Sarah Ewing
11/09/2024
Module 3.2 Assignment

Version Control Guidelines

Version control is an essential tool in software development. Version control tracks and manages changes to software code. Many websites discuss "Version Control Guidelines," and some refer to them as best practices.

The first resource I found that talks about version control was published by Modern Requirements: https://www.modernrequirements.com/blogs/version-control-best-practices/. In it, they talk about five best practices developers and organizations should follow when using version control. The first best practice talks about version numbering. They suggest that version numbering follows a Major.Minor.Patch format. The Major version is for significant changes that break the API, the minor version is for minor changes that do not break the API, and the patch number is for bug fixes. The second best practice is having a central repository for managing and storing code. This helps teams collaborate and keep track of changes. The third best practice is branching and merging strategies for simultaneous version development. Using these strategies, teams can work on features, bug fixes, experiments, and integrations simultaneously without conflict. The fourth best practice is having good documentation and commit tags/messages. When team members document their changes well, it is easier for others to understand the changes that are taking place. The last best practice is security and intellectual property. Protecting intellectual property by having backup and failover mechanisms and implementing access control is essential.

The second resource I found that talks about best practices for version control was published by Perforce: https://www.perforce.com/blog/vcs/8-version-control-best-practices. This article lists eight best practices. The first is to commit changes atomically. This means that all files in a commit are committed together or not at all. The second is to commit files with a single purpose - not as a backup. All commits should have a single purpose, like fixing a bug or adding a new feature. The third best practice is to write good commit messages. Each commit should make it easier for the others to understand the purpose of the review. The fourth is don't break builds. It's important to do tests so that commits don't break into other's work. The fifth best practice is reviewing before committing to a shared repository. Having an extra set of eyes will help prevent problems. The sixth is to make sure every commit is traceable. Keeping track of changes allows for backing out if necessary. The seventh best practice is to follow branching best practices. These best practices include keeping things simple, having well-defined code branching policies, giving code lines an owner, using branches for releases or milestones, protecting your mainline, and merging up and copying down. The final best practice of this article is to protect your assets. This means to implement security measures to protect intellectual property.

The final resource I found is published by Gitlab: https://about.gitlab.com/topics/version-control/version-control-best-practices/. This article talks specifically about Git but also has guidelines that can be applied to other version control systems. The first best practice is to make incremental, small changes. This means dividing the work into small batches that can be solved in small amounts of code to decrease the chance of integration conflicts. The second is to keep commits atomic. As mentioned earlier, this means commits should involve one task or fix. The third best practice is to develop using branches. As mentioned earlier, branching separates work in progress from stable, tested code. The fourth is to write descriptive commit messages. The article suggests starting the commit with a verb in the present tense to indicate the purpose of the commit. The fifth best practice is to obtain feedback through code reviews. Feedback helps ensure code quality. The sixth best practice is to identify a branching strategy. The article lists four common branching strategies: centralized workflow, feature branching, GitFlow, and personal branching.

When reviewing these articles, I did not identify any guidelines that are not relevant today. I saw a lot of overlap between the articles. If I were to put together a list of the most important guidelines, I would implement a central repository, have good branching strategies, set standards for commit messages, have code reviews, and secure your intellectual property. I chose these because they were standards I repeatedly saw within the articles I reviewed.