

# James Holmson Report

*by* James Holmson

---

|                |                                                                   |                 |       |
|----------------|-------------------------------------------------------------------|-----------------|-------|
| FILE           | 1004_JAMES_HOLMSON_JAMES_HOLMSON_REPORT_93264_188308253<br>5.DOCX |                 |       |
| TIME SUBMITTED | 23-MAR-2018 05:01PM (UTC+0000)                                    | WORD COUNT      | 10649 |
| SUBMISSION ID  | 85474284                                                          | CHARACTER COUNT | 64934 |

James Holmson

P150037

Application Design and  
Development

Mike Dacey

23/03/2018

## Table of Contents

|                                                           |           |
|-----------------------------------------------------------|-----------|
| <b>Table of Figures .....</b>                             | <b>2</b>  |
| <b>Introduction .....</b>                                 | <b>3</b>  |
| <b>Object-Oriented Programming Principles.....</b>        | <b>3</b>  |
| a. The Difference Between a Class and an Object.....      | 3         |
| b. The Implications of Inheritance and Polymorphism ..... | 7         |
| c. The Use of Interfaces vs Classes .....                 | 11        |
| d. The Dependency-Inversion Principle.....                | 13        |
| OOP Principles Conclusion.....                            | 16        |
| <b>Program Testing.....</b>                               | <b>16</b> |
| <b>Programming Exercises .....</b>                        | <b>19</b> |
| Exercise 1 – Spoon Machine.....                           | 19        |
| Exercise 2 – Overlapping Circles .....                    | 22        |
| Exercise 3 – Shopping Cart .....                          | 25        |
| Exercise 4 – Word Search .....                            | 27        |
| <b>Design .....</b>                                       | <b>37</b> |
| <b>Reference List .....</b>                               | <b>40</b> |
| <b>Bibliography .....</b>                                 | <b>42</b> |
| <b>Appendix A – Code from the exercises.....</b>          | <b>43</b> |
| Exercise 1 .....                                          | 43        |
| Exercise 2 .....                                          | 45        |
| Exercise 3 .....                                          | 47        |
| Exercise 4 .....                                          | 50        |

## Table of Figures

|                                                                                                |           |
|------------------------------------------------------------------------------------------------|-----------|
| <b>Figure 1.</b> The Person class is used to create multiple objects.....                      | <b>3</b>  |
| <b>Figure 2.</b> Data encapsulation visualisation.....                                         | <b>6</b>  |
| <b>Figure 3.</b> The Athlete Class inherited variables and a method from the Person Class..... | <b>7</b>  |
| <b>Figure 4.</b> A breakdown of the SOLID design principles.....                               | <b>13</b> |
| <b>Figure 5.</b> Planning and dividing the problems for exercise one.....                      | <b>19</b> |
| <b>Figure 6.</b> Use case diagram for the application.....                                     | <b>20</b> |
| <b>Figure 7.</b> The plotting coordinates for Exercise 2.....                                  | <b>22</b> |
| <b>Figure 8.</b> Exercise 3 Class diagram.....                                                 | <b>25</b> |
| <b>Figure 9.</b> An overview of the minimum requirements for the sections identified.....      | <b>27</b> |
| <b>Figure 10.</b> Sequence diagram for Testers and Developers Console Application.....         | <b>37</b> |
| <b>Figure 11.</b> Class diagram for the program provided.....                                  | <b>39</b> |

## Introduction

The contents of this report investigate the fundamental elements of object orientated programming. In addition to this, research into the testing of software applications has been carried out, the results of which are applied in the practical exercises and the design of the program provided.

## Object-Oriented Programming Principles

In object orientated programming (OOP), it is established that there are four key components, and McGrath asserts the principle three are Encapsulation, Inheritance and Polymorphism [1]. This section ascertains their significance in software development using classes and objects. An interface can provide extra functionality, and its usage is compared to a class. Finally, a brief overview of design principles is given, before examining the benefits and drawbacks of the Dependency-Inversion principle.

### a. The Difference Between a Class and an Object

It is accepted that the creation of a class has similarities to a blueprint or a template [1]–[4], the contents of which Whitaker [2] informs us are usually “modelled after real-world objects”, for example, a person or a house. These have a specific purpose, which is to categorise the requirements for the class and define which problem it is meant to solve [5]. The class is only used when an object is created, which also has a unique identifier [6]. Many objects can be created from the same template and in figure 1, the objects named Jane, Mark and John are created using the person class.

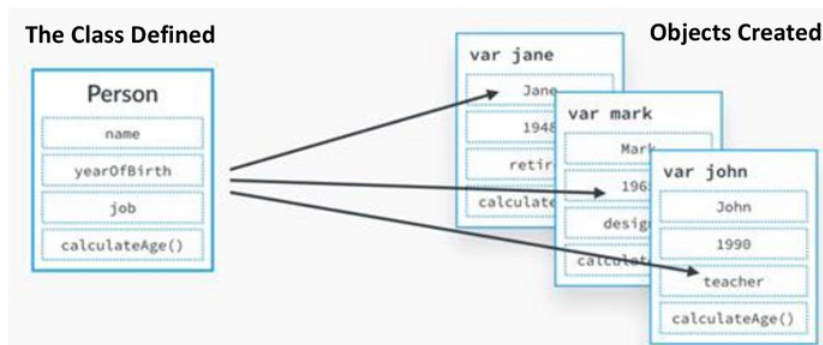


Fig 1. The Person class is used to create multiple objects [6]

Allardice [3] summarises that the “Class is the idea and the objects are the thing itself”, hence the attributes are used to define the characteristics of the class, and behaviours are what the class can do. Whitaker [2] advises that a class be created in a way that makes sense to the programmer, hence modelling them on real-world objects is effective. A class is abstract in the sense that the programmer provides the settings for the four key aspects, these are the attributes, constructors, properties, and methods [3]. Thus, once an object is created, there is an instance of a class, and data can be stored in the variables. The properties can get the values from the variables or set them, whereby they are set initially or changed at a later point. The constructor is a specific method used to set the attributes when an object is created [4]. Finally, the methods complete a particular task using the attributes, related to the class.

Whitaker [2] notes that classes are structured in a way that should make them understandable to anyone who wishes to use them “being in manageable chunks”. Consequently, this assists with the understanding of the construction and maintenance of programs.

The cross-section of code below shows figure 1 the Person class being declared in C#. Each of the attributes has a type and a name. The keyword private is an access modifier; more detail is provided on page 5. The constructor can take arguments to set the attributes at the point of object creation, or they can be stipulated, as seen in the current year attribute. However, there is never a return type, and the constructors are declared using the name of the class.

```
class Person
{
    //Attributes
    private string name;
    private int yearOfBirth;
    private string job;
    private int currentYear;

    //Constructors
    public Person(string name, int yearOfBirth, string job)
    {
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.job = job;
        currentYear = 2018;
    }
}
```

Only the get and set properties of the job attribute are shown here; there is a shorthand version of this code which uses the `get` and `set` keywords. The outcome is the same, get has the same return type as the attribute and is used to retrieve the value of a specific attribute of an object.

```
//Properties
public string GetJob()
{
    return job;
}
```

The set property takes in an argument value of the same data type, which when used will set the value of the object that uses it.

```
public void SetJob(string value)
{
    job = value;
}

//Methods
public int calculateAge()
{
    return currentYear - yearOfBirth;
}
```

Below is the declaration of three objects as seen in figure 1. The specified details of each attribute are input via the constructor to create an object.

```
Person jane = new Person("Jane", 1948, "retired");
Person mark = new Person("Mark", 1963, "designer");
Person john = new Person("John", 1990, "teacher");
```

To use these objects the name identifier is required, and the names of the property or method with any arguments matching the data types. This process is seen below the three objects now use the properties and method declared earlier.

```
jane.GetName(), jane.GetYearOfBirth(), jane.calculateAge(), jane.GetJob();
mark.GetName(), mark.GetYearOfBirth(), mark.calculateAge(), mark.GetJob();
john.GetName(), john.GetYearOfBirth(), john.calculateAge(), john.GetJob();
```

From the code above a standard message to the console, with each of the objects unique attributes being displayed.

```
Hello Jane you were born in 1948 and you are 70 years old and your current job is retired
Hello Mark you were born in 1963 and you are 55 years old and your current job is designer
Hello John you were born in 1990 and you are 28 years old and your current job is teacher
```

As previously mentioned, marking attributes and properties with keywords give programmers the ability to control access to the members, this is known as encapsulation[2]. Whitaker describes this as keeping everything “that the class is responsible for safe and protected inside of the class”. Allardice [3] examines this in his infographic figure 2, whereby the properties and methods are boxed up. He describes these two as being the data and the code to be used on this data.

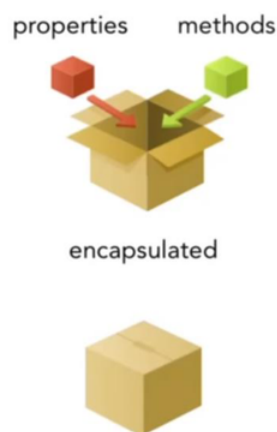


**Fig 2.** Data encapsulation visualisation [3].

In C# the members of a **class** are denoted by keywords such as **public**, **private**, **protected** and **internal** [4]. It is advisable that the attributes or variables use the **private** keyword, in doing so, access is prevented. The properties methods and constructors, usually adopt the **public** keyword, although there are circumstances where other keywords are applied. Whitaker [2] investigated why this is necessary and found that there is protection from outside interference and requests to change the values of an attribute is negated.

In summary, a class is a user-defined type, and the programmer is setting the members for the template. Then when this type is required, the programmer creates an object, and all the attributes and behaviours can be used. The benefits include a reduction in code repetition using the same pattern many times. Programs should be more understandable and maintainable for the programmer and attributes are protected by limiting access by using encapsulation. Part b.explores expanding

classes and how this also relates to a real-world object and harnessing additional functionality using abstraction.

#### b. The Implications of Inheritance and Polymorphism

Inheritance and polymorphism are two of the pillars of OOP and are recognised as being related [7]. Whitaker [2] explains that the application of inheritance is to “Reuse classes by expanding them into more specific types of classes”. The attributes and behaviours of these will be inherited from the parent, or base class to any child or subclass. Sharp [8] informs us that when implementing inheritance, the programmer is outlining the relationship between classes. In figure 3 Schmedtmann expands upon the person class in figure 1, with a specific type of Person: The Athlete class. The parent class has generalised the attributes and methods whereas the child class has attributes and methods that make it specialised. The athlete class has unique attributes and methods, however, those of a person still apply [7]. Therefore, they are not recreated, as the commonalities from its parent class are available. Subsequently, unnecessary code repetition is avoided, as there is no need to rebuild a class.

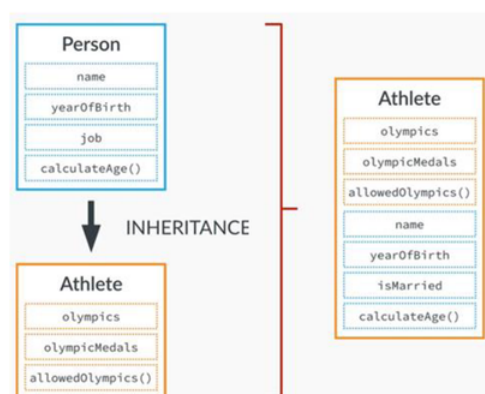


**Fig 3.** The Athlete Class inherited variables and a method from the Person Class [6]

The etymology of the term polymorphism McGrath [1] originates from the Greek and means, “many forms”. Whitaker [2] states it allows any child or derived classes to use methods “to do the same task in their own custom way” [1]. Therefore, the classes and the objects that derive from them are more flexible, inasmuch that each class that inherits from the base class can have unique methods.

