Distributed Project Teams and Software Development

An Introduction to the use of Git and GitHub for ASP.NET MVC Development

Thom Luce
luce@ohio.edu
Analytics and Information Systems Department
Ohio University
Athens, OH 45701

Abstract

This paper describes changes, precipitated by the Covid-19 pandemic, to a capstone MIS class using Microsoft ASP.NET MVC for team development with live-clients. The advent of the pandemic required that the entire development effort of the class immediately transition from a largely in-class development effort with local SQL Server and Web Server Instances to one requiring all development be done in a virtual desktop interface (VDI). The VDI was the only way for students to get to both the SQL Server instance and the web server where they published their applications. Code availability, version control and joint development issues were resolved with Git and the Visual Studio interface to GitHub. This paper summarizes the development issues faced by the student teams, how they were resolved and provides a brief introduction to the use of GitHub from within the current version of Visual Studio. The paper is descriptive, and the subjective nature of the live-client project deliverables made any significant statistical analysis impossible.

Keywords: Version control, Source code management, Git, GitHub, Capstone course, Pandemic

1. INTRODUCTION

A few years ago, there was an active discussion on the Computer Science Educator's Stack Exchange (Anon, n.d.) about why more universities didn't teach revision control. While this discussion focused on computer science and engineering programs, many of the considerations also apply to IS and MIS programs.

The (Anon n.d.) article says that one reason may be that it is hard to teach some of the concepts if students don't have an opportunity to practice the concepts and that normally requires team projects and longer projects than are typically assigned. They also say that students typically don't see a reason for using source code management (version control) until they get into more complex problems than those they typically get in the classroom. The course described in this paper fits those criteria with a three-part individual project where each part builds on the previous part and a major team project involving the development of an application for an external client.

Uzunbayir (2018) points out that source code management is especially appropriate for continuous software development projects which are based on agile practices. The course and project described here and further documented in (Luce 2017, 2020) use Scrum with two-week Sprints and a published, potentially deliverable product at the end of each Sprint.

According to Andersson (2018) and Marko (2019) good version control should include, among other things

1. Making and committing small changes
2. Committing frequently
3. Not committing generated code
4. Only committing verified, test code

5. Making good commit comments.

The pandemic induced version of the class, using GitHub as described here forced students to make and commit small changes frequently because they have to commit every time they left the VDI.

The use of. gitignore allows the students to control which files are committed each time. The one exception is the .ddl file created when the application is built, and this can be handled by essentially ignoring it and always keeping the local version.

Point number 4 is harder to police but students in general understand the idea of garbage-in, garbage-out and soon learn that committing untested and unvalidated code leads to the propagation of errors among their team members.

The last point becomes clearer when students try to find and reverse changes that they or a teammate committed to the repository.

2. PROBLEM DEFINITION

Eighty-two students, most of them graduating seniors and all taking a capstone live-client software development project, left campus for spring break on March 7, 2020. They never returned to campus. While they were on break the Covid-19 pandemic struck with full force causing the University to shut down all face-to-face classes, close campuses and move all classes online.

The students, working in teams of 4-6, were four weeks into an eight-week live software development effort, developing an application for a remote client using Microsoft ASP.NET MVC tools (Note: We are planning to more to MVC Core during the next academic year) and managing their projects with Scrum. Prior to the campus closure all students had access to computer labs with all the development software they required, the college network with a full SQL Server instance and a shared web server where they could publish their applications.

Additionally, the students had access to a virtual desktop interface (VDI), implemented with VMWare Horizon (VMWare Horizon 7. (n.d.)). The VDI gave them access to the college network and servers, including the SQL Server instance and a web server from anywhere in the world and allowed them to work on projects when they weren't working on a lab computer. Unfortunately, the VDI installation at the time was configured to present a clean environment every time a student logged in. This meant students were unable to store files on the virtual desktop or anywhere on the virtual machine. They could pull files into the VDI from their local computer but that didn't help when they were working remotely. There were two UNC (universal naming convention) networked drives available from the labs and in the VDI environment where students could store files. Unfortunately, students could not work directly on applications from the network drives because a number of important ASP.NET MVC development operations, such as data migration and publishing, didn't work properly when the source code was stored on a networked drive.

Access to the VDI and the networked drives was password protected and limited to registered students, so sharing files with team members was difficult. MVC applications tend to be too large to email. For example, the compressed version of the sample application developed for this paper started at over 75Mbytes. As a result, students were unable to share work via email. Students were also unable to share network resources unless they were also willing to share passwords and, since these passwords were used in numerous University systems, most students weren't willing to do that.

Students learned how to develop ASP webforms applications in earlier classes and we had established a series of shared folders on networked drives that would allow students to save applications and to share work. Unfortunately, work on MVC applications is more complex than work on webforms applications and shared folders didn't provide a good solution. As mentioned previously, a number of important development steps don't work correctly in the Visual Studio ASP.NET MVC environment if the source code is stored on a network drive. Because of this, the entire MVC application had to be copied back and forth between the network drive and a local drive to be used. Then there was the most important consideration of all, because of all the interrelated files in the MVC environment, only one student at a time could safely work on application development. Students could not simply work on different parts of the application and then copy the individual files to one central location. Additionally, copying the application folder back to the shared drive did not enforce any kind of version control and the newly revised work simply replaced the existing files on the drive. Size limitations on the networked drives made simple version control via

file and folder renaming impractical if not impossible.

While we had tried different approaches to version control software in the past, trying systems like Microsoft Team Foundation Server, now known as Azure DevOps Server ("Microsoft Team Foundation Server," n.d.) and git ("Git," n.d.), but our students preferred to work in our labs, finding the version control systems either too complex or too foreign to what they knew. One reason for this is that our program has always focused on solving business problems using technology, rather than learning technology for its own sake. We limit the number of software platforms and development environments students are exposed to and focus on what needed to be done by any set of tools. We had, in fact, standardized our MIS assignments around Microsoft Visual Studio, C# and SQL Server a number of years ago for precisely that reason. Because of this approach, most of our students have never seen a command line process like that used by git or the Package Manager in Visual Studio. While it could be argued that learning to use a command line tool would be good for their education, our program wanted to focus is on solving business problems, not learning technologies or new, or old, interfaces.

