Project Management

Software Engineering I

AY 24/25

Week 12

Outline

- Project Management Overview
- Risk Management
- Managing People
- Teamwork
- Continuous Improvement

1.1 Project Management Importance

- 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 cannot guarantee project success BUT a bad management may 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
  - Software cannot be seen or touched.
- ✧ Large software projects are often ‘one-off’ projects
  - Lessons learned from previous projects may not be transferable to new projects.
- ✧ Software processes are variable and organization specific
  - software processes vary quite significantly from one organization to another -> cannot reliably predict when a particular software process is likely to lead to development problems.

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

1.5 Abilities of A Good Project Manager (cont.)

- **Problem solving.** An effective software project manager can diagnose the technical and organizational issues
- **Managerial identity.** A good project manager must take charge of the project. They must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.
- **Influence and team building.** An effective project manager must be able to “read” people; they must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals.

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.

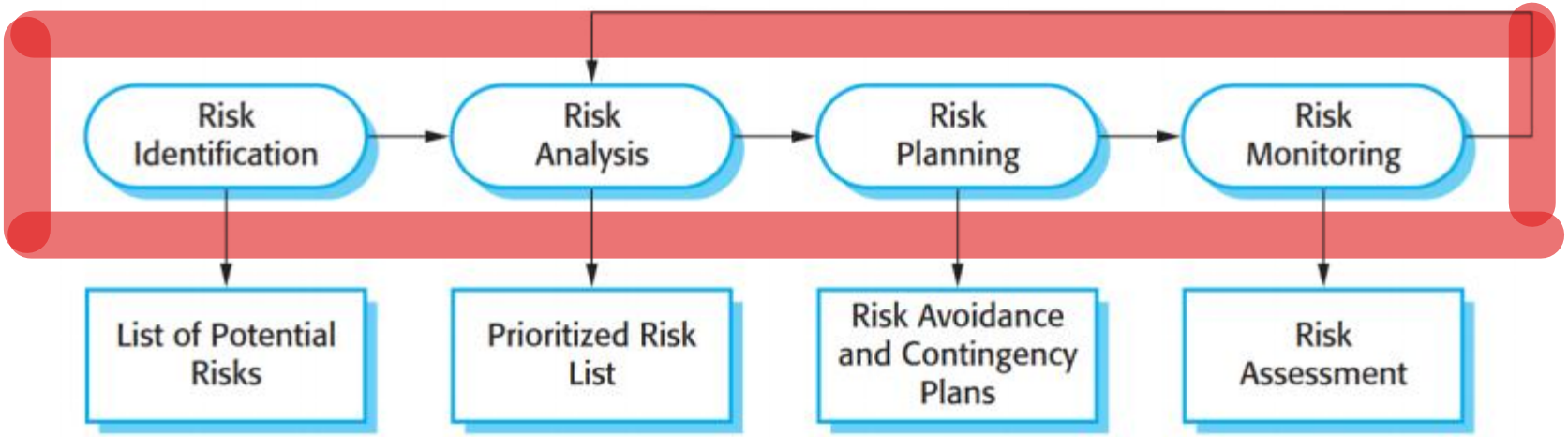
✧ Three related types of risks:

- **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 to perform as expected).
- **Business risks:** Risks that affect the organization developing or procuring the software (e.g., a new product from competitors).

2.1 Risk Management (cont.)

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

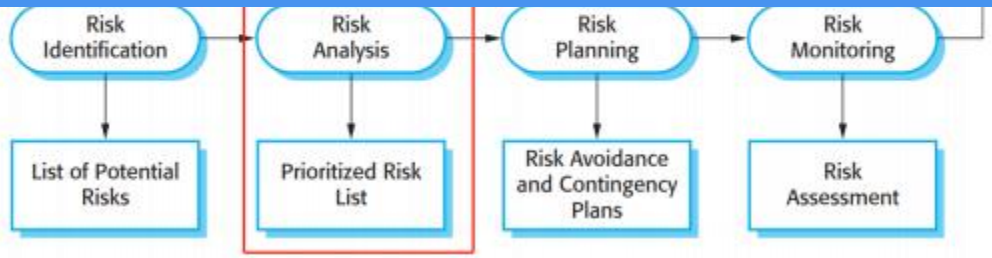
2.2.1 Risk Identification

- 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, the software being developed, and the development organization.
- Team process **and/or** project manager's call

