

Application Design and Development David Price P131188

by David Price

FILE	4305_DAVID_PRICE_APPLICATION_DESIGN_AND_DEVELOPMENT_DAVID_PRICE__P131188_93264_362948929.DOCX		
TIME SUBMITTED	20-MAR-2018 08:40PM (UTC+0000)	WORD COUNT	3675
SUBMISSION ID	85170290	CHARACTER COUNT	25318

Application Design and Development
Application Design and Development
Report
M5X01025

Student: David Price

Student number: P131188

Lecturer: Mike Dacey

Table of Contents

Table of figures.....	3
Abstract.....	4
Introduction.....	4
Part A	4
Question a.....	4
Question b	5
Question c.....	6
Question d	6
Part B	8
Exercise 1	8
Exercise 2	9
Exercise 3	10
Exercise 4.....	10
Part C	10
Part D	11
Conclusion	13
References.....	14
Appendix.....	16
Exercise 1 code.....	16
Exercise 2 code.....	17
Exercise 3 code.....	18
Exercise 4 code.....	25

Table of figures

Figure 1 - Class example[3]	4
Figure 2 - Object Modules[4]	5
Figure 3 - Overview of interface alongside class[9]	6
Figure 4 - Simple layers [13]	7
Figure 5 - Abstract layers[13]	7
Figure 6 - White box testing representation[17]	12
Figure 7 - Black box testing representation[17]	13

Abstract – This report will show demonstration of systematic understanding of software engineering concepts and practice of applications, an overlook at the principles taken in object orientated.

Introduction

This report will look at innards of the design and development process of building applications regardless of scale, looking at benefits and implications of small and large applications. There will also be diagrams explaining and describing the relationship between classes and interfaces by using UML diagrams, along with different approaches of testing such as white box testing, black box testing and unit testing mainly. There will be evidence showing the thought process behind the applications and exercises involved in this report, along with multiple screenshots showing the expected outcome.

Part A

Question a

Firstly, object orientated programming is “a technique for programming – a paradigm for writing “good” programs for a set of problems”[1]. Object orientated programming (OOP) gives the programmer the freedom to take a large problem and find a solution by breaking it down into smaller more manageable problems.

Classes play a vital part in object orientated programming, “they specify sets of objects with common properties.”[2], a class is essentially an extension template for creating objects. “A class is a specification of the state and behaviour of objects. In the context of a system, an object’s state is part of the system state and an object’s behaviour is part of the system behaviour”[2]. A class allows the programmer to group variables together, the can be used to create new data types.

A class must have the following specifications:

- Have a name
- Defines attributes and types
- Instances of a class are objects, share features described by the class
- Defines properties/fields and methods

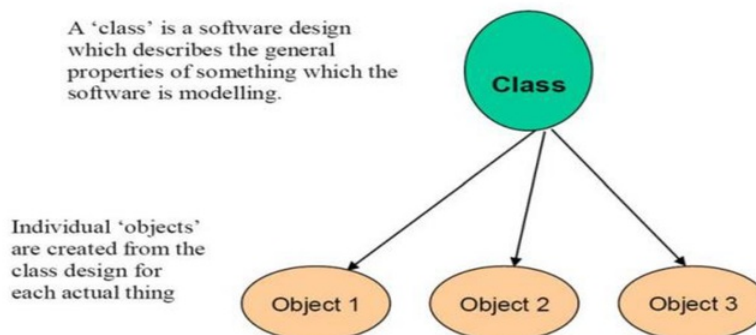


Figure 1 - Class example[3]

Objects used in OOP “are collections of operations that share a state. The operations determine the messages (calls) to which the object can respond, while the shared state is hidden from the outside world and is accessible only to the object's operations” [4].

An object must have the following specifications:

- Be created from class
- Contain unique values for each attribute
- Hold attributes and values
- It's an instance of a class

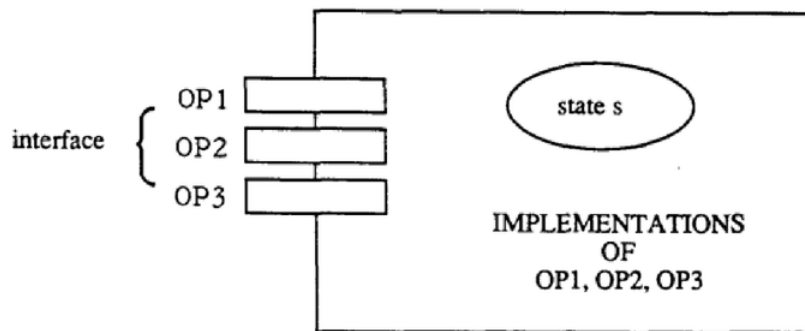


Figure 2 - Object Modules[4]

From the comparison detailed above, looking at what each concept must have in order for it to exist, the main difference between a class and an object is, that a class is “is a description of one or more similar objects.” [5] whereas an object are “entities that combine the properties of procedures and data since they perform computations and save local state.” [5].