So, despite our attempts to introduce version control software the biggest criticism of the course, semester after semester, was the inability to have more than one person working on the application at a time.

Things started to change in the fall of 2019 when we realized that Visual Studio (VS) had improved support for git and GitHub within the Visual Studio environment (Nadagouda, 2020). We introduced the Visual Studio interface to GitHub at the beginning of a three-part individual software development assignment designed to help prepare students for the team development effort that would follow. We attempted to sell it on the idea that they would now be able to work from anywhere, would eventually be able to do co-development, wouldn't be limited to working in our computer labs and wouldn't need to copy large applications back and forth to servers every time they wanted to work.

As might be expected, students generally didn't see the purpose while they were working on individual assignments but started to understand how it might be useful when they began the team project. During that first semester they could, however, avoid the VDI environment and work on our lab computers where their files were typically preserved for the entire semester and, being busy students, many didn't do much project work outside class and lab times.

The pandemic outbreak in the spring of 2020 changed all of that. Students could no longer use lab computers. They could no longer meet face-to-face with their teammates. They could no longer avoid the VDI environment with its access to SQL Server, the web server and networked drives. They could no longer avoid the problem that things left on the VDI desktop or on a virtual disk wouldn't be there after they left the environment. Finally, they could no longer avoid the fact that multiple team members needed to work on the project from different locations, often at the same time.
The solution to these problems was the improved graphic interface to GitHub built into Visual Studio 2019. This paper is a short overview for instructors on how to use GitHub in Microsoft Visual Studio Community Edition 2019 (Version 16.5.x). Much of the information contained here has been converted to videos to help our students use GitHub.

3. GETTING STARTED

The latest version of Microsoft Visual Studio Community Edition generally comes with support for git and GitHub preinstalled but if they aren't installed, you need to start the Visual Studio Installer, select the current version of the software and then select modification of the installation. This should bring up a window (Figure 1, Appendix) where you can select the Workloads you wish to install or modify. For most web development in ASP.NET MVC and ASP.NET Core, the ASP.NET and web development workload should work. You can also include Python, Node.js and other workloads if you use those.

To be sure you have the components for git and GitHub, click the Individual Components link and make sure the "Git for Windows" and "GitHub Extensions for Visual Studio" are selected.

Once the extensions are installed, start Visual Studio and create a new project. At the bottom right-hand side of the window (Figure 2) you will see a message that says, "Add to Source Control" and a popup that says Git when you mouse over it and click the small white arrow on the right.

Click on Git and the Team Explorer window should open to something like Figure 3A. Visual Studio supports different source code management

systems including Azure DevOps and GitHub. Click "Publish to GitHub".

You will then be asked to login to GitHub, or create an account, and to name the new repository – EdSigCon2020 in this case (Figure 3B). Once you have done that, press Publish.

After the project is published the Team Explorer window should look something like Figure 3C.

At this point you can switch back to the Solution Explorer. As shown in Figure 4A, the Solution Explorer now has a small icon at the top that allows you to switch between Folder View (shown in 4B) and solution view (Shown in 4A). The window needs to be in Solution view for normal development work (creating Controllers, testing MVC pages, etc.). Folder view allows you to see all the files in the folder, including files that are hidden in Solution View.

Among the hidden files is .gitignore. This file lists files and folders that should be ignored, not copied, when a local repository is synchronized with the remote repository. The list includes various system files and compiled files that should not be replaced when a repository is cloned to a local computer. Many of these are bin and system package files. Unfortunately, ASP.NET MVC uses a number of binary files and package files that won't necessarily be on a local computer and that do need to be synchronized. To see that this happens we must edit .gitignore and comment out two specific lines.

The exact location of these lines varies but one says

```
**/packages/*
```

and the other is

```
[Bb]in/
```

To convert these lines to comments, type ## in front of each line and save the file.

Once the edited .gitignore file is saved the entire application should be saved (committed) to the local repository and to the distributed repository on GitHub. The process requires several steps, but the steps will be essentially the same every time changes are committed.

To commit changes, first switch to the Team Explore window and click the small house icon under the title bar. The Team Explorer windows should look something like Figure 5A. Next click Changes and enter a comment in the yellow comment box (now white because a comment has been entered). The comments are an important part of the changes you are committing, and you won't be able to actually commit the work until a comment is entered. As you can see in Figure 5B, the Team Explorer window shows you what changes you are about to commit. After entering a comment, press Commit Staged (sometimes this button will just say Commit All).

The commit process saves/commits changes to the local repository. Prior to the commit all code modifications since the last commit can be undone and haven't been added to the local repository. After committing changes, you need to synchronize the changes with the remote distributed repository. First press the Sync link shown in Figure 6A and then Push the changes to the remote depository (Figure 6B). Things may not always work quite as smoothly as this and you may need to merge changes (discussed below).

4. CLONING A PROJECT

One major benefit of distributed source code management is the ability to make an exact copy, a clone, of the project at any time. In the virtual desktop interface environment (VDI) described earlier this means a student can login to the VDI, clone a project and resume work. Once done the project must once again be committed and pushed to the server as previously described.

Figure 7 shows the Visual Studio 2019 startup screen where you can select to Clone an existing project. After selecting Clone, the Clone or check out code window opens (Figure 8, underneath). To clone from a remote repository, click GitHub. If you aren't already signed into GitHub you will have to do that first and then select the project you wish to clone (Figure 8, upper).

