SOFTWARE ENGINEERING

UNIT - 5

TOPIC – 1

RISK MANAGEMENT

Risk Management in Software Engineering

Risk management is the process of identifying, analyzing, and addressing potential problems that could disrupt a software project.

It ensures the project is completed on time, within budget, and with fewer surprises by reducing the impact of unexpected problems.

Why is Risk Management Important?

Risk management is important because it prepares teams to handle challenges effectively, ensuring a smoother workflow and successful project delivery.

Software projects often face unexpected problems. These might include bugs in the code, delays in getting resources, or changes in customer requirements. Risk management prepares teams to handle these issues smoothly.

Think of it like planning a big event. You think about possible problems like rain or guests arriving late, and you prepare solutions, such as setting up tents or having a backup plan. In software engineering, planning for risks ensures the project stays on track and avoids bigger problems later.

Without risk management, projects can become chaotic. Delays, higher costs, and unhappy stakeholders are common results when risks are ignored. Managing risks means teams are better equipped to face challenges and deliver successful results.

What is Risk Management?

Risk management means planning ahead for problems and taking steps to reduce their impact.

It involves:

1. Identifying what could go wrong.
2. Figuring out how serious the problem could be.
3. Deciding on ways to prevent or fix the problem.

For example, if there's a chance that a software feature won't work on older devices, the team can test early and adjust the feature if needed. By being proactive, the team avoids surprises and ensures the project runs smoothly.

Steps in Risk Management

Risk management steps are the actions teams take to find, assess, and handle problems in a project.

Managing risks is a step-by-step process. Here's how it works:

1. **Identify Risks**: Make a list of everything that could go wrong.
    - Example: The software might not work on all devices, or a team member might leave the project.
2. **Assess Risks**: Decide how likely each problem is and how serious it would be if it happened.
    - Example: A delay in testing might be likely and could slow down the entire project.
3. **Plan for Risks**: Create solutions to avoid or reduce the impact of risks.
    - Example: Train multiple people to handle critical tasks so work doesn't stop if someone leaves.
4. **Monitor Risks**: Keep an eye on risks throughout the project and check for new ones.
    - Example: Regular updates and testing can help spot problems early.
5. **Communicate Risks**: Share information about risks and plans with the team and stakeholders.
    - Example: If a delay is likely, inform everyone involved and explain the solution.

Risk Identification

Risk identification is the process of finding and listing potential problems that could affect the project.

This is the first step in risk management, where teams think about what could go wrong and record all possible risks. The goal is to make sure no risk is overlooked.

How to Identify Risks:

1. **Review Past Projects**: Look at issues faced in similar projects.
    - Example: A common issue might be delays during testing.
2. **Brainstorm with the Team**: Discuss potential problems together.
    - Example: The team might identify that a new tool could cause compatibility issues.
3. **Analyze the Project Plan**: Check for areas that depend on tight schedules or limited resources.
    - Example: A critical task might depend on hardware delivery, which could be delayed.
4. **Consult Experts**: Get insights from people experienced in similar projects.
    - Example: A senior developer might warn about specific technical risks.

Risk Projection

Risk projection, also called risk estimation, is predicting how likely each risk is to happen and how serious it could be.

Steps in Risk Projection:

1. **Describe the Risk**: Clearly explain what the risk is.
    - Example: "There is a risk of delays in hardware delivery."
2. **Estimate Likelihood**: Decide how likely it is for the risk to happen (e.g., high, medium, low).
    - Example: Hardware delays might have a medium likelihood.
3. **Assess Impact**: Determine how much trouble the risk would cause if it happens.

- Example: Hardware delays might have a high impact because they could delay testing.

4. **Prioritize Risks**: Focus on risks that are both likely and have a big impact.
    - Example: A table helps organize risks based on likelihood and impact for easier decision-making.

Risk Description	**Likelihood**	**Impact**	**Priority**
Hardware Delay	Medium	High	High
Minor Code Bugs	High	Low	Medium
Team Member Leaves	Low	High	Medium

SOFTWARE ENGINEERING

UNIT - 5

TOPIC – 2

RMMM AND RMMM PLAN

Principles of Risk Management in RMMM

RMMM (Risk Mitigation, Monitoring, and Management) is a structured approach to handling risks in software projects.

