

# CMPS 3390 Homework 3

*Spring 2026*

## Version Control

### 1. Git Resources

- **[GIT](#)**: The industry standard for tracking changes in code over time and managing project history. Git was developed by Linus Torvalds and is widely used for its flexibility, branching capabilities, and ability to handle large projects efficiently.
- **[Github](#)**: GitHub is a cloud-based platform where you can store, share, and work together with others to write code. Storing your code in a "repository" on GitHub allows you to:
  - Showcase or share your work.
  - Track and manage changes to your code over time.
  - Let others review your code, and make suggestions to improve it.
  - Collaborate on a shared project, without worrying that your changes will impact the work of your collaborators before you're ready to integrate them.
- **[Github Gists](#)**: Think of a gist like a tiny little single file repo. A gist is a simple way to share code snippets, text, or other information via GitHub. Gists also support versioning, so you can see how your gist changes over time.
- **[Bitbucket](#)**: Bitbucket is another cloud-based GIT platform owned by Atlassian. While Bitbucket is free for personal use, Bitbucket is primarily marketed to larger software development companies and teams. This is due to the fact that Atlassian also owns Jira, Trello, and Confluence which allows them to integrate version control directly into these project management tools.

### 2. Version Control Key Concepts

- **[Initialize](#)**: In order to begin tracking/committing the changes to your project, you must first initialize a new git repo.
- **[Stage](#)**: Once you have initialized your repo, you can start adding/staging files that you would like to track. Any time changes are made to your files you must stage the changes before they will be committed.
- **[Commit](#)**: Once you have staged your changes, you can commit them to the repo. This is what actually SAVES the changes to your repo and creates a historical change record.
- **[Branch](#)**: Any time you plan on making significant changes to your project, or if you have multiple environments (like DEV, STAGING, PRODUCTION) branching allows you to create a copy of the project that you can commit to without affecting the main branch.
- **[Merge](#)**: Once you have successfully committed to a branch, you may eventually want to apply those changes to the source branch with a merge. If no changes have been made to the source branch, you can fast-forward, otherwise you may need to resolve any merge conflicts.

### 3. Working With Remote Repositories (source)

- **Clone:** If you would like to make a local copy of a remote repository (and you have proper permissions) you can use this command.
- **Remote Add/Remove:** If you have an existing local repo that you would like to associate with a specific remote repo, you would use the remote command.
- **Fetch:** If you would like to download/update the history of your repo from the remote source WITHOUT doing a pull/merge.
- **Pull:** Does the same thing as fetch, but will ALSO attempt to merge the most recent commit from the remote repo. This is good to do BEFORE you start making changes.
- **Push:** Uploads and merges your local commit history to the remote/source repository

### 4. Remote-Specific Concepts:

- **Forking:** If you do not have permission to push to a repo on github, you can fork a copy of it to your remote account instead. Then you can clone YOUR copy and make changes.
- **Pull Requests:** When you would like the original project (that you forked from) to incorporate your changes, you can request that they PULL your changes.

## Requirement Analysis

### 1. Understanding Requirement Analysis

- The process of defining what an application should do before development begins.
- Ensures that the final product meets client, user, and business needs.

### 2. Types of Requirements

- Functional Requirements: Features the system must have (e.g., user authentication, data storage).
- Non-Functional Requirements: Performance, security, usability, scalability.
- Business Requirements: How the product will benefit the clients.
- User Requirements: How end users will interact with the application.

### 3. Requirement Gathering

- Interviews: Speaking with clients and users to understand needs.
- Surveys & Questionnaires: Collecting feedback from potential users.
- Observation: Watching users interact with similar systems. Look for service gaps.
- Prototyping: Creating mockups to clarify expectations.
- Use Cases & User Stories: Describing system interactions from the user's perspective.

### 4. Analyzing and Prioritizing Requirements

- Must-have vs. Nice-to-have: Prioritizing essential features. Avoid feature creep.
- Feasibility Study: Checking technical, economic, and legal feasibility.
- Conflicting Requirements: Resolving different user/client expectations.

### 5. Analyzing and Prioritizing Requirements

- Software Requirement Specification (SRS): A formal document detailing the requirements.
- User Stories & Acceptance Criteria: Agile approach to defining requirements.

## Application Architecture

- [Layers, Types, Principles, Factors](#) (Commercial Article)
- [A Comprehensive Guide to Mobile App Architecture](#) (Personal Blog)

While these articles are primarily focussed on MOBILE application architecture, these same principles apply to general application architecture and design. You are not expected to read and retain ALL of this information, however both of these resources are fairly current. Use them as a reference during your application research, design, and develop.

- [Software Requirement Specification \(SRS\) Format](#) (GeeksforGeeks)

## Additional Reading

- [What's in your stack: The state of tech tools in 2025](#) (Noam Segal, Lenny's Newsletter)

This is a great writeup discussing current trends in software tooling and development tech stacks. Lenny Rachitsky is a former product lead at Airbnb turned writer, advisor, and investor, known for his insights on product management and startups. He shares practical advice through his popular *Lenny's Newsletter* and *Lenny's Podcast*, where he interviews top tech leaders. His work focuses on helping product managers and founders build and scale successful businesses.

- [Context Switching Kills Productivity](#) (Tech World With Milan)

This article explains how constant interruptions and context-switching kill developer focus, slow progress, and hurt code quality. TIME IS YOUR ONLY RESOURCE AS A STUDENT. Develop strategies to organize and protect it.