Application Development MCOMD1ADC

Dr. Alexios Louridas

2021-02-09

Contents

Introduction

Module Structure

We are going to have 12 weeks of teaching. What we would like at the end of this module is for you to be able to create a software application from scratch. From an idea to developing and maintaining it.

To achieve this we are going to introduce tools that would help you in developing software followed by introducing concepts to help you organise your ideas. We then going to look at developing the application front end(graphical user interface) and back end. Last you will be introduced to testing and application management for easy maintaining your software application.

The indicative schedule we are going to follow can be seen in table 0.1. Although we shall try and follow this schedule it all depends on you. If you believe that you need an extra session to describe and master something then we are here to listen. Please tell us and we shall try to change it accordingly.

Assignment

A. Version Control

While developing software, development teams use tools to help them reduce development times and produce more stable and better solutions. The most popular of these are version control or source control tools. The aim of these tools is to keep a record of all changes that happen on the source code of an application or applications being developed.

A version control tool keeps a record of all changes while recording the person or team responsible, the time of a change and the content that has been altered. All of these are recorded in a specialised database usually called a **repository** or repo for short, that can be accessed at any point to review code and provide a fix or an addition to the source code by identifying the best place, time and/or person to do so.

Table 1: Indicative Content Schedule.

Session	Indicative Content
0	Version Control
1	Revision
2	Introduction to Classes - Basics
3	Arrays and Collections
4	Objects and Structures
5	Algorithms
6	GUI and Forms
7	Methods Again
8	Errors and Exceptions
9	Testing and Data Validation
10	File System
11	Summary and Revision

Try and remember how many times have you lost work that you wish you have backed up or you have but you do not remember where it is located, or you have not backed up as frequently as you would have wanted with the specific change you are looking. All of these cases would disappear or be minimised by using a version control tool and having a repository to store all your changes.

In this module you will learn how to use one of the most popular version control tool. Before we go on to play and start creating or obtaining repositories let us see the types of repositories that exist.

Local based model

The simplest way of having version control is to keep multiple copies of the same file at different stages of its development. You try to keep versions by changing the filename appropriately or keeping track of the last time modified. Although, simple it is associated with a lot of errors and large disk space.

To improve this method some decided to create a small database to track changes of the files under version control. If you know a version control system called RCS you probably are a bit older. This system kept the database on a special section of your hard drive, thus at any point you wer able to go back and recreate the files at specific times.

Client based model or centralised systems

A single repository exists that users connect and exchange information. The repository is centralised and all repository functions are done centrally. The users connect and obtain a **part** of the working copy of the repo. The users have the ability to change the source files but any changes to the repository

would need to be communicated directly to the central repository.

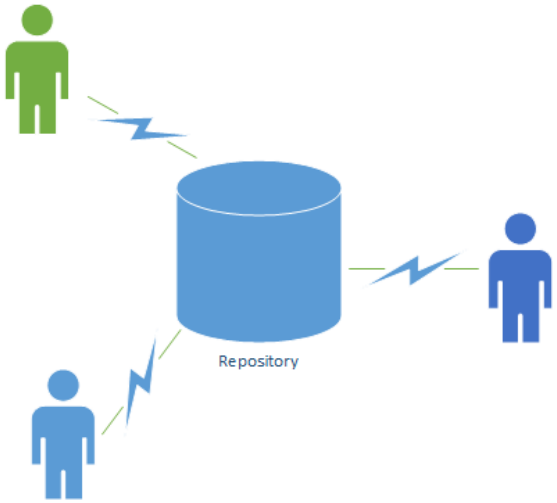

Figure 1: Client data model

One of the main disadvantage of these system is that it has a single point of failure. If the server that contains the repository goes down then collaboration and version control does not work arnymore.

Distributed based model

Each user has a working **full** copy of the repository. Each user can make changes to the repository locally and thus work offline if required. Once back online the users can distribute all changes to the repository where all users would then have access to view.

B. Git and GitHub

Git is a distributed version control system that has approximately 90% usage in the industry. The main reason of its popularity is its performance compared to its competitors. It provides an outstanding flexibility and security and everything is inbuilt and free. In addition to Git advantages, because of its popularity there are many third party software tools and services that have integrated with Git including IDEs such as visual studio, issue and project tracking software, such as Jira, and code hosting services like GitHub.