After the project is cloned the Team Explorer window will look something like Figure 9A. Click on the Solution Explorer and then the Folder View Icon (Figure 9B) and select the sln view before you start working on the project.

5. CHANGING AN APPLICATION

Any number of people can clone and work on a project at the same time. Each developer commits changes to their local repository and then attempts to sync them to the remote repository. Conflict occurs when one developer has successfully pushed changes to the remote repository that the current developer doesn't have. Figure 10A shows the error message that appears when this happens.

The error message referred to in the window says: "Error encountered while pushing to the remote repository: rejected Updates were rejected because the remote contains work that you do not have locally. This is usually caused by another repository pushing to the same ref. You may want to first integrate the remote changes before pushing again."

To resolve this problem, it is necessary to first Pull the remote changes and then resolve any conflicts between the local and the remote versions. Figure 10B shows the Team Explorer after Pulling the remote code. Notice that three conflicts are reported. To fix this, first click on the Conflicts link and then click on one of the files reported to be in conflict, in this case EdSigCon2020.dll.

Figure 10C shows the conflict resolution window that appears. This window has a "Merge" button, a Compare Files button and buttons that allow you keep the remote version or the local version of the code. In this case the file in question is a ddl file created when the project was built and one that will be recreated every time the project is built and run so it doesn't matter which version we use, but we will Keep Local.

The next file selected was called EdSigCon2020.csproj, a file used to manage the project. In this case there are differences between the remote and local code that need to be resolved for the project to work correctly. To see the differences, click the Compare Files link shown in Figure 10C.

Figure 11 shows the resulting display. The version on the left, with the code highlighted in pink is code from the remote repository while the version on the right with code highlighted in green is in the local repository. In this example the developers of both the local and the remote versions of the application had added a View to the application.

To resolve the differences, click the Merge button shown in Figure 10C. The display in Figure 11 is replaced with a similar display shown in Figure 12. The new display lists the two files side by side and highlights the differences both with color and with boxes that highlight the difference. When you click the checkbox next to one of the code samples it is automatically copied to merged code at the bottom of the screen. When you click the checkbox next to the other set of code it is merged with the first set of code.

At this point you can manually edit the merged code to make any final adjustments. Once you are satisfied with the changes, click the appropriate button to take the remote version or keep the local version, but which one is it? Notice the green color in the merged section of Figure 12. This is the local version, so you need to click Keep Local.

After merging change to EdSigCon2020.csproject there is one other file to check. The two versions of this file, HomeController.cs, have different, new methods that can be merged into one file as shown in Figure 13.

6. BRANCHING

The senior capstone project class where this was introduced has a three-part individual development assignment prior to the team project. Each part of the assignment builds on the previous parts and students often get to part 2 or part 3, make a coding mistake they can't undo and have to go back and repeat all the previous parts of the assignments before they can continue. This problem is only magnified once the team development project begins.

Git's branching capability can help solve this problem. With branching the user creates a copy of the current project, a branch, and then works on the branch while leaving the original alone. If the user makes a mistake and needs to start over, they can simply delete the branch and continue to work on the original code or make a new branch and work on it. Once the code in the branch is thoroughly tested it can be merged back into the master branch as illustrated in Figure 14

Figure 15 shows how to start the process in Visual Studio. When the Task Manager's Branches button is clicked (Figure 15A) the Task Manager displays a Create Branch window (Figure 15B) and after the branch is created the resulting list of branches in the repository is displayed (Figure 15C). To switch between a branch and the master, simply double click on the desired branch. If there are unsaved changes you will have to commit them but once that is done, you will be working on the selected branch. Figure 16 shows the bottom of the Visual Studio window with the SampleBranch selected.

A branch can be modified and tested without affecting the master or any other branch. It is also possible to create a branch from a branch.

Once the branch is tested and complete it can be merged back into the master. Figure 17A shows the Branches view of the Task Manger configured to merge the current branch back into the master branch. Figure 17B shows a conflict window since the branch contains code not found in the master branch and Figure 17C shows the conflict resolution window. The process from this point is exactly the same as the conflict resolution process outlined above.

7. CONCLUSIONS

The latest GitHub additions to Visual Studio make source code management easier than ever, especially for students with limited technical backgrounds. Students can now add projects to source code management, commit changes to local repositories and push them to remote repositories, resolve code conflicts and merge files, create and work with branches and eventually merge branches, all within Visual Studio.

The ability to perform these operations easily means students can practice version control techniques and it no longer matters where a student works. They can work on campus computers if those are available, they can work at home or even work in environments like the VDI that don't allow them to save anything and provide a clean desktop every time they run.
Students with continuing projects can use source version control on projects that build on each other, can easily create code branches, work on the branch and merge back with the original after testing is complete. In these situations, they can also avoid large amounts of rework after disastrous changes by discarding a damaged branch, reverting to an earlier branch and then moving forward with a new branch.

Students working on large, individual projects can use these tools to implement good version control practices while students working on team projects no longer have to take turns coding or figure out how to copy and paste different sets of code together. Team members can clone a project, work on their own from anywhere and then commit changes to the common shared remote repository.

All of these are skills that students will need if they continue with development careers after graduation, but they are skills that even non-technical students can learn and use now.

8. REFERENCES

Anon (n.d.), "Why don't more universities teach revision control?" (n.d.), retrieved from https://cseducators.stackexchange.com/questions/3590/why-dont-more-universities-teach-revision-control

Andersson, Mikka, "7 Version Control Best Practices for Developers,", 2018, retrieved from https://resources.collab.net/blogs/7-version-control-best-practices-for-developers, July 2020.

Git. (n.d.) retrieved from https://en.wikipedia.org/wiki/Git

Luce, T., "Adding MVC to the MIS Capstone," *Issues in Information Systems*, Vol 18 (1), pp 118-127, 2017.

