Git Basic

Software Design Document

Prepared by Matthew Ristine

January 27, 2018

COMP 495

Contents

[1. Introduction 3](#_Toc504831922)

[1.1. Note to the Professor 3](#_Toc504831923)

[1.2. Overview 3](#_Toc504831924)

[1.3. Resources 3](#_Toc504831925)

[1.3.1. Git 3](#_Toc504831926)

[1.3.2. .NET 3](#_Toc504831927)

[1.3.3. MVVM 3](#_Toc504831928)

[2. Design Considerations 3](#_Toc504831929)

[2.1. Assumptions 3](#_Toc504831930)

[2.2. System Environment 4](#_Toc504831931)

[2.3. Design Methodology 4](#_Toc504831932)

[2.4. Risks and Volatile Areas 4](#_Toc504831933)

[2.5. Third Party 4](#_Toc504831934)

[3. Architecture 4](#_Toc504831935)

[3.1. Overview 4](#_Toc504831936)

[3.2. Component Details 5](#_Toc504831937)

[3.2.1. View and View-Model 5](#_Toc504831938)

[3.2.2. Model 5](#_Toc504831939)

[4. User Interface Design 6](#_Toc504831940)

[4.1. Application Layout 6](#_Toc504831941)

[4.2. Screen Components 7](#_Toc504831942)

[4.2.1. Diff Window 7](#_Toc504831943)

[4.2.2. File Status Control 8](#_Toc504831944)

[4.2.3. Console 8](#_Toc504831945)

[4.2.4. Command Buttons 9](#_Toc504831946)

[4.2.5. Status Bar 9](#_Toc504831947)

# 1. Introduction

## 1.1. Note to the Professor

This Design Document was adapted from my COMP 325 project. As explained before, I started on the concepts for this software in that class. I have modified and refined this document to better reflect the software application we are going to build.

## 1.2. Overview

This document presents a broad description of the software design for Git Basic. It is meant to be a top-level view of design. It is not intended to provide an in-depth description of any particular component. Additional design artifacts may be produced as necessary for individual components.

[Section 1](#_1._Introduction) – (this section) introduction.

[Section 2](#_2._Design_Considerations) – considerations which must be kept in mind for the architecture and design of the project.

[Section 3](#_3._Architecture) – overview of the software architecture.

[Section 4](#_4._User_Interface) – graphical user interface design.

## 1.3. Resources

### 1.3.1. Git

To download Git SCM and for more information about Git, go to: <https://git-scm.com/>

Refer to <https://git-scm.com/docs/git> for Git documentation.

For a 15 minute interactive Git tutorial go to: <https://try.github.io/levels/1/challenges/1>

The following Git tutorial from Tutorialspoint may be useful for learning Git: <https://www.tutorialspoint.com/git/index.htm>

### 1.3.2. .NET

C# tutorial: <https://www.tutorialspoint.com/csharp/index.htm>

WPF tutorials:

* <http://www.wpf-tutorial.com/about-wpf/what-is-wpf/>
* <https://www.tutorialspoint.com/wpf/index.htm>
* <https://www.wpftutorial.net/>

### 1.3.3. MVVM

An MVVM tutorial to learn the design pattern: <https://www.tutorialspoint.com/mvvm/index.htm>

# 2. Design Considerations

## 2.1. Assumptions

Network considerations - Git is a distributed source control system. However, Git can be used with a local repository and no network connection. With this in mind, the software should be designed in such a way so as to operate without a network connection. However, lack of a network connection will, of course, preclude interaction with remote repositories.

## 2.2. System Environment

The product will run on the following operating systems:

* Windows 7 (32 and 64 bit)
* Windows 8 (32 and 64 bit)
* Windows 10 (32 and 64 bit)

The client presupposes an installation of Git SCM.

## 2.3. Design Methodology

The application is to be developed in C# as a WPF desktop application. Object-oriented design and programming is to be applied. Domain models and architecture will be designed primarily in UML.

## 2.4. Risks and Volatile Areas

Integrating a console control into a WPF application could present some challenges. As such it might be worthwhile investigating some third-party options. I have done some initial research on third-party controls without success. However, I currently have a working (although basic) solution for the console.