How does Git Work

To start understanding what git provides let us look at the following scenario:

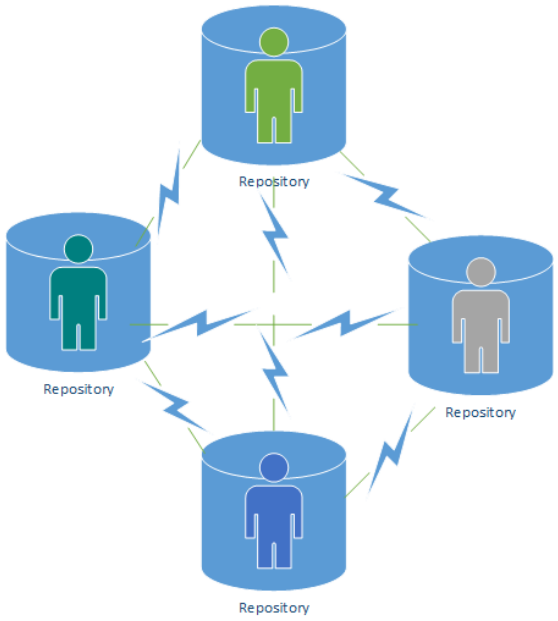

Figure 2: Distributed data model

A software application is being developed and it is currently in version 1.5 and the developers are working on version 2.0. Eventually the development finishes and version 2.0 is released. A customer though does not want to go to version 2.0 as they do not have a budget to support the update, but they would like a feature that they have seen in version 2.0 to be included in their version of their software (1.5).

The developers only have to go to Git history and "copy" the required change associated with the feature needed and add it to the release version of 1.5 and thus create a new release 1.5.1. No new code really needed to be developed for what the customer needed.

From this example you can see that Git offers flexibility to control your changes and releases without time constraints.

How does a Git repository work?

The database that contains all the information of the files we want to include for tracking their version and changes is called the repository. The repository in Git is stored in the folder *.git*. If you delete the folder you would lose all the history of this repository. On the other hand it does not mean that you have deleted the repository, as it is stored centrally but also to other users hardware who might have copied the repository.

As a repository is created there are no files associated initially with it so there is really nothing to track. So we need to add files to the repository. Once they are added they are now being tracked. If we now modify a file in our computer that is part of a repository it is in a state of *modified* for Git. If we want these changes to be included in our repository we need to flag them and place them in a state usually called *staged*. Once the file is in the staged state we can tell our repository to update the file accordingly inside our repository. The file is then in the *committed* state.
Although that is all done and we have included the modified file in the repository it does not mean that everyone can see it. We need to make sure that we flag our repository has changed to other users. To accomplish this we need to *push* our new repository to the central store. The central store can either be a dedicated server that we have access and full control over or we can use a web service that allows us to store our repositories. Such a web service is GitHub. As a university student you have access to GitHub.

Start using Git

First and foremost we need to install Git in our computer. Download the latest Git Version from:

https://git-scm.com/downloads

When you've successfully started the installer, you should see the Git Setup wizard screen. Follow the Next and Finish prompts to complete the installation. The default options are sensible for most users.

Run the Git Command Prompt (Git CMD) or Git Bash depending on installation

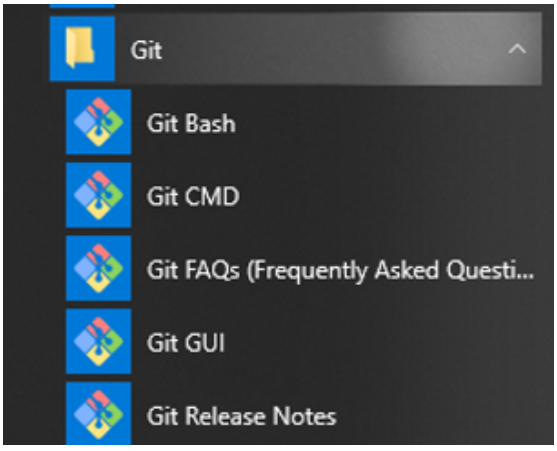

Figure 3: Run Git Enviroment

Personally I prefer running Git Bash as bash is one of the most widely used shell.