Luce, T., "Evolution of an IS Capstone Class," Information Systems Education Journal, Vol 18 (1), pp 40-47, Feb 2020.

Marco, "Best Practices for Version Control in 8 steps", 2019, retrieved from https://ruleoftech.com/2019/best-practices-for-version-control-in-8-steps, July 2020.

Microsoft Team Foundation Server (n.d.) retrieved from https://docs.microsoft.com/en-us/azure/devops/server/whats-new?view=azure-devops

Nadagouda, Pratik, "Improved Git Experience in Visual Studio 2019", retrieved from https://devblogs.microsoft.com/visualstudio/improved-git-experience-in-visual-studio-2019/ , Mar 2020.

Uzunbayir, 2018, "A Review of Source Code Management Tools for Continuous Software Development", 3ed International Conference on Computer Science and Engineering, IEEE, 2018, pop 414-419

VMware Horizon 7. (n.d.) retrieved from https://www.vmware.com/products/horizon.html

Appendix – Figures

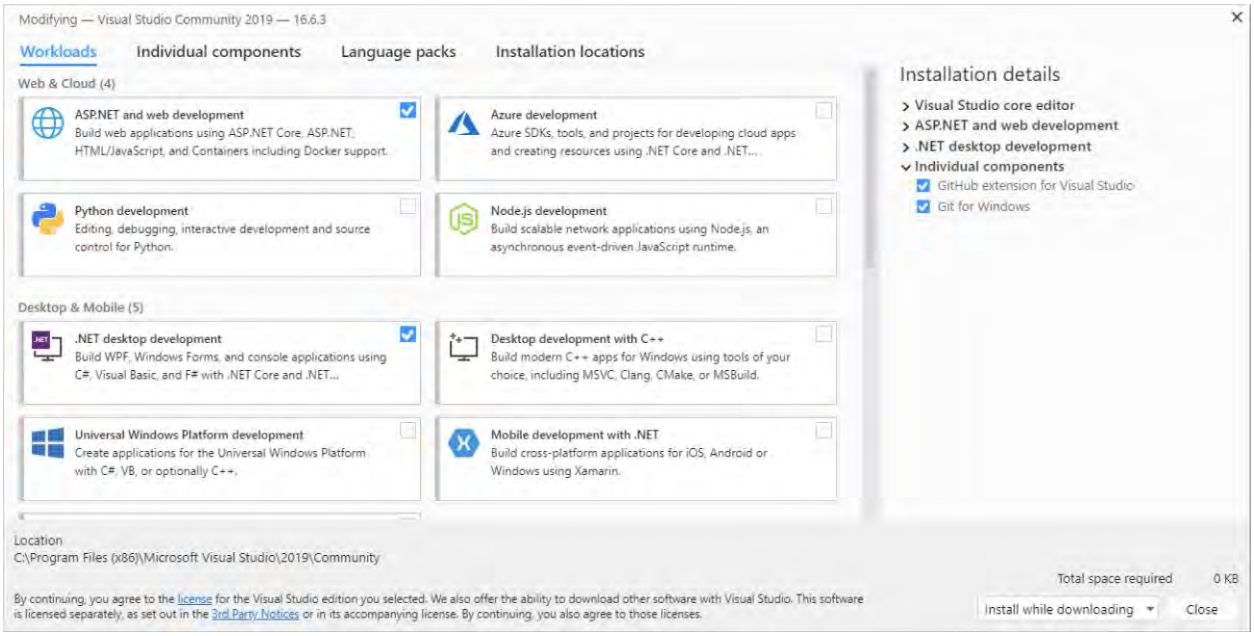


Figure 1. Visual Studio Workload Installer window

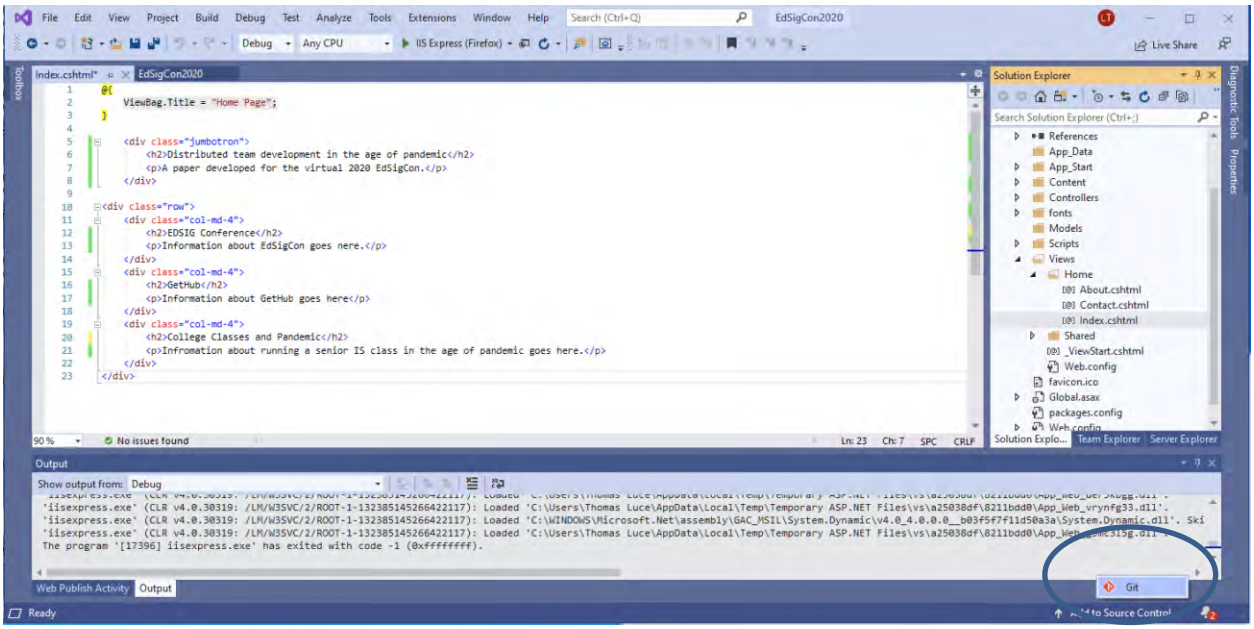


Figure 2. Visual Studio showing Add to Source Control

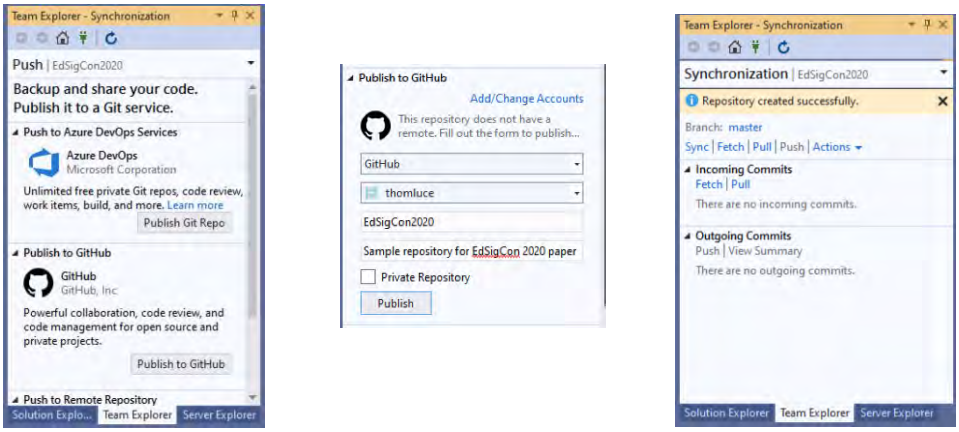


Figure 3. Adding GitHub for source code management.

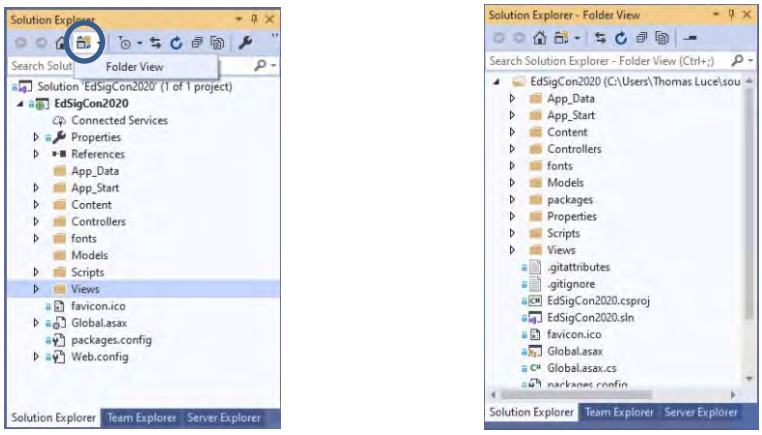


Figure 4. Solution folder options.

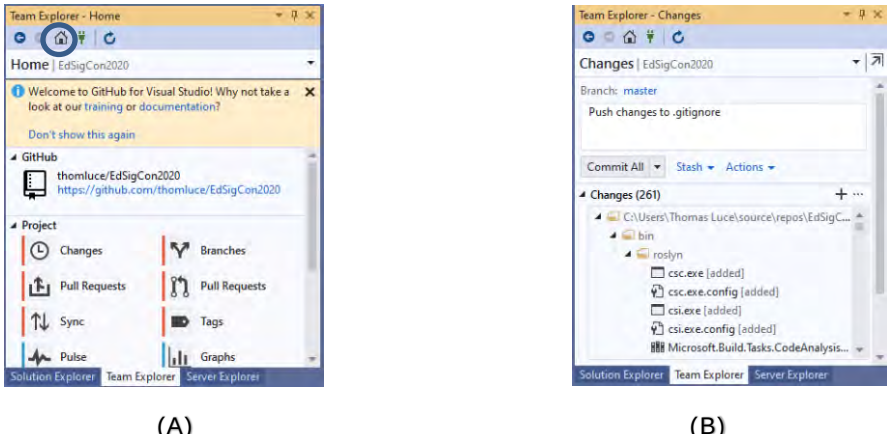
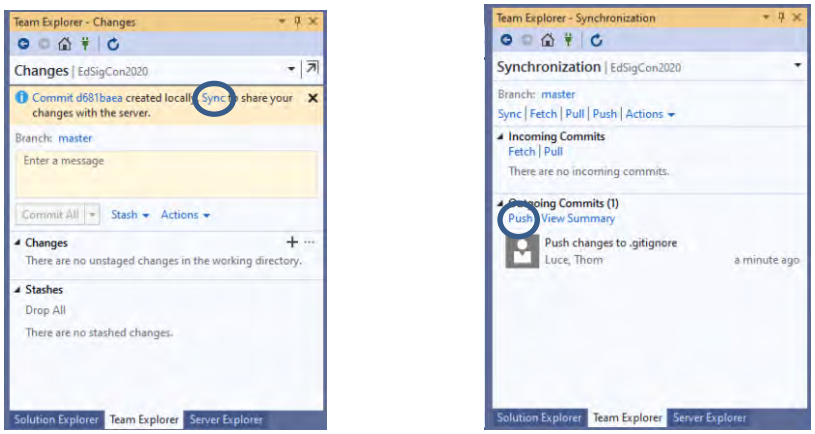


Figure 5. Team Explorer after clicking the Home icon and the window after clicking Changes



(A) (B)