When implementing polymorphism in C#, there are several ways to do so.

The keyword `virtual` is added to the method in the parent class, with the keyword `override` added to the child class. The word virtual is in effect granting permission to the child class to overwrite a method in the parent class. In doing so, it now has a unique implementation of the method, and it can be applied to many derived classes [2].

The class can be made `abstract` by using the keyword methods can be marked abstract as well, no code is required in the method body as the programmer only wishes to use the implementation of the child classes.

If the class is marked as abstract, then objects of this type cannot be created as seen below an error will be thrown [4].

```
abstract class Person
{
    abstract public void SayHello();
}

Person bob = new Person(); //error produced
```

However, if a class inherits from person the code below is acceptable.

```
Person bob = new Firefighter();
```

The behaviour of the method is polymorphic as there are many different types of person class, thanks to the many classes that derive from it. Thus, the method can stay empty, as it relies wholly on the subclass's implementation of the method.

On the following pages, two specific problems are analysed, both with and without using inheritance and polymorphism.

In this example program, there is a single class, that uses an: `if else` statement to determine how much an employee's salary will be inclusive of the bonus.

```
class Employee
{
    //Attributes
    private string name;
    private string jobtitle;
    private double salary;

    //Constructor
    public Employee(string name,
                    string jobtitle, double salary)
    {
        this.name = name;
        this.jobtitle = jobtitle;
        this.salary = salary;
    }
    //Methods
    public double CalculateBonus()
    {
        if (jobtitle == "manager")
        {
            return salary * 1.3;
        }
        else if(jobtitle == "programmer")
        {
            return salary * 1.2;
        }
        else if(jobtitle == "tester")
        {
            return salary * 1.1;
        }
        else
        {
            return salary;
        }
    }
}
```

```
//new array of employee objects
Employee[] employees = new Employee[3];

//Object creation

employees[0] = new Employee("Fred", "manager",
45000);
employees[1] = new Employee("Bob", "programmer",
30000);
employees[2] = new Employee("Dave", "tester",
32000);

//Calculating each employees bonus and
displaying this to the console
{
    Console.WriteLine("{0}'s salary including the bonus
is {1}", employees[0].GetName(),
employees[0].CalculateBonus());
}
```

Results of the program

```
Fred's salary including the bonus is 58500
Bob's salary including the bonus is 36000
Dave's salary including the bonus is 35200
```

Instead of the previous solution inheritance and polymorphism could be used in the example below. The array of employee objects is now set to the specific type these child classes inherit from the employee class; these are the programmer, software testers and managers, each of which has unique elements, such as a Christmas bonus.

```
//Class Declaration
class Employee//Parent Class
{
    private string name;
    private double salary;

    public Employee(string name, double salary)
    {
        this.name = name;
        this.salary = salary;
    }
    //Original method can be overwritten
    public virtual double CalculateBonus()
    {
        return salary;
    }
}

class Programmer : Employee
{
    private double ontimepayment; //Unique attribute
    //Constructor also allows for the setting of parent
    class attributes

    public Programmer(string name, double salary,
    double ontimepayment) : base(name, salary)
    {
        this.ontimepayment = ontimepayment;
    }
    //The programmer class implementation of the method
    public override double CalculateBonus()
    {
        return GetSalary() * 1.2 + (ontimepayment + 150);
    }
}

class Tester : Employee //Child Class
{
    private double xmaspayment; //Unique attribute
    //Constructor not shown.
    //Implementation of the method from the parent class
    public override double CalculateBonus()
    {
        return GetSalary() * 1.1 + xmaspayment;
    }
}

class Manager : Employee //Child Class
{
    private double performancepayment; //Unique attribute
    //Unique implementation of the method
    public override double CalculateBonus()
    {
        return GetSalary() * 1.3 + (performancepayment * 2);
    }
}
```

```
Employee[] employees = new Employee[6];

employees[0] = new Manager("Fred", 45000, 1000);
employees[1] = new Programmer("Bob", 30000, 200);
employees[2] = new Tester("Dave", 32000, 400);
employees[3] = new Programmer("Jim", 37000, 750);
employees[4] = new Tester("Steve", 29000, 200);
employees[5] = new Programmer("Jon", 29500, 500);

foreach (Employee emp in employees)
{
    Console.WriteLine("{0}'s salary including the bonus
    is {1}", emp.GetName(), emp.CalculateBonus());
}
```

```
Fred's salary including the bonus is 60500
Bob's salary including the bonus is 36350
Dave's salary including the bonus is 35600
James's salary including the bonus is 45300
Steven's salary including the bonus is 32100
Jason's salary including the bonus is 36050
```

With this solution, there is no need to go through if else statements as it automatically determines the appropriate overriding method to execute. Adding a new type of employee is a straightforward process, with the ability to use any unique attributes. Any changes made to the overriding method is applied to all objects, which is pertinent as the classes will likely have more complex implementations than those shown.

The disadvantages of inheritance are considered by Gamma et al. [9] who note that "it breaks encapsulation" inasmuch that usually they define part of the child class, and when alterations are applied to the parent class, it forces the child class to change. Thus, the dependency means any changes made must be suitable for the child class. However, Gamma et al. [9] state that inheriting from abstract classes will overcome this problem, due to "providing little or no implementation", ergo each subclass will not be reliant on the base class. The programming language C# allows classes to inherit from a single parent Class, which could be seen as a drawback. However, Whitaker(p.159) [2] does summarise that in this situation, the intuitiveness does diminish due to increased complexity.

To summarise, inheritance facilitates in reducing code repetition enabling reusability across parent and child classes. Additionally, it assists when planning applications as there is a hierarchy between classes. It is therefore established that polymorphism complements inheritance overriding a method with a unique application.

### c. The Use of Interfaces vs Classes

An interface is described by Joseph and Ben Albahari(p.83)[10] as being a "specification rather than an implementation for its members". The specification is initially given a name, and the members go within, which are either methods or properties. The return type and the arguments and their data types are also specified. The declaration of an interface is abstract, and there is no code within the method body, as seen below.

```
interface INumbers
{
    int NumberCalculator();
}
```

The interface can be applied to any class where it is required or given the contract [2] as seen below with the Person class example.

```
class Person : INumbers
{
    public int NumberCalculator()
    {
        return currentYear - yearOfBirth;
    }
}
```

The Dog class is also given the contract although its application is unique.

```
class Dog : INumbers

public int NumberCalculator()
{
    return _humanyears * 7;
}
```

Thus, the implementation is within the classes themselves, which have been given this capability. There is a similar functionality between the two which in this example involves returning an integer. With regard to whether there is a preference to use an interface over a class, Córdoba [11] informs us that programmers could abide by his basic rule:

“Use abstract classes and inheritance if you can make the statement “A is a B”. Use interfaces if you can make the statement “A is capable of [doing]”.



He implies that using a class lends itself to where inheritance is used, and coincidental similarities exist between a base and subclass. Interfaces have self-imposed limitations; they do not need to know about other classes and their members. However, they enable robust functionality, which is sharable across classes. Córdoba [11] concedes that there are situations where his statement may not be applicable, and common sense is required. For example, a problem may be too complicated for an interface, or it may be easier to solve smaller problems using multiple interfaces. The way interfaces are declared can follow this approach of breaking down programs into a series of small problems and helping programmers develop the application. Sharp adds that in doing so “you can truly separate the ‘what’ from the ‘how’[8]. Although perhaps it is more effective to decide upon the ‘how’ or the implementation and then define the ‘what’ after this. Justin [12] declares in doing, so it aids the clarity of the code due to the “self-documenting” application of interfaces.

Interfaces can negate the problem identified in section b, regarding a class only being able to inherit from a single parent class [2] as a single class can have many interfaces.

The restrictions are investigated by Sharp who notes that because the application is with the class itself, the use of fields constructors is not permitted. In addition to which it is assumed that methods within the interface will be public [8].

Deciding on the proper utilisation of an interface and a class is situational, and experience will aid the decision-making process. Although, both work well together especially when different classes need the ability to perform the same overlapping function in their own way.

#### d. The Dependency-Inversion Principle

The Dependency Inversion Principle or DIP is one of the five SOLID principles, with each of the five concepts being listed in figure 4.

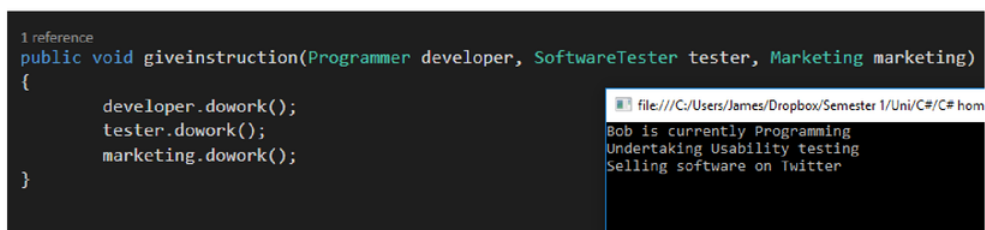
|          |                                 |
|----------|---------------------------------|
| <b>S</b> | Single responsibility principle |
| <b>O</b> | Open/Closed principle           |
| <b>L</b> | Liskov substitution principle   |
| <b>I</b> | Interface segregation principle |
| <b>D</b> | Dependency inversion principle  |

**Fig 4.** A breakdown of the SOLID design principles.

These principles are not rules, and Martin [13] explains how they should be seen as “good advice”, describing how they provide “common-sense solutions to common problems”. Although there may be instances where they are not suitable to solve a problem, he informs us that “they have been observed to work in many cases, but there is no proof that they always work, nor any proof that they should always be followed” [13]. The following is the statement which programmers should follow to implement DIP:

“High-level modules should not depend upon low-level modules. Both should depend on abstractions” [14].

Firstly, a high-level module is code that is written to solve a general problem, whereas low-level modules are for specific problems [15]. The primary drawback of not using DIP is that if dependencies are present, any alterations will have to be applied to both high and low-level modules. Martin [14] refers to this as the rigidity of the system, and if the programmer is making alterations in many places whether expected or unexpected, it leads to making the program fragile and potentially results in the application failing. The third point is a lack of flexibility within a system, where parts of the application cannot be reused because of the dependencies that exist. The solution is in the statement, which informs us that abstraction is to be used. The example below will be set out to explore the drawbacks of not using DIP. Each type of staff member inherits from an abstract employee class then has the unique implementation of the dowork function. The manager class has a method that passes in three team members of the three different classes, and their unique application is seen below.



The screenshot shows a code editor with a C# method call. The code is as follows:

```
1 reference
public void giveinstruction(Programmer developer, SoftwareTester tester, Marketing marketing)
{
    developer.dowork();
    tester.dowork();
    marketing.dowork();
}
```

On the right side of the editor, there is a console window showing the output of the program:

```
file:///C:/Users/James/Dropbox/Semester 1/Uni/C#/C# hom
Bob is currently Programming
Undertaking Usability testing
Selling software on Twitter
```

While the program functions, it is tightly coupled and reliant on other classes, in the manager class. To process the method call, above it must have the three different objects of the specified class types. Also, it is closed for extension should they have to manage a new type of employee or manage a team of five both of which would have to be added to the code.