Table 2: Bash Most Common Commands.

Command	Operation
pwd	Show current directory
mkdir ex_dir	make directory ex_dir
cd ex_dir	Change directory to ex_dir
cd ..	Go up a directory level
ls	list all files in the current directory

I have summarised The usual commands for bash that you would probably need for running git commands in the table below.

To run any command in git you require to write *git* followed by a space and the git command, followed by any required command requirements.

`git <<git command>> <<git command parameters>>`

config

To configure Git with your preferences you would require to run the command $config$. So run the following commands using YOUR NAME and YOUR UNIVERSITY EMAIL to setup your git installation.

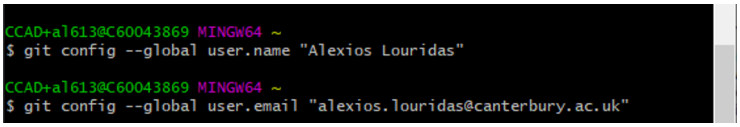

Figure 4: Config command example

Before you can start creating and cloning repositories we need a place to store them. Lucky for you, Terry from IT has done a wonderful job and we all have access to GitHUb via our university credentials.

DO NOT GO TO THE MAIN GITHUB WEBSITE INSTEAD GO TO:

http://github.canterbury.ac.uk

Use your university credentials to sign in and create an account. If you already have an account please either create a new account or add your university credentials to that account. Once you enter GitHub I shall assign you to the appropriate team/teams.

PLEASE use your whole name when creating your account

We now going to introduce you to the most usual git command:

clone

This command will clone a repository. What that means is that you will download a working copy of a repository.

Open Git Bash and go to a directory in your current machine that you wish to clone a repository. To change directory in Bash use the cd command. For example: $ cd c:\temp Git Bash informs you all the times your current directory. You will see that before your promt. Using Git Bash run the following command: $ git clone https://github.com/cccu-uk/MCOMD1ADC2021.git"

Congratulations you have cloned your first repository!

Now that you have been setup to use GitHub and Git you are more than welcome to start playing around with creating your own private repositories and start learning a bit about git and github.

I would recommend to go to https://try.github.io/

Try and do as many of the tutorials as you can. Especially the section "Learn by doing".

Beyond Source Code

The use of Git and GitHub does not have to stop on code only. As a student it can be a valuable tool for all of your assignments. I would recomend to start using these tools not only to famirialise yourself with the tool but also to get to know a community in GitHub and open source code. Below is a simple example of how you can use it for group work in any assignment.

You are a student and a member of a group that is working on a group assignment set by Gordon. You have two progress checks to pass before your final submission. Each member of the group needs to do certain aspects of the assignment but each depends on a part of each others work in order to be able to complete your parts. You make changes to your group assignment, adding some files for the upcoming progress check of your assignment, then **commit** and **push** those changes with descriptive messages in order for your fellow students in your group to be able to do their parts. You then work on a second part of your assignment that is required only for the end report and **commit** those changes too. As you have committed these separately they are also stored separately in the version history of your Git repo.

Another member of the group is having issues with a part of the assignment that is needed to be submitted in the progress check. You decided to help them out so you **pull** the latest repository and fix some of the issues your fellow student has and **commit** and **push**. The other student can then **pull** the latest and finish his task. Once all students of the group finish the all their changes they **commit** and **push** at their own time and the end product is complete.

When marking Gordon is able to see the **log** and understand how much work everyone has done and how valuable you have been on the team by looking at the **commits** you have done.

At some point in the near future you complete your degree and you are being interviewed for a position in a company. The interviewer asks if you have done something similar to the group assignment that Gordon had set several year ago. You answer “yes” and share your repo which resides in your GitHub account. Another candidate has similar experience but has never used Git or GitHub and thus they cannot share any of their work.

Although you and the other candidate have similar experience and knowledge you get the job just because you can easily show and demonstrate your work. That is all that it takes sometimes to gain or lose a work position.

Chapter 1

Recap

1.1 Revision

The purpose of revision is not just to remember what you have done previously. Revision helps you understand what you could not on a first pass of a material, it helps you identify concepts and material you did not see before, it helps you deepen your knowledge on a subject matter and of course it helps you remember.

