Version Control Systems

* **Christina: Introduction, Thesis Statement, 1ST Generation VCS**

Hi everyone, My name is Christina. Today, I’m joined with my teammates Michael, Travis, and Sarah to talk about the evolution of version control systems. Since our project is to develop a web-based code review tool integrated with Git which is a distributed version control system, we had a lot of practice using VCS and understanding how they work in order to keep track of our progress as a team.

 For those who don’t know what VCS is, it’s essentially a software tool that many programmers use to manage and track their source code. VCS stores different versions of code and is especially helpful when working on projects in a large group. With code being constantly updated from one person to another, VCS aims to track these changes and manage different versions of source code, files, and documents that multiple users work on at once. Funny enough, as our team was working on our project, we were sending files back and forth through our Microsoft teams chat which was really confusing and difficult to keep track of who was sending what, further demonstrating the need and importance of version control.   |

The first-generation VCS provided the foundation for many modern VCS tools that many developers use today. Unlike modern VCS, The first generation VCS were intended for users to track changes for individual files and could only be edited locally by one user at a time. They were built so that all users would log into the same shared Unix host with their own accounts.

SCCS (Source Code Control System) is considered to be one of the first VCS tools created. SCCS consists of two parts: SCCS commands and SCCS files. Some common operations SCCS has are the ability to track file history, check out specific file revisions for editing, reviewing, commenting or compilation, reverting changes, and basic branching and merging of changes. These operations can be done through various basic commands, which is common among many modern VCS. Here is a list of some of the most common SCCS commands.

On the other hand, a SCCS file is a special type of file used in SCCS also called an s-file or a history file. It is created when a file is added for tracking with SCCS. The files have a unique format prefix s., which is controlled by SCCS commands. The purpose of this file type is so that when created, the history file will contain the initial content of the original file as well as some metadata to assist with version tracking. So for instance, a file called test.txt would get a history file created in the ./SCCS/ directory with the name s.test.txt

In an SCCS file, there are 3 components: A delta table, access, and tracking flags, and a body of the text.  instead of creating a separate file for each version of a file, the SCCS file system only stores the changes for each version of a file. These changes are referred to as deltas which are stored in a delta table, where each delta represents a single revision in a file. Each entry in the delta table contains information about who created the delta, when they created it, and why they created it. Control and tracking flags in SCCS files are somewhat self-explanatory. These flags are used to track the various access and tracking options of every SCCS file. Some of the SCCS flag operations include: designating users who may edit the files and locking certain releases of a file from editing. Finally, the SCCS file body contains the text for all the different versions of the file. In other words, portions of the SCCS file body, otherwise known as the control characters indicate the portions of text that correspond to each delta. Altogether, SCCS commands and SCCS files work hand in hand to create a successful working version control system and have set the path for the following generations of VCS tools widely used today. Now I’m going to hand it off to Sarah to talk about how VCS has evolved since the first generation was released.

* **Sarah: 2ndgeneration VCS , lead into 3rd generation**

………

Thank you, Christina. We just learned about all the amazing functionalities of the 1st generation of VCS. But even with all those features, the 1st generation was still less permissive since it was not possible for geographically dispersed users to work on a given file at time.

This is the main issue that the second generation was made to solve and for the first time in version control history, the Concurrent Versions Systems allowed multiple developers to check out and work on the same files simultaneously.

This is essentially achieved by setting up a centralized repository model on a remote server so that projects can be imported in that repository. When a project is imported in CVS, each file is converted into a ,v history file and stored in a central directory known as a module.

This is the season that gave rise to network in version control systems history. The repository generally lives on a remote server, which is only accessible over a local network or the internet.

Unlike the 1st generation VC, with CVS no files are locked in the process of checking out the module, which means that there is no limit to the number of developers that can checkout/copy the module at one time. Developers can even modify checked out files and commit their changes as needed. And if a developer commits a change, others will just need to update their working copies by merging before making their commits. Obviously, merge conflicts may occur occasionally, but they can be easily resolved before the commit can be made. Like modern VCS, CVS also provides the ability to create and merge branches.

These are the basic commands used to set up the repository, import projects into the repo, copy s module, commit changes, and update the module.

However, even with all these features CVS was not robust enough and had be succeeded by Subversion. Both CVS and SVN have common features such as the centralized repository model and remote users must have a working network connection to commit changes to the central repository.

SVN introduced the functionality of atomic commits that can tell if a commit will either succeed or be completely abandoned if it fails. In CVS, if a commit operation fails midway, the repository can be left in a corrupted and inconsistent state.

Additionally, SVN introduced the features of committing multiples files and directories, which is really important since it allows developers to trach sets of related changes together instead of tracking changes separately for each file.

Also, empty folders can be committed in SVN. This is not achievable in other VCS, even in the modern ones where empty folders are unnoticed.

SVN does not use a conventional branching and tagging system. A normal SVN repository layout has 3 folders in the root:

The trunk folder used for production version of the application. The branches/ folder is used to store subfolders that correspond to individual branches. The tags/ folder is used to store tags which represent specific and significant project revisions.

These are the basic commands used to create an empty repository, import directories of files into the repo, copy a repository path to the desired working directory, commit changed files, and update the repository. Now I would like to hand over to Travis who will take you through the evolution of VCS since the second generation.

* **Travis: 3rd generation VCS, Conclusion and repeat of thesis**

……….

* **Michael: 4th paragraph demo presentation**

DEMO DEMO DEMO DEMO DEMO DEMO DEMO