Now the DIP will be applied, this time an interface is applied to all the classes that use it.

```
interface IStaff
{
    void dowork();
}
```

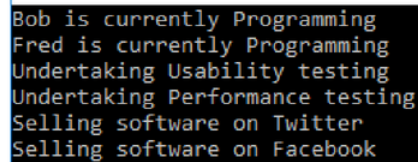
The manager class has a list that stores any objects that are given the `IStaff` contract:

```
//Attributes
private List<IStaff> _team;

//Constructor
public Manager()
{
    _team = new List<IStaff>(); //every object will have this list
}
```

It uncouples the Manager class from the different types of employees, also removing the self-imposed limits requiring one of each object. If any new types of staff are added, then as long as it implements the interface, the method below will still work if the manager has them on their team.

```
//Methods
public void manage()
{
    foreach(IStaff members in _team)
    {
        members.dowork();
    }
}
```



```
Bob is currently Programming
Fred is currently Programming
Undertaking Usability testing
Undertaking Performance testing
Selling software on Twitter
Selling software on Facebook
```

The usage of DIP is not without its drawbacks; it can become difficult to understand or follow the purpose of the code due to the more abstract nature and a lack of experience potentially exacerbating this further. Fowler confirms this [16] and states that “it leads to problems when you are trying to debug”.

To conclude when using the DIP, the frailties in programs can be reduced; this is a decisive factor when developing applications. Martin recognised that the likelihood of

further difficulties being encountered with maintenance and expandability would be increased if ignored. It warrants an investigation into the other design principles, as Martin made clear they assist with problems that regularly occur in software applications. However, for this reason, novice developers should be cautious in the application of these principles until they have the required knowledge.

### **OOP Principles Conclusion**

Object Orientated programming is reliant upon the four pillars [2], and together they enable many overlapping benefits such as a reduction in code repetition. In doing so, it leads to more robust systems that are relevant throughout the whole cycle in the development of a software application.

### **Program Testing**

The testing phase of programming is essential for creating resilient software that meets a client's expectations [17]. The requirements are provided by the customer or stakeholder, although without obtaining these a project is unlikely to have a satisfactory outcome [18]. Zafar [19] confirms that without testing, there is no validation or verification of the software, which will harm customer satisfaction.

Boehm and Papaccio inform us that the cost of fixing an error in the early stages of development can promote savings by a factor of 50 to 200 times [20]. Thus, these facts highlight a strong case for enforcing stringent software testing. Zafar defines the process as being an evaluation of the elements of the program with the objective being "to detect differences between given input and expected output" [19]. When devising tests, it is vital to include those designed to make the application fail and not just perform tests that handle the expected input [21] (test to pass, test to fail). The boundaries of a program should be tested. For example, if a program can accept a minimum temperature of -5 and a maximum of 50, testers should input values just above and below these figures to confirm proper functionality.

It has been established that testing usually falls into two categories, these are Black Box Testing and White Box Testing [19]. Greg Fournier (p.9-10) defines each as:

“Black box testing refers to testing a software item without knowing anything about its inner workings”. “White box testing focuses on the internal structure of the system under test.”

The table below [19] presents Zafar’s list that includes many areas of application testing and development.

| Categorising the types of testing [19]                                          |                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| White Box                                                                       | Grey Box                                                                                              | Black Box                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Unit Testing – This is to ensure the components the developer builds work [18]. | Integration Testing<br>Low-level testing which relates to groups of the components from unit testing. | <b>Functional Testing</b> – test from the viewpoint of the user, data is input to see if it matches the expected output [18].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|                                                                                 |                                                                                                       | <b>Non-Functional</b> – areas the program should achieve specific standards outside of the functional elements. <ul style="list-style-type: none"> <li>• System Testing – security, auditing and error handling [18]</li> <li>• Performance Testing such as the following: <ul style="list-style-type: none"> <li>○ Stress Testing – under extreme circumstances, for example, a significantly higher number of users logging on to buy a product.</li> <li>○ Usability Testing – focused on how the end users find the product [22]</li> <li>○ Acceptance Testing – meeting the user's requirements</li> </ul> </li> </ul> |

The table outlines developers and software testers solely undertake white and grey box testing, it was noted that developers need to be stringent at this stage and not be afraid to break code they have written [7].

Black box testing requires no knowledge focusing on the application behaviours under normal usage, so developers should adopt the mindset of their end-users and try to encompass any mistakes they could make into the test plan [7].

Code can be written to carry out specific tests, Fowler [23] refers to these objects as Test Double, stating that the objects created are for testing rather than the finished production. Martin explains that like a stunt double on a film set, the different types of Test Doubles listed below have a specific task to carry out, thus, standing in, and with the dummy, it will fill-in any parameters in the lists of functions but as Martin informs they are never used [23].

- Dummy
- Fake
- Stubs
- Spies
- Mocks [23]

Over time, many different testing strategies/practices have been developed, a combination of which can be used as required.

Fournier [18] strongly endorses use cases for essential testing, and he provides the following definition of their application:

“Use cases are used to package, at a minimum, the functional requirements of a system. They are described via sequences of interaction between one or more Actors, who represent users, or other systems that interact with the system and the System being specified”. He alludes that by breaking down a program in this way, developers have a good starting point to decide upon tests to run due to understanding formed of the functions of the program and flow in order to meet requirements of the client [18].

Use Stories are also informal and take the form of about three sentences written in the language of users of the system [18].

Agile is a practice commonly adopted in the industry, Fournier explains how this works. The building and integration of new parts of the program are done in short-time spans, with team-collaboration and feedback from customers being critical [18].

Test-driven Development involves writing the tests first then developing the product, although this could seem counter-intuitive McLaughlin et al. [7] inform writing the code should be efficient as developers have devised how the code is going to behave as part of this practice.

To conclude, the standard feature throughout these practices is the planning, as though doing so there is a clear path to follow, which should highlight areas where weaknesses exist. Consequently, faults are rectified and there is confirmation that the application meets the customer's requirements without which it leads to businesses being liable to rectify faults at a high cost to themselves.

## Programming Exercises

Each of the exercises has three stages. Firstly, different strategies are used to plan; these include mind- maps, use cases diagrams and computerised sketches to provide a visual reference point of the elements required. The testing process is established before development, with the actual tests being on-going throughout the exercises. During the development, the author's current programming experience is used to devise a working solution by focusing on where different aspects could be applied using the previous two stages as a reference point. The full code for each exercise is available in Appendix A.

### Exercise 1 – Spoon Machine

The four areas of this program and the plan to create their components are shown in figure 5.

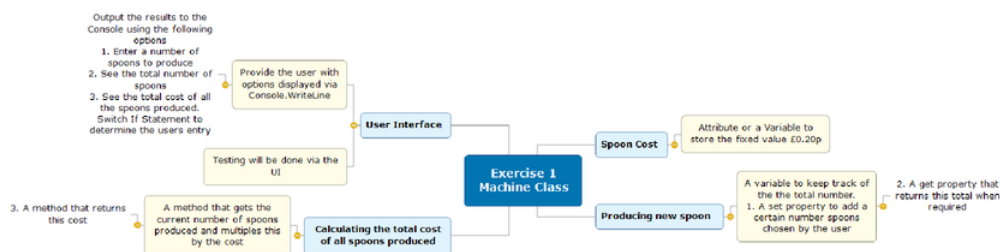


Fig 5. Planning and dividing the problems for exercise one.

This diagram outlines that one class is required, two attributes to store the values and five properties, four to get and set the values with an additional one to get the product spoon. A single method will calculate the cost using the get property and then return the final value. The user interface utilises the object produced from the class to properties and method, which is done via switch statement based on the user's choice.

In figure 6, the interactions between the user and spoon machine were noted.

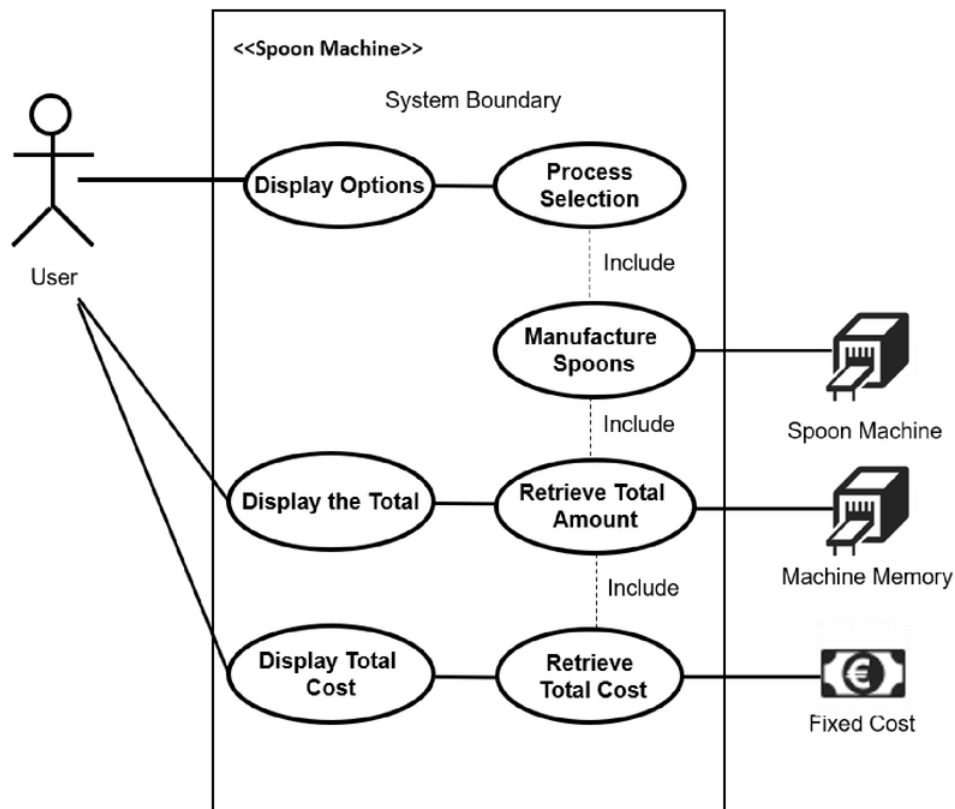


**Fig 6.** Use case diagram for the application

This information is used to form the solution in code by mapping the tasks to current programming knowledge in the table below.

| Task                   | Possible Solution                                                                            |
|------------------------|----------------------------------------------------------------------------------------------|
| Display Options        | <code>Console.WriteLine</code><br>Put into a method to be repeatedly called                  |
| Process Selection      | Switch Statement that controls the choice via<br><code>Console.ReadLine</code>               |
| Manufacture            | <code>+=</code> The amount Set by the user                                                   |
| Total Number of spoons | Get Property                                                                                 |
| Total Cost             | Method Call and attribute                                                                    |
| Machine                | Attributes are set via a Constructor to create and set the initial object (only one machine) |

On the next page is the documentation process for the testing the completed program.