Principles of RMMM:

1. **Focus on High-Priority Risks**: Address risks that are most likely to happen or have the biggest impact first.
    - Example: Start by solving risks that could delay the entire project.
2. **Continuous Monitoring**: Keep track of risks throughout the project lifecycle.
    - Example: Use weekly meetings to review risks and spot new ones.
3. **Prepare Contingency Plans**: Have backup plans ready for critical risks.
    - Example: If a server crashes, switch to backup servers immediately.
4. **Involve the Team**: Ensure everyone understands the risks and the strategies to handle them.
    - Example: Share risk details with all team members so they are prepared.
5. **Document Everything**: Keep records of risks, mitigation plans, and outcomes.
    - Example: Use risk information sheets to track each risk.

Using a Risk Table to Track Risks

A risk table is a tool to organize risks by listing their likelihood, impact, and the plans to handle them.

A risk table simplifies risk tracking by summarizing key information in a clear format. It helps teams quickly see which risks need attention and what steps are planned to manage them.

Example of a Risk Table

Risk Name	Likelihood	Impact	Mitigation Plan
Bugs in Code	High	Medium	Regular code reviews and testing.
Team Member Leaves	Medium	High	Cross-train team members.
Server Downtime	Low	High	Set up backup servers.

Each row represents a specific risk. The "Likelihood" column shows how likely it is to occur (e.g., high, medium, low), and "Impact" shows how much trouble it might cause. The "Mitigation Plan" describes steps to prevent or reduce the risk's effect. This table acts as a quick reference to prioritize risks and ensure the team is prepared.

Risk Refinement

Risk refinement means breaking down large risks into smaller, detailed parts so they are easier to manage and address.

Why is Risk Refinement Important?

Big risks can be overwhelming to deal with. By refining them into smaller parts, teams can understand the root causes, create focused solutions, and monitor progress more effectively.

Steps in Risk Refinement:

1. **Analyze the Risk**: Look at the big risk and identify the specific issues causing it.
    - Example: If the risk is software crashing, the causes might include memory problems, coding errors, or hardware compatibility issues.
2. **Create Specific Solutions**: For each issue identified, plan solutions that directly address it.
    - Example: For memory issues, optimize how the program uses memory.
3. **Monitor Changes**: As the project moves forward, keep refining your understanding of the risk and update your plans as needed.
    - Example: If testing reveals new bugs, include those in the refined risk plan.

Risk Information Sheet

A risk information sheet is a document that records detailed information about a specific risk and how to handle it.

What Does a Risk Information Sheet Include?

1. **Risk Name**: A short title to identify the risk.
    - Example: "Server Crash Risk"
2. **Description**: A clear explanation of the risk.
    - Example: "The server may fail during high traffic periods, leading to downtime."
3. **Likelihood**: How likely the risk is to happen (e.g., high, medium, low).
    - Example: "Medium likelihood during peak usage."
4. **Impact**: How much trouble the risk might cause.
    - Example: "High impact because downtime would delay user transactions."
5. **Mitigation Plan**: Actions to reduce the risk or its effects.
    - Example: "Set up backup servers and monitor server load regularly."
6. **Contingency Plan**: What to do if the risk happens.
    - Example: "Switch traffic to backup servers immediately and notify the team."

Example of a Risk Information Sheet:

Risk Name	Description	Likelihood	Impact	Mitigation Plan
Bug Surge Risk	Risk of increased bugs affecting stability	High	Medium	Regular code reviews and tests
Server Crash	Servers might fail before launch	Medium	High	Backup servers and monitoring
Key Team Absence	Critical member may leave mid-project	Low	High	Cross-training and backups

The risk information sheet ensures that teams have all the necessary details about risks and are prepared to act quickly and effectively.

SOFTWARE ENGINEERING

UNIT - 5

TOPIC – 3

INTRODUCTION TO CLOUD AND ITS SERVICES

Introduction to Cloud Computing

Cloud computing is a way to use computer resources like storage, applications, and processing power over the internet instead of owning and managing physical hardware. It lets you access these resources whenever you need them, and you only pay for what you use.