## 2.5. Third Party

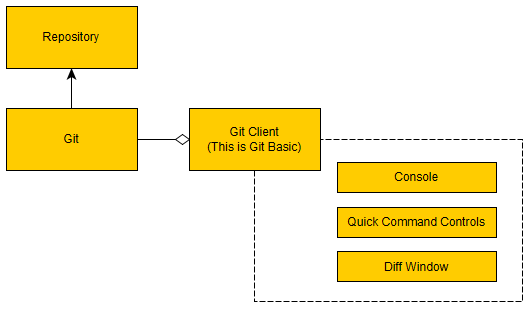
The following third-party libraries should be considered. We will most likely be using Git and diffing features from one or more of these libraries.

* GitSharp – <http://www.eqqon.com/index.php/GitSharp>
* DiffPlex – <https://github.com/mmanela/diffplex/>

# 3. Architecture

## 3.1. Overview

The primary architectural components of the system are as follows.



All version control operations are handled by Git. Git “operates” on the files in a repository. The purpose of Git Basicis to act as a frontend on top of Git. It simply facilitates the calls to Git and provides an interface to make particular operations more user-friendly.

Git Basic is broken down into three primary components.

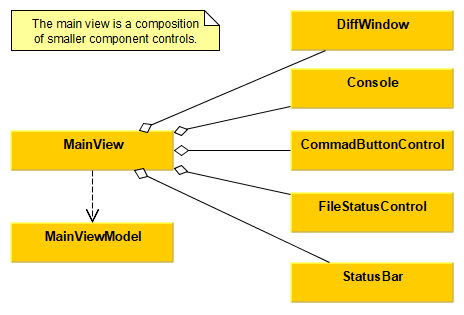
* **Console:** All Git commands are executed from here. Feedback is displayed in the console.
* **Commands:** These are the command controls which Git Basic provides to facilitate the execution of Git commands.
* **Diff Window:** The view windows for file diffs and the file staging controls.

## 3.2. Component Details

### 3.2.1. View and View-Model

The application will be built using the MVVM (Model View View-Model) design pattern. Under this pattern, controls in the view are data-bound to properties in a view-model.

The view and view-model architecture is as follows.

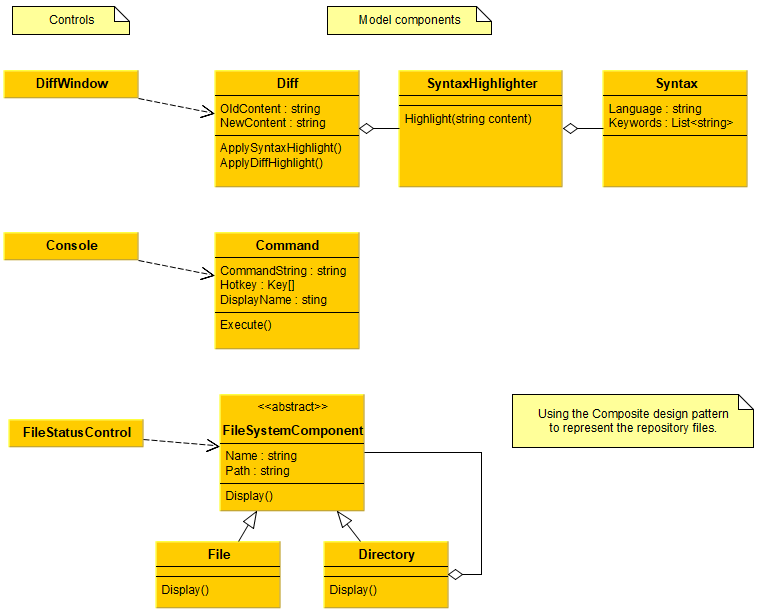


As seen above the main window is a composition of smaller components. This allows the application to be assembled from small reusable controls.

Properties on the individual controls will be data bound to properties in the view-model (specific properties not shown in the diagram). Where necessary sub view-models may be created for the individual component controls.

### 3.2.2. Model

Models representing the primary objects of the system are to be created. Some example models are as follows. Note that the models presented here will most likely change significantly when the individual components are designed. These are meant to serve more as a starting point for component design.



Controls interface with model components as seen in the above diagram.