2.2.1 Risk Identification (cont.)

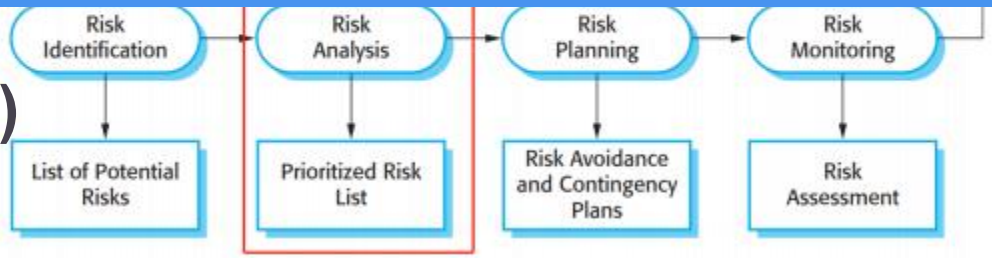
Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. (1) Reusable software components contain defects that mean they cannot be reused as planned. (2)
People	It is impossible to recruit staff with the skills required. (3) Key staff 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 categories of strategies

- **Avoidance strategies:** Following these strategies means that the probability that the risk will arise will be reduced. (e.g., Defective components)

Defective components	Replace potentially defective components with bought-in components of known reliability.

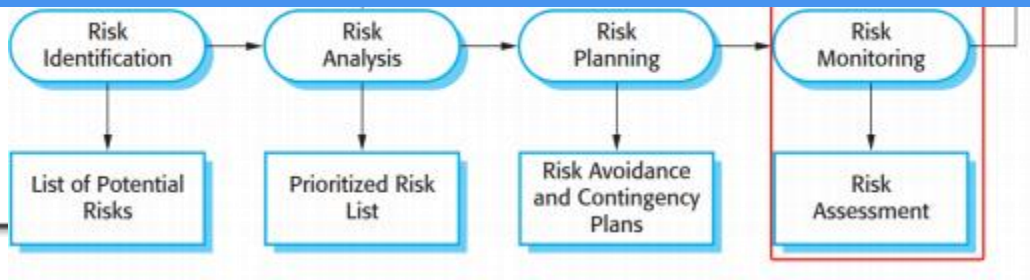
- **Minimization strategies:** Following these strategies means that the impact of the risk will be reduced. (e.g., Staff illness)

Staff illness	Reorganize team so that there is more overlap of work and people therefore understand each other's jobs.

- **Contingency plans:** Following these strategies means that you are prepared for the worst and have a strategy in place to deal with it. (e.g., Organizational financial problems)

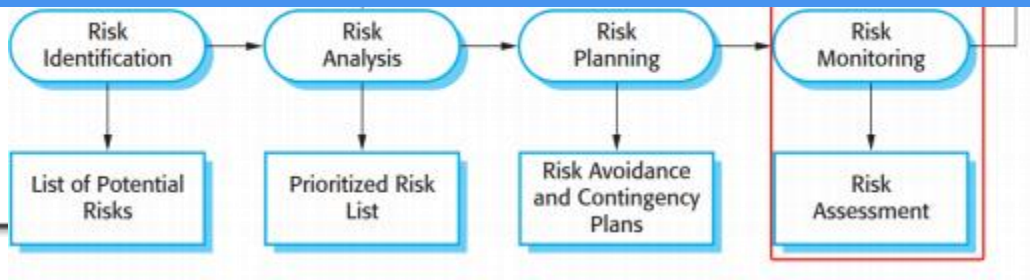
Organizational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business and presenting reasons why cuts to the project budget would not be cost-effective.

2.2.4 Risk Monitoring



- Risk monitoring is the process of checking that your assumptions about the product, process, 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.)



