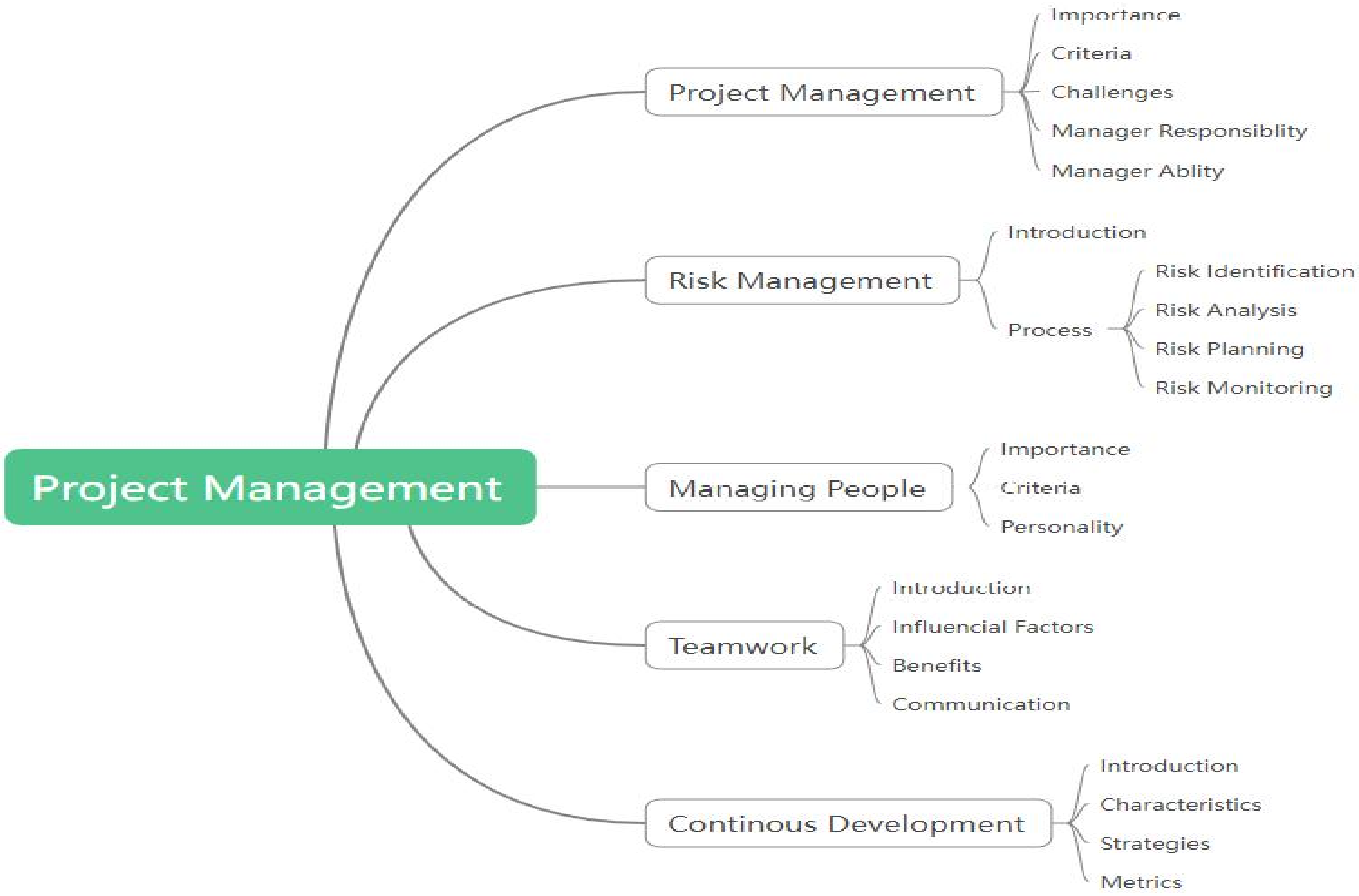
Project Management

Software Engineering I
AY 25/26
Week 12

Outline

SOMMERVILLE
SOFTWARE ENGINEERING
9


Project Management
Project Management
Importance
Criteria
Challenges
Manager Responsiblity
Manager Ablity
Risk Management
Introduction
Process
Risk Identification
Risk Analysis
Risk Planning
Risk Monitoring
Managing People
Importance
Criteria
Personality
Teamwork
Introduction
Influencial Factors
Benefits
Communication
Continous Development
Introduction
Characteristics
Strategies
Metrics

1.1 Project Management Importance

It is an essential part of software engineering

- Projects need to be managed because professional software engineering is always subject to organizational budget and schedule constraints
- Ensure that the software project meets and overcomes these constraints as well as delivering high-quality software.

Good management may not guarantee project success **BUT** a bad management can result:

- Late deliver
- Increase cost
- Fail to meet the expectations of customers

1.2 Criteria for Project Management

- Deliver the software to the customer **at the agreed time**.
- Keep overall **costs within budget**.
- Deliver software that **meets the customer's expectations**.
- Maintain a **happy and well-functioning development team**.

1.3 Challenges in Project Management

The product is **intangible**

- You cannot inspect software by visual examination; defects only appear when executing the program.
- A hardware fault (e.g., a cracked screen) is visible, but a bug/error has no visual appearance until the system is executed

Large software projects are often **'one-off' projects**

- Each large system reflects unique business rules, stakeholders, and integration constraints.
- A solution built for a hospital (patient records, privacy laws, emergency workflows) cannot be reused for an airline reservation system (seat allocation, scheduling, pricing algorithms)

Software **processes are variable and organization specific**

- Variation in culture, regulation, tools, and approval workflows makes process behavior unpredictable.
- A fintech company requires strict multi-step code audits for compliance, while a startup may deploy directly from a developer's branch

1.4 Manager's Responsibilities

Project Planning

- Planning, estimating and scheduling project development, assigning people to tasks, and supervising them.

Reporting

- Reporting on the progress of a project to customers and to the managers of the company developing the software.

