Chapter 6

Related Work

6.1 Other Applications of the Theory of Conceptual Design

In [4], the authors used Conceptual Design Theory and applied it to version control systems. The authors studied Git to understand why it seems to fall short of users' expectations. The authors analyzed those issues by using Conceptual Design to explain Git's design issues as operational misfits of its underlying concepts. They then fix those operational misfits by constructing a new conceptual model and system, called *Gitless*, built on top of Git.

In [8], the author performed another conceptual analysis case study, this time focusing on Dropbox. The author first researched and polled users for the areas of Dropbox they find most confusing. They then used Conceptual Design Theory to find the operational misfits that caused this confusion. The result was a remade conceptual model of Dropbox, one that was cleaner and easier to understand.

In both of these case studies, the theory of conceptual design is applied to a specific system for analysis. The design of Snapstore, on the other hand, involved using the theory of conceptual design to design a new version control system from scratch. We did, however, use the purposes of version control enumerated in [4] to guide the design of Snapstore.

6.2 Version Control and File Syncing

We also studied the file syncing and version control system spaces as case studies. Systems such as Dropbox, Google Drive, Git, Mercurial and more were studied from a user's perspective to see how they accomplished various tasks that Snapstore would cover.

6.2.1 File Syncing Tools

Continuous saving is supported in Google Drive and Dropbox, but they do not allow a user to group changes or record coherent points. These users can leave comments on changes but cannot do so across multiple files. Snapstore accomplishes this with the group and tag concept.

Merging is also not well supported by many file syncing tools. Google Drive preserves every edit made on a shared document through a process called operational transformation [7]. When a conflict arises, a newline is inserted. This is fine for some text documents, but it can break code documents. Dropbox does not allow files to be merged at all. With these systems, any merging must be done manually. Snapstore simplifies the shared document by keeping the line of development perfectly linear, so as not to adversely affect white space sensitive documents. It also allows file merging between branches to give it the power of a version control system.

The file concept is prohibitive in many file syncing tools because the file name is a unique identifier. In Dropbox, for example, renaming a file offline and reconnecting to the network will upload an entirely new file with a new history. Files in snapstore are identified with a unique id, not a file name. Because of this, Snapstore can track renames and maintain file history more robustly.

File syncing systems have limited offline support. Google Drive allows minimal offline capabilities on Chromebooks, and Dropbox does not monitor offline changes. Snapstore uses the local repository to track offline edits exactly the same way it tracks online edits. This allows Snapstore to push those edits once a network connection is restored.

6.2.2 Version Control Systems

Version control systems tend to couple together the motivating purposes of grouping changes together and recording coherent points. Git, for instance, combines the commit with the act of labeling the commit. Snapstore decouples these purposes by allowing users the granularity of a single snapshot and by allowing them to later group snapshots and tag those groups.

Creating branches and copies of files varies greatly between version control systems and file syncing systems. Git allows a user to create a branch, a fork, or a clone. This array of concepts is simplified to just the branch in Snapstore.

Similar to file syncing systems, the file name in a version control system is often a unique identifier. In Git, if a user renames a file without using the “git” command, Git sometimes recognizes that as deleting a file and creating a new one. It is similarly handled in Gitless. This limits the history of commits and history of the file. Snapstore achieves a seamless history by using the file id as an identifier. It is able to log renames to files as rename snapshots and keep the snapshot history consistent.

Finally, version control systems typically cannot save or pull in changes without a direct command from the user. Git and Gitless require the explicit “commit” command to do so. The push/pull model of Git is unnecessary. Snapstore automatically pushes out and pulls in changes to a branch using the upstream repository.