Worksheet 4
Object Oriented Programming

Josh Wainwright
UID:1079596

Object oriented programming (OOP) is defined as

> *using a methodology which enables a system to be modelled as a set of objects which can be controlled and manipulated in a modular manner.*[4]

This means that complex situations can be broken down and implemented in manageable pieces and allows code to be grouped into semantically related sections.

Object oriented programming is designed to address the issue of code re-use in larger programming projects. Even at the smaller level, re-implementation of code wherever it is needed leads to problems, for the developer, the maintainer and, ultimately, the end user.

This conceptual model of programming offers a number of advantages over other programming methods and models, such as functional and procedural programming, as well as a few disadvantages.

1 Advantages

1. Clear modular structure[1].

   A program written using OOP is forced to be clearly defined in the way that it is structured. There can be no ambiguity regarding the classification of parts of the code since it must be understandable by the compiler and hence organisation is a requirement.

   This organised structure applies to all programming languages, but when using OOP, it is enforced more strongly since every new type that is created must have rules applied at creation time, thus enforcing the use of that type to the purpose originally intended.

2. Easier maintenance and modification of code.

   Since the modular structure keeps related code together and allows a simple and clear hierarchy within the program, it is much easier to get an understanding of the whole program through high level descriptions of the code, rather than having to understand the workings of every line.

   This improves the ability to go back to previously written code, or another's code and edit it without having to understand how a particular job is achieved, but simply knowing the interface that is used to achieve it. From this, it is then also easier to upgrade software, or make improvement, since a method or action can be changed without needing to know what makes use of it, just what they expect the results to be.

   For example, consider the case of calling a method that accepts an array and returns a sorted copy of it from a main program. The particular way that the array is sorted is of no consequence to the program and so it can be modified to use any other sorting algorithm without that programming knowing, or needing to know, that anything has been changed.

3. Faster and cheaper development[2].

   Following from the first two points, the development of programs making use of OOP will naturally be faster to build a substantial code base, and cheaper to achieve it since code does

not have to be rewritten and so less developer time is required, and code modifications are easier, meaning that upgrade and bug fixes are simpler.

2 Disadvantages

Though OOP is widely used, there remain many languages that do not make use of it and instead prefer to use functional or procedural structures. The most widely used object oriented languages are C++ and Java, and some use these as examples of how OOP is non-beneficial[5].

1. Slower programs[3].

   During the translation from written code to binary execution, OOP generally requires more machine level instructions than functional programing equivalents and so, even when not using a virtual machine language like Java, the program is likely to run more slowly. This is usually because of the use of many small objects that leave the codebase fragmented.

   This is also generally the case for the written code before compilation, meaning that object oriented projects often require more lines of code.

2. Steep learning curve to learn languages.

   Since the structure of the program must be broken down so finely, the learning curve required to start programming with and OO language is generally higher than for a procedural language. This is particularly a problem when a one-off project requires that use of OOP since the developer, if they are not already familiar with OOP, is unlikely to want to spend time learning the programming paradigm as well as the language, and so the result might be substandard quality code.

3. There are several other more specific disadvantages to do with the particular language implementation that comes from OOP. For example, the fact that when using OOP, everything must be an object, i.e. a "time" must be represented by an object, whereas a non-OOP language can represent a time as an instance of a data type.

   Also, the essential linking of data and functions can be confusing. Data structures hold information, functions take an input and perform operations, often, it is not necessary, or desired, to have these two linked.

There are many and varied opinions on the virtues and failings of object oriented programming. However, it is likely to remain in widespread use simply because there are some programming problems that are well solved using it, just as there are some problems that are better solved using functional programming, procedural programming, logical programming, machine code, literate, declarative, imperative, mathematical or symbolic programming paradigms. The issue is not with any one of these in particular, but the implementation and the use to which it is put.

References

[1] Martin Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1996.

[2] Scott Ambler. A realistic look at object-oriented reuse. `http://www.drdobbs.com`, 1998.

[3] L. Cardelli. Bad engineering properties of object-orient languages. *ACM Comput. Surv.*, 28(4es), December 1996.

[4] Oxford Dictionaries. `(http://www.oxforddictionaries.com)`, 2013.

[5] Paul Graham. Why arc isn't especially object-oriented. `http://www.paulgraham.com/noop.html`, 2006.