Risk management

- Assess the risks that may affect a project, monitor these risks, and take action when problems arise

People management

- Choose people for their team and establish ways of working

Proposal writing

- Writing a proposal to win a contract to carry out an item of work

1.5 Abilities of A Good Project Manager

- **Motivation.** Ability to encourage (by “push or pull”) co-workers to produce to their best ability.
  - Recognize team burnout and adjusts workload so developers stay productive
- **Organization.** Ability to shape/design work processes that turn ideas into a deliverable product.
  - Setting up a weekly build–review–deploy cycle so features move smoothly from design to release
- **Ideas or innovation.** Ability to help colleagues generate creative solutions within project constraints.
  - Create an environment where team members feel safe to propose new ideas and try creative approaches.

1.5 Abilities of A Good Project Manager (cont.)

- **Problem solving.** Ability to diagnose technical and organizational issues that hinder project progress.
  - Identify whether a delay comes from unclear requirements, resource constraints, or a technical bottleneck
- **Managerial identity.** Ability to take ownership of the project, make necessary decisions, and grant experts the autonomy to work effectively.
  - When two teams disagree on how to proceed, stepping in to clarify priorities while letting the technical experts decide the details
- **Influence and team building.** Ability to read people, interpret verbal and nonverbal cues, and respond to team needs to maintain a healthy working environment
  - Notice when someone seems stressed or disengaged and initiating a supportive conversation to address the issue

2.1 Risk Management

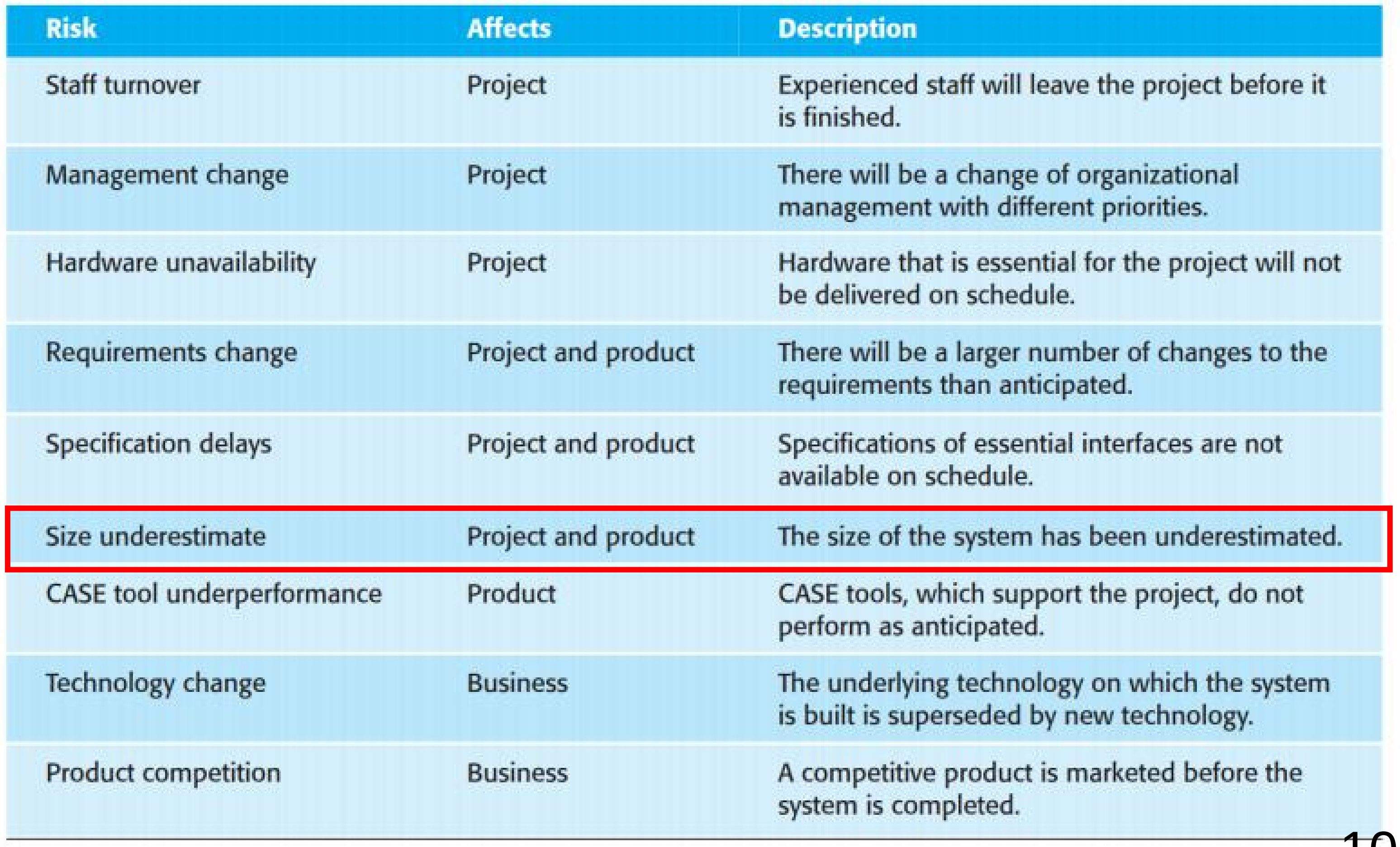
Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks.

Three major types of risks:

- **Project risks:** Risks that affect the project schedule or resources (e.g., loss of an experienced designer).
- **Product risks:** Risks that affect the quality or performance of the software being developed (e.g., failure of a purchased component (e.g., a server) to perform as expected).
- **Business risks:** Risks that affect the organization developing or procuring the software (e.g., a new product from competitors, ChatGPT v.s. Deepseek).