| Testing Documentation Exercise 1                                                                                                                                                                                                                       |                                                                                                                                    |                                                                                                                                           |                                                                                       |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Test                                                                                                                                                                                                                                                   | Outcome                                                                                                                            | Alterations                                                                                                                               | Re-testing                                                                            |
| <b>Main Menu Entry:</b><br><br>Input with fully capitalised letters for each of the menu options. aDD ADD toTal<br><br>Add space to each of the options. For example<br>" add" "add "<br>" add "<br><br>Input numbers both integers and floating point | <b>FAIL</b> - Default message returned<br><br><b>FAIL</b> - Default message returned<br><br><b>PASS</b> – Default message returned | Add the <code>ToLower()</code> function added at point of user entry<br><br>The <code>Trim()</code> function added at point of user entry | <b>PASS</b> – Proceed to<br><br><b>PASS</b> – Proceed to                              |
| <b>Adding spoons and printing the total:</b><br><br>The initial value<br><br>After spoon input integers: 232, 500, 1000, 1000, 7, 1428<br><br>Negative integers                                                                                        | <b>PASS</b> – 1000 printed<br><br><b>PASS</b> - Confirmed the total updated<br><br><b>FAIL</b> – Subtracts from the total          | N/A<br><br>N/A<br><br>< 0 conditional statement added after entry point, adding method in the else                                        | N/A<br><br>N/A<br><br><b>PASS</b> – The message "Number must not be negative" printed |

|                                                    |                                                                                               |                                                               |                                                                              |
|----------------------------------------------------|-----------------------------------------------------------------------------------------------|---------------------------------------------------------------|------------------------------------------------------------------------------|
| Enter floating point numbers 2.99 10.00            | <b>FAIL</b> - Unhandled exception error                                                       | Current solution placed inside a <b>try block</b>             | <b>PASS</b> – The message "Invalid Entry" handles all input                  |
| Entering string values "Two" "One Hundred" "HELLO" | <b>FAIL</b> - Unhandled exception error                                                       | the <b>catch block</b> is used to present an error message    |                                                                              |
| The Quit Option                                    | <b>PASS</b> – Program exits<br><b>FAULT</b> found with UI<br>Printing the menu before exiting | Solution wrapped in if statement else providing a new message | <b>PASS</b> – The message "Machine is powered down Goodbye" No menu is shown |

## Exercise 2 – Overlapping Circles

The location of the following coordinates was marked onto a grid in figure 7: In this exercise.

Circle 1 or **C1** (x6 and y6) and Circle 2 or **C2** is (x1 and y3)

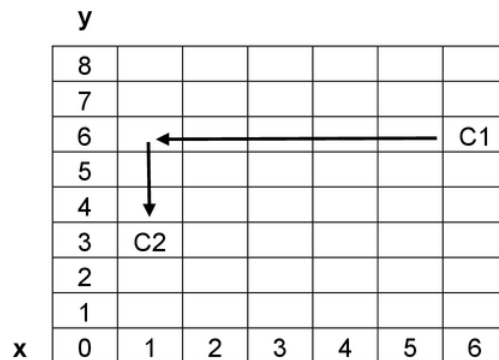


**Fig 7.** The plotting coordinates for Exercise 2.

The distance between each circle's coordinates is calculated  $(x1 - x2)$  and  $(y1 - y2)$ , this is the basis of the total distance.

$$\left. \begin{array}{l} (x1 - x2) \ 6 - 1 = 5 \\ (y1 - y2) \ 6 - 3 = 3 \end{array} \right\} 5 + 3 = 8$$

Each circle has a radius and are added together to give the radii; this is subtracted from the previous answer.

$$\left. \begin{array}{l} \text{C1 Radius} = 3 \\ \text{C2 Radius} = 2 \end{array} \right\} 3 + 2 = 5$$

$$8 - 5 = 3$$

In this instance, the centre point is further away than the sum of the radii, and there is no overlap.

From this it is identified that three attributes are needed:

- X-coordinate
- Y-coordinate
- Radius

The solution was to create the three attributes which would be set by a constructor. Subsequently, two objects were created.

```
Circle circle1 = new Circle(6, 6, 3);  
Circle circle2 = new Circle(1, 3, 2);
```

Confirmation was obtained that the values set were those that were set above, the necessary get properties were created.

```
Circle 1 x = 6 y = 6 r = 3  
Circle 2 x = 1 y = 3 r = 2
```

The next step was to add these values together then compare the results to determine if the two circles are overlapping.

A method was created that accepts two circle objects.

```
public void OverlapTest(Circle circle1, Circle circle2)
```

Using the get properties would get the attributes of the object, the coordinates were subtracted ( $x1 - x2$ ) and ( $y1 - y2$ ) with the results added together. The radius of the two objects was also added together.

With a comparison operator it confirms if the distance between the circle centre points was higher than the radii:

```
total > radii
```

If true, this meant there was no overlap, and if false, it would mean there was an overlap. The two initial tests confirmed a working program for happy path testing.

```
No Overlap  
X = 5 Y = 3 Total = 8 Radii = 5
```

```
Overlap Detected
X = 5 Y = 3 Total = 8 Radii = 11
```

On the next page is the testing documentation from this point onwards. Any erroneous outcomes were examined, and alterations applied to the program to produce an acceptable outcome.

| Testing Documentation Exercise 2                                                                   |                                                                                                                                                                                                                                      |                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Test                                                                                               | Outcome                                                                                                                                                                                                                              | Alterations                                                                                                                                                                   |
| The integer in x2 or y2 is higher in the value of that in x1 and y2 resulting in a negative number | <b>FAIL</b> – False returned despite being an overlap<br><pre>Overlap Detected X = -5 Y = -6 Total = -11 Radii = 2</pre> <b>Re-testing</b><br><pre>No Overlap X = 5 Y = 6 Total = 11 Radii = 2</pre> Correct output message received | Method created with a Conditional statement check if the total is negative from the calculation. If true, this is multiplied by -1 giving the distance between the two points |
| <b>Boundary testing</b>                                                                            |                                                                                                                                                                                                                                      |                                                                                                                                                                               |
| The sum of the distance is the same as the same as the radii 6 and 6                               | <b>PASS</b> – Overlap detected acceptable as on the border<br><pre>Overlap Detected X = 2 Y = 4 Total = 6 Radii = 6</pre>                                                                                                            |                                                                                                                                                                               |
| The radii are +1 -1 of the sum of the coordinates                                                  | <b>PASS</b> – +1 Overlap detected<br><pre>Overlap Detected X = 5 Y = 6 Total = 11 Radii = 12</pre> <b>PASS</b> -1 Result<br><pre>No Overlap X = 5 Y = 6 Total = 11 Radii = 10</pre>                                                  |                                                                                                                                                                               |
| The sum of the distance is +1 -1 the radii                                                         | <b>PASS</b> +1 Result<br><pre>No Overlap X = 5 Y = 6 Total = 11 Radii = 10</pre> <b>PASS</b> -1 Result<br><pre>Overlap Detected X = 8 Y = 1 Total = 9 Radii = 10</pre>                                                               |                                                                                                                                                                               |
| The sum of the distance is 0 radii                                                                 | <b>PASS</b> –No overlap and overlap                                                                                                                                                                                                  |                                                                                                                                                                               |

|                      |                                                     |  |
|----------------------|-----------------------------------------------------|--|
| 1 the swapped around | No Overlap<br>X = 1 Y = 0 Total = 1 Radii = 0       |  |
|                      | Overlap Detected<br>X = 0 Y = 0 Total = 0 Radii = 1 |  |

### Exercise 3 – Shopping Cart

The prerequisites inform that each item for sale should be in a separate class; the common attribute between the products is the price. Therefore, there would be a Product class with the CD, DVD and Book classes using inheritance. An additional class to work out the total price would be used to return the final values to the user, by passing in an array of Product objects of Type CD, DVD and Book. In figure 8, a class diagram was created to distinguish the attributes, properties, and methods also where abstraction was required.

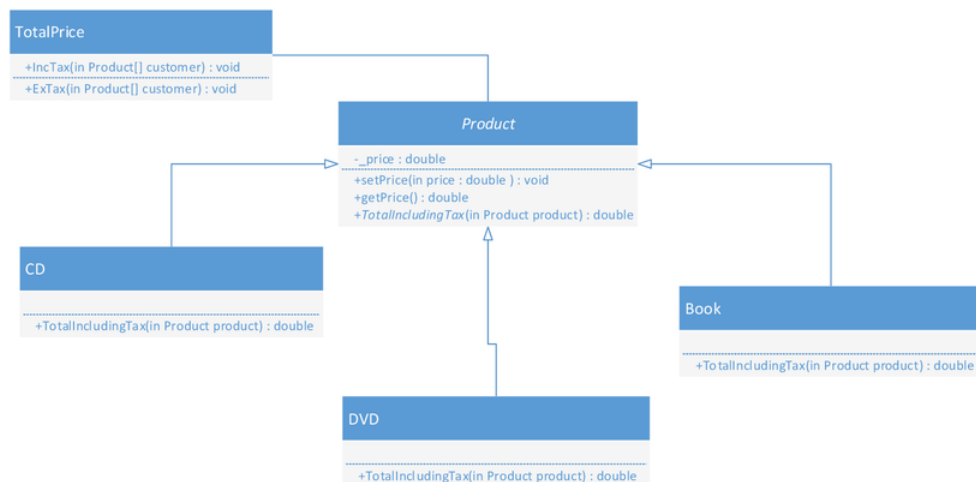


**Fig 8.** Exercise 3 Class diagram

From this diagram, the following was ascertained:

- Every product has a price, although this is different.
- Each item would be created using the following syntax:  
Product = new CD(), Product = new DVD() Product = new Book().
- The Product is the base class, and CD, DVD, Book are the subclasses.

- The tax amount varies for the different classes and abstraction was used.
- Two methods for total calculation one using getPrice() and one using overriding method TotalincludingTax().

From the class diagram and notes, the tests were written, after which each section was writing in code whereby each test was actioned to develop the program recording the test developing each area as necessary and applying any alterations.

| Planned Test                                                                  | Outcome                                                                                                                                                                                    | Alterations                                                                                                                                                                  |
|-------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. User Interface Test                                                        | A message displayed informing the user of the available options. The user specifies the many items they wish to purchase this number is used for the size of an array of Product objects.  | A for loop was applied to perform this action the duration of which is set from the input from the number of products                                                        |
| 2. Product Selection                                                          | Console Message informs the user of option and the switch statement within the loop to choose a product. It steps through each case successfully<br><br>The default case reveals a problem | ToUpper() is applied to match user input with the case<br><br>Within the default case, it now reduces the count by one to allow the correct number of products to be entered |
| 3. Price for the product entry                                                | Entered and saved to the variable outcome printed                                                                                                                                          | Changed data type from int to double                                                                                                                                         |
| 4. Saving an item                                                             | From 2. The CD DVD or Book is selected and saved to the array of Product objects using the using the counter of the for loop to determine the position to save in                          |                                                                                                                                                                              |
| 5. Saving the price for this item                                             | The price is now passed into the setPrice property again using the count enforcing correct product in the array                                                                            |                                                                                                                                                                              |
| 6. Exiting the loop and displaying the total both including and excluding tax | Test the overriding method for adding Tax for each product                                                                                                                                 |                                                                                                                                                                              |

## Exercise 4 – Word Search

There are three distinct parts to this exercise; comprising the words, the players, and the gameboard, the three of which are interlinked. In figure 9, these areas are expanded to provide visualisation of the finished design.