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
  - Costs 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:** People in a project team should all be treated in a comparable way.
- **Respect:** Different people have different skills and managers should respect these differences.
- **Inclusion:** People contribute effectively when they feel that others listen to them and take account of their proposals.
- **Honesty:** Should always be honest about what is going well and what is going badly in the team.

3.3. Motivating People - Motivation

- A project manager, you need to motivate the people that work with you so that they contribute to the best of their abilities.
- Motivation means organizing the work and the working environment to encourage people to work as effectively as possible.

3.3. Motivating People - Motivation (cont.)

✧ Personality type also influences motivation.

- **Task-oriented people**, who are motivated by the work they do. In software engineering, these are people who are motivated by the intellectual challenge of software development.
- **Self-oriented people,** who are principally motivated by personal success and recognition. They are interested in software development as a means of achieving their own goals.
- **Interaction-oriented people,** who are motivated by the presence and actions of co-workers. As software development becomes more user-centered, interaction-oriented individuals are becoming more involved in software engineering.

4.1.1 Teamwork - Why Important

- It is clearly impossible for everyone in a large group to work together on a single problem, large teams are usually split into a number of groups
- Putting together a group that has the right balance of technical skills, experience, and personalities is a critical management task.
- In a cohesive group, members think of the group as more important than the individuals who are group members.

4.1.2 Teamwork - Benefits

- Benefits of creating a cohesive group:
  - **The group can establish its own quality standards** Because these standards are established by consensus, they are more likely to be observed than external standards imposed on the group.
  - **Individuals learn from and support each other** People in the group learn from each other. Inhibitions caused by ignorance are minimized as mutual learning is encouraged.
  - **Knowledge is shared** Continuity can be maintained if a group member leaves. Others in the group can take over critical tasks and ensure that the project is not unduly disrupted.
  - **Refactoring and continual improvement is encouraged** Group members work collectively to deliver high-quality results and fix problems, irrespective of the individuals who originally created the design or program.

4.1.3 Teamwork - Influencial Factors

- Whether or not a group is effective depends, to some extent, on the nature of the project and the organization doing the work.
- However, apart from project and organizational issues, there are three generic factors that affect team working:
  - The people in the group
  - The group organization
  - Technical and managerial communications

4.1.4 Teamwork - A Team of Cohesiveness

- Good project managers should always try to encourage group cohesiveness
  - Organize social events
  - Naming the group and establishing a group identity and territory
  - Explicit group-building activities such as sports and games
  - Be Inclusive

4.2 Selecting group members

- Many software engineers are motivated primarily by their work. Software development groups, therefore, are often composed of people who have their own ideas about how technical problems should be solved.
- A group that has complementary personalities may work better than a group that is selected solely on technical ability.
  - People who are motivated by the work are likely to be the strongest technically.
  - People who are self-oriented will probably be best at pushing the work forward to finish the job.
  - People who are interaction-oriented help facilitate communications within the group

4.3 Group organization

- The way that a group is organized affects the decisions that are made by that group, the ways that information is exchanged, and the interactions between the development group and external project stakeholders.
- Organizational Questions to be considered:
  - Should the project manager be the technical leader of the group?
  - Who will be involved in making critical technical decisions, and how will these be made?
  - How will interactions with external stakeholders and senior company management be handled?
  - How can groups integrate people who are not colocated?
  - How can knowledge be shared across the group?