Question b

The implications that occur with polymorphism and inheritance in developing large scale applications, as the code gets bigger and more elaborate also does the hierarchy of your code.

A look at polymorphism, “polymorphism refers to a programming language's ability to process objects differently depending on their data type or class.” [6]. Polymorphism can cause obstacles and implications when used in large scale development, “new components can be used in the same environment as old components without having to modify the calling environment” [7].

- A problem is when it comes to testing the code, the same method will then be tested repeatedly, which is a waste of time.
- Polymorphism supports re-usable code, by doing this the subclass creates a method which could be an extension or new version of this method all while keeping the same name as the method, so you're left with subclasses with methods named the same that might do different operations this is called Overriding.
- When implementing polymorphism into code, it can be harder to read and follow
- “new components can be used in the same environment as old components without having to modify the calling environment” [7].

Inheritance allows “the programmer to build class hierarchies where subclasses inherit characteristics from simpler, more general parent classes.”[8].

Inheritance can cause implications is:

- "it can be difficult to gain a proper understanding of how the subclass is constructed. Because objects can send messages to themselves, an methods up and down the hierarchy can execute, all the superclasses to the subclass have to be examined in order to acquire an understanding of the whole inheritance structure.”[7]
- Inheritance violates encapsulation
- Re-usable code
- More hierarchy there is, the less likely optimized the application will be

Question c

An interface is “purely abstract class; only signatures, no implementation.”[9], it’s like an abstract base class, they encompass only the bare minimum to function properly. We use interfaces to define what things an object can do but doesn’t imply how the behaviour of the object is done. Having an interface allows to have many implementations, there is no need for an individual method for each implementation of a class. Interfaces are there to oversee that the correct functions within an interface are implemented in the parent class, “Implementing an interface allows a class to become more formal about the behaviour.”[10].

- “Abstract classes are permitted to contain implementations for some methods while interfaces are not.”[11]
- “A class must be a subclass of the abstract class. Any class that defines all of the requires methods and obeys the general contract is permitted to implement an interface”[11]

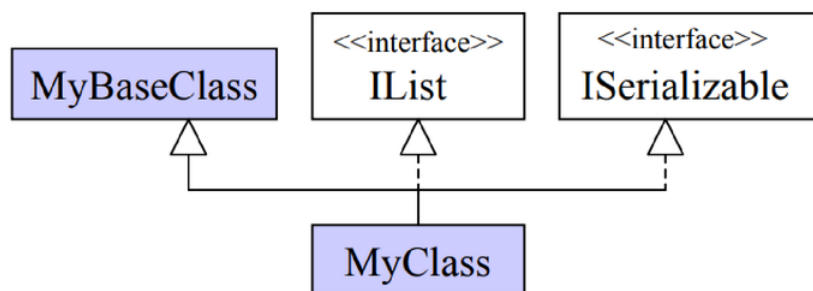


Figure 3 - Overview of interface alongside class[9]

To summarise, the difference between classes and interfaces is that, classes will specify what an object is and an interface is where it specifies what the object can do.

Question d

Dependency-Inversion principle is the “principle of dependency inversion is the fundamental low-level mechanism behind many of the benefits claimed for object-oriented technology. Its proper application is necessary for the creation of reusable frameworks”[12], it is the decoupling of software modules, by adding an abstraction layer between them.

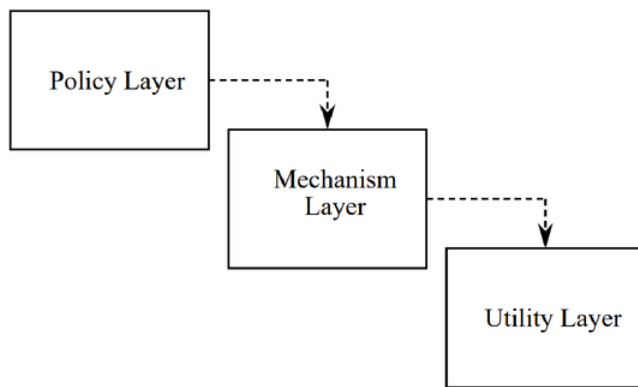


Figure 4 - Simple layers [13]

