Differences between Object-Oriented Programming and Procedural Programming using C# and Python

Trac Duc Anh Luong - 103488117

August 19, 2022

1 Abstract

This research was conducted on the topic of finding contrast perspectives between OOP and Procedural Programming. To give a practical view on the 2 concepts, we built a small program step-by-step that implemented each concept respectively. The program was a simple calculator that can do 4 basic mathematical operations, including addition, subtraction, multiplication, and division. Testings were done based on the performance and output of each programming structure, with detailed analytics based on the data collected. Overall, the calculator that applied Procedural Programming provided better numbers in terms of efficiency. The research's discussion section was written based on observation on the program structures and ease of implementing new features into the program. From that, this paper would provide an extensive insight on the industrial use of OOP and Procedural Programming. While OOP proved its application in numerous computer technology segments, Procedural Programming served as a foundation to the field of programming and the 2 programming methods can be used in correlation with one another.

2 Introduction

This paper will include research and findings on the topic of differentiating between 2 popular programming approaches: Object-Oriented and Procedural. As of the time that this paper is being written, many programming languages have already provided support for OOP, including Ruby, Python, Java, C++, C#, JavaScript, etc. The latest technological trends are also gravitating towards Object-Oriented Designs, which the pattern can be found in many pieces of software and systems. Even though the aforementioned languages also support procedural programming, its application tends to cover a smaller area of technology, which will be further discussed in this paper.

3 Method

The research methodology used for this article will be an analytic method, as we explore the differences between two programming strategies using a small program that is developed from both perspectives, to find out which one is more optimized using standard programming metrics, including code efficiency and structure, complexity and runtime, and scope of implementation.

For this research, we will create a simple console calculator program that can be used to do 4 mathematical functions of adding, subtracting, multiplying, and dividing between 2 numbers that are passed in within the program.

3.1 Object-Oriented Programming

This demonstration was carried out using a highly Object-Oriented language, which is C#. Below is the UML diagram used for this demonstration.

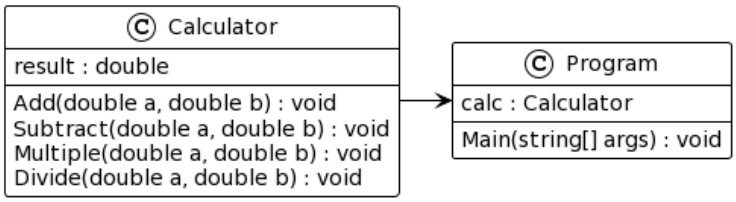

Figure 1: Class diagram

The code is as followed:

```
using System;

namespace Calculator_OOP
{
    class Calculator
    {
        private double result;

        public Calculator()
        {
            result = 0;
        }
```

```
        public double Add(double a, double b)
        {
            result = a + b;
            return result;
        }

        public double Subtract(double a, double b)
        {
            result = a - b;
            return result;
        }

        public double Multiply(double a, double b)
        {
            result = a * b;
            return result;
        }

        public double Divide(double a, double b)
        {
            result = a / b;
            return result;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Calculator calc = new Calculator();

            Console.WriteLine(calc.Add(1, 5));
            Console.WriteLine(calc.Subtract(7, 3));
            Console.WriteLine(calc.Multiply(6, 2));
            Console.WriteLine(calc.Divide(8, 4));
        }
    }
}
```

The output should be:
6
4
12
2

Explain: Firstly, we created a Calculator class that contained the main methods that were used as the 4 mathematical operators, including addition, subtraction, multiplication, and division. Secondly, we declared the variable "result" as a double, set its visibility to private, and initiated a Calculator object to assign the value of result to 0. Thirdly, we created 4 methods that were used to handle the logic behind the program, each took in 2 parameters of a and b (which the value type was double), did the math and returned the result.
Next, we move to the Program class, which contained the Main() method. Firstly, we create a new "calc" object using the Calculator class. Secondly, we used the object to call its constructors, and passed in different values for each mathematical functions. Thirdly, we used the Console.WriteLine() method to print the results to the screen.

