Building your Allocated Component

Your Git Hub day

[Building your Allocated Component 1](#_Toc58413680)

[Before you Begin 1](#_Toc58413681)

[Revisit your System Specifications 1](#_Toc58413682)

[Copy and Adapt 1](#_Toc58413683)

[So where do we go next? 1](#_Toc58413684)

[The GitHub Workflow 2](#_Toc58413685)

[Create a clean folder for this work cycle 3](#_Toc58413686)

[Clone the repository to the folder 4](#_Toc58413687)

[Create a new branch (Don’t edit the master directly) 8](#_Toc58413688)

[Do some work on your part of the system 12](#_Toc58413689)

[Save your work 18](#_Toc58413690)

[Scan for changes 19](#_Toc58413691)

[Add notes 20](#_Toc58413692)

[Commit changes 20](#_Toc58413693)

[Sync 20](#_Toc58413694)

[Go to Git 21](#_Toc58413695)

[Create a new pull request (complete the process with appropriate comments from another team member) 22](#_Toc58413696)

[Delete branch 24](#_Toc58413697)

[Delete the local folder you created at the start of the day 24](#_Toc58413698)

# Building your Allocated Component

## Before you Begin

Make sure that you have completed the work for last week and that you have watched the eLectures for this week before starting this work.

## Revisit your System Specifications

Remember at the start of the module you created system specifications identifying what your application is going to do and who is allocated which section?

Now is the time to return to these documents to remind you of who is working on what component.

I will continue to use my address book examples but remember you don’t simply copy my examples, you need to copy and adapt such that it meets your personal requirements.

## Copy and Adapt

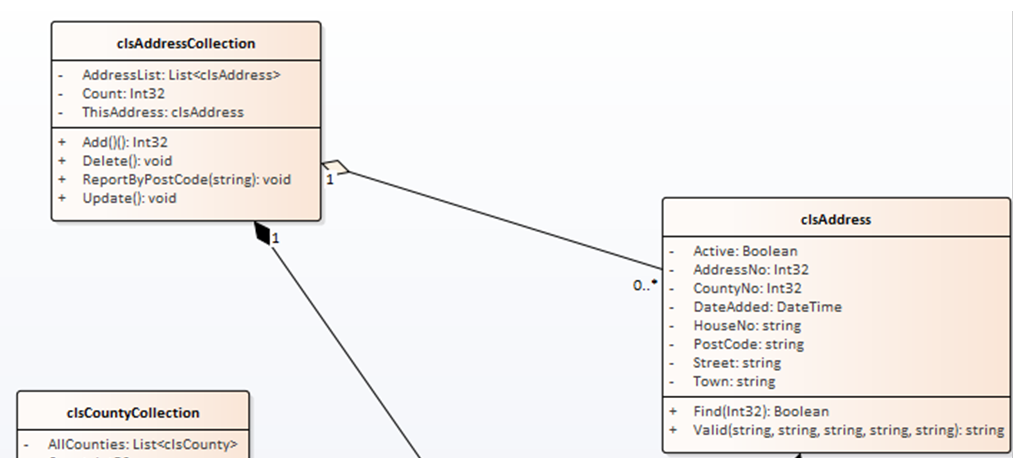
One approach we will use to introduce programming concepts is copy and adapt.

The idea is that I present you with an example from my own design and then you need to adapt that design to match your own requirements.

For example, if you are creating a system to manage staff you need to look at my address book code and adapt it such that it follow your own requirements not mine. If I say clsAddress you need to create clsStaff etc..

## So where do we go next?

We will implement the class pair clsAddressCollection and clsAddress



As above let’s start with something simple. Since clsAddressCollection uses clsAddress we will start with clsAddress.

The problem we have now is that via GitHub you will all be working on your systems as a team so we need to establish a work flow such that you don’t keep treading on each other’s toes.

## The GitHub Workflow

Here is a quick check list of the GitHub workflow we will use as you work together...

Initial Setup (You should all have done this by now!)