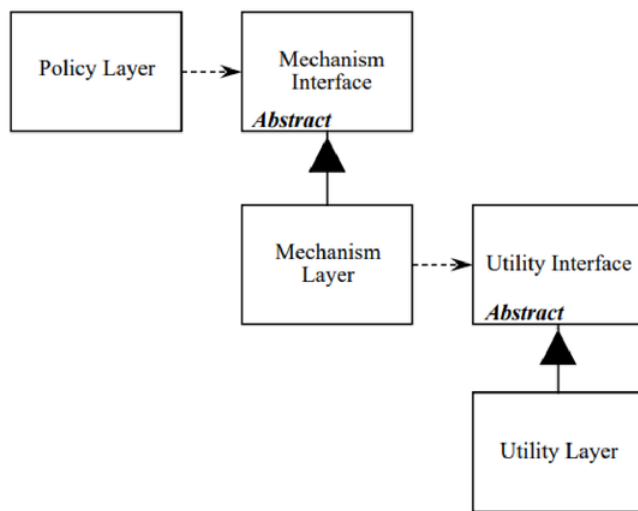


Figure 5 - Abstract layers[13]

Benefits of Dependency Inversion:

- “Abstractions should not depend upon details. Details should depend upon abstractions.”[12]
- “Its proper application is necessary for the creation of reusable frameworks”[13]
- “the abstractions and details are all isolated from each other, the code is much easier to maintain.”[13]
- No coupling between layers
- Multiple different layers can be used

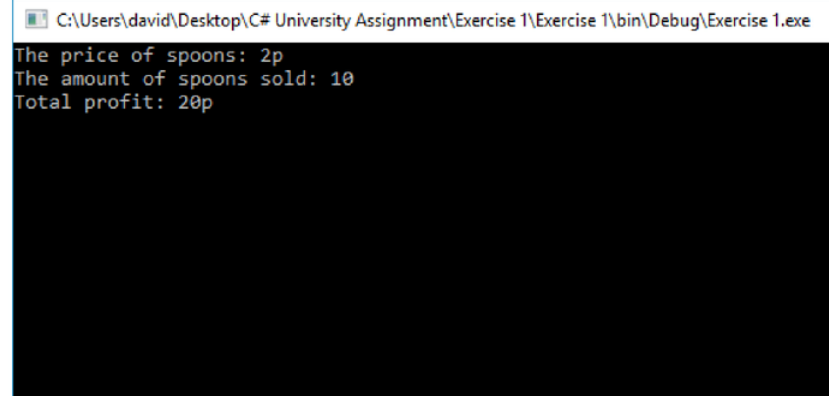
Drawbacks of Dependency Inversion:

- Hard to read code
- Quite risky, unstructured files.
- Testing becomes quite difficult

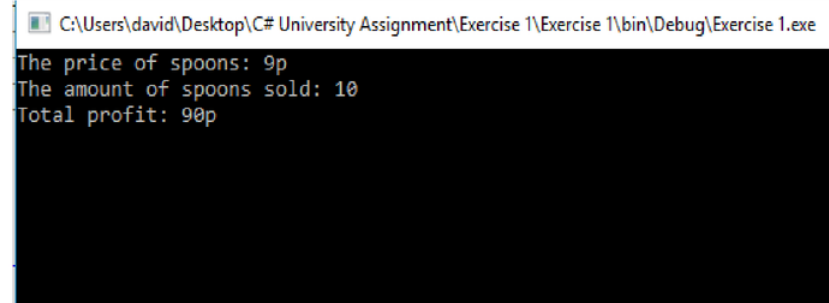
Part B

Screenshots of the following exercises, the code for each exercise is at the end of this report in the appendix.

Exercise 1



```
C:\Users\david\Desktop\C# University Assignment\Exercise 1\Exercise 1\bin\Debug\Exercise 1.exe
The price of spoons: 2p
The amount of spoons sold: 10
Total profit: 20p
```



```
C:\Users\david\Desktop\C# University Assignment\Exercise 1\Exercise 1\bin\Debug\Exercise 1.exe
The price of spoons: 9p
The amount of spoons sold: 10
Total profit: 90p
```

Exercise 2

C:\Users\david\Desktop\C# University Assignment\Exercise 2\Exercise 2\bin\Debug\Exercise 2.exe

```
Do they overlap?: False  
Area of circle is: 1256cm2
```

C:\Users\david\Desktop\C# University Assignment\Exercise 2\Exercise 2\bin\Debug\Exercise 2.exe

```
Do they overlap?: True  
Area of circle is: 1011.08cm2
```

Exercise 3

```
C:\Users\david\Desktop\C# University Assignment\Exercise 3\Exercise 3\bin\Debug\Exercise 3.exe
Which product would you like: CD|DVD|Book
CD
Enter a number of products:
50
Enter a price:
10

The Price of your purchase before Tax is: £500
The amount of tax taken off: £50
The final price is: £450
```

```
C:\Users\david\Desktop\C# University Assignment\Exercise 3\Exercise 3\bin\Debug\Exercise 3.exe
Which product would you like: CD|DVD|Book
DVD
Enter a number of products:
50
Enter a price:
10

The Price of your purchase before Tax is: £500
The amount of tax taken off: £60
The final price is: £440
```