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

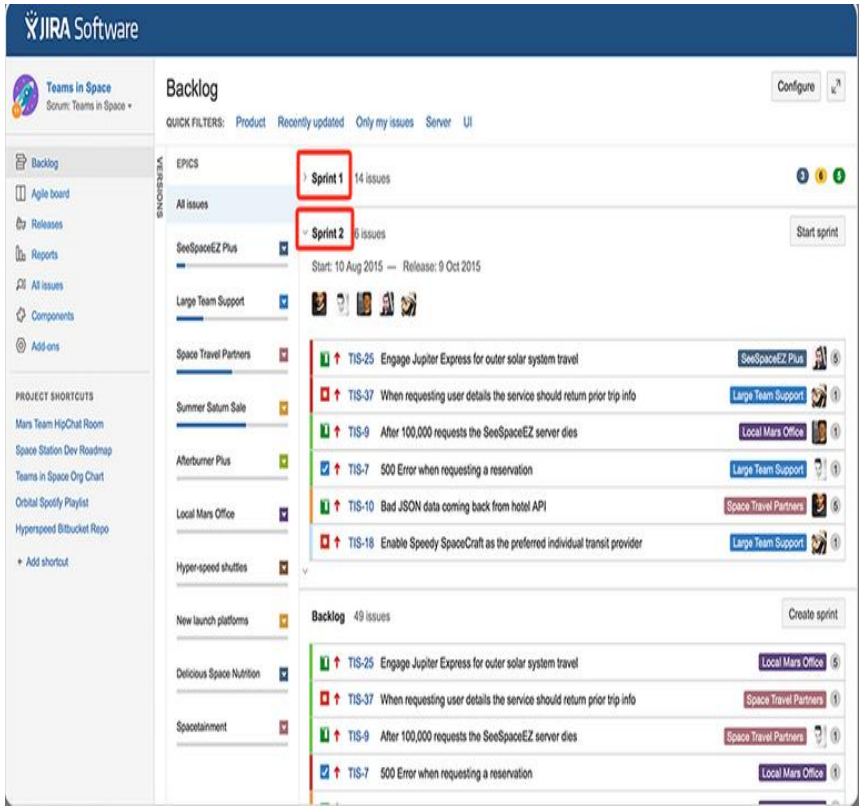
- **Definition of Continuous Improvement**
  - Refers to the ongoing effort to enhance processes, products, or services incrementally over time or through breakthrough improvements.
  - It is a systematic approach aimed at increasing efficiency, reducing waste, improving quality, and delivering greater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

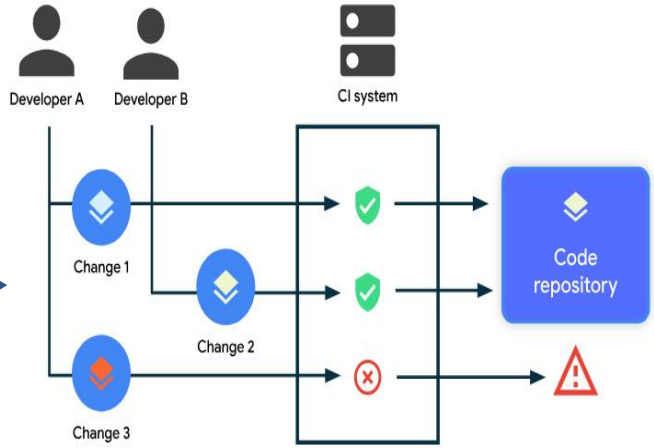
5.3.1 Strategies for Continuous Improvement

- **Agile Methodologies**
  - Emphasizing iterative development, collaboration, and responsiveness to change. Teams deliver small, incremental improvements through cycles called sprints or iterations.
  - *Scrum:* Conduct sprint retrospectives to evaluate and improve team performance.
    - Example: Use JIRA to identify unclear task descriptions and requriements in each Sprint
  - Similarily: *Kanban* - Visualize tasks to identify bottlenecks



5.3.2 Strategies for Continuous Improvement

- **Continuous Integration and Continuous Delivery (CI/CD)**
  - CI/CD automates the process of integrating code changes (CI) and deploying them to production environments (CD), ensuring faster delivery and higher reliability.
  - *Continuous Integration:* Developers frequently integrate their code changes into a shared repository. Automated builds and tests are triggered for each integration to ensure that the new changes do not break the existing codebase.
    - Example: Automated Builds - For a Java application, use Maven or Gradle to compile the project after each commit in the repository
  - *Continuous Delivery:* Continuous Delivery extends CI by automatically deploying code changes to a staging environment for further testing. It ensures that the application is always in a deployable state.
    - Example: Deploy to Staging - Automatically deploy the build artifacts (e.g., Docker images) to a staging environment (pre-production environment that mimics the production setup).



5.3.3 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

- **Metrics** help track the effectiveness of processes, identify areas for improvement, and measure the progress of continuous improvement

- **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