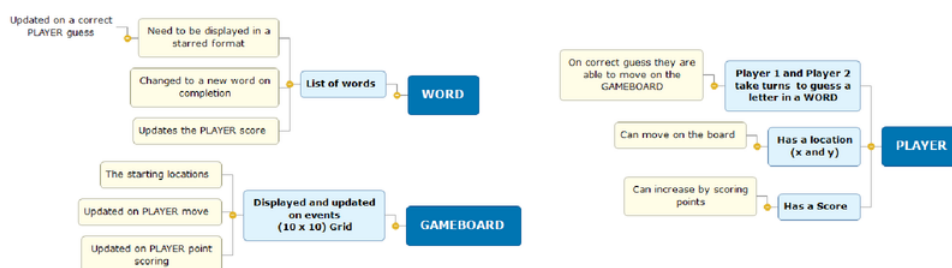


**Fig 9.** An overview of the minimum requirements for the sections identified.

From this initial stage it confirmed the Word, Player and Gameboard would be the classes, the Players are reliant on the Board, and consequently the former would be the sub-class with the latter being the base class.

The game is reliant upon guessing a letter, as correct entry provides the players with moves around the board. A use case lists the flow of the process below.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>User Cases Name:</b> Guessing a Letter                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Stakeholders and Interests:</b> Players                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Brief Description:</b><br>Player input a letter that is present in the word that is in-play each correct guess they are given a move on the board and the letter is revealed in a starred-out version of the word.                                                                                                                                                                                                                                                 |
| <b>Trigger:</b> Player inputs a letter                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Normal Flow:</b> <ol style="list-style-type: none"> <li>1. User inputs letter</li> <li>2. This input is checked if it is in the word             <div style="margin-left: 40px;">               If the letter is in the current word in play, the letter is displayed in the starred-out word, and the S1 player is given a chance to move there.<br/>               If the letter is not present, it is the next player's turn S2.             </div> </li> </ol> |
| <b>Subflows:</b><br><b>S-1:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                       |

1. The player enters the direction they wish to move their piece on the board.
2. This process repeats albeit for the next player

Initially, there will be ten preset words saved to an array, starting from array location zero, each correct word would lead to this being increased by one. The ToCharArray() Method is used on this array location to store the word as individual letters two char array are required. Firstly to store the word to check against on user entry, and secondly to display the word as stars, where letters can be revealed on the entry of a character that exists in the word.

|                                      |   |   |   |   |   |   |   |   |   |
|--------------------------------------|---|---|---|---|---|---|---|---|---|
| <b>The Word to Guess</b>             | C | H | R | I | S | T | M | A | S |
| <b>Starred Out In the char array</b> | * | * | * | * | * | * | * | * | * |
| <b>Array Location</b>                | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

The console then displays the word to guess as below.

Word: \*\*\*\*\*

Upon entry of a letter A for loop goes through each letter of the guess word comparing the entry against each letter, if it finds a match it then reveals this in the starred out version, this links to play moves the flow of which is on page ?.

Correct Word: C\*\*\*\*\*

The next problem was changing the word once every letter has been revealed; this was examined in the use cases.

|                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>User Cases Name:</b> All letters in the in-play word have been guessed                                                                                                                                                                              |
| <b>Stakeholders and Interests:</b><br>A list of pre-set words                                                                                                                                                                                          |
| <b>Brief Description:</b><br>The new word is presented to the user, which they guess until wholly revealed and the process is repeated.                                                                                                                |
| <b>Trigger:</b><br>After the last letter in the word is guessed in the word.                                                                                                                                                                           |
| <b>Normal Flow:</b> <ol style="list-style-type: none"><li>1. New word in the list is selected.</li><li>2. It is displayed on the screen in the starred format</li><li>3. The user guesses are correctly processed until the word is revealed</li></ol> |
| <b>Subflows:</b><br><b>S-1:</b>                                                                                                                                                                                                                        |
| <b>Alternate/Exceptional flows:</b><br><b>Player inputs an</b>                                                                                                                                                                                         |

The word in the array is changed via a function call, which is made to add one to a counter to the increment it by one in the array containing all the words. Using the length ToCharArray is used twice to set the new word which users guess against for comparison, and to the Char array used to star the word, this would have star stars reapplied.

```
Word: CHRISTM*S
choose next letter >a
WORD FOUND!!
Word: ****
```

The next problem would be to display the game board, and as this is a console application, a multidimensional array is an appropriate way of creating a two-dimensional (2D) game board. Using console.write and two counting loops seen below the grid was created, the locations needed to access the locations within the multidimensional array were printed to the console to assist with overcoming future problems relating to the grid.

```
0,9 1,9 2,9 3,9 4,9 5,9 6,9 7,9 8,9 9,9
0,8 1,8 2,8 3,8 4,8 5,8 6,8 7,8 8,8 9,8
0,7 1,7 2,7 3,7 4,7 5,7 6,7 7,7 8,7 9,7
0,6 1,6 2,6 3,6 4,6 5,6 6,6 7,6 8,6 9,6
0,5 1,5 2,5 3,5 4,5 5,5 6,5 7,5 8,5 9,5
0,4 1,4 2,4 3,4 4,4 5,4 6,4 7,4 8,4 9,4
0,3 1,3 2,3 3,3 4,3 5,3 6,3 7,3 8,3 9,3
0,2 1,2 2,2 3,2 4,2 5,2 6,2 7,2 8,2 9,2
0,1 1,1 2,1 3,1 4,1 5,1 6,1 7,1 8,1 9,1
0,0 1,0 2,0 3,0 4,0 5,0 6,0 7,0 8,0 9,0
```

From this point the two player pieces (1, 2), the point scoring coin (\$) and dashes to represent (-) empty grid locations were input into the array to produce the following starting locations:

```
- - - - -
- - - - -
- - - - 2
- - - - -
- - - $ -
- - - - -
1 - - - -
- - - - -
```

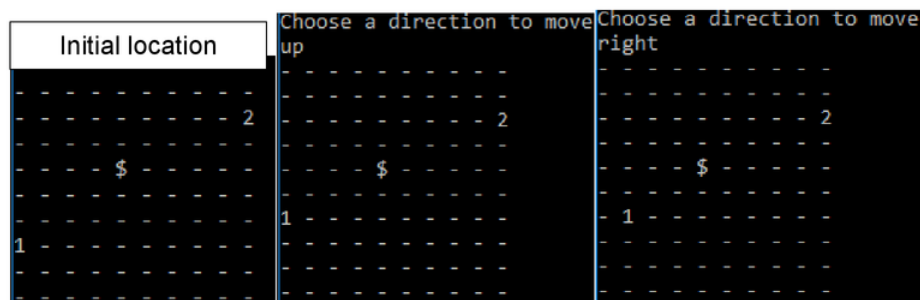
When the initial game board displayed, it would have to be updated between player moves and on point scoring.

|                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>User Cases Name:</b> Move a Piece                                                                                                                                                                                                                                                                                                                         |
| <b>Stakeholders and Interests:</b><br>Players                                                                                                                                                                                                                                                                                                                |
| <b>Brief Description:</b><br>On a correct guess, the player can move their piece on the 2D gameplay board either "up, down, left, right."                                                                                                                                                                                                                    |
| <b>Trigger:</b> Every correct guess of a letter                                                                                                                                                                                                                                                                                                              |
| <b>Normal Flow:</b> <ol style="list-style-type: none"> <li>1. The user enters the command to move their piece to the desired location.</li> <li>2. The piece now appears in the new location.</li> </ol> <p>If the piece is in a location where the move requested is not permitted.<br/>The player arrives at a location which is occupied by the coin.</p> |
| <b>Subflows:</b><br><b>S-1:</b> <ol style="list-style-type: none"> <li>1. The piece is not moved, the user is informed</li> <li>2. Point Score use case</li> </ol>                                                                                                                                                                                           |
| <b>Alternate/Exceptional flows:</b><br><b>Player inputs an</b>                                                                                                                                                                                                                                                                                               |

It was surmised that by continually keeping track of each where a piece is, by storing their x and y coordinates, all that would be needed would be to add or subtract one to these figures depending on the direction of movement. A '-' symbol would be recorded at the present position with the piece moved to the location of choice using the calculation on the coordinates. A cross-section was examined to determine this.

|             |                                |
|-------------|--------------------------------|
| 0,9 1,9 2,9 | + 1 for Up or -1 for Down      |
| 0,8 1,8 2,8 |                                |
| 0,7 1,7 2,7 | Applied to the leading number  |
| 0,6 1,6 2,6 |                                |
| 0,5 1,5 2,5 |                                |
| 0,4 1,4 2,4 |                                |
| 0,3 1,3 2,3 | -1 for left or +1 for Right +1 |
| 0,2 1,2 2,2 |                                |
| 0,1 1,1 2,1 | Applied to the secondary digit |
| 0,0 1,0 2,0 |                                |

This would have to be joined to where the letter is guessed, however, just using player 1s piece the locations were entered and moved around the board.



It was highlighted that in the sub-flow of the previous use case, there are instances when users could get to the edge of the board. At this point, if they enter a move that takes their piece off the board, it would crash the game as the multidimensional array size has already been set. Thus, it would attempt and write a piece to an array location that does not exist. By adding a conditional statement on the move event at the locations where this applies, it removes the option to from the player. Firstly, the board was examined below to observe at which array locations the bans would be imposed.

|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|-----------------------------------------------------|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| Secondary digit is 9 - The move UP to be disabled   |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
| Leading Digit is 0 -<br>The move LEFT is disabled   |  | 0,9 1,9 2,9 3,9 4,9 5,9 6,9 7,9 8,9 9,9<br>0,8 1,8 2,8 3,8 4,8 5,8 6,8 7,8 8,8 9,8<br>0,7 1,7 2,7 3,7 4,7 5,7 6,7 7,7 8,7 9,7<br>0,6 1,6 2,6 3,6 4,6 5,6 6,6 7,6 8,6 9,6<br>0,5 1,5 2,5 3,5 4,5 5,5 6,5 7,5 8,5 9,5<br>0,4 1,4 2,4 3,4 4,4 5,4 6,4 7,4 8,4 9,4<br>0,3 1,3 2,3 3,3 4,3 5,3 6,3 7,3 8,3 9,3<br>0,2 1,2 2,2 3,2 4,2 5,2 6,2 7,2 8,2 9,2<br>0,1 1,1 2,1 3,1 4,1 5,1 6,1 7,1 8,1 9,1<br>0,0 1,0 2,0 3,0 4,0 5,0 6,0 7,0 8,0 9,0 | Leading Digit is 9 -<br>The move RIGHT is disabled |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
|                                                     |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |
| Secondary digit is 0 - The move DOWN to be disabled |  |                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                    |

The graphical representation of the board provided the details of the locations to add to the if statement within the movement functions.