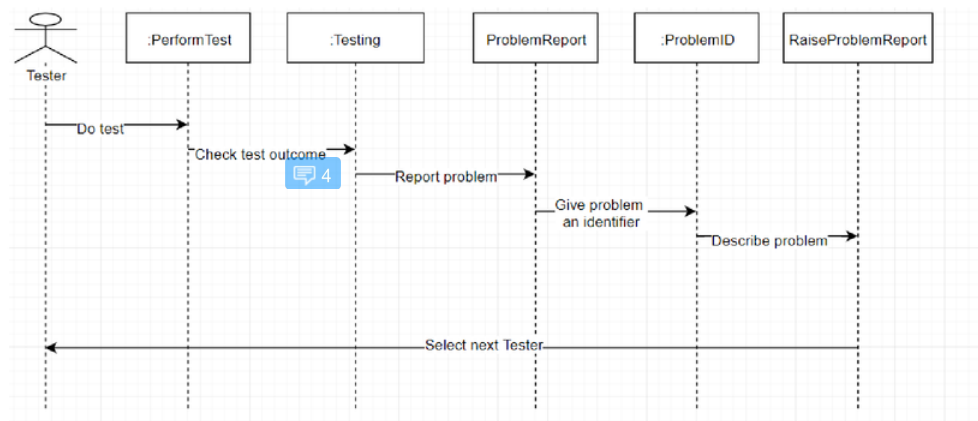
Exercise 4

Part C

UML (Unique modified language) “was to provide the development community with a stable and common design language that could be used to develop and build computer applications”[14], the way this was achieved was by creating diagrams showing the process/lifecycle of an application or the steps taken by a user/customer on how the application will be planned on being used.

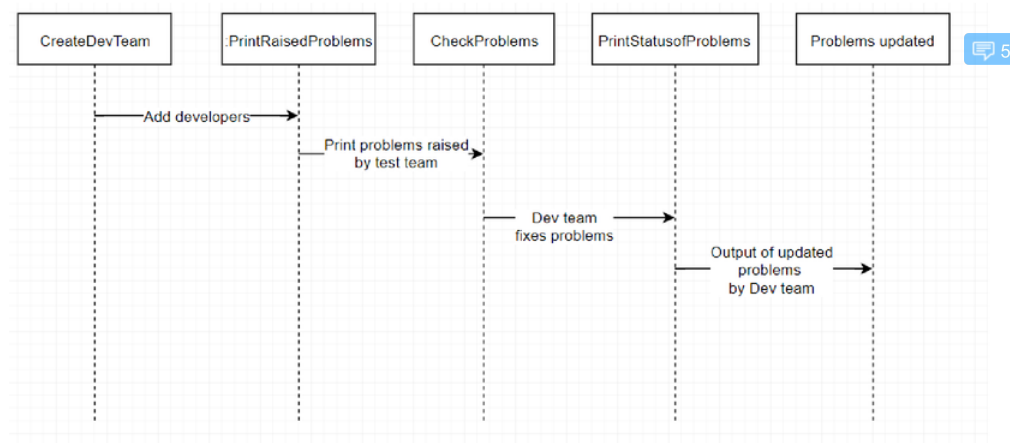
It gives a valuable insight and a more direct clear plan on what occurrences might happen and how a user would act towards the application, by identifying and creating a diagram will give the developer a more clearer idea of what the application will look and act like but also to identify possible flaws before the design phase begins. UML now is the industry standard of visualising and specifying an overview of a system software, UML “has been applied to a wide range of domains from aerospace to e-commerce underlining its potential for modelling systems that are complex yet very different in nature. The generality of the language is strengthened by a drive for its standardisation.”[15].

Tester lifeline



This shows the testers lifeline from when they are first called to do the testing, as you can see they perform the testing, come back with an outcome whether if it passed or not. Then moved on to reporting said problem if there was one, proceed to give that report found an identifier and describe the problems faced when testing the software. After a description of the problem has been given, it returns to the next tester until there are no testers left.

Dev team lifeline



Developers lifeline shows when their first added to a team, and when they are called to fix problems that the testers might have found, the problems are printed for the dev team to see and the severity of the problems, dev team attempts to fix problems and then prints an updated outcome of the problems they've attempted and if they are fixed and the severity of those problems after the attempted fixes.

