# Minimum Viable Product Requirements for Practicum

The SRS document contains the requirements and features for the complete version of the Git Basic software. To develop a product of this scope would take more time and resources than we have in this practicum course.

With this in mind, our team will focus on building a minimum viable product as the first release of this software. This release will be a fully functional product. However, its feature set is limited to the core features only, leaving non-essential features for future releases. This is how many software teams following agile processes operate (I am familiar with this process from work.) If we were to continue developing this software after the practicum, (which I may do as a hobby project) it would take about two more releases to finish and polish the product described in the SRS.

This document contains a copy of the requirements from the SRS. Those which are crossed out are not included in this first release. All the other requirements will be met as part of this practicum.

Note that these are just requirements to be satisfied. To better understand exactly what we will be doing as part of the practicum, the project plan should be referenced. The project plan will include coding tasks which cover both these specific requirements and tacit requirements.

***The following section was copied directly from the SRS:***

# 3. Product Requirements

## 3.1. External Interfaces

### 3.1.1. Git SCM

* 3.1.1.1. Git Basic will interface with Git repositories, as such it requires that Git SCM be installed on the user’s computer.

## 3.2. Functions

### 3.2.1. Installer

* ~~3.2.1.1. If Git is not installed, the installer will notify the user and provide a link to~~ [~~https://git-scm.com/~~](https://git-scm.com/)

### 3.2.2. Command Line Interface

* 3.2.2.1. The integrated console (command line interface) will support the execution of all Git commands.
* 3.2.2.2. The console will support selection and copying of text.
* 3.2.2.3. The console will support pasting text input into the input region.
* ~~3.2.2.4. The console will output color coded text for easy interpretation.~~
* 3.2.2.5. Error output in the console window will be displayed in red text.
* 3.2.2.6. The console will display the current working directory by the text input region.
* ~~3.2.2.7. The font size in the console may be adjustable.~~
* 3.2.2.8. Double click in the console should select the word underneath the cursor.
* ~~3.2.2.9. Triple click in the console should select an entire line.~~
* ~~3.2.2.10. Right click on the console should display a context menu with the following options: Copy, Paste, Select All~~
* 3.2.2.11. The console window will be resizable.
* 3.2.2.12. Pressing tab in the console input region, should toggle through the working directories contents (subdirectories and files).
* 3.2.2.13. Pressing the up/down arrow keys in the console input region, should cycle through command history.
* ~~3.2.2.14. Pressing Ctrl + Space in the console input region, should open auto-complete for branch names.~~

### 3.2.3. Review and Diffs

* 3.2.3.1. The diff viewer will support ~~unified and~~ dual viewer comparisons.
* 3.2.3.2. The diff viewer will only compare text based files.
* 3.2.3.3. The diff viewer is resizable.
* 3.2.3.4. The text in the diff viewer is selectable and can be copied.
* 3.2.3.5. Additions are highlighted in green in the diff viewer.
* 3.2.3.6. Removals are highlighted red in the diff viewer.
* ~~3.2.3.7. Line numbers should be displayed in the diff viewer.~~
* 3.2.3.8. The name of the file being diffed should be displayed in the diff viewer.
* 3.2.3.9. The file to be displayed in the diff viewer should be selectable from the file status control.
* ~~3.2.3.10. The diff viewer will be collapsible.~~
* 3.2.3.11. The file status control will display all modified files.
* 3.2.3.12. The file status control will display which files are staged and which files are unstaged.
* 3.2.3.13. The files status control will allow the user to select a file and stage or unstage it.
* 3.2.3.14. The file status window will support a show in Windows Explorer option for the selected file.
* 3.2.3.15. The file status window is resizable.

### 3.2.4. Productivity Assitance

* 3.2.4.1. Ten buttons will be mapped to common Git commands.
* 3.2.4.2. Clicking on the buttons will execute their corresponding commands in the console window.
* 3.2.4.3. The buttons will be mapped to default commands~~, but the user will have the ability to change these commands.~~
* 3.2.4.4. The buttons will have the hotkeys Ctrl + 0-9 mapped to them so that the user can invoke the commands without clicking on the buttons. The first button will be 1 and the last will be 0, so as to spatially map the keyboard to the layout.
* 3.2.4.5. The status bar should display the current repository.
* 3.2.4.6. Switching to a different repository should be possible by clicking on the repository name in the status bar.
* 3.2.4.7. The status bar should display the current branch.
* 3.2.4.8. Switching to a different branch should be possible by clicking on the branch name in the status bar.
* ~~3.2.4.9. When typing “Git branch” into the console control an autocomplete box should be displayed which allows the user to complete a branch name without fully typing it.~~
* 3.2.4.10. The working directory of the application should be persisted between sessions.

## 3.3. System Requirements

### 3.3.1. System

* 3.3.1.1. The software should function on Windows 7, 8, and 10.

### 3.3.2. Resources

Since Git Basic simply acts as a client on top of Git SCM, it should not be resource demanding.

* 3.3.2.1. Excluding Git’s time to execute a command, no Git Basic specific action should take more than 1 second to perform.
* 3.3.2.2. The application should use no more than 250 MB of memory at any given time.