Thus far, only one player has been moved around the board, with two player objects already being declared. The technique used to distinguish between the two players would be to have a counter variable, then a conditional statement would check if the number was devisable by two, or if it was an even or odd number. It was decided to

have a third player object called current; this would take on the attributes of player 1 or 2 each turn. Otherwise, for each move, it would have to check which player is in play then call player1.Move() and call player2.Move(). The current object is passed into a method to move, and there is a counter that initiates the switch between the two players. It can get and update the correct locations for the correct piece, and this is seen below.

```

- - - - -
- - - - - 2
- - - $ - - -
- - - - -
1 - - - - -
- - - - -
Coins P1 = 0 P2 = 0
Word: C*****
Player 1 choose next letter >c
Correct
Choose a direction to move
up
- - - - -
- - - - - 2
- - - $ - - -
- - - - -
1 - - - - -
- - - - -
Coins P1 = 0 P2 = 0
Word: C*****
Player 2 choose next letter >h
Correct
Choose a direction to move
down
- - - - -
- - - - - 2
- - - $ - - -
- - - - -
1 - - - - -
- - - - -
Coins P1 = 0 P2 = 0
Word: C*****
Player 2 choose next letter >h
Correct
Choose a direction to move
down
- - - - -
- - - - - 2
- - - $ - - -
- - - - -
1 - - - - -
- - - - -
Coins P1 = 0 P2 = 0
Word: CH*****
Player 1 choose next letter >r
Correct
Choose a direction to move
down
- - - - -
- - - - - 2
- - - $ - - -
- - - - -
1 - - - - -
- - - - -
Coins P1 = 0 P2 = 0
Word: CHR*****
Player 2 choose next letter >_

```

The objective of the game is to get to the coin displayed as a '\$' symbol on the 2D board; the process is examined below.

|                                                                                                                                                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>User Cases Name:</b> Player piece lands on the coin piece                                                                                                                                                                                                                                                                       |
| <b>Stakeholders and Interests:</b><br>Player Piece                                                                                                                                                                                                                                                                                 |
| <b>Brief Description:</b><br>After a correct guess of the letter, the user selects the move that lands their piece on the same square as the coin.                                                                                                                                                                                 |
| <b>Trigger:</b><br>On a player move to the location of the coin.                                                                                                                                                                                                                                                                   |
| <b>Normal Flow:</b> <ol style="list-style-type: none"><li>1. The player's score is increased by one S1.<br/>If the score is now five S2 the current game is won.</li><li>2. The coin is sent to a new random location on the board.</li><li>3. Players continuing guessing trying to reach the new location of the coin.</li></ol> |
| <b>Subflows:</b><br><b>S-1:</b> The scoreboard is updated for the appropriate player.<br><b>S-2:</b> The game ends or players choose to play again.                                                                                                                                                                                |
| <b>Alternate/Exceptional flows:</b><br>The coin is sent to the location of one of the player pieces and overwrites.                                                                                                                                                                                                                |

With the player's individual pieces, the coin position needs attributes to store the x and y coordinates. It can then be compared to the player's piece to see if it is at the same location. In addition to this, it must be written to a new location, so the players may continue to score points. A solution was devised to use the `Random` class, to generate a new location between 0 and 9 at each coordinate. In doing so, the coin can appear at any of the one hundred places on the board. It was noted from the use case that there was a remote possibility of this action that it overwrites a player piece.

A scoreboard is required, this should be printed at the same time as the game board is printed and is seen below.

```
Coins P1 = 0 P2 = 0
```

The score is an attribute for the player objects with Console.WriteLine used to print this out and to update the score; a function was created that adds one to the score attribute on detecting this event.

The other sub-flow occurs when a player reaches the score of five. The program is within a while loop to continually step through each step involved to play the game. To break the loop, the variable which is set to true at the beginning of the game there is a check at the end of the iteration. If the current player's score is equal to five, the function returns false and causing the game to end.

| Testing Documentation Exercise 4                                |                                                                                                       |                                                                            |            |
|-----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|------------|
| Test                                                            | Outcome                                                                                               | Alterations                                                                | Re-testing |
| <b>Word</b>                                                     |                                                                                                       |                                                                            |            |
| Enter a letter in the current in-play word                      | <b>PASS</b> – Correct message displayed, and the player told to enter their move.                     |                                                                            |            |
| Enter a letter, not in the word                                 | <b>PASS</b> – Incorrect message displayed, and next players turn                                      |                                                                            |            |
| Enter other symbols Including the *                             | <b>PASS</b> – Incorrect message displayed, and next players turn                                      |                                                                            |            |
| Enter more than one character                                   | <b>FAIL</b> – Program error message                                                                   | Try Catch Block or Conditional statement checking the length of the entry. |            |
| Enter a letter that has already been revealed                   | <b>FAIL</b> – Move permitted                                                                          | Letters guessed put into a list.                                           |            |
| A new word is displayed once the current word is fully revealed | <b>PASS</b> – New word appears at the correct length in the starred format                            |                                                                            |            |
| <b>Movement</b>                                                 |                                                                                                       |                                                                            |            |
| Moving the piece around the board in every direction            | <b>PASS</b> – Piece displayed in the correct location                                                 |                                                                            |            |
| Moving outside the outer edges of the board                     | <b>PASS</b> – Move Disabled message displayed                                                         |                                                                            |            |
| Other string entry other than that of UP DOWN LEFT RIGHT        | <b>PASS</b> – Default message display move not permitted<br><b>FAULT</b> - Player has lost their move | A function that called to that repeatedly asks for acceptable input.       |            |

|                                                             |                                                                                                                  |                                                                                                                    |  |
|-------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|--|
| Player piece 1 moves to a square occupied by player piece 2 | <b>FAIL</b> – Piece overwritten although it will reappear on subsequent moves away from each other               | A method could be applied similar to the one used to check for coin detection.                                     |  |
| <b>Coin</b>                                                 |                                                                                                                  |                                                                                                                    |  |
| Testing the location of the coin on pickup                  | <b>CONDITIONAL PASS</b> – Coin never appeared on top of a player piece and appeared in many different locations. | A method to check on this event that it does not appear on the piece, and the random placement method is recalled. |  |
| Testing Score increases on coin pickup                      | Score increases by one for the correct player                                                                    |                                                                                                                    |  |

Although some features still require work and subsequent testing to minimise failure from erroneous entry. The program should be made more abstract as some problems have already been solved but are required elsewhere. Thus, using interfaces is one such way this could be achieved rather than copy the code.

## Design

This section of the report examines the application of the Unified Modeling Language or UML, which is used on the console application provided. UML was developed to assist developers by providing an overview of the application regarding the structure [2]. It is relevant, as developers work together in teams to complete programs, and in addition to this, it assists in meeting the obligations to the customer, without listing details that are not relevant at the early stage [7].

Initially, a sequence diagram (figure 10) was created to understand the underlying flow of events in the program. The class admin can fulfil both tester and developer roles.

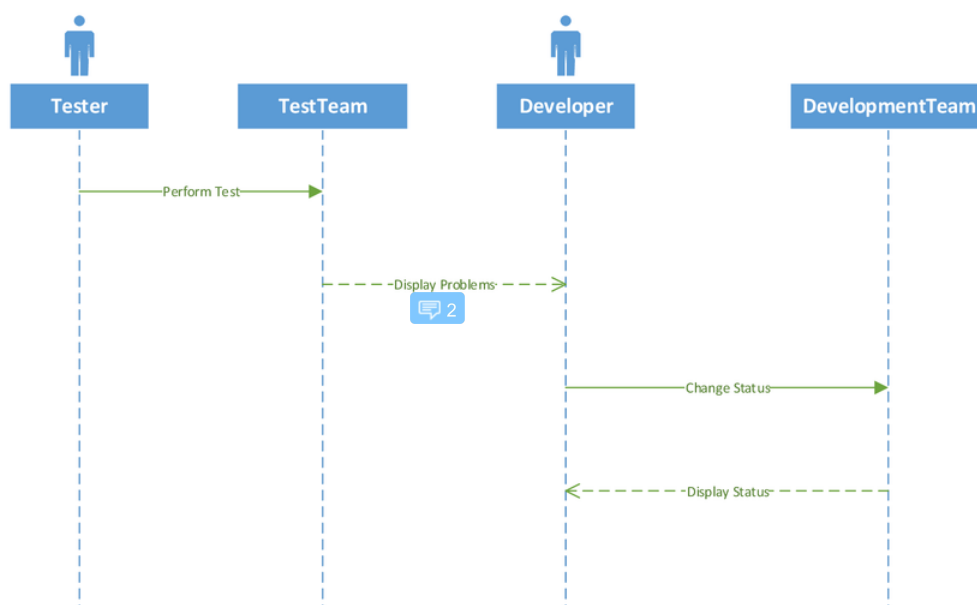


**Fig 10.** Sequence diagram for Testers and Developers Console Application.

The cross-section of code on the next page shows this sequence from trialling the program. It was observed that yes, no responses are required for each test, with the tester providing a string description of the problem found. Subsequently, any test that failed is assigned an ID and importance from the enumeration, and as observed in the sequence diagram it is printed out on-screen. The developers then get a chance

to change the status of these unfixed problems yes tries to change the earlier status, although upon testing the message failed to change status apart from the admin account can cancel problems.

```
karl is performing test Test X
Did test pass?
n
Describe the problem you encountered?
core overload
mary is performing test Test Y
Did test pass?
n
Describe the problem you encountered?
overheated
admin is performing test Test Z
Did test pass?
y
##### Print all problems #####
<XML>
<id>1</id>
<importance>NORMAL</importance>
<description>core overload</description>
</XML>
<XML>
<id>2</id>
<importance>NORMAL</importance>
<description>overheated</description>
</XML>
core overload
NORMAL
fred Change status of problem?
n
overheated
NORMAL
bob Change status of problem?
y
Fixed?
y
Failed to change status
##### Print status of unfixed problems #####
1
NEW
fred
```

In figure 11 the class diagram was created from the code provided to examine the interactions within the program.



Fig 11. Class diagram for the program provided.

Together the admin, developer and testers classes and their associated interfaces appear to adopt the open/closed principle. As if another class is added it can assume the responsibilities of a developer or tester making the program open to extension.

## Reference List

- [1] M. McGrath, *C# Programming in easy steps*. In Easy Steps Limited, 2016.
- [2] RB Whitaker, *The C# Player's Guide*, 2nd ed. Starbound Software, 2015.
- [3] S. Allardice, "Computer programming: What is object-oriented language?," 2011. [Online]. Available: <https://www.youtube.com/watch?v=SS-9yOH3Si8>. [Accessed: 29-Oct-2017].
- [4] Jamie Chan, *C#: Learn C# in One Day and Learn It Well*. Learn Coding Fast, 2015.
- [5] A. Stellman and J. Greene, *Head First C#: A Learner's Guide to Real-World Programming with C#, XAML, and .NET*, 3rd ed. O'Reilly, 2013.
- [6] J. Schmedtmann, "The Complete JavaScript Course," 2016. [Online]. Available: <https://www.udemy.com/the-complete-javascript-course/>. [Accessed: 02-Nov-2017].
- [7] D. McLaughlin, Brentt; Pollice Gary; West, *Head First Object-Oriented Analysis & Design*. O'Reilly, 2006.
- [8] J. Sharp, *Microsoft Visual C# 2013 Step by Step*: . Pearson Education M.U.A., 2013.
- [9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*, 1st ed. Addison Wesley, 1994.
- [10] J. Albahari and B. Albahari, *C# 5.0 Pocket Reference: Instant Help for C# 5.0 Programmers*. O'Reilly, 2012.
- [11] J. Córdoba, "Abstract classes and interfaces," 2017. [Online]. Available: <https://www.devfields.com/abstract-classes-and-interfaces/>. [Accessed: 11-Feb-2018].
- [12] J. James, "When to use interfaces instead of classes," 2010. [Online]. Available: <https://www.techrepublic.com/blog/software-engineer/when-to-use-interfaces-instead-of-classes/>. [Accessed: 10-Mar-2018].
- [13] R. Martin, "Getting a SOLID start," 2009. [Online]. Available: <https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start>. [Accessed: 10-Feb-2018].
- [14] R. Martin, "The Dependency Inversion Principle," *Object Mentor SOLID Des. Pap.*, 1996.
- [15] S. Oloruntoba, "S.O.L.I.D: The First 5 Principles of Object Oriented Design," 2015. [Online]. Available: <https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>. [Accessed: 10-Feb-2018].
- [16] M. Fowler, "Inversion of Control Containers and the Dependency Injection pattern," 2004. [Online]. Available: <https://martinfowler.com/articles/injection.html>. [Accessed: 10-Feb-2018].
- [17] A. LOZANČIĆ, "Benefits of Software Testing," 2016. [Online]. Available: <http://gauss-development.com/benefits-software-testing/>. [Accessed: 10-Jan-2018].