Part D

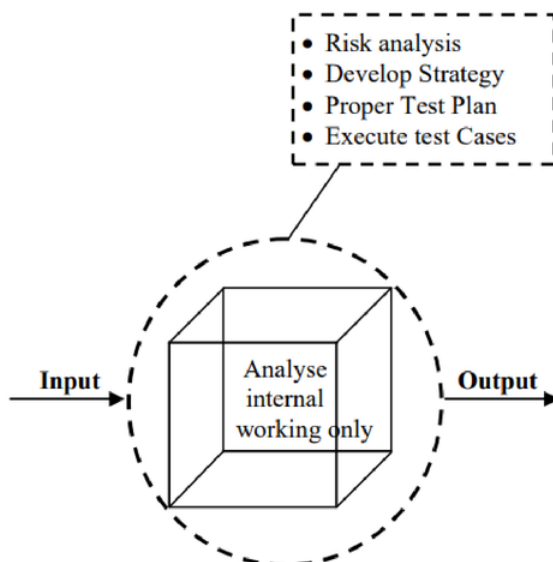
Testing in software development plays a vital part in how coding is coded nowadays, "Software testing is a significant activity of the software development life cycle (SDLC). It helps in developing the confidence of a developer that a program does what it is intended to

do so.”[16]. Developers know that the code produced is going to be put through vigorous series of repetitive testing before the software goes out to the public, “role of testing involves checking, that there should be no discrepancy in the software development process anywhere throughout”[17]. There are many types of testing techniques, the ones that are focused on in this section will be:

- White box testing
- Black box testing
- Unit testing

These testing techniques changes the way developers code, looking at these practices in detail and reflect on how important it is to test code at all stages of development, and how testing changes how code is coded in the future.

Firstly, white box testing is “the testing of a software solution's internal coding and infrastructure.”[17], white box testing is used when the developers know the internal coding and structure of the code. This allows the developer to make a test harness on the code themselves as they have a complete understanding of the internal code and the structure, they can test every aspect of the code as they have full understanding of it. White box testing is “typically very effective in validating design, decision, assumptions and finding programming errors and implementation errors in software”[18].



representation[19]

Figure 6 - White box testing

Secondly, black box testing is “conceptually simpler and can be easily automated. It is a commonly followed strategy for testing a wide variety of software systems”[20]. Black box testing is typically used on multiple systems, the difference between this and white box testing is that with black box testing we don’t really know the entire internal coding structure, there is a limitation and not specified how much code is covered in the testing bubble of black box testing. Black box testing “only examines the fundamental aspects of the system and has no or little relevance with the internal logical structure of the system”[19].

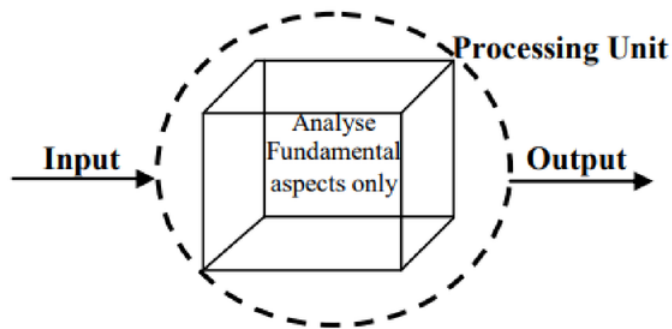


Figure 7 - Black box

testing representation[19]

Lastly, Unit testing is “just one of the levels of testing which go together to make the “big picture” of testing a system”[21], where a small collection of code grouped with each other that can be tested this limits individual testing for each segment of code. It is more logical to tested multiple modules or programs together than to create a test for each one, this is more logical for larger applications. “At the lowest level, testing is considered no different to debugging. At the higher levels, testing becomes a mindset which aims to maximise the system reliability”[21].

Conclusion

This report has shown the importance of testing and planning an intended software application whether it be large or small application, the strategies to use when testing and which practice to use depending on the application size and resources. There has been systematic understanding of developing applications and a look at concepts within in OOP such as classes, objects, interfaces and inheritance and when to implement which concept at what time. These practices will carry on in development of applications for many years to come, they provide such abstract and elaborate ways of coding which also gives the freedom to many developers and businesses to create new applications and push the boundaries of what can be achieved in computing. By constantly making the process of building small or large applications more and more elaborate and complex, you need a way of simplifying the planning and design phase, by achieving this more complex applications can be built with fewer problems happening and more reliable testing can be made.

