Chapter 7:9. Why is it difficult to establish a set of privacy standards that can be applied internationally in software products?

Privacy: privacy is the right to be let alone, free from interference or intrusion. Privacy is not just about confidentiality, but also about having control over our own domains and knowledge about what is done with those domains. The importance of privacy has changed over time and individuals have their own views on what degree of privacy is important.

Example: Some people may be willing to reveal information about their friends and colleagues by uploading their contacts list to a software system; others do not wish to do so.

- To maintain privacy, you need to have a secure system. However, security and privacy are not the same things. People have different opinions on privacy, so it is impossible to establish objective "privacy standards" based on a definition of "sensitive personal information."
- It is troublesome to set up a set of security measures that can be connected globally to computer program items since universal program items have higher protection and security and have suitable protection approaches and are a fundamental organization that ought to have input sometime recently the software phases.
  The previous will guarantee satisfactory consideration and assets are committed to the protection, and the last mentioned will serve as the premise for suitable security necessities at the start of the program improvement process.
- So that they fulfill essentials it's troublesome to set up a set security handle in program designing.

The following are the 3 challenges in creating a set of privacy standards for software products that can be widely used internationally:

i. Copyright/Downloading: Programs like Napster and LimeWire, which enable users to download music, software, and videos for free, have made copyright/downloading one issue into a serious issue. Many people, especially kids, are unaware of the significant effects of their conduct.

ii. Hacking: One of the problems is when someone intentionally harms a different machine or computer network. This can involve modifying a website without authorization, acquiring passwords to access a site, or stealing classified material. Because computers govern the world, it's crucial to stop hackers. They might produce viruses that take down crucial websites or computer networks. These severe repercussions must be explained to kids.

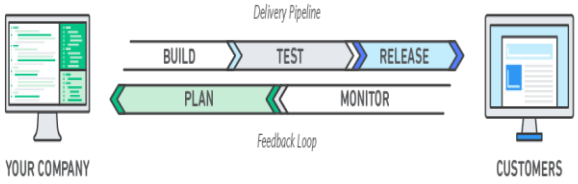
iii. Cyberbullying: This problem is spreading, and people are starting to realize how it affects kids. This problem, which many people believed to be innocuous bullying, was brought to light by the Megan Meier case. This teenage girl allegedly committed herself as a result of online bullying that she experienced over email and MySpace.

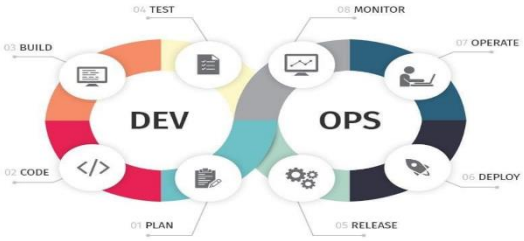
10.1. Explain why adopting DevOps provides a basis for more efficient and effective software deployment and operation.

DevOps: DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

Under a DevOps model, development and operations teams are no longer "siloed." Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.





Your business success hinges on the efficient collaboration of these two functions to improve software delivery speed and quality. DevOps service is critical to faster and more efficient software delivery. DevOps, a portmanteau of **"Development"** and **"Operations"**, is a set of tools, processes, and practices that combine software development and IT operations to improve software delivery. Due to the tendency for development and operations teams to work in silos, DevOps acts as the go-between to improve collaboration efficiency.

Reasons to adopt DevOps.

1. Efficient Development and Deployment Planning: One of the biggest headaches of any IT Manager is managing cross-functional teams to develop and deploy software in good time. DevOps practices help you plan how both teams will work cohesively to deliver products, meet customer needs and stay competitive.

DevOps methods such as **Scrum** and **Kanban** provide practices that define how teams work together. For example, Scrum practices include working in sprints, holding Scrum meetings, and designating time boxes. These work to reduce blockers quickly and complete work within a fixed period before starting another project. Kanban manages workflows by visualizing work on a **Kanban Board**, and there's real-time communication of your work's progress.

2. Continuous Improvement and Software Delivery: Adopting DevOps practices improves the quality of your software when deploying new features and allows you to make changes rapidly.

Continuous Integration and **Continuous Deployment (CI/CD)** is the practice where you make incremental changes to your code and quickly merge to the source code. This DevOps practice delivers your software to the market faster while addressing customer issues rapidly.