What is Cloud Computing?

Cloud computing means you can:

- **Store files** like photos, videos, and documents on the internet, so you don't need a big hard drive at home.
- **Run programs** that require a lot of computing power without buying expensive computers.
- **Use software** like email or document editors directly through the internet without downloading or installing them.

Imagine renting a car instead of buying one. You use the car only when needed and pay for it based on how long you use it. Similarly, cloud computing gives you the resources you need without the hassle of owning them.

Features of Cloud Computing

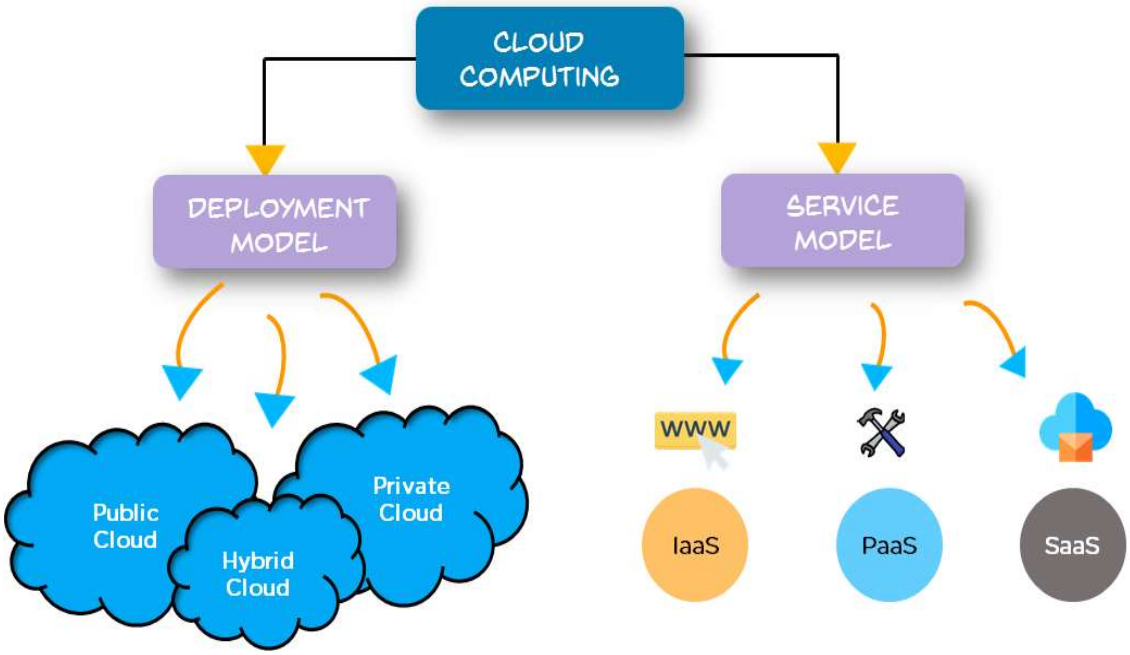
1. **Availability**: Your data and applications are always accessible. Cloud providers work hard to make sure there is little to no downtime.
    - *Example*: Like water from a tap—always ready whenever you need it.
2. **Scalability**: You can adjust the number of resources based on your needs. If you need more storage or computing power, you can get it instantly, and when you don't need as much, you can scale back down.

- *Example*: Like a balloon you can inflate or deflate depending on how much air you need.

3. **Pay-as-you-go**: You only pay for what you use, saving money by not having to invest in expensive hardware upfront.
    - *Example*: Like a phone bill, where you pay for the minutes and data you use.

Types of Cloud Computing

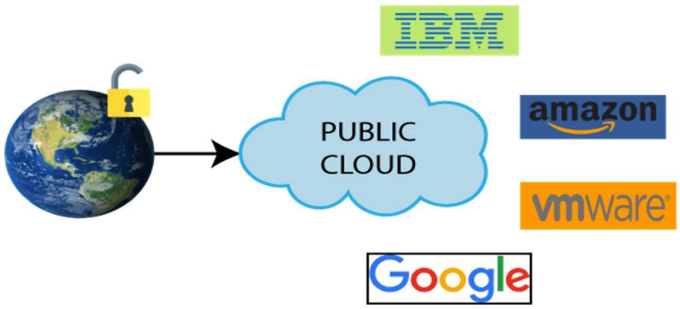
Cloud computing can be categorized based on **how it is deployed** and **the services it offers**.