References

- [1] B. Stroustrup, "What is Object-Oriented Programming?," *IEEE Softw.*, vol. 5, no. 3, pp. 10–20, 1988.
- [2] T. Reenskaug, "Roles and Classes in Object Oriented Programming," 2007.
- [3] S. Kendal, *Object Oriented Programming Using C# .NET*. 2016.
- [4] P. Wegner, "Concepts and Paradigms of Object-Oriented Programming Expansion of Oct 400PSLA-89 Keynote Talk Peter Wegner, Brown University 1.," 1990.
- [5] M. Stefik and D. G. Bobrow, "Object-Oriented Programming: Themes and Variations," *AI Mag.*, vol. 6, no. 4, p. 40, 1985.
- [6] J. M. Ross, "Polymorphism in decline?," *J. Comput. Sci. Coll.*, vol. 21, no. 2, pp. 328–334, 2005.
- [7] P. Paetau, *Object-Oriented Software Reuse – On The Benefits and Problems*, no. 151. 2005.
- [8] A. Lake and C. Cook, "Use of factor analysis to develop OOP software complexity metrics," *6th Annu. Oregon Work. Softw. Metrics*, ... , pp. 1–15, 1994.
- [9] H. Mössenböck, "Advanced C #," *Syntax*, pp. 1–62.
- [10] D. Goodwin, "Object Oriented Analysis & Design," *Dep. Comput. Sci*, 2012.
- [11] J. Bloch, E. Java, and T. Java, "Classes and Interfaces 6.170," pp. 1–6, 2005.
- [12] R. C. Martin and M. Micah, *Agile Principles, Patterns, and Practices in C#*, no. Robert C. Martin Series. 2006.
- [13] R. C. Martin, "The Dependency Inversion Principle," *C++ Rep.*, vol. 8, pp. 61–66, 1996.
- [14] D. Bell, "UML basics : An introduction to the Unified Modeling Language," *Ration. Softw.*, pp. 1–11, 2003.
- [15] S. De Cesare and M. Lycett, "Business Modelling with UML : Distilling Directions for Future Research," *Int. Conf. Enterp. Inf. Syst.*, pp. 570–579, 2002.
- [16] S. Nidhra, "Black Box and White Box Testing Techniques - A Literature Review," *Int. J. Embed. Syst. Appl.*, vol. 2, no. 2, pp. 29–50, 2012.
- [17] M. E. Khan and F. Khan, "Importance of Software Testing in Software Development Life Cycle," *Int. J. Comput. Sci.*, vol. 11, no. 2, pp. 120–123, 2014.
- [18] M. E. Khan, "Different approaches to white box testing technique for finding errors," *Int. J. Softw. Eng. its Appl.*, vol. 5, no. 3, pp. 1–14, 2011.
- [19] M. E. Khan and F. Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques," *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 6, pp. 12–15, 2012.
- [20] H. Kantamneni, S. Pillai, and Y. Malaiya, "Structurally guided black box testing," pp. 1–21, 1998.
- [21] R. Parkin, "Software Unit Testing," *IV V Aust. Indep. Softw. Test. Spec.*, pp. 1–4, 1997.

Bibliography

<http://www.ddegjust.ac.in/studymaterial/mca-3/ms-17.pdf>

<https://lists.gnu.org/archive/html/gnumed-devel/2005-12/pdfDUNizNkWN6.pdf>

<https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20010071445.pdf>

http://ansymore.uantwerpen.be/system/files/uploads/courses/SE3BAC/p_05_01RunesonUnitTestPractices.pdf

https://lib.dr.iastate.edu/cgi/viewcontent.cgi?referer=https://scholar.google.co.uk/&httpsredir=1&article=1337&context=cs_techreports

<https://pdfs.semanticscholar.org/15d1/cb9abb6f48234a7a8ad692fea5722814edb9.pdf>

http://www.pst.informatik.uni-muenchen.de/~mayer/papers/2006_02_TAVWEB_Towards_A_BPEL.pdf

<http://dspace.elib.ntt.edu.vn/dspace/bitstream/123456789/7251/1/UML%20Distilled.pdf>

http://sutlib2.sut.ac.th/sut_contents/H76936.pdf

<https://pdfs.semanticscholar.org/1d9c/68b162b47c6466aa08fcad2519f80e211f4f.pdf>

https://www.researchgate.net/profile/William_Cook6/publication/234784610_Interfaces_for_strongly-typed_object-oriented_programming/links/0deec52b4d740efc91000000/Interfaces-for-strongly-typed-object-oriented-programming.pdf

http://cds.cern.ch/record/798217/files/1930934122_TOC.pdf

<http://orbit.dtu.dk/files/4269120/Bahn.pdf>

Appendix

Exercise 1 code

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_1
{
    class Program
    {
        static void Main(string[] args)
        {
            Spoons Spoon1 = new Spoons(9, 10);

            Spoon1.GetSpoonPrice();
            Console.WriteLine("The price of spoons: {0}p", Spoon1.GetSpoonPrice());
            Spoon1.SpoonCount();
            Console.WriteLine("The amount of spoons sold: {0}", Spoon1.getTotal());
            Spoon1.totalSale();
            Console.WriteLine("Total profit: {0}p", Spoon1.getTotal());

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

Spoons.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_1
{
    class Spoons
    {
        //Attributes
        private double _spoonPrice;
        private int _spoonsSold;
        private double _total;
        private double _totalSale;

        //Constructors
        public Spoons(double x, int w)
        {
            _spoonPrice = x;
            _spoonsSold = w;
        }

        //Properties
        public void SetSpoonPrice(int price)
        {
            _spoonPrice = price;
        }
    }
}
```