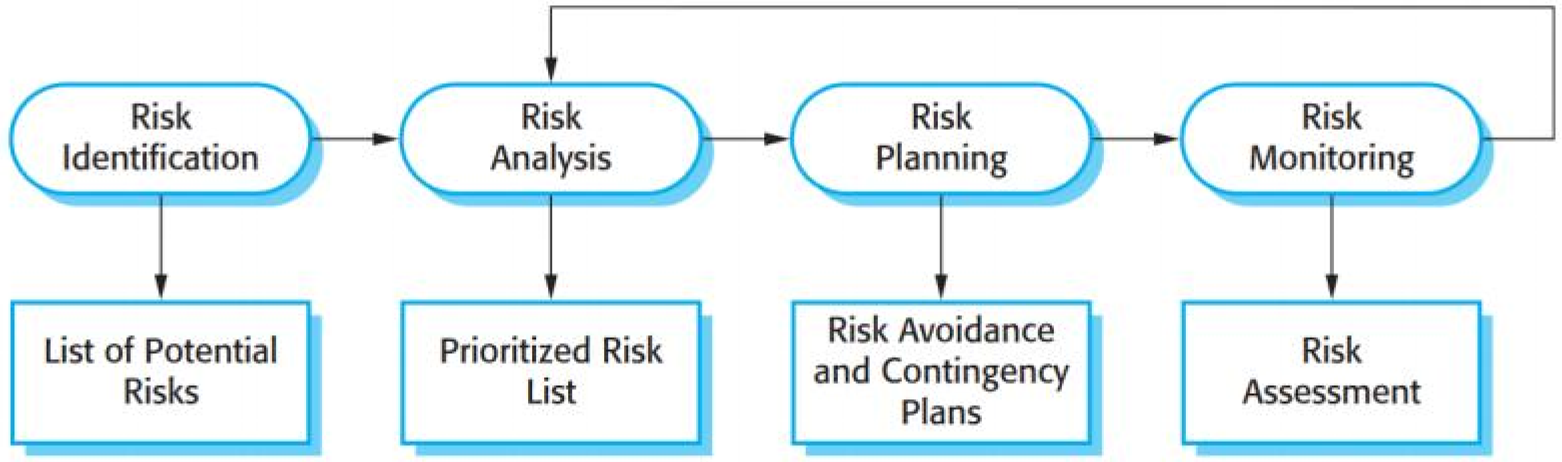
2.1 Risk Management (cont.)

Sometimes, one issue can belong to two or more different types of risks



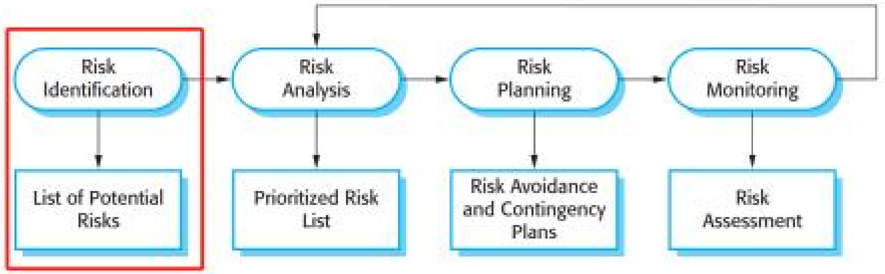
Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organizational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool underperformance	Product	CASE tools, which support the project, do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

2.2 Risk Management Process



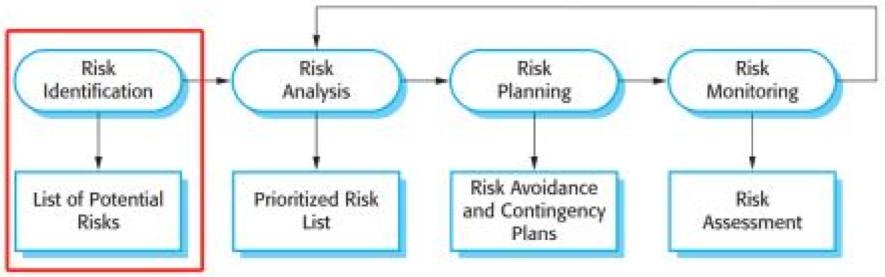
- **Risk identification:** identify possible project, product, and business risks
- **Risk analysis:** assess the likelihood and consequences of these risks
- **Risk planning:** plans to address the risk, either by avoiding it or minimizing its effects on the project
- **Risk monitoring:** regularly assess the risk and your plans for risk mitigation and revise these when you learn more about the risk