- [18] G. Fournier, *Essential Software Testing A Use-Case Approach*, 1st ed. CPC Press, 2008.
- [19] R. Zafar, "What is software testing? What are the different types of testing?," 2012. [Online]. Available: <https://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty>. [Accessed: 08-Feb-2018].
- [20] B. Boehm and P. Papaccio, "Understanding and Controlling Software Costs," *IEEE Trans. Softw. Eng.*, vol. 14, no. 10, pp. 1462–1477, 1988.
- [21] J. Palermo, "Guidelines for Test-Driven Development," 2006. [Online]. Available: [https://msdn.microsoft.com/en-us/library/aa730844\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx). [Accessed: 01-Mar-2018].
- [22] "Usability Testing." [Online]. Available: <https://www.usability.gov/how-to-and-tools/methods/usability-testing.html>. [Accessed: 07-Mar-2018].
- [23] M. Fowler, "TestDouble," 2006. [Online]. Available: <https://martinfowler.com/bliki/TestDouble.html>. [Accessed: 12-Mar-2018].

## Bibliography

- [1] M. McGrath, *C# Programming in easy steps*. In Easy Steps Limited, 2016.
- [2] RB Whitaker, *The C# Player's Guide*, 2nd ed. Starbound Software, 2015.
- [3] S. Allardice, "Computer programming: What is object-oriented language?," 2011. [Online]. Available: <https://www.youtube.com/watch?v=SS-9y0H3Si8>. [Accessed: 29-Oct-2017].
- [4] Jamie Chan, *C#: Learn C# in One Day and Learn It Well*. Learn Coding Fast, 2015.
- [5] A. Stellman and J. Greene, *Head First C#: A Learner's Guide to Real-World Programming with C#, XAML, and .NET*, 3rd ed. O'Reilly, 2013.
- [6] J. Schmedtmann, "The Complete JavaScript Course," 2016. [Online]. Available: <https://www.udemy.com/the-complete-javascript-course/>. [Accessed: 02-Nov-2017].
- [7] D. McLaughlin, Brentt; Pollice Gary; West, *Head First Object-Oriented Analysis & Design*. O'Reilly, 2006.
- [8] J. Sharp, *Microsoft Visual C# 2013 Step by Step*. Pearson Education M.U.A., 2013.
- [9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*, 1st ed. Addison Wesley, 1994.
- [10] J. Albahari and B. Albahari, *C# 5.0 Pocket Reference: Instant Help for C# 5.0 Programmers*. O'Reilly, 2012.
- [11] J. Córdoba, "Abstract classes and interfaces," 2017. [Online]. Available: <https://www.devfields.com/abstract-classes-and-interfaces/>. [Accessed: 11-Feb-2018].
- [12] J. James, "When to use interfaces instead of classes," 2010. [Online]. Available: <https://www.techrepublic.com/blog/software-engineer/when-to-use-interfaces-instead-of-classes/>. [Accessed: 10-Mar-2018].
- [13] R. Martin, "Getting a SOLID start," 2009. [Online]. Available: <https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start>. [Accessed: 10-Feb-2018].
- [14] R. Martin, "The Dependency Inversion Principle," *Object Mentor SOLID Des. Pap.*, 1996.
- [15] S. Oloruntoba, "S.O.L.I.D: The First 5 Principles of Object Oriented Design," 2015. [Online]. Available: <https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>. [Accessed: 10-Feb-2018].
- [16] M. Fowler, "Inversion of Control Containers and the Dependency Injection pattern," 2004. [Online]. Available: <https://martinfowler.com/articles/injection.html>. [Accessed: 10-Feb-2018].
- [17] A. LOZANČIĆ, "Benefits of Software Testing," 2016. [Online]. Available: <http://gauss-development.com/benefits-software-testing/>. [Accessed: 10-Jan-2018].
- [18] G. Fournier, *Essential Software Testing A Use-Case Approach*, 1st ed. CPC Press, 2008.
- [19] R. Zafar, "What is software testing? What are the different types of testing?," 2012. [Online]. Available: <https://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty>. [Accessed: 08-Feb-2018].

- [20] B. Boehm and P. Papaccio, "Understanding and Controlling Software Costs," *IEEE Trans. Softw. Eng.*, vol. 14, no. 10, pp. 1462–1477, 1988.
- [21] J. Palermo, "Guidelines for Test-Driven Development," 2006. [Online]. Available: [https://msdn.microsoft.com/en-us/library/aa730844\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx). [Accessed: 01-Mar-2018].
- [22] "Usability Testing." [Online]. Available: <https://www.usability.gov/how-to-and-tools/methods/usability-testing.html>. [Accessed: 07-Mar-2018].
- [23] M. Fowler, "TestDouble," 2006. [Online]. Available: <https://martinfowler.com/bliki/TestDouble.html>. [Accessed: 12-Mar-2018].

## Appendix A – Code from the exercises

### Exercise 1

```
static void Main(string[] args)
{
    bool quit = true; //used for the while loop
    string options; //for the switch staement
    Machine machine = new Machine(1000); //object produced with initial value set
    Console.WriteLine("Welcome to the spoon machine please input one of the
following options\n{0}", machine.MachineOptions()); //Greeting the options provided by
the function

    options = Console.ReadLine().ToLower().Trim();

    while (quit)
    {
        switch (options)
        {
            case "add":
                Console.WriteLine("How many spoons");
                int usernumber;
                try
                {
                    usernumber = int.Parse(Console.ReadLine());
                    if (usernumber < 0)
                    {
                        Console.WriteLine("Number must not be negative");
                    }
                    else
                    {
                        machine.AddToSpoonTotal(usernumber);
                    }
                }
                catch
                {
                    Console.WriteLine("Invalid Entry");
                }
                break;
            case "total":
```



```

        return _price;
    }

    //Methods

    public double CalculateTotalCost()
    {
        return _spoonstotal * _price;
    }

    public string MachineOptions()
    {
        return " add - to manufacture new spoons\n total - to see the total number\n produced\n cost - to see the total value of spoons\n quit - to exit the application";
    }
}

```

## Exercise 2

```

static void Main(string[] args)
{
    Circle circle1 = new Circle(6, 6, 3);
    Circle circle2 = new Circle(1, 3, 8);
    circle1.OverlapTest(circle1, circle2);
    //Testing
    //Console.WriteLine("Cicle 1 x = {0} y = {1} r = {2}", circle1.getX(), circle1.getY(),
    circle1.getRadius());
    //Console.WriteLine("Cicle 2 x = {0} y = {1} r = {2}", circle2.getX(), circle2.getY(),
    circle2.getRadius());
    Console.ReadKey();
}

class Circle
{
    private int _xcoordinate;
    private int _ycoordinate;
    private int _radius;

    public Circle(int x, int y, int radius)
    {
        _xcoordinate = x;
        _ycoordinate = y;
        _radius = radius;
    }

    public int getX()
    {
        return _xcoordinate;
    }

    public int getY()
    {
        return _ycoordinate;
    }

    public int getRadius()
    {

```

```

        return _radius;
    }

    public int NegativeValueCheck(int total)
    {
        if (total < 0)
        {
            return total * -1; //circle2.getx() / gety() is the higher value
        }
        else
        {
            return total; //circle1.getx() / gety() is the higher value
        }
    }

    public void OverlapTest(Circle circle1, Circle circle2)
    {
        int xvalues = NegativeValueCheck(circle1.getX() - circle2.getX());
        int yvalues = NegativeValueCheck(circle1.getY() - circle2.getY());
        int total = xvalues + yvalues;
        int radii = circle1.getRadius() + circle2.getRadius();

        if (total > radii)
        {
            Console.WriteLine("No Overlap \nX = {0} Y = {1} Total = {2} Radii = {3}", xvalues, yvalues, total, radii);
            //return false; //circles do not overlap
        }
        else
        {
            Console.WriteLine("Overlap Detected \nX = {0} Y = {1} Total = {2} Radii = {3}", xvalues, yvalues, total, radii);
            //return true; //they overlap;
        }
    }
}

```

### Exercise 3

```
static void Main(string[] args)
{
    int basketsize = 0; //for loop
    string productname; //switch statement
    Console.WriteLine("Please enter the basket size");
    basketsize = int.Parse(Console.ReadLine());

    Product[] customer = new Product[basketsize]; //setting the size

    TotalPrice totalvalue = new TotalPrice(); //used at the end the basket will be
    passed in

    for(int count = 0; count < basketsize; count++) //loops through the size
    of the users basket size
    {
        Console.WriteLine("Please enter the name of the product to add to basket");

        productname = Console.ReadLine().ToUpper();
        double value;
        switch (productname)
        {
            case "CD":
                Console.WriteLine("Please enter the value");
                value = double.Parse(Console.ReadLine());
                customer[count] = new CD();
                customer[count].setPrice(value);
                break;
            case "DVD":
                Console.WriteLine("Please enter the value");
                value = double.Parse(Console.ReadLine());
                customer[count] = new DVD();
                customer[count].setPrice(value);

                break;
            case "BOOK":
                Console.WriteLine("Please enter the value");
                value = double.Parse(Console.ReadLine());
                customer[count] = new Book();
                customer[count].setPrice(value);
                break;
            default:
                Console.WriteLine("No such product try again");
                count--; //take the loop back one to permit entry of the
                amount of products intially choosen
                break;
        }
        totalvalue.ExTax(customer); //method in the Total Price class
        totalvalue.IncTax(customer); //method in the Total Price class

        Console.ReadKey(); //Pause
    }
}
```

```

abstract class Product
{
    private double _price;

    public void setPrice(double price)
    {
        _price = price;
    }

    public double getPrice()
    {
        return _price;
    }

    abstract public double TotalIncludingTax(Product product);
}

class Book: Product
{
    public override double TotalIncludingTax(Product cart)
    {
        return cart.getPrice();
    }
}

class CD : Product
{
    public override double TotalIncludingTax(Product cart)
    {
        return (cart.getPrice() / 100 * 10) + cart.getPrice();
        //getting 10% of the value and adding back on the intial value
    }
}

class DVD: Product
{
    public override double TotalIncludingTax(Product cart)
    {
        return (cart.getPrice() / 100 * 12) + cart.getPrice();
    }
}

class TotalPrice
{
    public void IncTax(Product[] customer)
    {
        double price = 0;
        foreach (Product product in customer)
        {
            price += product.TotalIncludingTax(product);
        }
        Console.WriteLine("The total cost including tax is {0}", price);
    }
}