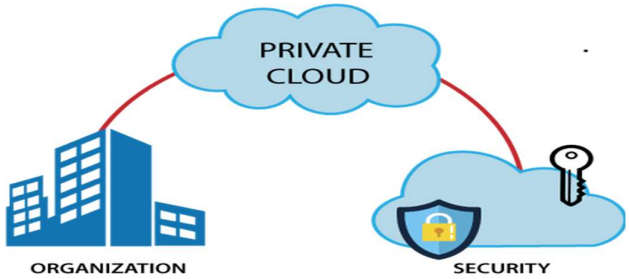
Cloud Deployments

Deployment refers to how and where cloud services are made available. The three types of deployments are:

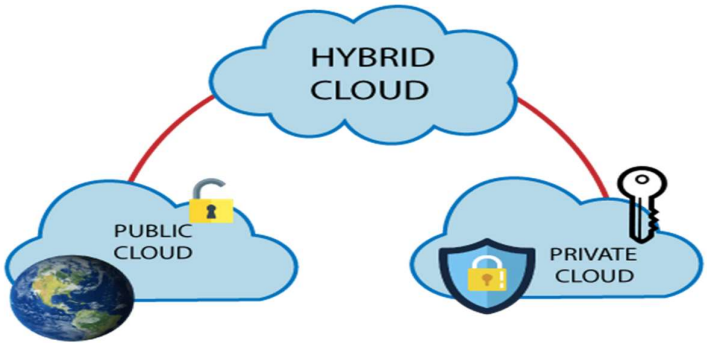
1. **Public Cloud**:
    - Resources like servers and storage are shared by multiple users and managed by companies like Amazon, Microsoft, and Google.
    - *Example*: A public library where anyone can borrow books.



2. **Private Cloud**:
   - Resources are used by a single organization, offering more control and security. It can be managed by the organization or a third-party provider.
   - *Example*: A private library owned by a school, only accessible to students and staff.

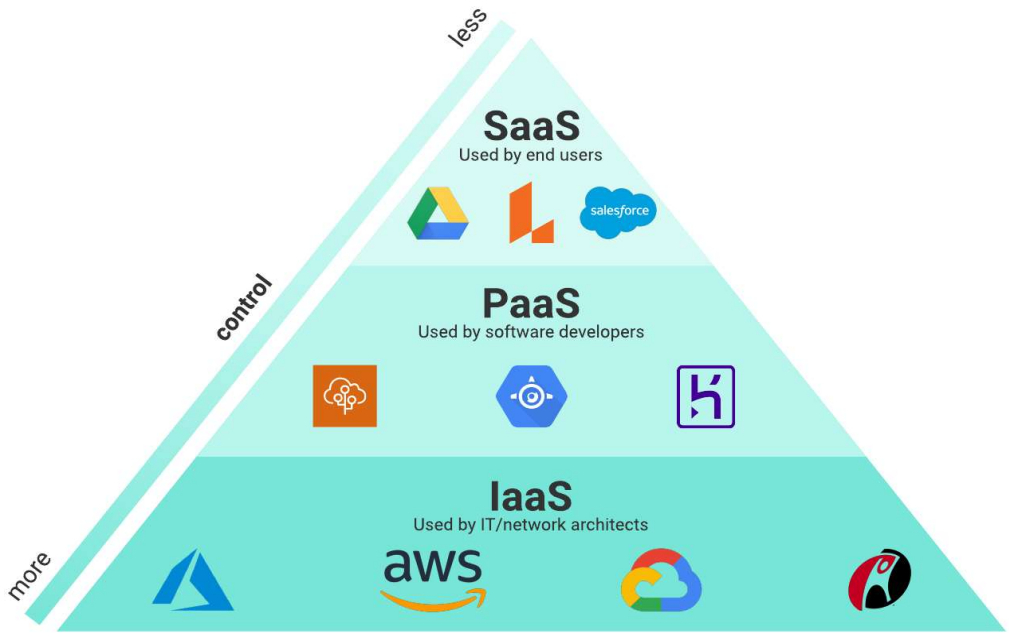


3. **Hybrid Cloud**:
   - A combination of public and private clouds. Organizations can use private clouds for sensitive tasks and public clouds for less critical activities.
   - *Example*: A bakery with a private kitchen for secret recipes and a public shop to sell the goods.



