Brett Fuller

CSD-380 - Assignment 3.2

6/15/2025

Version control is the process of managing source code changes during software development. Version Control would involve using tools to track and log changes collaboratively. Version Control is a complicated subject with many philosophies of how-to best approach maintaining versions of your code. To better understand the various philosophies, I chose to look at articles from the University of Washington, Modern Requirements, and the Software Engineering Guy with an objective of looking to find a solid set of guidelines that can be easily followed to successfully manage Version Control.

Michael Ernst of The University of Washington has amassed a good collection of guidelines which he began curating back in 2012 and continues to maintain and update as recently as April of 2024. He has chosen to limit his guidance to more basic concepts, avoiding complicated topics such as branching and the different strategies for maintaining code in large projects or small ones. Some of the best practices that Michale mentions are to Ensure that you use descriptive Commit messages. Leaving a good message makes it easier for yourself and others to understand what was done with each commit. He also recommends making sure that each commit to the main branch only serves a single purpose. He suggests that separating bug fixes, feature updates, and typo fixes into their own commits makes it easier to track those separate kinds of tasks and keeping commits to a single purpose makes it easier to back out changes if necessary. In this same spirit he recommends making sure you do not commit more changes than you mean to. He also suggests that you should regularly bring in changes from other users by making sure you regularly refresh your local copy or clone and that you likewise frequently commit your changes back to the repository allowing your coworkers to utilize the changes you have made. He also recommends regularly checking in with your coworkers to understand what everyone is working on so that people are not stepping on each other's toes so to speak. Finally, he gives some general tool-based guidance on how to use the tools and avoiding checking in generated files that could cause conflicts if grabbed by other users.

Arunabh Satpathy of Modern Requirements begins discussing their recommendations by talking about using version numbers to have a better understanding of where you are at with your software and make it easier to communicate what version of the code you are talking about in any discussion. They recommend using a Major.Minor.Patch numbering system where the numbers before the first dot describe the major version of the software you are on or often those changes that may break api’s, the numbers after that dot and before the second dot describe the minor changes to the software or those that don’t break the api’s, and the final numbers following the second dot indicate the patch version. They then touch on how a team could implement branching for complex software stacks or even for complex organizations with many teams working on the software, though I found their actual guidance on this subject to be little more than a mention of its importance and would have liked to see a little more detail here. Arunabh then expresses similar guidance to Michael on leaving good messages in your commits or tags. They also recommend maintaining high quality requirements documentation. Finally, they give some guidance around securing your version control solution ranging from ensuring that you have high quality backups and failover capabilities, to maintaining your Access lists by defining who should have what level of access to which resources and finally mentioning the need to keep track of the activity of people using the repository to identify when something seems off.

Finally, we have the Software Engineering Guy from his medium blog, his post focuses on GIT and not more broadly on various Version Control methods. He begins by discussing branching strategies, by first giving a brief description of what branching is and why you might want to use it when he writes “Branches: Ever wanted to try out a crazy new idea without messing up your main codebase? That’s what branches are for! They let you create parallel universes of your project where you can experiment freely.” He then goes on to mention three different strategies you can use with Git to manage the various code branches. The first is GIt Flow which is where you have several different branches for the various stages of your software, something like main, develop, feature, hotfix, release and so on. He finds this to be great for large projects but can be too complicated for small teams. Next, he mentions GitHub flow which he describes as a “more laid-back cousin of Git Flow” Here you still have your main branch which crucially is always deployable, and then you have several feature branches that come directly off the main branch instead of branching off each other to make versioning more manageable. He says this is better for teams that ship changes frequently and want to avoid the complexity of a fully blown Git Flow strategy. The final branching Strategy he discusses is Trunk Based Development where all changes go directly into the main branch, which he acknowledges is riskier and requires more discipline from the team, but he likes the simplicity of just managing a single branch. Next, he focuses on Commit messaging with similar guidance to Michael from UW. With some additional guidance around what makes a good comment from advising on using the Imperative voice when writing comments to limiting comments to 50 characters and referencing Issue numbers. He then gives similar guidance around Pull requests from making sure you limit the scope of your requests, to providing thorough explanations of what you are doing and why, linking to issues, and requesting peer reviews from the right people. His final piece of advice that I found helpful was focussed around learning the tool you are using in his case Git and making sure to take advantage of advanced features and leverage automation where you can.

Out of the three sources I chose to reference they all had a lot of similar information and while I really liked the first and third articles I found the second one to be a little light on detail, and while I was initially drawn to the site because of the high quality web design and the use of helpful pictures I found the actual content of the article to be a little lacking. If I were to build guidance around Version Control I would take the UW article as a basic framework because I thought all of their suggestions to be really good for establishing a baseline. From there I would flesh out the sections on commit messages by including the guidance on messaging both in commits and pulls from The Software Engineering Guy. I would also include his guidance around branching, and getting to know the Version Control Software you are using to be able to take advantage of the advanced features along with leveraging automation to increase productivity and consistency. Finally I would include the security guidelines from Arunabh Satpathy of Modern Requirements. With this framework I think you would find yourself with a very nice outline of guidelines around Version Control.

**References**

Ernst, M. (2024, April 8). *Version control concepts and best practices*. Version control concepts and best practices. Retrieved June 15, 2025, from https://homes.cs.washington.edu/~mernst/advice/version-control.html

Satpathy, A. (2023, August 21). *Version Control Best Practices for Efficient Software Development*. Modern Requirements. Retrieved June 15, 2025, from https://www.modernrequirements.com/blogs/version-control-best-practices/

Software Engineering Guy!! (2025, January 1). *Version Control Best Practices for Modern Software Development*. Medium. Retrieved June 15, 2025, from https://medium.com/@paulosejithu/version-control-best-practices-for-modern-software-development-67bdfeddd74a