* Set up the local skeleton system for the team project containing one class library, one test project a customer facing front end and a staff facing back end
* Make sure it compiles, and the folder structure is correct
* As a team select one person to host the master repository on GitHub
* Invite team members to collaborate
* Add the local system skeleton to source control such that the code is in the master repository
* Each member should test that they are able to make a clone of the skeleton system and that the folder structure is identical for each team member
* Remember – the master repository is sacred do not edit directly

The next stage is to forget that you are a student and start thinking like an employed software developer.

Imagine now that you work within a company, clock in at nine o’clock and leave at five.

At 9am you start the working day

* Create a clean folder for today’s work cycle
* Clone the repository to the folder
* Create a new branch (NEVER edit the master directly)
* Do some work on your part of the system

At 5pm you finish the working day

* Save work
* Scan for changes
* Add notes
* Commit changes
* Sync
* Push
* Go to Git
* Create a new pull request (complete the process with appropriate comments from another team member)
* Delete branch
* Delete the local folder you created at the start of the day

IMPORTANT!

When you are working on code within your own local branch, that code is unavailable to other members of your team until you have pulled the changes into the master branch.

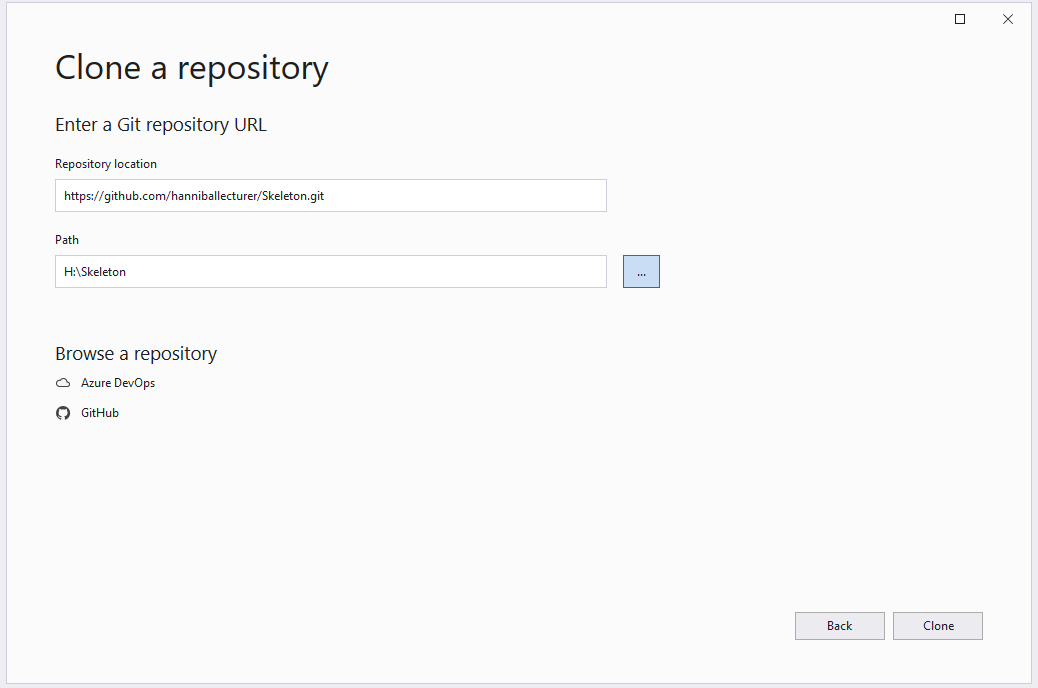
There are two issues to think about here.

1. Never put code into the master that is buggy or doesn’t compile, this is a really good way of spreading bad code to your team members
2. Keep the life of your branches short. What I mean is complete a small section of work, say create a few tests with corresponding changes to the class library then merge the new code with the master once you are confident it is debugged and free of compilation errors.
3. 24 hours is a long time in Git. The longer you sit on code in your own branch the longer other team members have to wait to access your code.
4. Always talk to each other! Let each other know what work you are about to undertake such that you may anticipate any potential problems.

As above, we will start with the most basic test i.e. creating an instance of the required class. This time we will do it within the context of the GitHub work flow.

