Object-Oriented Programming vs. Functional Programming

Research Report

Lucas Gehlen

Fontys University of Applied Sciences

Venlo, January 14, 2019

Summary

In the programming world there are to big programming paradigm. OOP and FP both are different and have their own advantages and disadvantages. But were exactly are difference between these two paradigm. Which paradigm solves which problem better? In this research I take a brief look at these two paradigms and show the main differences and try to conclude which paradigm is better.

Contents

List of Listings

List of Abbreviations

SoFa Software Factory

FP Functional Programming

OOP Object-Oriented Programming

1 Introduction

In this research report the topic of object-oriented programming vs. functional programming is handled. OOP and FP both are programming paradigm that are used to create software in today's world. Currently there are programming languages that were mainly developed for OOP like *C++*, *C#* and *Java* and programming languages that were mainly developed for FP like *Haskell*, *Clojure* and *Scala*. There are also Programming languages that support both OOP and FP like *JavaScript* and *Ruby*.

But what are the differences between these two paradigms and what are the main use cases for them?

In the following chapters, first the research was planned. Therefore the research questions were created and the research methods were explained.

The next chapter deals with the research results of this research. In the results both paradigms were compared to each other and the advantages, disadvantages and use cases for them were found.

Finally, the report ends with the conclusion of this research. In the conclusion all the results are reflected and the answer for the question which programming paradigms is better will be given.

2 Research Definition

This chapter explains why this topic was researched and which method was used to complete the research. First the motivation for this research is given. After that the Research Questions are listed to show what questions this research tries to answer. Finally the last chapter deals with the methods that were used during this research.

2.1 Research Motivation

This research is part of the Software Factory Project *Connected.Football* at Fontys Venlo. *Connected.Football* is a mobile application developed in *React-Native*. *React-Native* is a front-end framework developed by Facebook. The goal of the Software Factory Project was to add new features to the *Connected.Football* application.

At the start of the SoFa project the project group realized that the *Connected.Football* application was developed by using the programming paradigm 'Functional Programming'. Therefore the students had to learn about FP to understand the application and to add new features.

This research was done to compare FP and OOP against each other and to conclude in which use case FP is better and in which use case OOP is better.

2.1.1 Research Questions

For this research the following research questions were defined:

1. What are the differences between OOP and FP?
2. Which paradigm is better?

2.1.2 Research Methods

To answer the research questions there are several methods that can be used to complete this research. This research is based completely on literature research.

Literature that is used will show how both paradigm work and what the main differences are.

3 Research Results

In this chapter the results of the research are presented. First the two programming paradigm OOP and FP will be described and explained how they work on their own. The second chapter then deals with the differences between both paradigms. Finally the last chapter focuses on the advantages and disadvantages of OOP and FP.

3.1 Object-Oriented Programming

"*As the name would imply, object oriented programming is all about, big surprise... objects.*" (Thomas De Leon 2012).
In the programming paradigm OOP everything is about objects. In most OOP languages objects are instances of a class that can interact with each other. These objects contain both data and the functionality that executes operations on that data.
OOP has four main features:

1. Abstraction
   It helps in letting the useful information or relevant data to a user, which increases the efficiency of the program and make the things simple.
2. Inheritance
   It helps in inheriting the methods, functions, properties, and fields of a base class in derived class.
3. Polymorphism
   It helps in doing one task in many ways with help of overloading and overriding which is also known as compile time and run time polymorphism respectively.
4. Encapsulation
   It helps in hiding the irrelevant data from a user and prevents the user from unauthorized access.

(educba 2019)

3.2 Functional Programming

FP focuses on the computation of pure functions and tries to avoid the concepts of states and mutable data. For FP only the values that are passed to the function matter. The function should always return the same results with the same values. These functions doesn't depend on local or global states.
"*Functional programming is a declarative paradigm because it relies on expressions and declarations rather than statements.*" (Cody Arsenault 2017) Not being depend on states makes software created with FP far more predictable.

3.3 Differences

To compare the differences between OOP and FP an example in Ruby is given that uses an array of cars. The goal is to reduce the price of every car in this array by 1000$.

3.3.1 Car Example OOP

```ruby
class Car
    attr_accessor :brand, :color, :price

    def initialize(brand, color, price)
        @brand = brand
        @color = color
        @price = price
    end

    def increasePrice(priceToIncrease)
        @price += priceToIncrease
    end
end

volkswagen = Car.new("Volkswagen", "Grey", 15000)
opel = Car.new("Opel", "Green", 10000)
bmw = Car.new("BMW", "White", 30000)

carArray = [volkswagen, opel, bmw]
```

Listing 1: Class Car

In OOP the first thing that has to be done is to create a class. In this example the class is called *Car*. The class *Car* in this example has the attributes *Brand, Color and Price* that will be initialized using the constructor of the class *Car*. To increase the price of a car later a method was introduce that allows to increase the price of a car by a given value. After that three objects of the class *Car* are created and initialized with new and packed into an array. The next step would be to increase the price of every car by 1000$. Therefore we have to iterate over all the cars in the array and call their *increasePrice* method and pass it a value of 1000.

```ruby
carArray.each do |car|
    car.increasePrice(1000)
end
```

Listing 2: Increase Price

Now all the cars in the *carArray* have their price increased by 1000.

3.3.2 Car Example FP

To recreate the car example in FP, first an array of arrays with the data is created.

```ruby
carArray = [
    ["Volkswagen", "Grey", 15000],
    ["Opel", "Green", 10000],
    ["BMW", "White", 30000]
]
```

Listing 3: FP Car

Because FP doesn't want to mix data and behaviour there will be no class that stores the data from the cars. So instead of creating an object of cars and use their internal methods,

standalone methods are introduced that take over that part.

```
1 def increasePrices(carArray, price)
2     carArray.each do |car|
3         car[2] += price
4     end
5 end
6
7 increasePrice(carArray, 1000)
```

Listing 4: FP methods

The standalone method *increasePrices* takes as parameters an array and how much the price should be increased. In this method we also could put in an other array as long as its similiar to the carArray.

4 Conclusion

Answering the question of which is the better paradigm OOP or FP is quite hard. Because each paradigm has its own advantages and disadvantages and both try to deliver bug-free, easy to understand and manageable code.
OOP has the advantage of inheritance that allows the programmer to reuse code and encapsulation that bounds methods and data to an object and therefore makes it easier to manage and change the object. Also in OOP state changes are normal a specific input does not always lead to the same output.
FP on the other hand is great for multi-threading because of the immutability. For a specific input in FP there will be always the same output. Also the separation of data and methods leave less room for errors in FP.
Therefore I come to the following conclusion after doing this research. OOP and FP both have their advantages and disadvantages. Both have their use cases and topic where they shine. So there is no possibility to say one paradigm is better than the other. The final answer to this question is it depends on the use case.

References

Cody Arsenault (2017). *Differences Between Functional Programming vs OOP.* Available at: `https://www.keycdn.com/blog/functional-programming` [Accessed 11 Jan. 2019].

educba (2019). *Differences Between Functional Programming vs OOP.* Available at: `https://www.educba.com/functional-programming-vs-oop/` [Accessed 10 Jan. 2019].

Thomas De Leon (2012). *What is Object Oriented Programming? – A Basic Explanation.* Available at: `http://teknadesigns.com/what-is-object-oriented-programming/` [Accessed 10 Jan. 2019].