2.2.1 Risk Identification

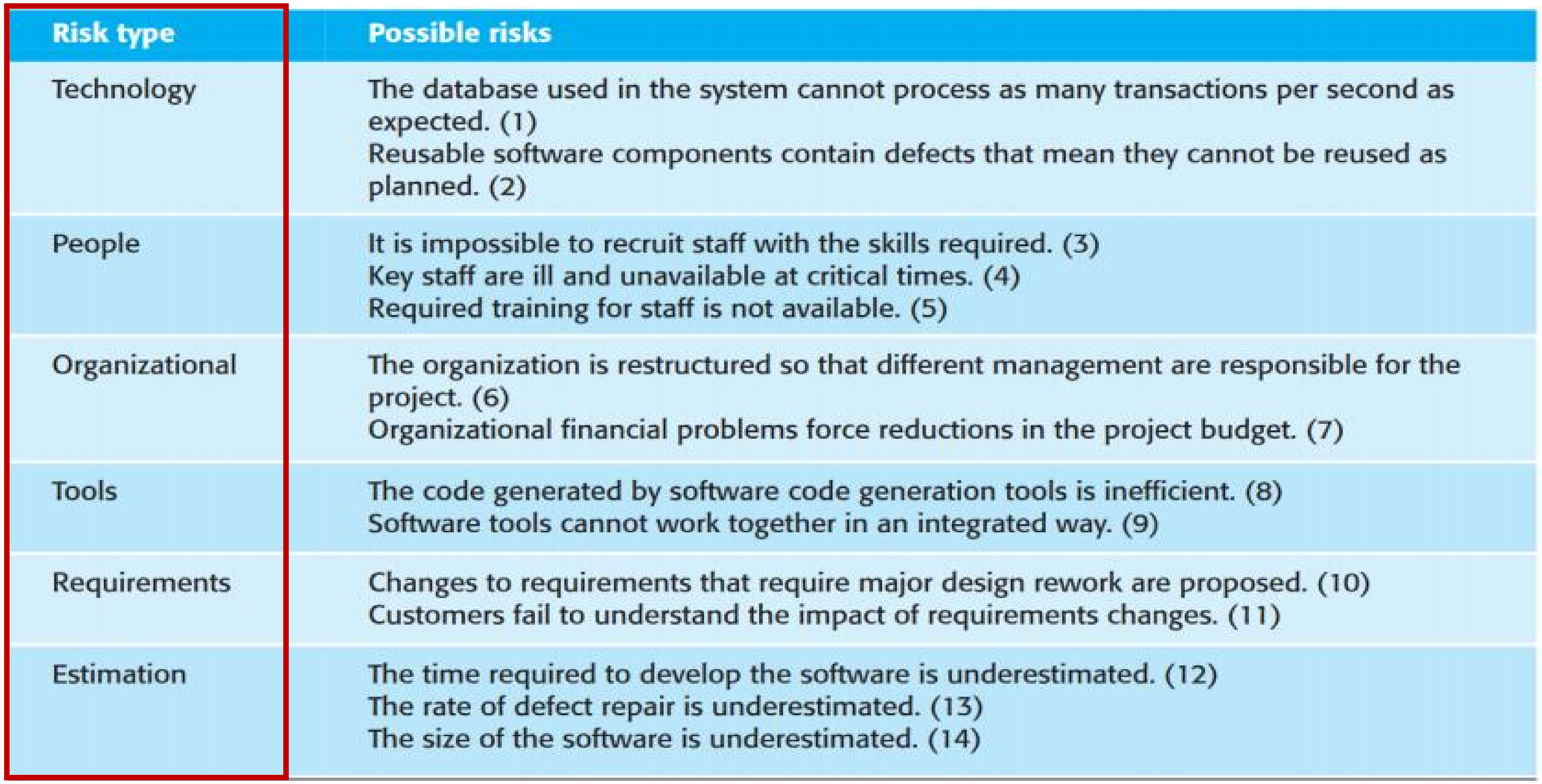


- Risk identification is the **first stage** of the risk management process.
- It is concerned with identifying the risks that could pose a major threat to the software engineering process (project), the software being developed (product), and the development organization (business).
- The identification is usually team process, or sometimes, the project manager’s call

2.2.1 Risk Identification (cont.)

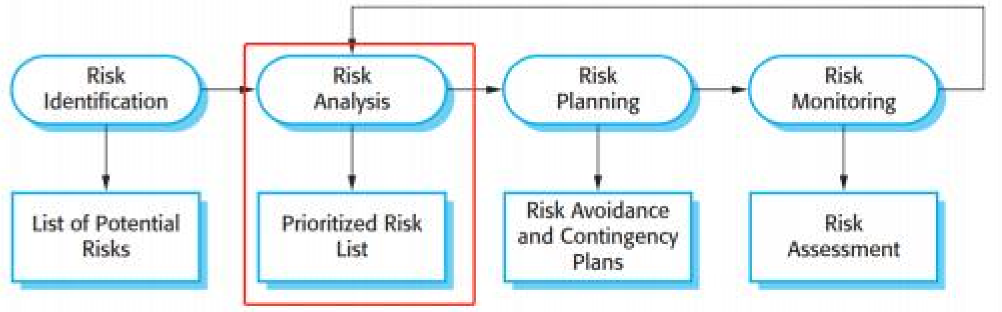


Risk Checklist



Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. (1) Reusable software components contain defects that mean they cannot be reused as planned. (2)
People	It is impossible to recruit staff with the skills required. (3) Key staff are ill and unavailable at critical times. (4) Required training for staff is not available. (5)
Organizational	The organization is restructured so that different management are responsible for the project. (6) Organizational financial problems force reductions in the project budget. (7)
Tools	The code generated by software code generation tools is inefficient. (8) Software tools cannot work together in an integrated way. (9)
Requirements	Changes to requirements that require major design rework are proposed. (10) Customers fail to understand the impact of requirements changes. (11)
Estimation	The time required to develop the software is underestimated. (12) The rate of defect repair is underestimated. (13) The size of the software is underestimated. (14)

2.2.2. Risk Analysis



To consider each identified risk and make a judgment about the probability and seriousness of that risk

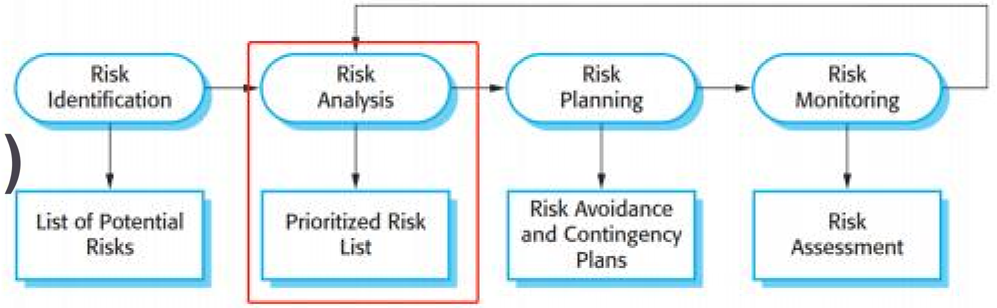
Probability

- Very low (<10%),
- Low (10–25%),
- Moderate (25–50%),
- High (50– 75%), or very high (>75%)

Seriousness

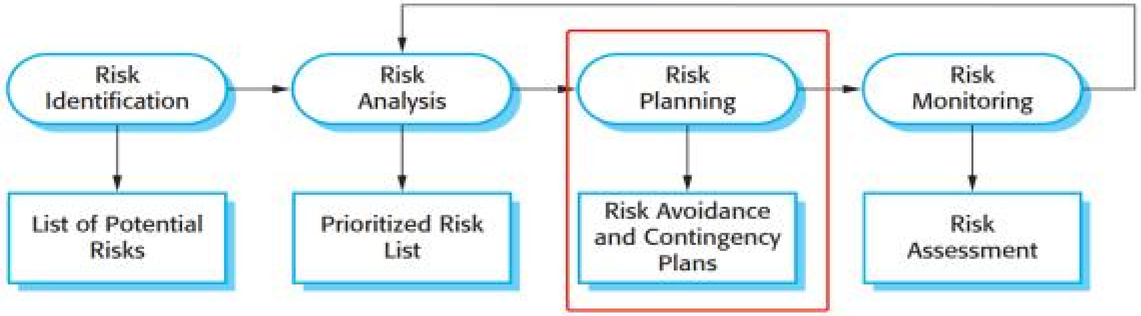
- Catastrophic (threaten the survival of the project),
- Serious (would cause major delays),
- Tolerable (delays are within allowed contingency),
- Insignificant