## Clone the Repository

As you start up VS you should see the option to clone the existing repository, select it and complete the details like so…



IMPORTANT

* Make sure you enter the URL of the shared repository
* Make sure that you create the clone in an empty folder of your choice DO NOT LET VISUAL STUDIO DECIDE THIS FOR YOU AS THIS IS REALLY GOOD WAY OF LOOSING YOUR WORK!

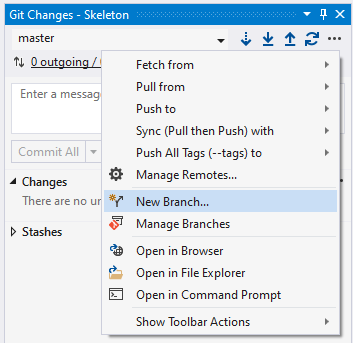
## Create a new branch (Don’t edit the master directly)

As things stand you have a copy of the master repository, if you edit this you will be directly editing the system and potentially create problems for your team members.

The only time you edit the master repository directly is for high level admin operations ensuring that there are no active branches open on the work.

To do today’s work you need to create a new branch for yourself to isolate your changes from the code belonging to other members.

In Visual Studio under Git changes select New Branch from the following drop down menu ...



You should see on the next screen that the master branch is selected...

Graphical user interface, text, application, email

Description automatically generated

Press create to add the branch.

## Do some work on your part of the system

Let’s create a test class.

I am going to claim section 6 as my allocated part of the system. I plan to create an address book of clients.

So, I will rename section 6 to the following…



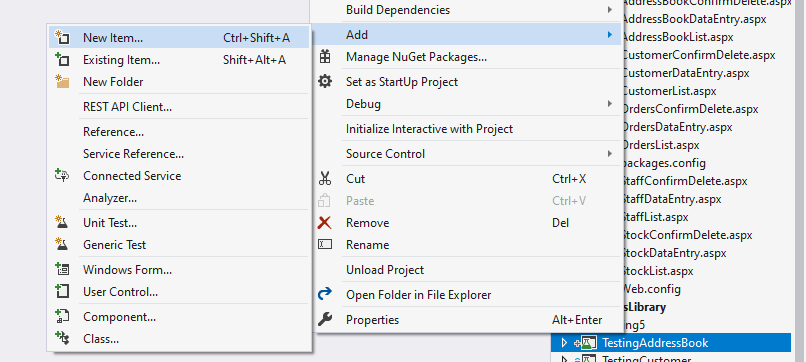
And my test project like so…



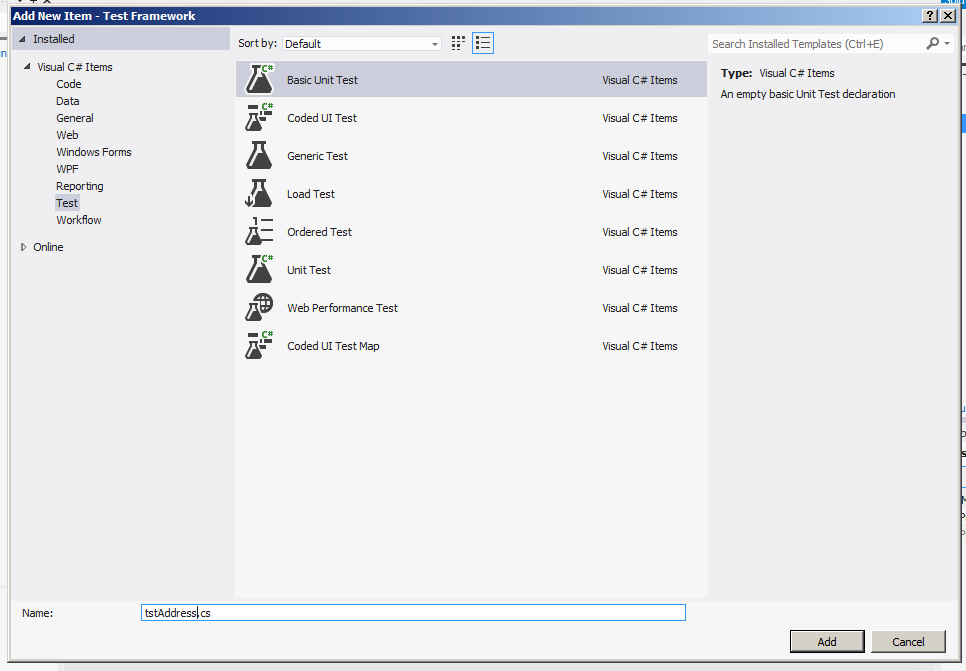
Clearly you should have done this part when your SCRUM master set up the master repo.