# 4. User Interface Design

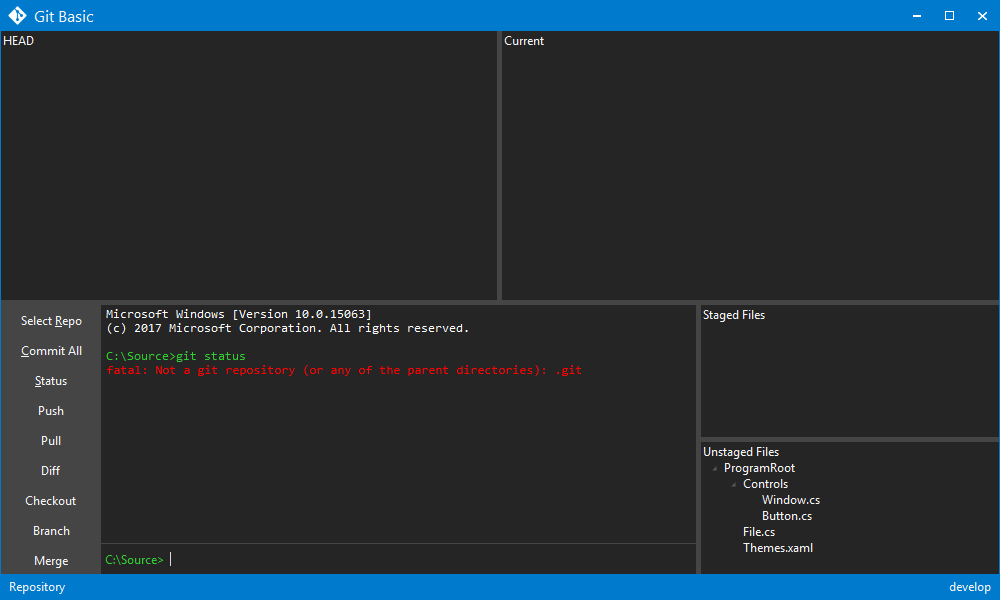
## 4.1. Application Layout

The primary theme of the application will be a dark theme. (A light theme may also be provided.)

The diff window should be positioned at the top of the application. This position is ideal for reading the text.

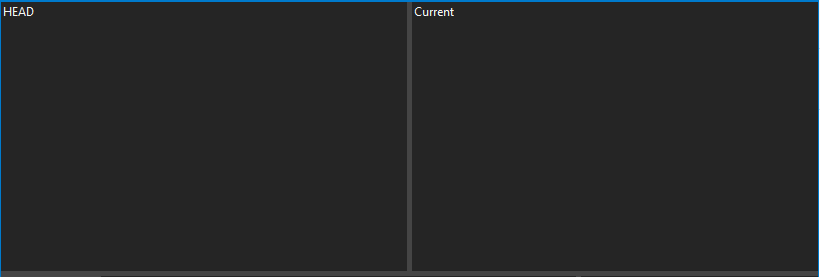
The console control should be positioned at the bottom of the application. The command buttons should be adjacent to the console as they are used in conjunction with the console.

Here is a mockup demonstrating a possible layout for the application.



## 4.2. Screen Components

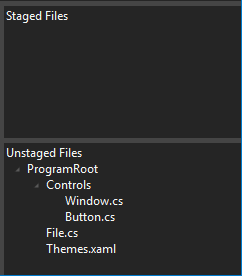
### 4.2.1. Diff Window



The side-by-side diff panel allows for easy comparison of changes.This diff control should also provide the ability to view unified diffs.

To view a diff, the user selects the file from the file status control.

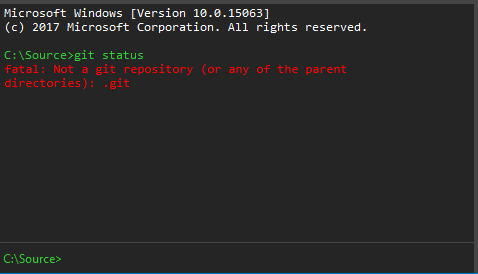
### 4.2.2. File Status Control



This control allows the user to stage or unstage files. The user should be able to drag and drop individual files or entire directories.

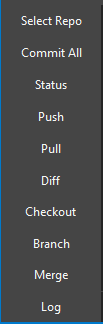
Also, selecting a file in this view displays it in the diff control.

### 4.2.3. Console



This is the primary input control of the application. This is where the user enters Git commands. This console should be a fully functional console such as Window’s cmd.exe.

### 4.2.4. Command Buttons



These buttons should be in proximity to the console. They input user-specified git commands into the console. The user sets up the button names, command mappings, and hotkeys from an options menu.

### 4.2.5. Status Bar



The user should be able to select the current repository (directory selector) and the current branch from the status bar.