2.2.2 Risk Analysis (cont.)



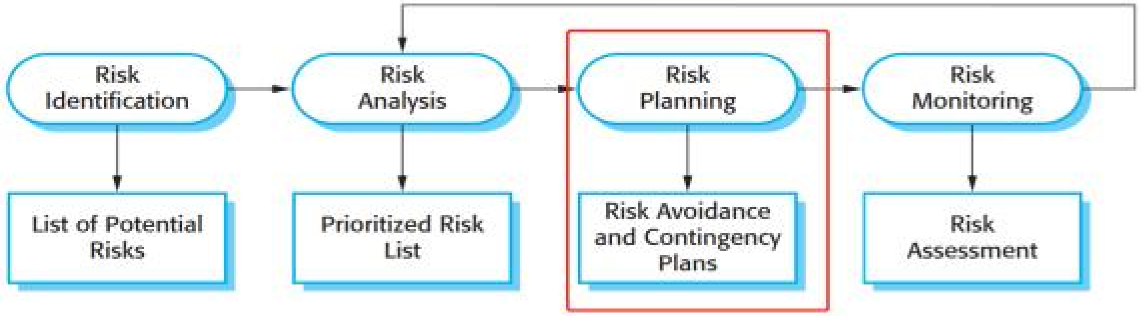
Risk	Probability	Effects
Organizational financial problems force reductions in the project budget (7).	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project (3).	High	Catastrophic
Key staff are ill at critical times in the project (4).	Moderate	Serious
Faults in reusable software components have to be repaired before these components are reused. (2).	Moderate	Serious
Changes to requirements that require major design rework are proposed (10).	Moderate	Serious
The organization is restructured so that different management are responsible for the project (6).	High	Serious
The database used in the system cannot process as many transactions per second as expected (1).	Moderate	Serious
The time required to develop the software is underestimated (12).	High	Serious
Software tools cannot be integrated (9).	High	Tolerable
Customers fail to understand the impact of requirements changes (11).	Moderate	Tolerable
Required training for staff is not available (5).	Moderate	Tolerable
The rate of defect repair is underestimated (13).	Moderate	Tolerable
The size of the software is underestimated (14).	High	Tolerable
Code generated by code generation tools is inefficient (8).	Moderate	Insignificant

2.2.3. Risk Planning



- To consider each of the key risks that have been identified, and develops strategies to manage these risks
- To think of actions that you might take to minimize the disruption to the project if the problem identified in the risk occurs
- To think about information that you might need to collect while monitoring the project so that problems can be anticipated

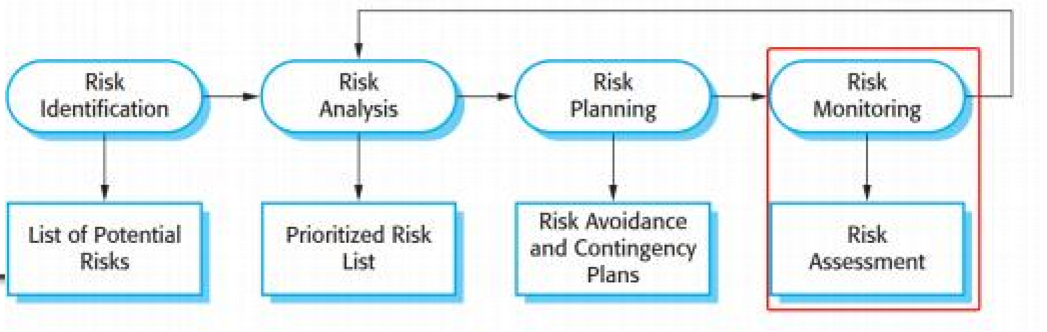
2.2.3 Risk Planning (cont.)



✧ Three strategies

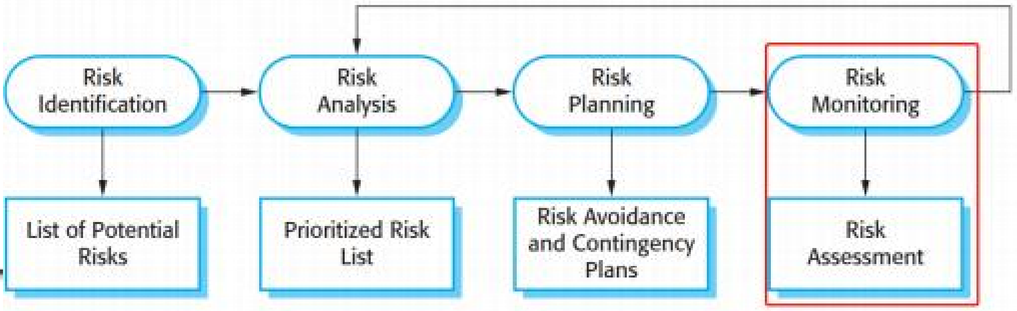
Strategy	Concept	Goal	Example
Avoidance	Change the plan so the risk does not happen.	**Reduce the probability** that the risk will occur.	The team worries that a new library might cause errors, so they decide not to use it and choose a stable one they already know.
Minimization	Take measures to reduce negative effects if the risk happens.	**Reduce the impact** of the risk.	The team fears the server may get slow, so they add simple speed checks and show a loading message to users instead of letting the app freeze.
Contingency Plan	Have a backup plan ready in case the worst happens.	Make sure the team can **recover quickly** when the risk occurs.	If the new version has problems, the team can press one button to go back to the older stable version.