```

        public double GetSpoonPrice()
        {
            return _spoonPrice;
        }

        public int GetSpoonsSold()
        {
            return _spoonsSold;
        }

        public double getTotal()
        {
            return _total;
        }

        //Methods
        public void totalSale()
        {
            _total = _spoonPrice * _spoonsSold;
        }

        public void SpoonCount()
        {
            _total = _spoonsSold * _totalSale;
        }
    }
}

```

Exercise 2 code

Program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_2
{
    class Program
    {
        static void Main(string[] args)
        {
            Circle C1 = new Circle(1, 10, 20);
            Circle C2 = new Circle(1, 5, 20);

            Console.WriteLine("Do they overlap?: {0}", C1.OverlapTest(C1, C2));

            Console.WriteLine("Area of circle is: {0}cm²", C1.CalcArea(C1, C2));
            //Testing to see if code worked with Area

            Console.ReadKey();
        }
    }
}

```

Circle.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_2
{
    class Circle
    {
        //Attributes
        private double _x1;
        private double _x2;
        private double _y1;
        private double _y2;

        private double _radius;

        //Constructors
        public Circle(double x, double y, double r)
        {
            _x1 = x;
            _y1 = y;
            _radius = r;
        }

        //Properties

        //Methods
        public double CalcArea(Circle C1, Circle C2)
        {
            double area = (3.14) * C1._radius * C2._radius;
            return area;
        }

        public bool OverlapTest(Circle C1, Circle C2)
        {
            double distanceBetween = ((C1._x1 - C2._x2) * (C1._x1 - C2._x2) + (C1._y1
- C2._y2) * (C1._y1 - C2._y2));
            double TotalRadii = C1._radius + C2._radius;

            double result = TotalRadii - distanceBetween;

            if (result < 0)
            {
                return false;
            }
            return true;
        }
    }
}

```

Exercise 3 code

Program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace Exercise_3
{
    class Program
    {
        static void Main(string[] args)
        {
            Basket basket = new Basket();
            double BeforeTaxResult;
            double AfterTaxResult;
            double Total;

            basket.UserProduct();
            basket.InputProducts();
            basket.CalculateBeforeTax();
            BeforeTaxResult = basket.getResult();
            basket.CalculateAfterTax();
            AfterTaxResult = basket.getResult();
            basket.CalculateTotalPrice();
            Total = basket.getResult();

            Console.WriteLine();
            Console.WriteLine("The Price of your purchase before Tax is: £{0}",
BeforeTaxResult);
            Console.WriteLine("The amount of tax taken off: £{0}", AfterTaxResult);
            Console.WriteLine("The final price is: £{0}", Total);

            Console.ReadKey();
        }
    }
}

```

Basket.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_3
{
    class Basket
    {
        //Attributes
        private CD _CD;
        private DVD _DVD;
        private Book _Book;
        private List<int> _products;
        private double _result;
        public int price;
        private string _ProductChoice = "";

        //Constructors
        public Basket()
        {
            _CD = new CD();
            _DVD = new DVD();
            _Book = new Book();
            _products = new List<int>();
        }
    }
}

```

```

        _result = 0;
    }

    //Properties

    //Methods
    public int InputProducts()
    {
        int products;

        Console.WriteLine("Enter a number of products: ");
        products = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter a price: ");
        price = int.Parse(Console.ReadLine());
        int TotalPrice = products * price;
        _products.Add(TotalPrice);

        return TotalPrice;
    }

    public string UserProduct()
    {
        string userChoice;

        Console.WriteLine("Which product would you like: CD|DVD|Book ");
        userChoice = (Console.ReadLine());
        _ProductChoice = userChoice;

        return userChoice;
    }

    public double getResult()
    {
        return _result;
    }

    public double CalculateBeforeTax()
    {
        if (_ProductChoice == "CD")
        {
            foreach (int products in _products)
            {
                _CD.InsertProduct(products);
            }
            _CD.BeforeTax();
            _result = _CD.getTotal();
        }
        if(_ProductChoice == "DVD")
        {
            foreach (int products in _products)
            {
                _DVD.InsertProduct(products);
            }
            _DVD.BeforeTax();
            _result = _DVD.getTotal();
        }
        if (_ProductChoice == "Book")
        {
            foreach (int products in _products)
            {
                _Book.InsertProduct(products);
            }
            _Book.BeforeTax();
        }
    }

```