3. Improves Software Security: A subset of DevOps is **DevSecOps** which combines **development, security,** and **operations**. Adopting DevOps practices allows you to **build security** into your software continuously.

The IT security team is involved in the software development cycle from the start rather than at the deployment stage. Outdated security practices integrate infrastructure security features independently. With DevOps, software security is a collaborative practice and is considered the foundation before any product development.

4. Improves Customer Experiences: A major benefit to adopting DevOps practices is it **lowers the failure rates** of new features while **improving recovery time**. The continuous deployment, testing, and feedback loop ensure faster service delivery and happier customers.

By automating the software pipeline, the development teams focus on creating superior products while the operations team can improve business delivery.

5. Better Collaboration Among Teams: DevOps practices dismantle silos between teams and improve collaboration. By adopting methods such as **Scrum** and **Kanban**, teams collaborate more efficiently, friction among colleagues is dealt with quickly, and communication flows seamlessly across the organization. Since DevOps discourages hierarchies among teams, everyone is responsible for software quality and speedy delivery. This approach **reduces** cases of **low-effort contributors** and **blame games**.

6. DevOps Practices Create More Time to Innovate: DevOps automates **repetitive tasks**, improves **service delivery**, and reduces the **software development cycle**. The methodology frees up time for teams to develop new products and better features. Unfortunately, inefficient teamwork leads to time wasted fixing errors rather than innovating. Building better products and services lead to a competitive advantage.

Companies that innovate continuously are also better able to react to market changes and take advantage of new opportunities.

7. DevOps Enhances Decision-Making: DevOps helps you leverage data to **make better decisions** and **removes hierarchies** that require layers of approvals. In addition, DevOps practices **reduce the time spent** between design and execution. **Scrum** and the **Kanban** approach address blockers as they come, manage workflows efficiently and tackle projects in sprints.

Fast decision-making based on accurate data is a competitive advantage. Slow decision-making delays go-to-market plans and throttles the implementation of new ideas. According to a **McKinsey Global Survey**, only **20% of organizations** are good at **decision-making,** demonstrating **room for improvement by adopting DevOps**.

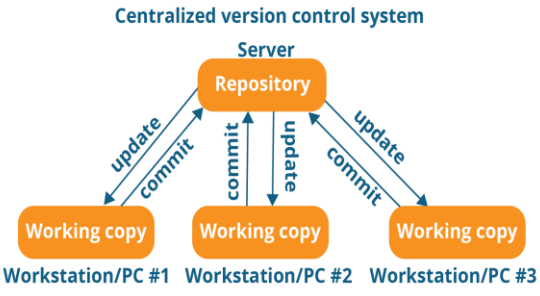
8. DevOps Encourages Higher Trust and Better Collaboration: Due to improved collaboration across development and IT operations, teams trust each other more and produce higher quality work. Higher trust leads to improved morale and better productivity. Conversely, teams working in silos mistrust each other's new suggestions and ideas.

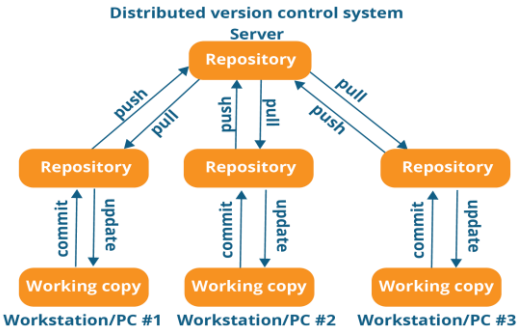
A culture of trust and collaboration creates transparency in workflows and gives visibility into projects. This approach leads to better development and deployment planning. The net effect is a collaborative environment that works in unity, solves problems faster, and delivers superior products to the market.

10.3. What is the fundamental difference between distributed and centralized code management systems? How does this difference lead to the most significant benefits of distributed code management systems?

Code management is a set of software-supported practices used to manage an evolving codebase. It is needed to ensure that changes made by different developers do not interfere with each other and to create different product versions. Code management was originally called "software configuration management" and this term is still widely used. However, "configuration management" is now commonly used to refer to the management of server infrastructure. Early source code management systems had a centralized repository architecture that requires users to check in and check out files.