3.1.1 Runtime Test

To record the runtime, we used the System.Diagnostics Namespace. First, we accessed the package.

```
using System.Diagnostics;
```

Second, we created a Stopwatch local variable named "MyTimer".

```
Stopwatch MyTimer = new Stopwatch();
```

Third, we used the Start() and End() method to wrap around the code that led to execution and record the time. The code should be as followed:

```
MyTimer.Start();

Calculator calc = new Calculator();
Console.WriteLine(calc.Add(1, 5));
Console.WriteLine(calc.Subtract(7, 3));
Console.WriteLine(calc.Multiply(6, 2));
Console.WriteLine(calc.Divide(8, 4));

MyTimer.Stop();
```

Finally, we printed the execution time out to the screen.

```
Console.WriteLine("Duration for OOP: " + MyTimer.Elapsed);
```

3.1.2 Build Test

For testing, we used the Windows PowerShell. As this terminal supported measuring command runtime, we utilized this feature and ran the following command:

```
>(Measure-Command {csc .\speedtest_oop.cs}).ToString()
```

3.2 Procedural Programming

For procedural programming, we used Python, a beginner-friendly programming language that is often used to work with procedural programming. The code is as followed:

```
def add(a, b):
    result = a + b
    return result

def subtract(a, b):
    result = a - b
    return result

def multiply(a, b):
    result = a * b
    return result

def divide(a, b):
    result = a / b
    return result

print(add(1, 5))
print(subtract(7, 3))
print(multiply(2, 6))
print(divide(8, 4))
```

The output should be:
6
4
12
2.0

Explain: First, we defined 4 functions in relation to its 4 operator counterparts. These functions had a parameter that took in 2 integers of a and b, did the mathematical logic, and return the result. Secondly, to get the result, we passed in 2 values to the parameters of each function. Thirdly, we called the print() function to print out the result.

3.2.1 Runtime Test

For this test, we utilized the datetime module to record the time taken to run the program.

```
from datetime import datetime
```

We will wrap the datetime.Now() function around the block of code that is used to run the program, store them in 2 values called start_time and end_time. Next, we take the end_time value, subtract it to start_time and print out the result. The code should be as followed:

```
start_time = datetime.now()

# block of code

end_time = datetime.now()
print('Duration for PP: {}'.format(end_time - start_time))
```

3.2.2 Execute Test

Similar to the build test for OOP, the execution test was also conducted using the Windows PowerShell. The command should be as followed:

```
>(Measure-Command {python .\Calculator_Procedural.py}).ToString()
```

4 Results

After running a total of 20 tests, 10 tests for each program, here is the data that we collected.

Note: The time was measured in milliseconds.

Test	OOP	Build	PP	Execute
1	51.64	206.91	46.15	168.88
2	57.19	280.84	61.29	116.49
3	59.59	189.60	41.42	96.47
4	59.27	218.69	22.10	111.43
5	56.58	212.66	28.04	108.72
6	36.75	226.53	24.74	171.40
7	23.12	221.33	20.28	155.90
8	59.43	227.48	44.05	113.22
9	36.34	204.51	68.19	112.01
10	60.46	212.58	66.27	161.08
Average	50.04	220.11	42.25	131.56