Figure 6. Commit, Sync Push

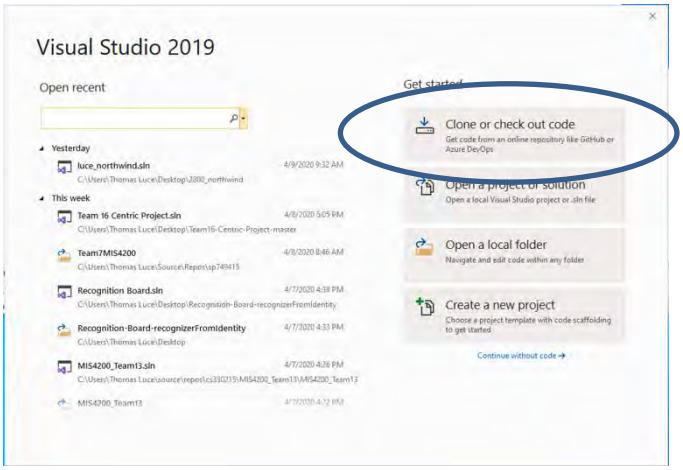


Figure 7. Visual Studio project startup

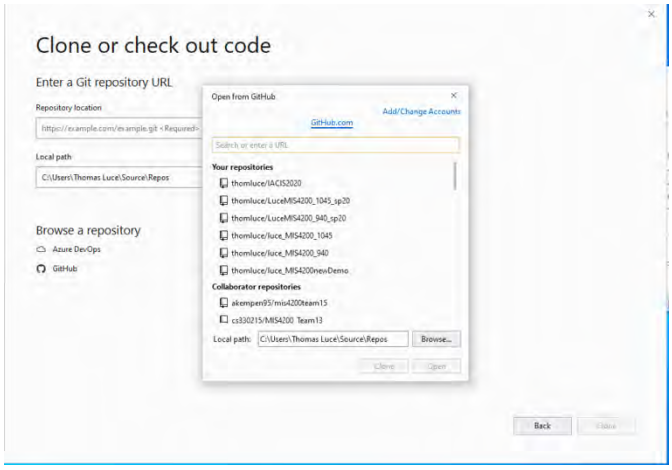
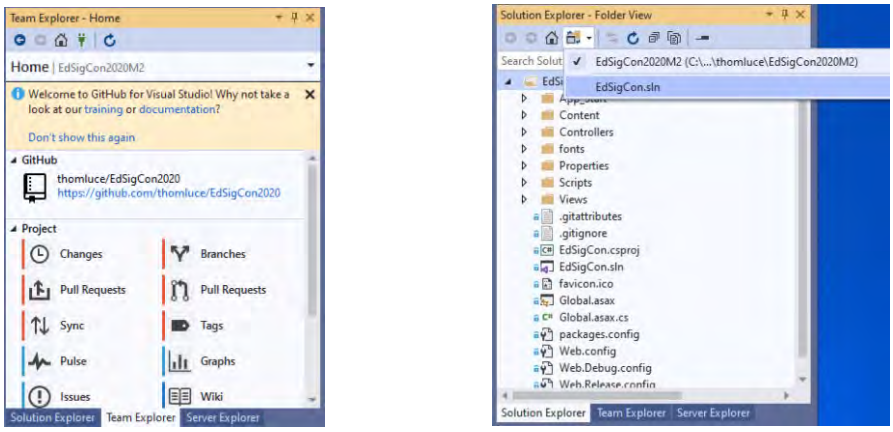
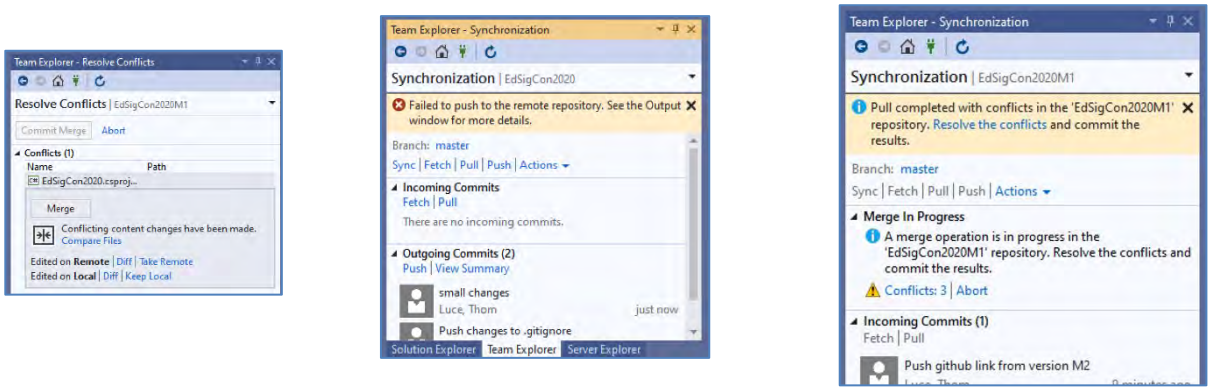


Figure 8. Clone or Check Out code window



(A) (B)

Figure 9. After cloning a project.



(A) (B) (C)