```

```
public void ExTax(Product[] customer)
{
    double price = 0;
    foreach (Product product in customer)
    {
        price += product.getPrice();
    }
    Console.WriteLine("The total cost excluding tax is {0}", price);
}
```

## Exercise 4

```
static void Main(string[] args)
{
    //Exercise 4 Word Search

    bool gameplay = true;
    Board board = new Board();
    Player player1 = new Player(0, 0, 2, "1"); //Score, Xcoor, Ycoor, Piece
    Player player2 = new Player(0, 9, 7, "2"); //Initial positions for each
    Player current = new Player(0, 0, 0, "?"); //passed into functions to for
the current player switched each turn

    string direction; //used for switch
    WordGuess wordguess = new WordGuess();
    wordguess.StarTheWord(); //initial staring of the word
    int progress = 0; //Alternates the players depending if the counter is
even or odd
    while (gameplay)
    {
        board.DisplayTheBoard(board.GetTheBoard()); //Print Game Board
        Console.WriteLine("Coins P1 = {0} P2 = {1}", player1.GetScore(),
player2.GetScore()); //Print Score Board

        wordguess.WordProgress();
        Console.WriteLine(""); //new line
        if (progress % 2 == 0)
        {
            current = player1; //current player is 1
            Console.Write("Player {0} ", current.GetPlayerPiece()); //Print
feedback to the user
        }
        else
        {
            current = player2; //current player is 1
            Console.Write("Player {0} ", current.GetPlayerPiece());
        }
        if (wordguess.GuessLetter()) //Player enters a letter if in the word,
they get to move
        {
            Console.WriteLine("Choose a direction to move");
            direction = Console.ReadLine().ToUpper();

            switch (direction)
            {
                case "UP":
                    current.MovePlayerPieceUp(board, current); //Will move the
player up on the board along with the current player is passed in
                    break;
                case "DOWN":
                    current.MovePlayerPieceDown(board, current);
                    break;
                case "LEFT":
                    current.MovePlayerPieceLeft(board, current);
                    break;
                case "RIGHT":
                    current.MovePlayerPieceRight(board, current);
                    break;
            }
        }
    }
}
```



```

    {
        return _tenbyten;
    }

    public int GetGoldX()
    {
        return _goldCoinX;
    }

    public int GetGoldY()
    {
        return _goldCoinY;
    }

    public void SetCoins(int x, int y)
    {
        _goldCoinX = x;
        _goldCoinY = y;
    }
}

class Player: Board
{
    //Attributes
    private int _score;
    private int _xcoord;
    private int _ycoord;
    private string _playerpiece;
    //Constructors
    public Player(int score, int xcoord, int ycoord, string playerpiece)
    {
        _score = score;
        _xcoord = xcoord;
        _ycoord = ycoord;
        _playerpiece = playerpiece; //used to write to the board
    }

    //Properties

    public int GetX()
    {
        return _xcoord;
    }

    public void SetX(int x)
    {
        _xcoord = x;
    }

    public int GetY()
    {
        return _ycoord;
    }

    public void SetY(int y)
    {
        _ycoord = y;
    }
}

```

```

public int GetScore()
{
    return _score;
}
public string GetPlayerPiece()
{
    return _playerpiece;
}

//Methods
public void PointScored()
{
    _score += 1;
}

//ends the while loop
public bool PlayerWon(Player player)
{
    if(player.GetScore() == 5)
    {
        return false;
    }
    else
    {
        return true;
    }
}

public void MovePlayerPieceDown(Board board, Player player)
{
    if(player.GetY() == 0) //if the secondary digit in the array is 0
    {
        Console.WriteLine("Move Disabled");
    }
    else
    {
        board.SetTheBoard(player.GetX(), player.GetY(), "-"); //Writing a
blank space to the current Coordinates
        board.SetTheBoard(player.GetX(), player.GetY() - 1,
player.GetPlayerPiece()); //Writing the playerpiece to the new location on the board
        player.SetY(player.GetY() - 1); //Updating the cuurent Coordinates
    }
}

public void MovePlayerPieceUp(Board board, Player player)
{
    if (player.GetY() == 9) //if the secondary digit in the multi array is 9
    {
        Console.WriteLine("Move Disabled");
    }
    else
    {
        board.SetTheBoard(player.GetX(), player.GetY(), "-");
        board.SetTheBoard(player.GetX(), player.GetY() + 1,
player.GetPlayerPiece());
        player.SetY(player.GetY() + 1);
    }
}

```

```

    }
    public void MovePlayerPieceLeft(Board board, Player player)
    {
        if (player.GetX() == 0) //if the leading digit in the multi array is 0
        {
            Console.WriteLine("Move Disabled");
        }
        else
        {
            board.SetTheBoard(player.GetX(), player.GetY(), "-");
            board.SetTheBoard(player.GetX() - 1, player.GetY(),
player.GetPlayerPiece());
            player.SetX(player.GetX() - 1);
        }
    }

    public void MovePlayerPieceRight(Board board, Player player)
    {
        if (player.GetX() == 9) //if the leading digit in the multi array is 9
        {
            Console.WriteLine("Move Disabled");
        }
        else
        {
            board.SetTheBoard(player.GetX(), player.GetY(), "-");
            board.SetTheBoard(player.GetX() + 1, player.GetY(),
player.GetPlayerPiece());
            player.SetX(player.GetX() + 1);
        }
    }

    public void FoundCoin(Board board, Player player)
    {
        //Both x and y must be the same to be on the coin
        if(player.GetX() == board.GetGoldX() && player.GetY() == board.GetGoldY())
        {
            Random rnd = new Random();
            int goldx = rnd.Next(0, 10); //use the random class to find a number
between 0 and 9
            int goldy = rnd.Next(0, 10); //variables used to set this position
below
            player.PointScored(); //player is on the coin point scored
            board.SetCoins(goldx, goldy);
            board.SetTheBoard(goldx, goldy, "$"); //Set the piece to an
alternative location
        }
    }
}

```

```

class WordGuess
{
    //Attributes
    private int currentword = 0; //1 is added each time a word is guessed
    private string[] _Wordchoices = new string[10] { "CHRISTMAS", "TREE", "SANTA",
"PUDDING", "TURKEY", "TINSEL", "MINCEPIES", "ELF", "SPROUTS", "PRESENTS"};
    public char[] _WordToGuess;
    private char[] _InPlayWord; //What the user sees
    private char _PlayersGuess; //Compare to Word To Guess
    //Constructors
    public WordGuess()
    {
        _WordToGuess = ReturnWord().ToCharArray(); //Start With Array 0
        _InPlayWord = new char [WordLength()]; //Create a Char Array Of This
Length
    }
    //Properties

    public int GetCurrentWordInArray()
    {
        return currentword;
    }
    //Methods
    public void NextWordInTheArray()
    {
        currentword += 1; //The word has been guessed so move to the next item in
the array
    }

    public int WordLength()
    {
        return _Wordchoices[GetCurrentWordInArray()].Length;
    }

    public string ReturnWord()
    {
        return _Wordchoices[0 + GetCurrentWordInArray()];
    }

    public void StarTheWord()
    {
        for(int i = 0; i < WordLength(); i++)
        {
            _InPlayWord[i] = '*';
        }
    }

    public bool GuessLetter()
    {
        int hits = 0;
        Console.Write("choose next letter >");
        _PlayersGuess = Convert.ToChar(Console.ReadLine().ToUpper());
        for (int count = 0; count < WordLength(); count++)
        {
            if (_PlayersGuess == _WordToGuess[count])
            {

```

```

        _InPlayWord[count] = _PlayersGuess; //Letter is in the word
        hits++;
    }

    }
    if(hits == 0)
    {
        Console.WriteLine("Incorrect"); //the word is searched with no matches
        return false;
    }
    else
    {
        Console.WriteLine("Correct");
        return true;
    }
}

public void WordProgress()
{
    Console.Write("Word: "); //Printing the users word as they make guesses
    for (int i = 0; i < WordLength(); i++)
    {
        Console.Write(_InPlayWord[i]);
    }
}

public void NewWord()
{
    _WordToGuess = ReturnWord().ToCharArray();
    //+1 on the word and put it in the char array
}

public void WordGuessed()
{
    int tracker = 0;
    for (int i = 0; i < WordLength(); i++)
    {
        if (_InPlayWord[i] != '*')
        {
            tracker++;
        }
    }
    if(tracker == WordLength()) //If no letter is a star the word has been
guessed
    {
        //Next Word
        NextWordInTheArray(); //Add one onto current array position
        StarTheWord(); //Star out all the letters
        NewWord(); //Set word to guess as the new array position
        Console.WriteLine("WORD FOUND!!");
    }
}

}

```

# James Holmson Report

---

## GRADEMARK REPORT

---

### FINAL GRADE

83/100

### GENERAL COMMENTS

#### Instructor

Knowledge OOPs concepts (30) 26

Discussion of design principles (10) 8

Knowledge of testing (10) 8

UML (20) 17

Exercise 1 (5) 5

Exercise 2 (5) 2

Exercise 3 (10) 9

Exercise 4 (10) 8

Classes vs Objects: excellent explanation

Inheritance and Polymorphism: very good, you have provided an example. However, you needed to talk about the benefits of abstraction a bit more i.e. it allows code to be more cleanly divided and the Open Closed Principle.

Interfaces: very nice description, could have talked a little more about the benefits of subdividing code using interfaces.

Dependency Inversion: a good description of DIP, I would have liked to have seen some discussion of the benefits it has in testing

Software Testing: you have covered testing approaches in some detail,.

UML: A good attempt at the class diagram, there are

a couple of minor issues. The sequence diagram is a bit too simplistic.

Ex2: you need to calculate the hypotenuse.

Ex9: You could have used a Basket class to contain and manage the array of Products

---

PAGE 1

---

PAGE 2

---

PAGE 3

---

PAGE 4

---

PAGE 5

---

PAGE 6

---

PAGE 7

---

PAGE 8

---

PAGE 9

---

PAGE 10

---

PAGE 11

---

PAGE 12

---

PAGE 13



### Comment 1

I like this description you have found

---

PAGE 14

---

PAGE 15

---

PAGE 16

---

PAGE 17

---

PAGE 18

---

PAGE 19

---

PAGE 20

---

PAGE 21

---

PAGE 22

---

PAGE 23

---

PAGE 24

---

PAGE 25

---

PAGE 26

---

PAGE 27

---

PAGE 28

---

PAGE 29

---

PAGE 30

---

PAGE 31

---

PAGE 32

---

PAGE 33

---

PAGE 34

---

PAGE 35

---

PAGE 36

---

PAGE 37

---

PAGE 38

---



### Comment 2

These should be method names

PAGE 39

---

PAGE 40

---



### Comment 3

Use relationship



### Comment 4

Use relationship



### Comment 5

should be unidirectional



### Comment 6

We don't need to show associations between enumerations and the classes, they are just types like integers

PAGE 41

---

PAGE 42

---

PAGE 43

---

PAGE 44

---

PAGE 45

---

PAGE 46

---

PAGE 47

---



### Comment 7

Not required, you forgot to square the difference and calculate the hypotenuse



### Comment 8

This won't work

PAGE 48

---

PAGE 49

---

PAGE 50

---

PAGE 51

---

PAGE 52

---

PAGE 53

---

PAGE 54

---

PAGE 55

---

PAGE 56

---

PAGE 57

---