Centralized: In a centralized, a server acts as the main repository that stores every version of code. Using centralized source control, every user commits directly to the main branch, so this type of version control often works well for small teams, because team members can communicate quickly so that no two developers want to work on the same piece of code simultaneously. Strong communication and collaboration are important to ensure a centralized workflow is successful.





Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository. Contributors then push commits to the server and resolve any merge conflicts on the main repository. As a client-server model, a centralized workflow enables file locking so that any piece of the code that's currently checked out will not be accessible to others, ensuring that only one developer can contribute to the code at a time.

Advantages:

- Works well with binary files: Binary files, such as graphic assets and text files, require a large amount of space, so software developers turn to centralized version control systems to store this data. With a centralized server, teams can pull a few lines of code without saving the entire history on their local machine.
- Offers full visibility: With a centralized location, every team member has full visibility into what code is currently worked on and what changes are made. This knowledge helps software development teams understand the state of a project and provides a foundation for collaboration since developers share work in the central server.

Distributed version control systems, like Git, use multiple repositories, which can decrease insight into projects.

- Centralized version control is easy to understand and use, so developers of any skill level can push changes and start contributing to the code quickly. Setting up the system and the workflow is also simple and doesn't require a significant amount of time investment to establish how the software development team should use the tool.

Disadvantages:

- A single point of failure risks data: The biggest disadvantage is the single point of failure embedded within the centralized server. If the remote server goes down, then no one can work on the code or push changes. The lack of offline access means that any disruption can significantly impact code development and even result in code loss. When storing all versions on a central server, teams risk losing their source code at any

time. Only the snapshots on local machines are retrievable, but that is a small amount of code compared to the entire history of a project.

Unlike a centralized, a distributed version control system enables every user to have a local copy of the running history on their machine, so if there's an outage, every local copy becomes a backup copy and team members can continue to develop offline.

- Slow speed delays development: Centralized version control system users often have a difficult time branching quickly, because users must communicate with the remote server for every command, which slows down code development. Branching becomes a time-consuming task and allows merge conflicts to appear because developers can't push their changes to the repository fast enough for others to view.
- Few stable moments to push changes: A centralized workflow is easy for small teams to utilize, but there are limitations when larger teams try to collaborate. When multiple developers want to work on the same piece of code, it becomes difficult to find a stable moment to push changes. Unstable changes cannot be pushed to the main repository, so developers must keep them local until they're ready for release.

Distributed:

These systems do not necessarily rely on a central server to store all the versions of a project file. In Distributed VCS, every contributor has a local copy or "clone" of the main repository i.e., everyone maintains a local repository of their own which contains all the files and metadata present in the main repository. Every programmer maintains a local repository on its own, which is the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

It uses a peer-to-peer approach to version control, as opposed to the client-server approach of centralized systems. Distributed revision control synchronizes repositories by transferring patches from peer to peer. There is no single central version of the codebase; instead, each user has a working copy and the full change history.

They can update their local repositories with new data from the central server by an operation called "**pull**" and affect changes to the main repository by an operation called "**push**" from their local repository.

Advantages :(compared with centralized systems)

- All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.
- Committing new change sets can be done locally without manipulating the data on the main repository. Once you have a group of change sets ready, you can push them all at once.
- Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.

- If the central server gets crashed at any point in time, the lost data can be easily recovered from any one of the contributor's local repositories.
- Allows users to work productively when not connected to a network.
- Common operations (such as commits, viewing history, and reverting changes) are faster because there is no need to communicate with a central server.
- Allows private work, so users can use their changes even for early drafts they do not want to publish.
- Working copies effectively function as remote backups, which avoids relying on one physical machine as a single point of failure.
- Allows various development models to be used, such as using development branches or a Commander/Lieutenant model.
- Permits centralized control of the "release version" of the project.
- On FOSS software projects it is much easier to create a project fork from a project that is stalled because of leadership conflicts or design disagreements.

Disadvantages (compared with centralized systems) include:

- Initial checkout of a repository is slower as compared to checkout in a centralized version control system because all branches and revision history are copied to the local machine by default.
- The lack of locking mechanisms that is part of most centralized VCS and still plays an important role when it comes to non-mergeable binary files such as graphic assets or too complex single file binary or XML packages
- Additional storage is required for every user to have a complete copy of the complete codebase history.
- Increased exposure of the code base since every participant has a locally vulnerable copy.