**AGILE:**

Theme: Get GiggleGit demo into a stable enough alpha to start onboarding some adventurous clients

Epic: Onboarding experience

User stories are general explanations of software written in the POV of the end user.

A version control system is basically what GitHub is.

1. Complete these user stories:
   * As a vanilla git power-user that has never seen GiggleGit before, I want to feel comfortable being able to interact with GiggleGit with logical and intuitive command syntax that isn’t confusing so that I may learn to use it quickly.
   * As a team lead onboarding and experienced GiggleGit user, I want to easily have access to my previous versions and branches of changes that I may better manage them and track their progress.
2. Create a third user story (1-3 sentences), one task for this user story (single phrase), and two associated tickets.
   * As a student working on projects and new to GiggleGit, I want to feel comfortable being able to give and take feedback on my code with other like-minded individuals so that I may better improve my work and learn from others.
     + Task: Be able to collaborate on public projects
       1. Ticket #1: Implement the ability to comment on people’s code
          - They should be able to highlight and comment on the code
       2. Ticket #2: Design the ability to see a dashboard of people’s comments
          - See a range of comments based on the lines of code for the project

3. This is not a user story. Why not? What is it?

* As a user I want to be able to authenticate on a new machine

The reason that this isn’t a user story is because it does not specify what kind of user it is and it does not describe what the benefit and goal of the action is (in this case authenticate on a new machine).

**FORMAL REQUIREMENTS**:

Using SnickerSync

Goal: Create an interface that allows to merge and sync different branches using SnickerSync ensuring proper and seamless integration and syncing with base GiggleGit packages.

Non-Goal: Redesign GiggleGit packages to better facilitate syncing and merging using SnickerSync.

Non-functional requirement #1: **Accessibility**

Functional Requirement #1: Make sure that each employee has their authentication information to access the components of the project.

Functional Requirement #2: Create role-based access for projects which are dependent on the specific employee or rank of employee and that require permission from higher ranks.

Non-functional requirement #2: **Maintainability**

Functional Requirement #1: Make sure that the work/code for SnickerSync is well partitioned and modulated so that it reduces coupling and dependencies.

Functional Requirement #2: The code must be as readable as possible to better adapt and maintain in the future.