2.2.4 Risk Monitoring

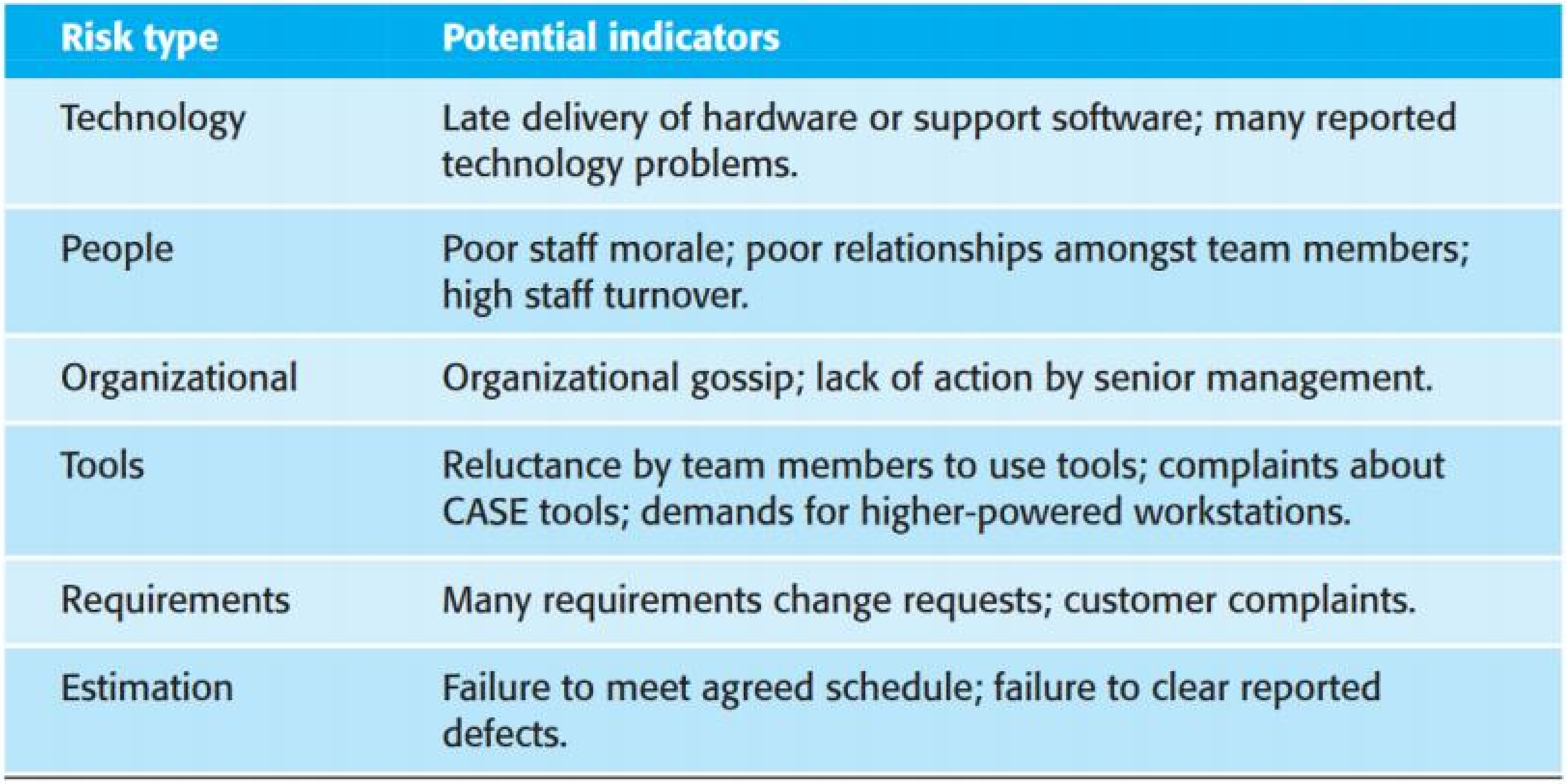


- Risk monitoring is the process of checking that your assumptions about the product, project, and business risks have not changed.
- Regularly assess each of the identified risks to decide whether or not that risk is becoming more or less probable. (Probabilty)
- Also think about whether or not the effects of the risk have changed. (Seriousness)

2.2.4 Risk Monitoring (cont.)



Indicators for monitoring



Risk type	Potential indicators
Technology	Late delivery of hardware or support software; many reported technology problems.
People	Poor staff morale; poor relationships amongst team members; high staff turnover.
Organizational	Organizational gossip; lack of action by senior management.
Tools	Reluctance by team members to use tools; complaints about CASE tools; demands for higher-powered workstations.
Requirements	Many requirements change requests; customer complaints.
Estimation	Failure to meet agreed schedule; failure to clear reported defects.

3.1 Managing People - Why Important

People working in a software organization are its **greatest assets**

- Cost a lot to recruit and retain good people
- Ensure that the organization gets the best possible return on its investment

3.2 Managing People - 4 Criteria

Four critical factors in people management

- **Consistency:** Managers should apply rules and expectations in a consistent way to all team members.
- **Respect:** Managers should recognize and value each person's individual skills and differences.
- **Inclusion:** Team members perform better when they feel their views are heard and considered.
- **Honesty:** Managers and team members must communicate openly about progress, problems, and limitations.

3.3. Personality matters

Personality is important when managing people.

- **Task-oriented people** They are motivated by the work itself, especially the intellectual challenge of software development.
  - Give them challenging technical tasks, allow autonomy in technical decisions, etc
- **Self-oriented people.** They are driven by personal success, recognition, and career advancement.
  - Set clear performance goals and reward mechanisms, provide chances for leadership roles, etc
- **Interaction-oriented people.** They enjoy working with others and are energized by teamwork and collaboration.
  - Assign them to roles requiring communication (e.g, user study), place them in collaborative tasks (e.g., participatory design)

4.1 Teamwork in Software Engineering

- Large teams **should be divided** into smaller working groups
  - In software development, it is not practical for every member of a large team to work on the same task.
  - Large teams are therefore organized into smaller groups, each responsible for a specific subsystem or component.