1.1.1 UML Activity Diagrams

Unified Modeling Language is used frequently in the industry to portray and visualise system characteristics of software. It is usually the job of system designers to design software using UML diagrams to portray concisely how a software is going to be organised and look at the different aspects before coding. UML is independent of programming language and thus it used widely on other areas as well. There numerous diagrams that UML support from class diagrams to system diagrams and activity diagrams, which we are going to briefly describe here.

Activity diagrams represents a workflow or activity of a part of a software system. When describing an algorithm or part of an algorithm it is essential to make and use simple symbols in order to quickly comprehend the operation. In activity diagrams you have two small circles that represent the initial state of the system and the end state of the workflow. The end usually is represented with two circles one inside the other, where the inner most circle is solid. The start is a single simple filled in (solid) circle.

Between these two circles there are actions which are represented by an elongated circle with text. All of these are then connected with arrows called transitions. The transitions indicate the flow of an activity. Sometime a square with notes inside it will appear within activity diagrams to signify notes.

1.1.2 Variables

When creating any variable this really means that you are creating a memory location. Where exactly it is stored depends on the variable type and memory availability of the computer where the program is running on. It is important to understand that anything you create in your program has to be stored somewhere when your application runs. This will help you create better applications and write better code.

Scalar and Arrays

1.1.3 Operators

1.1.4 Control Structures

Usually when you read a book or even this document you read line after line sequentially. In programming languages this is also the case, you write a series of commands and one follows the next. In real life or in books sometimes you will find statements that state that you might like to skip a part if something is satisfied or not or you might keep repeating the same section or revisit the same section. This is also true with our everyday lives. We wake up and we have to decide if we are going to wear the black socks or the red socks, we need to make our breakfast which we need to decide what are we going to have. Of course this is repeated mostly everyday.

In C# to achieve what we do in our everyday life, making decisions, repeating a process there are 3 types of control structures. Sequential structures, selection structures and iteration structures.

Sequential structure is already defined in C# by the compiler so we do not really have to do anything, besides knowing that one statement of code will be followed by another.

The selections structures that C# provides us are three. We shall now look at each one individually.

1.1.4.1 Single Selection

To chose an alternative course of action if something is satisfied within a program you would need to use an if statement. A simple example would be the pass grade of each module is 40%, so if you want to write a simple program that takes an input of your final mark and then outputs if you have passed only, you would probably write something like this:

```
if (input_mark >= 40)
{
  Console.write("Passed")
}
```

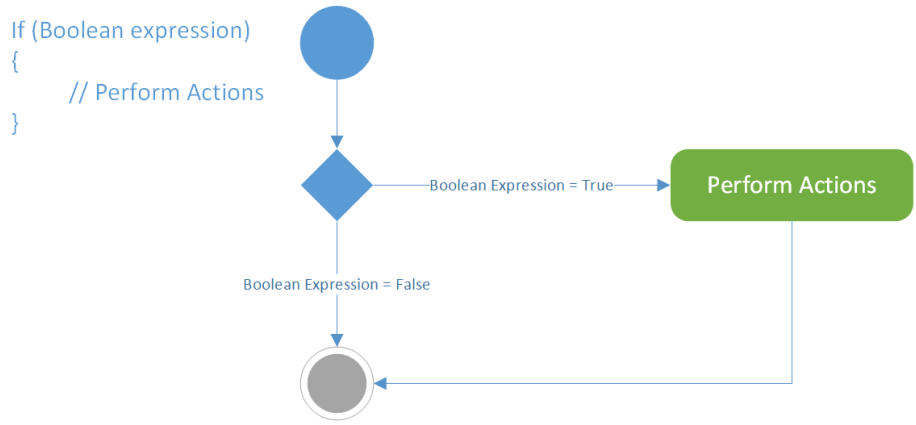

Figure 1.1: Client data model

1.1.4.2 Double Selection

To chose between a choice and alternative you can use the double selection contro sturcture of if..else A simple example would be the pass grade of each module is 40%, so if you want to write a simple program that takes an input of your final mark and then outputs if you have passed or failed, you would probably write something like this:

```
if (input_mark >= 40)
{
  Console.write("Passed")
}
else
{
  Console.Write("Failed")
}
```