You need to copy and adapt my examples from now on and apply them to your part of the system.

To create my first test class, I need to right click on my test project and add a new test.



I need to locate Basic Unit Test under the test section of the templates…



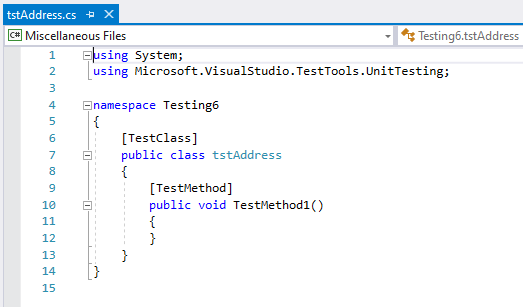
In this example I am creating a test to test another class.

The class I am really interested in is going to be called clsAddress and will be stored in my class library.

To make clsAddress I will create a test class called tstAddress, note the naming convention. tstAddress will be stored in my test project TestingAddressBook.

You need to adapt this to your own component. If you are working on staff management you would create tstStaff, if you are working on stock, tstStock etc.

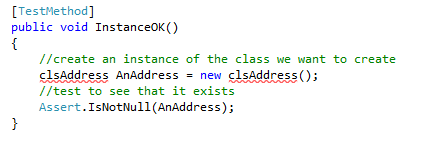
When the test has been created you should see something like this on the screen…



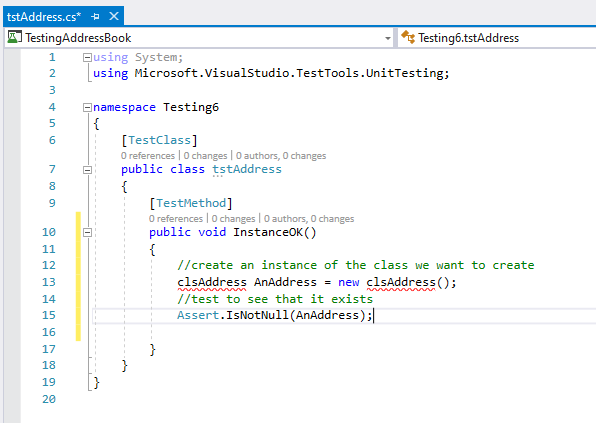
(The namespace will vary depending on your allocated component, don’t change this.)

The first test we need to create is the test for instantiating a new object based on clsAddress.

So next we create the test for instantiation…

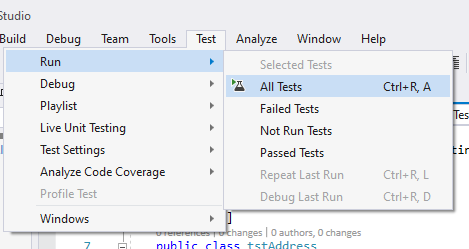


The code for the class should look like this…



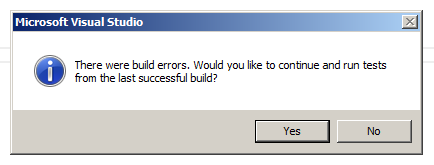
(The red underlining can be a bit slow at times to appear)

To run the test you will need to select the test option from the main menu…



Given that clsAddress doesn’t exist (as indicated by the red underlining) the test should fail.