```

        _result = _Book.getTotal();
    }

    return _result;
}

public double CalculateAfterTax()
{
    if(_ProductChoice == "CD")
    {
        foreach (int products in _products)
        {
            _CD.InsertProduct(products);
        }
        _CD.AfterTax();
        _result = _CD.getTotal();
    }
    if (_ProductChoice == "DVD")
    {
        foreach (int products in _products)
        {
            _DVD.InsertProduct(products);
        }
        _DVD.AfterTax();
        _result = _DVD.getTotal();
    }
    if(_ProductChoice == "Book")
    {
        foreach (int products in _products)
        {
            _Book.InsertProduct(products);
        }
        _result = _Book.getTotal();
    }

    return _result;
}

public double CalculateTotalPrice()
{
    if (_ProductChoice == "CD")
    {
        foreach (int products in _products)
        {
            _CD.InsertProduct(products);
        }
        _CD.TotalPrice();
        _result = _CD.getTotalPrice();
    }
    if (_ProductChoice == "DVD")
    {
        foreach (int products in _products)
        {
            _DVD.InsertProduct(products);
        }
        _DVD.TotalPrice();
        _result = _DVD.getTotalPrice();
    }
    if (_ProductChoice == "Book")
    {
        foreach (int products in _products)
        {

```

```

        _Book.InsertProduct(products);
    }
    _result = _Book.getTotal();
}
return _result;
}
}
}

```

Book.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_3
{
    class Book
    {
        //Attributes
        private List<int> _products;
        private double _total;
        private double _tax;
        private double _TotalPrice;

        //Constructors
        public Book()
        {
            _products = new List<int>();
            _total = 0;
            _tax = 0;
        }

        //Properties

        //Methods
        public void InsertProduct(int products)
        {
            _products.Add(products);
        }

        public void BeforeTax()
        {
            foreach (int products in _products)
            {
                _total = products + _total;
            }
        }

        public void AfterTax()
        {
            foreach (int products in _products)
            {
                _total = products * _tax;
            }
        }
    }
}

```

```

        public void TotalPrice()
        {
            foreach (int products in _products)
            {
                _TotalPrice = products - _total;
            }
        }

        public double getTotal()
        {
            return _total;
        }

        public double getTotalPrice()
        {
            return _TotalPrice;
        }
    }
}

```

CD.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_3
{
    class CD
    {
        //Attributes
        private List<int> _products;
        private double _total;
        private double _tax;
        private double _TotalPrice;

        //Constructors
        public CD()
        {
            _products = new List<int>();
            _total = 0;
            _tax = 0.10;
        }

        //Properties

        //Methods
        public void InsertProduct(int products)
        {
            _products.Add(products);
        }

        public void BeforeTax()
        {
            foreach (int products in _products)
            {
                _total = products + _total;
            }
        }
    }
}

```

```

    }
}

public void AfterTax()
{
    foreach (int products in _products)
    {
        _total = products * _tax;
    }
}

public void TotalPrice()
{
    foreach (int products in _products)
    {
        _TotalPrice = products - _total;
    }
}

public double getTotal()
{
    return _total;
}

public double getTotalPrice()
{
    return _TotalPrice;
}
}
}

```

DVD.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Exercise_3
{
    class DVD
    {
        //Attributes
        private List<int> _products;
        private double _total;
        private double _tax;
        private double _TotalPrice;

        //Constructors
        public DVD()
        {
            _products = new List<int>();
            _total = 0;
            _tax = 0.12;
        }

        //Properties
    }
}

```

```

//Methods
public void InsertProduct(int products)
{
    _products.Add(products);
}

public void BeforeTax()
{
    foreach (int products in _products)
    {
        _total = products + _total;
    }
}

public void AfterTax()
{
    foreach (int products in _products)
    {
        _total = products * _tax;
    }
}

public void TotalPrice()
{
    foreach (int products in _products)
    {
        _TotalPrice = products - _total;
    }
}

public double getTotal()
{
    return _total;
}

public double getTotalPrice()
{
    return _TotalPrice;
}
}

```

Exercise 4 code

FINAL GRADE

58/100

GENERAL COMMENTS

Instructor

Knowledge OOPs concepts (30) 23

Discussion of design principles (10) 7

Knowledge of testing (10) 6

UML (20) 11

Exercise 1 (5) 4

Exercise 2 (5) 5

Exercise 3 (10) 2

Exercise 4 (10) 0

Classes vs Objects: very nice explanation.

Inheritance and Polymorphism: again very good, although more discussion of the subdivision of code should have been provided

Interfaces: very good explanation. You could have talked a little more about the abstraction allowing better subdivision of the code, although it is implied.

Dependency Inversion: you have provided some interesting points but there is not much discussion.

Some example code would have helped.

Software Testing: you have briefly covered white, black and unit testing. But you could have outlined more of the benefits of these approaches.

UML: you have the basic idea, but because you didn't have a class diagram the sequence diagrams don't quite make sense.

Ex1: There is a `_totalSale` attribute which is not required

Ex3: You needed to use inheritance or an interface and store a list of products in the Basket class. Just add CDs, DVDs,, etc to this list because they are all types of Product then just use a foreach loop to read them all. No need for multiple lists.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

nice explanation

PAGE 6



Comment 2

Very good point



Comment 3

Nice explanation

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11



Comment 4

these should be method names



Comment 5

need a class diagram

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18



Comment 6

Not required

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25