Figure 10. Conflict resolution and merging

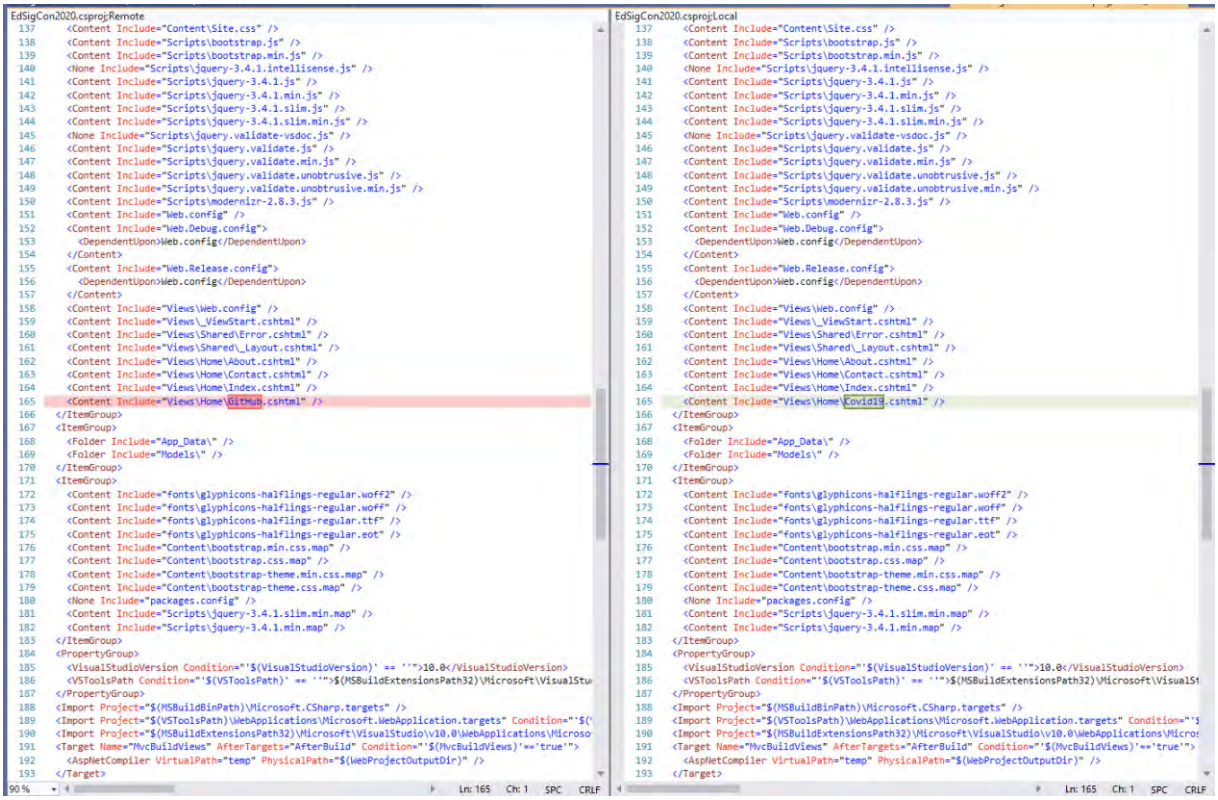


Figure 11. Comparing remote and local versions of EdSigCon.csproj

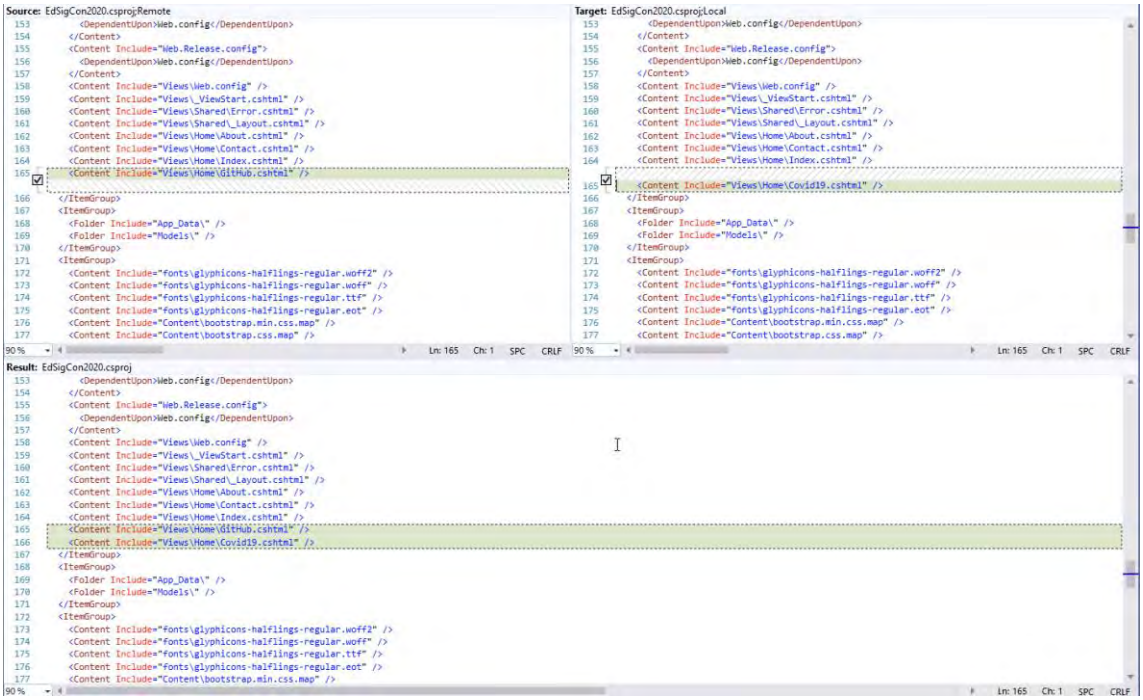


Figure 12. Merging files.