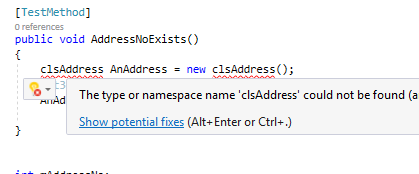
Like so…



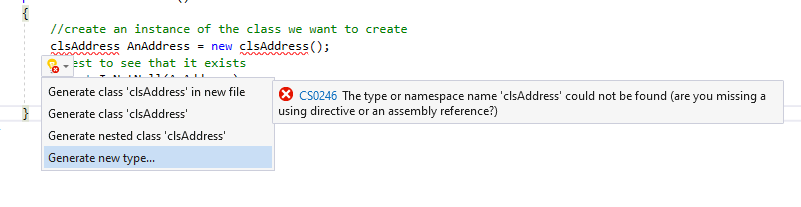
(Always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, always, press **NO** when you see this message!)

To fix the problem we need to create the class file.

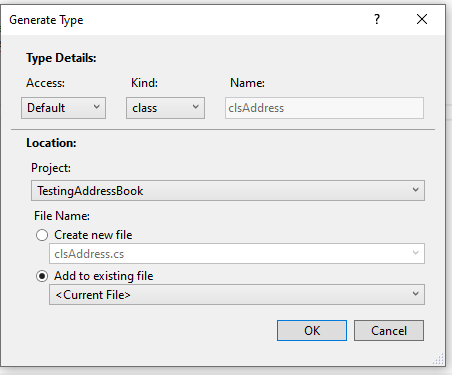
Hold the mouse over the red underlining and wait for the following menu to appear…



Click on “Show potential fixes” and these options should appear.

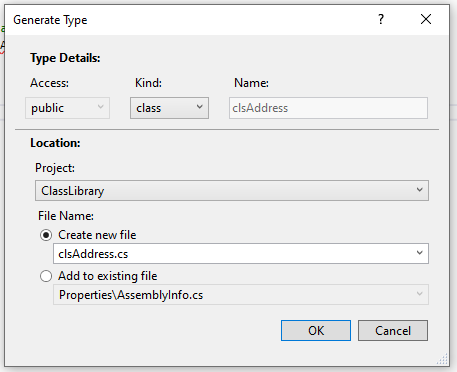


Select generate new type and you should see the following…



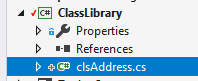
Note that the default setting on this screen are almost certainly wrong.

Change the settings like so…

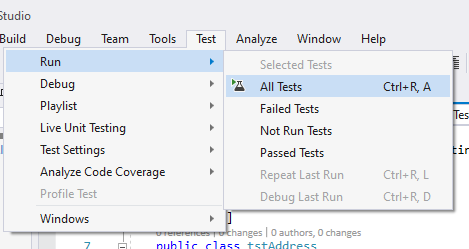


Make sure the settings are correct and that you are creating the class in your class library.

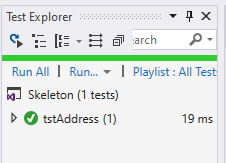
Once you press OK you should see the new class in your shared class library…



Run all test like before…



You should now see the test pass…



That’s enough work for one day!

## Save your work

Make sure everything is saved in Visual Studio.



## Switch over to Git Changes

It should list the changes you have made in the branch…

Graphical user interface, application

Description automatically generated

## Add notes

Before you are able to do anything else you need to add some notes explaining what you have changed. The idea is that you leave a paper trail so that other people know what has been changed in the system and by whom.

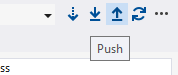
Graphical user interface, application

Description automatically generated

Press commit all…



## Push the Branch

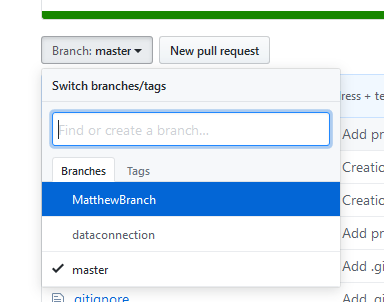


Your work is now stored in GitHub

## Go to Git

The next step is to check that the changes have been copied over to GitHub.

If you look at the repository you should see that the master branch and your own branch are listed...