1.1.4.3 Nested Selection

To chose between multiple choices more than 2 you can use the a nested selection control structure of if..elseif..else. Taking our simple example for passing you could increase it complexity by including the final mark category. A sample nested selection would probably write something like this:

```
if (input_mark >= 40 && input_mark < 50)
{
  Console.write("Passed with a 3rd")
}
elseif (input_mark >= 50 && input_mark < 60)
{
  Console.write("Passed with a 2.ii")
```

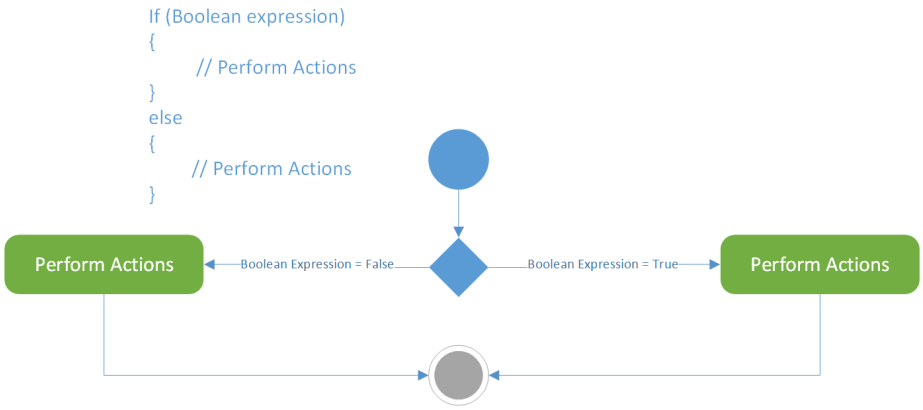

Figure 1.2: UML Activity Diagram for a Double Selection Statement

```
}
elseif (input_mark >= 60 && input_mark < 70)
{
  Console.write("Passed with a 2.i")
}
elseif (input_mark >= 70)
{
  Console.write("Passed with a First")
}
else
{
  Console.Write("Failed")
}
```

The activity diagram for a nested if statement can be seen below:

The activity diagram for a nested switch statement can be seen below:

The activity diagram for a while iteration statement can be seen below:

The activity diagram for a do..while iteration statement can be seen below:

The activity diagram for a for iteration statement can be seen below:

1.1.5 Methods and Modular Programming

1.2 Software Development

1.2.1 Creating or opening an application

How would you create an application what do you need to know before you create one?