Table 1: Runtime test

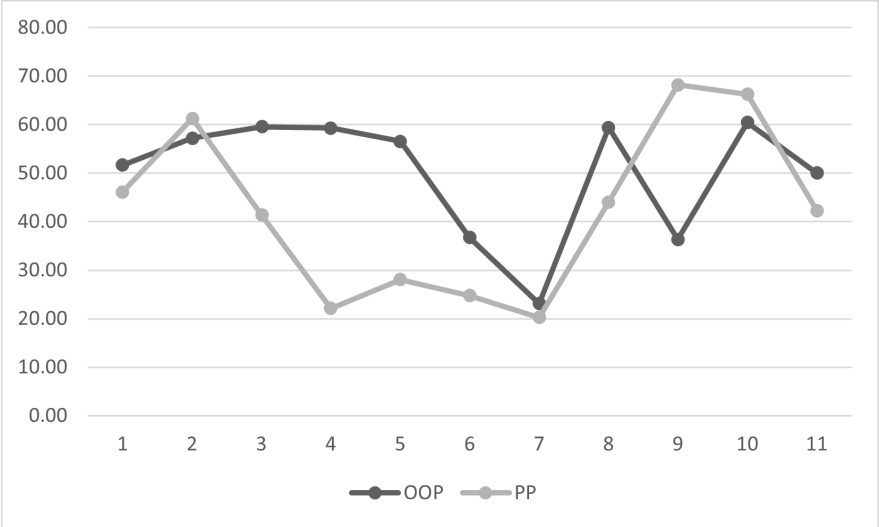

Figure 2: Performance based on runtime

5 Discussion

From observation, some speculations can be made about the 2 types of programming structure. Both structures have their own unique features, pros and cons that should be discussed with detail in this section.

Looking at the program code base, we can see that procedural programming took up less time as the code base is smaller in comparison to object-oriented programming, curbing the

time required to develop pieces of software and systems. In OOP, the structure is stricter, as class, field and method declaration is required to encapsulate data and behavior into objects. Even though the scale of the program can be small, this structure is always required, making efficient object-oriented design challenging to acquire. In contrast, procedural programming focus on following a top-down logic. The key aspect of procedural programming is the way that you work with your device, as you directly give instructions to the computer to execute every task step-by-step. This can be useful for web development, as procedural logic is widely applied in client-server systems and back-end platforms. The key attributes of procedural programming makes it a beginner-friendly concept that is used for educational purposes and serve as the foundation to learning other technological paradigm, including OOP as well. Some popular languages that are used for procedural programming include C, Pascal, and Java.

From the Results section, we can see the apparent differences between OOP and procedural programming. When comparing pure runtime, procedural programming was slightly faster, taking an average of 42.25 milliseconds to run, while its counterpart took approximately 50 milliseconds to run. However, we do need to note that we are using 2 different languages for testing here. While OOP is using a compiled language (C#), procedural programming is using an interpreted language (Python). Therefore, the runtime can varies due to each language different features and behaviors when translating into a machine language and running on the CPU. If compiling and execution time had been taken under consideration here, procedural programming was still more efficient, registering at an average of 131.56 milliseconds to run, while the counterpart took 220.11 milliseconds.

In terms of program structure, OOP seemed to triumph over procedural programming. Developers can implement various design patterns that falls under the category of creational, structural, or behavior patterns. These design patterns can enhance not only the modularity but the relationship among difference classes of the program. OOP provide access specifiers, including public, private, protected, etc. In terms of security, OOP is more secure with data hiding, while there is no efficient way to hide data in procedural programming. In the OOP Calculator program, we can see the declaration "private double result;". This is an implementation of data hiding, where the result value can only be interpreted via specific method calls. In contrast, the procedural program return the result after interpreting the a and b parameters. However, other parts of the program outside the parameter can assign different values to the result variable, making the program less stable.

When it comes to industry application, OOP is also the more favorable, as its application takes presence in various sectors of technology, including client-server systems, databases, modeling systems, automation, machine learning, AI, etc. In the field of software develop-

ment, the extensibility of a program is important for development and maintenance through updates, keeping a program relevant in the marketplace. Using an OO approach, we can easily add data and different features to an object in a program. On the hand, adding features will be challenging, as we have to get through various logical gates, guard clauses, and reduce code maintainability. Another programming principle that is key for this comparison is code reusability. Duplicating lines of code can cause confusion to other program collaborators and be time-consuming in the process of iteration, therefore, a well-designed program should have reusable methods to reduce coupling and enhance cohesion.

6 Conclusion

To conclude, both Object-Oriented Programming and Procedural Programming has its own unique features and implication in the world of programming. For OOP, this is an important concept as its scope of implementation widely covers multiple aspects of computer technology, from software development, system building, to machine learning and AI. The code reusability and extensibility facets of OOP designs make them an industrial standard to every developer. Data can easily be added, manage and protected with this form of programming. Furthermore, we can implement various design patterns to enhance the program structure and connection between its classes and functionalities. On the other hand, Procedural Programming is a fundamental concept that follows clear logic and gives direct instructions to the computer. This programming concept is often used for educational purposes, as it provides introductory knowledge to the field of programming. Both programming approaches are crucial and can be used interchangeably.