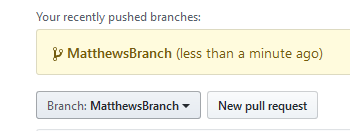


Select the new branch, not the master.

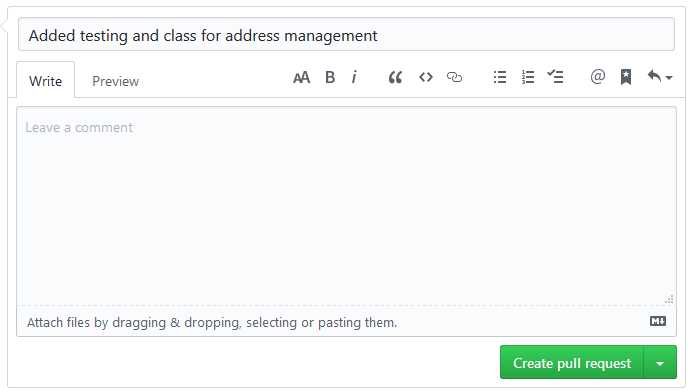
You should be able to see all of the changes listed in the repository.

## Create a new pull request (complete the process with appropriate comments from another team member)

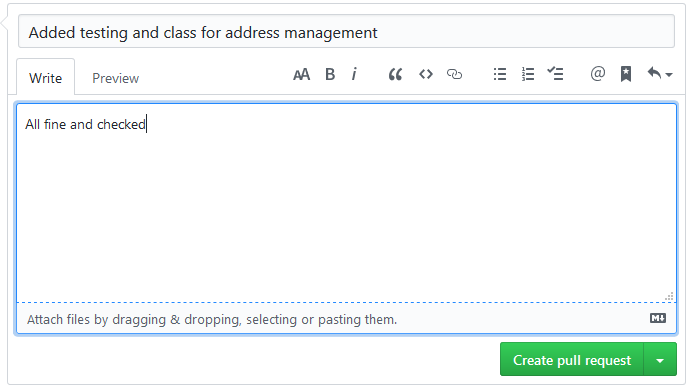
We now want to take the branch on GitHub and merge it back into the master branch, to do this we need to create a new pull request...



Press the button to create the pull request and you should see the following screen...

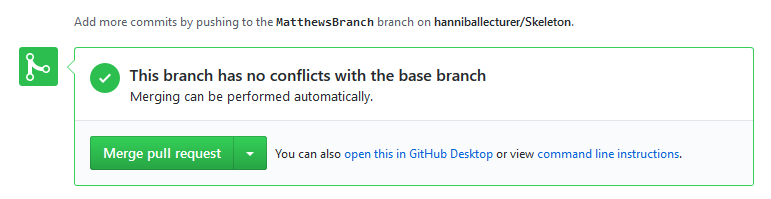


The idea here is that you ask another team member to review the changes and confirm that you are not adding bad code to the main repository. Once you are both happy you may add some comments.



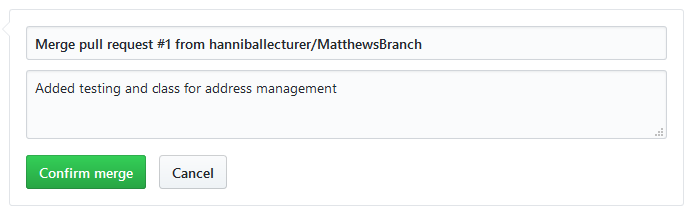
The select create pull request.

GitHub will check to see if there are any conflicts, hopefully not...



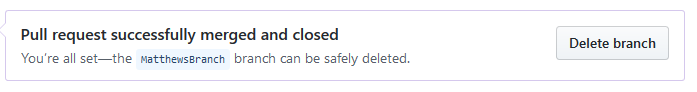
If there are conflicts you will have to resolve them (this is where the fun begins and why it is good to talk to each other!)

If all is well, select merge pull request followed by confirm merge (you have to really mean it!)



## Delete branch

Once the pull request is completed you need to delete your branch...



## Delete the local folder you created at the start of the day

I would also suggest that you tidy up your local machine by deleting or archiving your local folder.