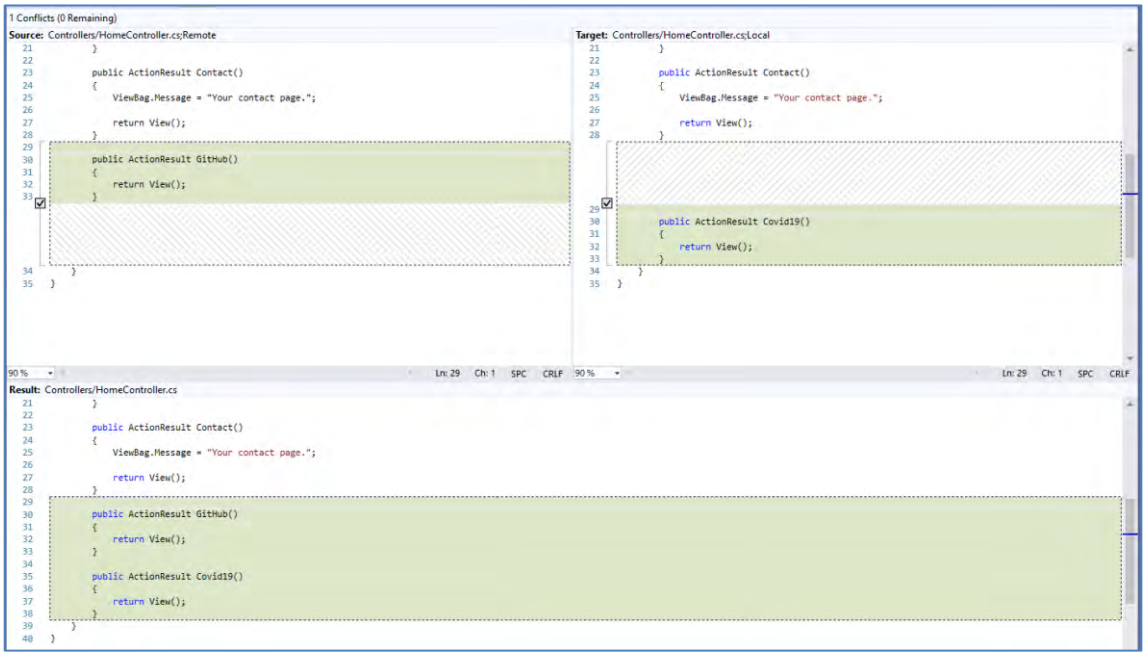


Figure 13. Merging changes in HomeController.cs

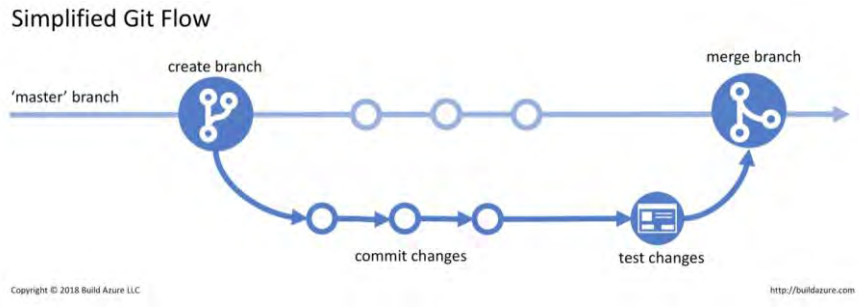


Figure 14. Simplified Git Flow (https://build5nines.com/introduction-to-git-version-control-workflow/)

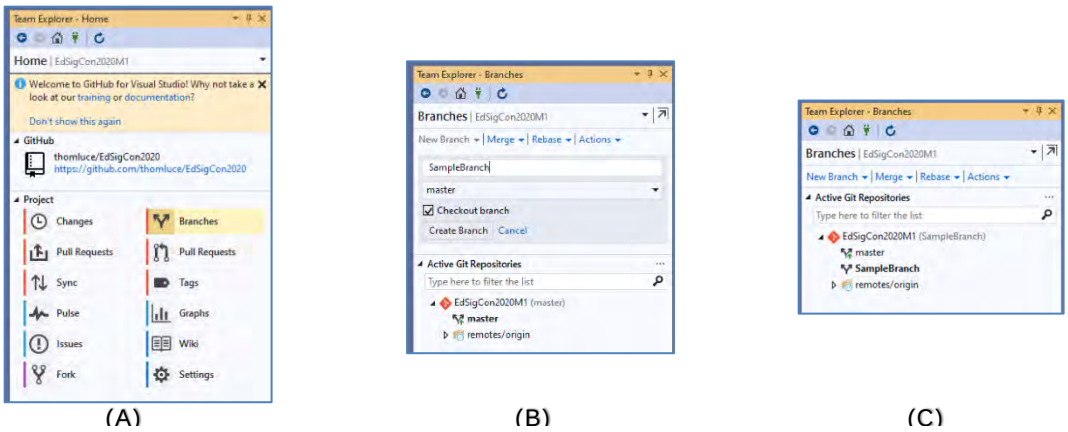


Figure 15. Creating a new branch

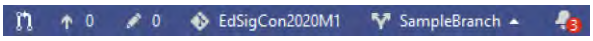


Figure 16. Visual Studio with SampleBranch selected

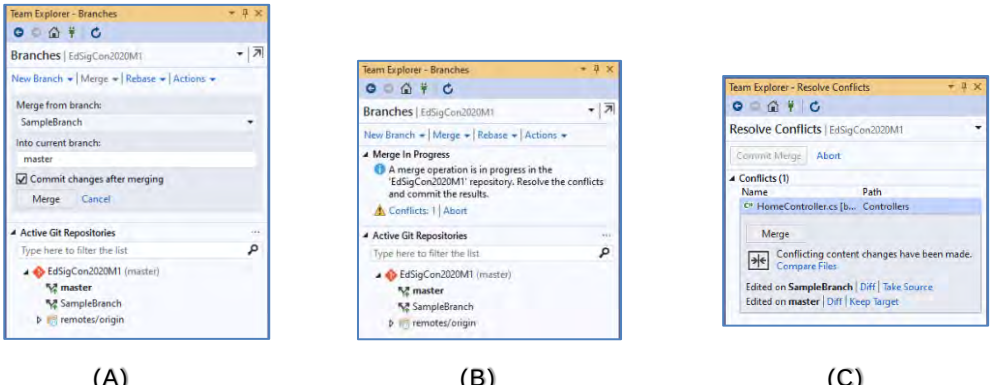


Figure 17. Steps in the Merge Process