- Creating the right group composition is a key managerial responsibility. When forming groups, managers must consider how to **distribute:**
  - technical expertise
  - experience levels
  - personality types

  This ensures each group has the capabilities needed for its assigned work

- Within each group, members need to **agree on**:
  - how tasks are shared
  - how decisions are made
  - how communication happens

  This internal coordination allows the group to work toward the same development goals

4.2 Teamwork - Influencial Factors

Project and organizational context influences team effectiveness

- How well a team performs partly depends on the nature of the project and the organization that supports it.
- E.g. A team building a safety-critical medical system must follow strict procedures, while a team doing a mobile app prototype can work more flexibly.

Besides, there are **three generic factors**:

- **The people in the group:** Team members' skills, experience, and personalities influence how the team functions
- **The group organization**: How the team is structured affects coordination and responsibilities (e.g., based on software modules, based or personal skills, or based on experiences?)
- **Technical and managerial communications:** Teams need clear communication about both technical issues and project management

4.3 Teamwork - Benefits of cohesive groups

- Benefits of creating a **cohesive group** (*a team where members feel connected, work well together, and see themselves as part of the same unit*):
  - **The group can establish its own quality standards**.
    - E.g. The group decides together that "every method must have comments," so all members naturally stick to this rule to improve the code quality.
  - **Individuals learn from and support each other.**
    - E.g. A junior developer who struggles with some problems gets guidance from other group members instead of working alone.
  - **Knowledge is shared.**
    - E.g. If the person who usually handles the database is sick, another teammate can update the database because they all know how it works.
  - **Refactoring and improvement is encouraged**.
    - E.g. Anyone can improve the code, not just the original author

4.4 Group communications

It is absolutely essential that group members communicate effectively and efficiently with each other and with other project stakeholders.

- Group members must exchange information on the status of their work, the design decisions that have been made, and changes to previous design decisions.
- They have to resolve problems that arise with other stakeholders and inform these stakeholders of changes to the system, the group, and delivery plans.
- Good communication also helps strengthen group cohesiveness. Group members come to understand the motivations, strengths, and weaknesses of other people in the group.

4.4 Group communications (cont.)

The effectiveness and efficiency of communications is influenced by:

- **Group size** As a group gets bigger, it gets harder for members to communicate effectively.
- **Group structure** People in informally structured groups communicate more effectively than people in groups with a formal, hierarchical structure.
- **Group composition** People with the same personality types may clash and, as a result, communications can be inhibited.
- **The physical work environment** The organization of the workplace is a major factor in facilitating or inhibiting communications.
- **The available communication channels** face-to-face, e-mail messages, formal documents, telephone, and Web 2.0 technologies such as social networking and wikis.

5.1 Continuous Improvement

- **Definition of Continuous Improvement**
  - Refers to the ongoing effort to enhance processes, products, or services incrementally over time or through breakthrough improvements.
  - It is a systematic approach aimed at increasing efficiency, reducing waste, improving quality, and delivering greater value to customers.

- **In the context of Software Engineering:**
  - Regularly evaluating and refining software development processes.
  - Leveraging feedback from stakeholders, users, and system monitoring.
  - Incorporating small, iterative changes or significant innovations to:
    - Enhance code quality.
    - Optimize workflows.
    - Reduce technical debt.
    - Improve collaboration among team members.

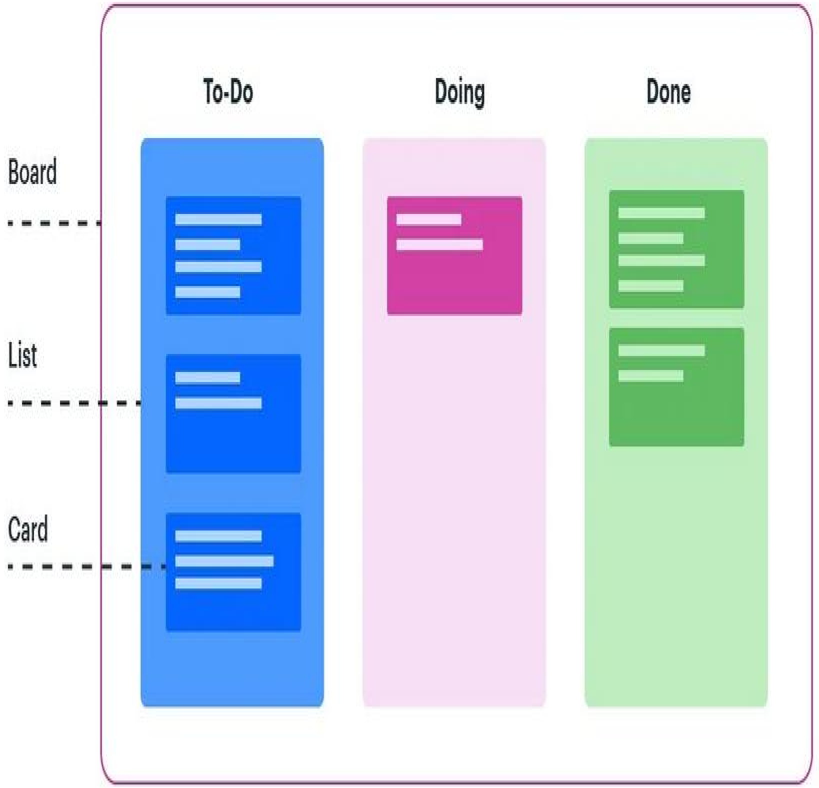
5.2 Continuous Improvement Charateristics

- **Incremental Progress:** Focus on small, consistent changes that accumulate over time.
- **Feedback-driven:** Relies on feedback loops from testing, users, and monitoring.
- **Goal-oriented:** Targets measurable improvements in quality, performance, or team productivity.
- **Collaborative:** Involves all stakeholders in the improvement process, promoting shared responsibility.
- **Cyclic:** Improvement is an ongoing, iterative process rather than a one-time effort.