Cloud Services

Cloud services are categorized based on the level of control they provide:



1. **Infrastructure as a Service (IaaS)**:
   - Provides basic resources like virtual machines and storage. You are responsible for managing everything else.
   - *Example*: Renting an empty plot where you can build anything you want.
2. **Platform as a Service (PaaS)**:
   - Offers tools and platforms for developers to build applications without worrying about the underlying hardware.
   - *Example*: A fully equipped kitchen where you can cook without buying the appliances.
3. **Software as a Service (SaaS)**:
   - Provides ready-made software that you can use directly over the internet.
   - *Example*: Using Gmail or Microsoft Word online without installing anything on your computer.

How AWS Helps with Cloud Computing

AWS (Amazon Web Services) is a platform that offers cloud services for building, managing, and scaling applications easily. It removes the need to own physical servers, allowing you to focus on your projects.

Key AWS Services

1. **Amazon EC2**:
    - Let's you rent virtual servers to run applications.
    - *Example*: Like booking a hotel room based on your needs—a single room or a suite.
2. **Amazon S3**:
    - Provides storage space to save and manage your files securely.
    - *Example*: A giant online locker where you can keep your files.
3. **Amazon RDS**:
    - Offers managed databases, so you don't have to worry about setting them up or maintaining them.
    - *Example*: Like hiring an assistant to organize and maintain your files.
4. **Amazon VPC**:
    - Allows you to create a private and secure section of the cloud for your resources.
    - *Example*: A private office within a shared building where only authorized people can enter.
5. **AWS Lambda**:
    - Runs your code automatically when needed without requiring servers.
    - *Example*: A light sensor that turns on a light only when it detects motion.

Steps to Create Cloud Infrastructure Using AWS

Here is how you can set up infrastructure on AWS step by step:

1. **Sign Up**:
    - Go to the AWS website and register for an account. Fill in your details and set up a payment method.

2. **Log In**:
    - Use your account credentials to access the AWS dashboard.
3. **Explore Services**:
    - Familiarize yourself with AWS services like EC2 for virtual servers, S3 for storage, and VPC for networking.
4. **Launch a Virtual Server**:
    - Go to EC2, choose an instance type, select an operating system, and launch your virtual machine.
5. **Set Up Storage**:
    - Use S3 to create a storage bucket where you can upload and manage your files.
6. **Configure Networking**:
    - Use VPC to design a secure network for your application.
7. **Monitor Resources**:
    - Use CloudWatch to track and optimize your usage, ensuring efficient performance.
8. **Secure Access**:
    - Set up user roles and permissions using AWS IAM to control who can access your resources.

Deploying a Web Application

To deploy a static web application using Docker and Nginx, follow these steps:

Steps to Deploy

1. **Push Code to GitHub**:
    - Store your application's code on GitHub. Initialize a Git repository, commit the files, and push them to a GitHub repository.
2. **Create a Virtual Machine**:
    - Launch an EC2 instance with Ubuntu or another operating system. Configure storage and security groups for access.
3. **Clone the Code and Install Docker**:
    - Use Git to clone your application's code from GitHub onto the virtual machine.

- Install Docker on the virtual machine. Docker is a tool that helps package and run your application in containers.

4. **Write and Build a Dockerfile**:
   - Write a Dockerfile to define how your application will run. Build the Docker image using the `docker build` command.
5. **Run the Application**:
   - Use the `docker run` command to create a container from the Docker image. Map it to port 80 so it can be accessed from the internet.
   - Use the public IP address of your EC2 instance to access your web application.