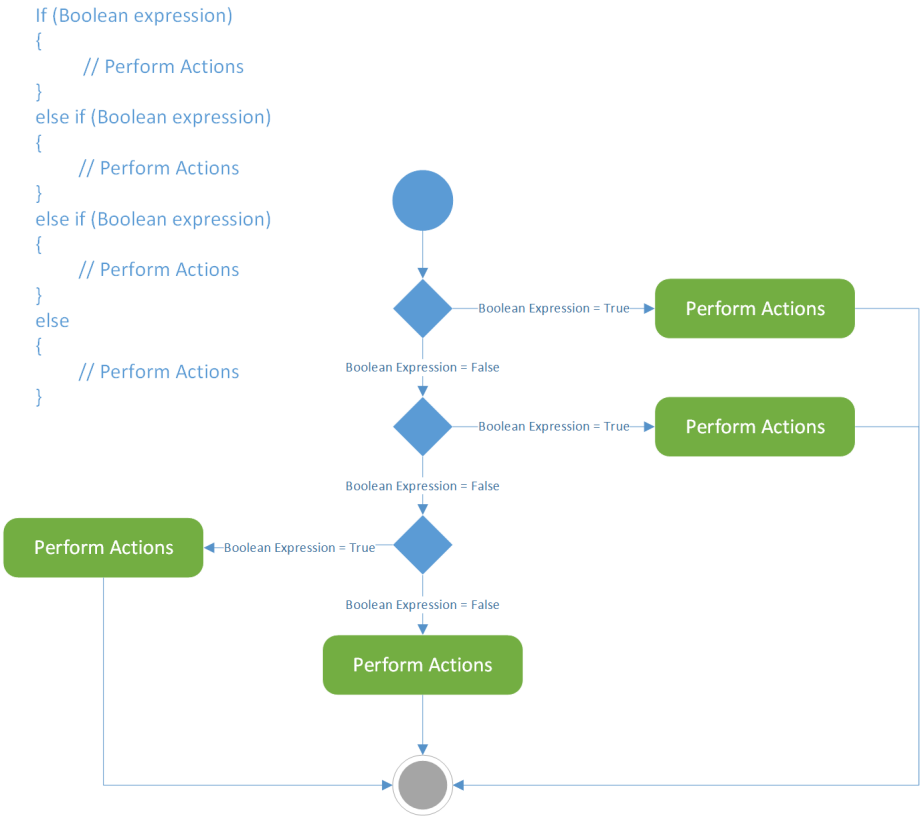

Figure 1.3: UML Activity Diagram for a Nested If statement

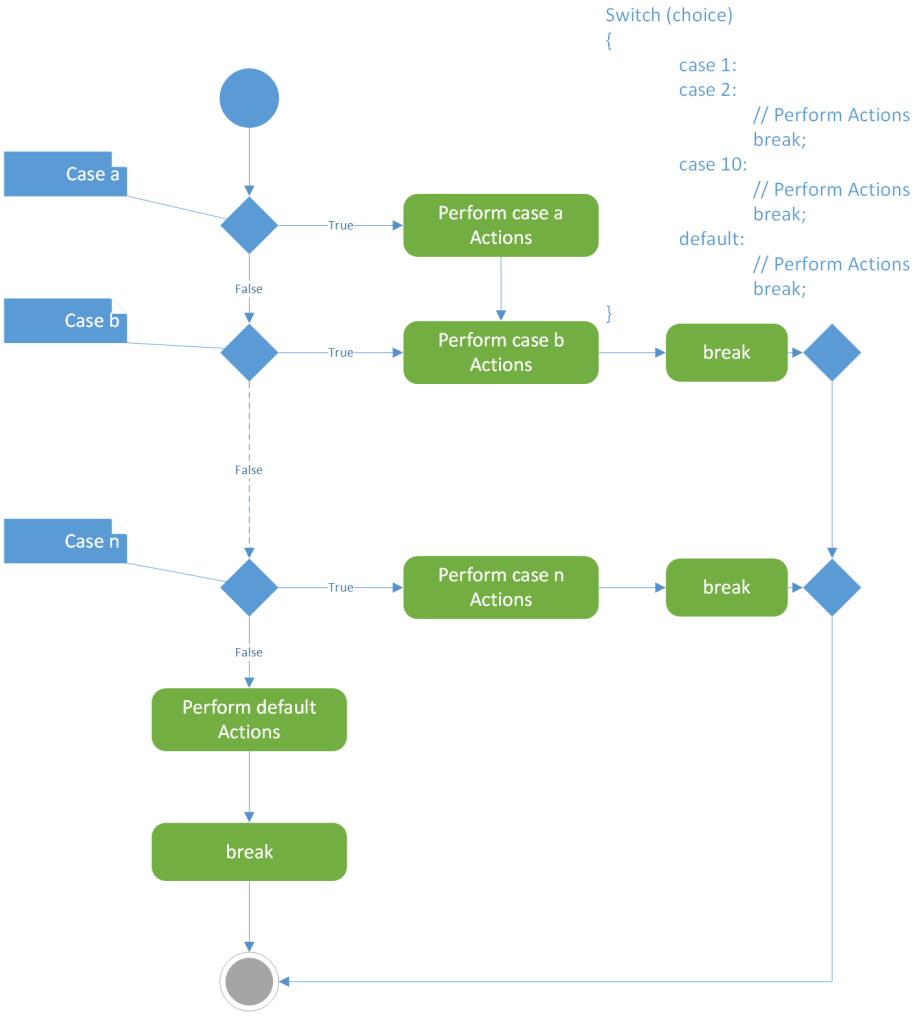

Figure 1.4: UML Activity Diagram for a nested switch statement

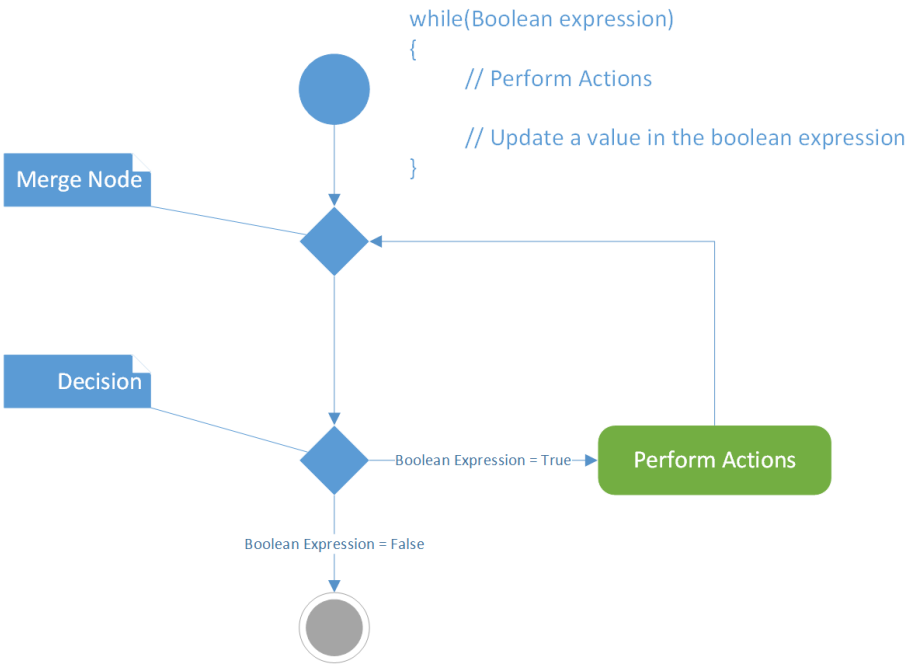

Figure 1.5: UML Activity Diagram for a while Statement

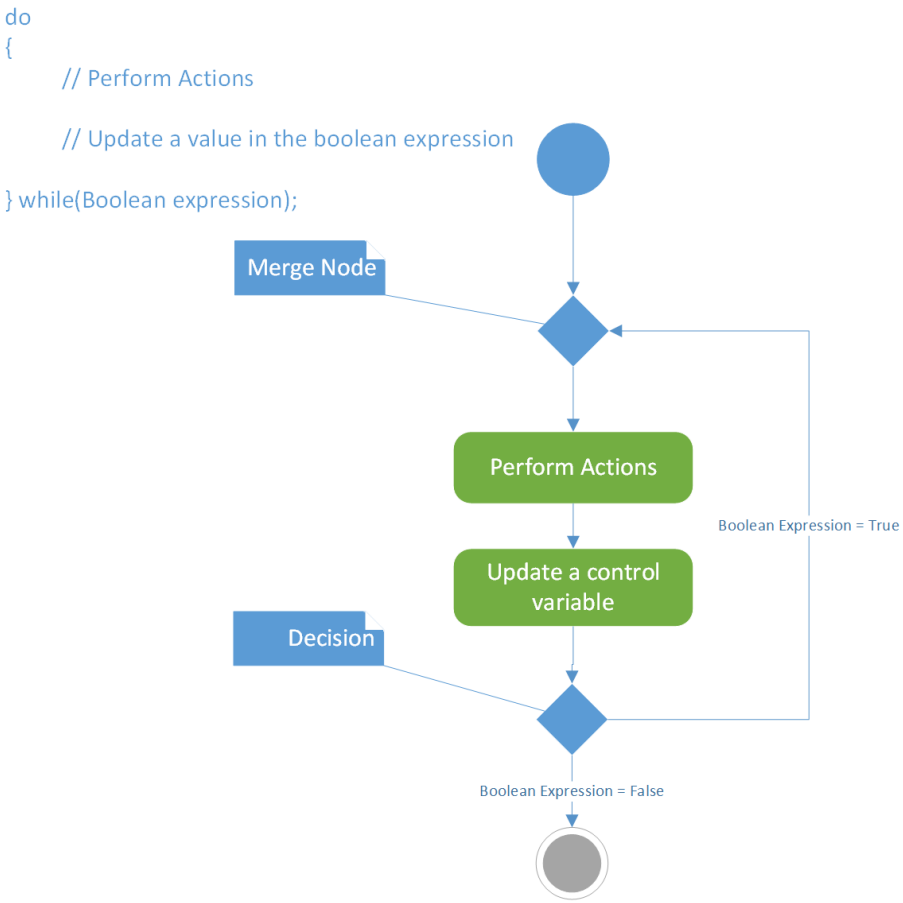

Figure 1.6: UML Activity Diagram for a do..while Statement

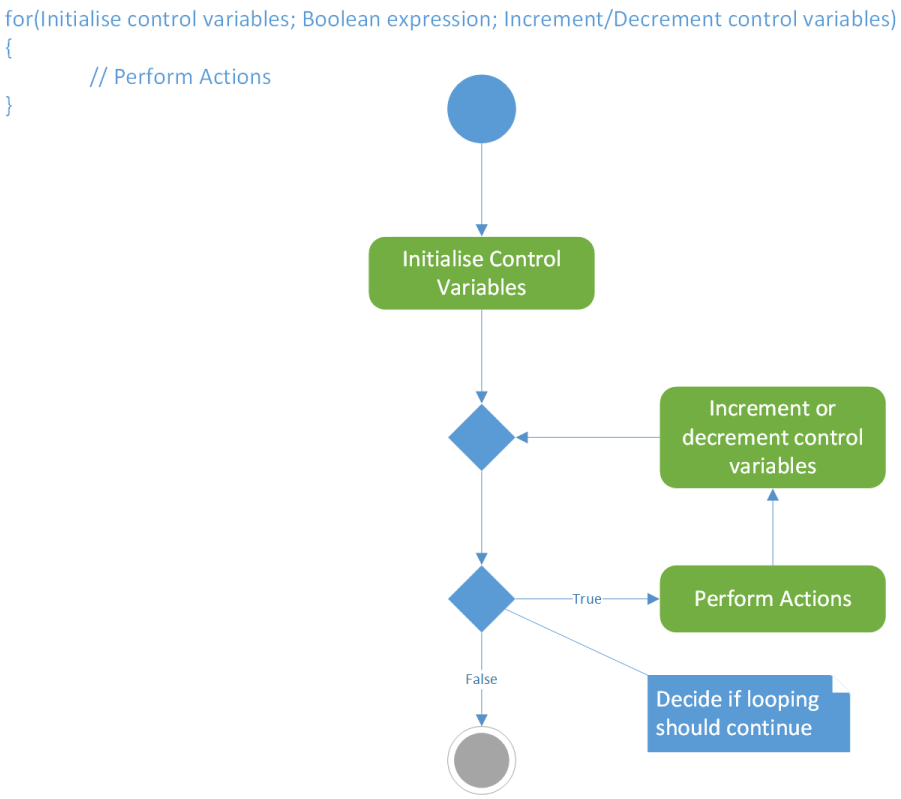

Figure 1.7: UML Activity Diagram for a for Statement

1.2.2 Software Development Lifecycle

1.2.3 Intellectual Property

1.2.4 Referencing

Chapter 2

Memory and Intorduction to Classes

Chapter 3

Arrays and Collections

3.1 Multi-Dimensional and Jagged Arrays

3.2 Lists

Chapter 4

Objects and Structures

4.1 Classes and Objects

4.2 Members

4.3 Structures

4.4 Methods and Properties

Chapter 5

Algorithms

Chapter 6

GUI and more on Classes

6.1 Introduction to access modifiers (Public and Private)

6.2 Overriding

6.3 User Interface Design Guidelines

6.3.1 General

6.3.2 Windows Forms Specific

6.4 Forms

6.4.1 Add controls in a form

6.4.2 Designer

6.4.3 Controls Types

6.4.4 Programmatically controlling them

Chapter 7

Testing and Data Validation

7.1 Introduction to Test Driven Development

7.2 Introduction to Unit Testing

Chapter 8

File System

8.1 Access files, directories and drives

8.2 Create files and directories

8.3 Write a simple

8.4 Introduction to object relationship