5.3.1 Strategies for Continuous Improvement - Agile

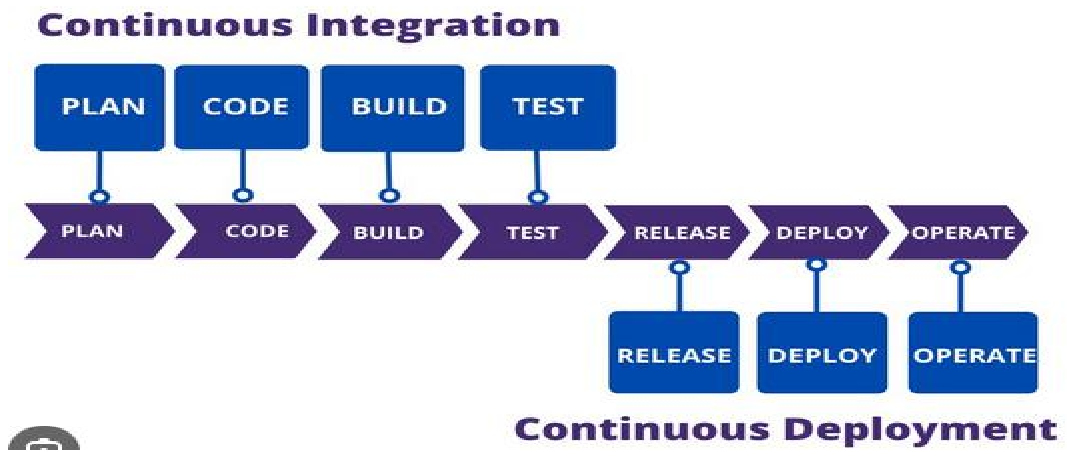
- **Agile Methodologies**
  - Agile focuses on small, frequent improvements, teamwork, and quick responses to change.

- **A popular way in Agile - Scrum:** Scrum uses short development cycles called sprints, followed by meetings (sprint retrospective) to reflect and improve.
  - Sprint – A short working period (e.g., 2 weeks)
  - Sprint Retrospective – A meeting after each sprint to discuss:
    - What went well
    - What went wrong
    - What can be improved next time
  - JIAR and Kanban are the commonly used tools in this process



5.3.2 Strategies for Continuous Improvement - CI/CD

- **Continuous Integration and Continuous Delivery (CI/CD)**
  - **CI/CD is an automated software engineering pipeline** that connects code integration, testing, and deployment into a continuous workflow.
  - **Aim:** to shorten development cycles, reduce integration problems, and ensure software can be released reliably at any time.
  - **Importance:**
    - Faster feedback: Issues are detected minutes after code is committed.
    - Less manual work: Builds, tests, and deployments run automatically.
    - Higher reliability: Every change is verified in a consistent environment.
    - Always-ready software: The system is kept in a deployable state.



**Upload code →
CI (build + test) →
CD (deploy to pre-release environment)**

5.3.2 Strategies for Continuous Improvement - CI/CD

Continuous Integration (CI)

- **CI** is a practice where developers frequently integrate their code into a shared repository. Each integration **automatically triggers a build and test process**

- **Importance:**
  Finds problems early:
  - Errors introduced by new code are detected right after integration.

  Prevents integration conflicts:
  - Frequent merging avoids large, messy conflicts at the end of development.

  Ensures consistent testing:
  - All code is tested in the same environment, not on individual laptops.

  Reduces manual work:
  - Build and test steps are automated, saving developer time.

- **How CI works (a simple case)**
  - A developer pushes their updated code to the shared repository
  - The CI server automatically compiles the code, runs unit tests and other automated checks, and analyzes results and reports failures.
  - The team immediately sees whether the change is safe to use.

5.3.2 Strategies for Continuous Improvement - CI/CD

Continuous Delivery (CD)

- **CD automatically deploys the successfully built code from CI to a testing or pre-release environment**.

- **Importance:**
  Ensures deployable software:
  - Every successful build is automatically packaged and deployed, so the application is always ready for release.

  Reduces release risks:
  - Smaller, frequent deployments reduce the chances of large failures when the software is finally released.

  Supports fast delivery:
  - Teams can release updates quickly because the deployment pipeline is already automated.

- **How CD works (a simple case)**
  - CD automatically packages the application (e.g., .zip or docker image).
  - The system deploys this package to a testing / pre-release environment that mimics production, so that they can release to production at any time with minimal manual steps.

5.3.3 Other Strategies for Continuous Improvement

- **Refactoring**
  - The process of restructuring existing code to improve its readability, maintainability, and performance without changing its external behavior.
- **Monitoring and Feedback**
  - Continuously tracking application performance, while feedback gathers user or system input to identify areas for improvement. (e.g., Track API response times and trigger alerts for anomalies and Collect insights from users about new features or bugs.)
- **Knowledge Sharing**
  - Sharing best practices, lessons, and expertise within the team to improve skills and consistency
- **A/B Testing**
  - Compares two versions of a feature to determine which performs better based on user interaction. (e.g., evaluate "Add to Cart" vs. "Buy Now" buttons)

5.4 Metrics for Continuous Improvement

- The metrics are used to **measure whether continuous development practices are working effectively** in different aspects**:**

- **For Development**
  - Time for Changes: Time taken from a code commit to deployment in production
  - Cycle Time: Time taken from the start of work on a task to its completion
  - Deployment Frequency: How often new code is deployed to production.

- **For Product Quality**
  - Defect Density: The number of defects per unit of code (e.g., defects per 1,000 lines of code)
  - Mean Time to Recovery: Average time taken to recover from a failure
  - Escaped Defects: Defects that are found after deployment to production

5.4 Metrics for Continuous Improvement

- **For Team Productivity**
  - Velocity: The amount of work completed by a team during a sprint, measured in story points or tasks.
  - Work in Progress: The number of tasks currently in progress
  - Flow Efficiency: The ratio of active time to total time (active + idle) spent on a task.

- **For Customer and User**
  - Customer Satisfaction: A direct measure of customer satisfaction with a product or feature
  - Net Promoter Score: Measures the likelihood of customers recommending the product to others
  - Feature Adoption Rate: Percentage of users actively using a newly released feature