Rationale for using Object Orienting Programming

What is object oriented programming?

Object oriented programming is a technique of computer programming through the use of objects. An object is a package of data structures and methods/algorithms encapsulated into a single structure. These packages/objects can be linked within the code through inheritance which allows interaction with one and other, but one object can not directly change the state of another object, the only way to change the state of an object is to access the methods within that object. For example if we want to manipulate the state of object C, we call the methods within object C. Methods from outside object C, can not manipulate the state of C directly, the methods within C have to be called. This as I will discuss later leads to secure coding. Objects are chosen using noun verb analysis of real world situations. This is one of the reasons object oriented programming became popular, as it was believed to be easy for programmers to represent real world environments within well structured code. Nouns are used to create class', which are the blueprints for the objects, and the verbs are used to create the methods that manipulate the state of an object. Some programmers no longer believe this to be the case as I will discuss later.

Problems addressed with object oriented programming

Before object oriented programming many developers used procedural programming techniques. Procedural programming results in the entire program being written in one long procedure, it may contain functions and subroutines making it more modular and maintainable, but the code itself is just a combination of data and methods in one long executable code. This results in difficulty when expand existing systems. New code is likely to conflict with previously written code, causing developers to have to modify previous code in order to get the system to run. Editing existing code is likely to introduce additional errors within the code, as well as taking time to initially locate the conflict. This means a developer will waste lots of time fixing conflicts which they introduce with new code, this is therefore not an efficient way to program. The final code is also likely to still have some undiagnosed vulnerabilities that the developer has missed. Object oriented programming attempts to reduce the conflicts introduced when expanding systems.

Object oriented programming allows expansion of a program through the addition of new objects, these new objects interact with previously coded objects through inheritance. This in theory means previous code does not need to be modified in order to provide additional functionality for a system, provided the initial objects have the correct functionality. This ease to expand current projects, also means that a developer does not have to have excessive knowledge of the previous code within a system before they can add additional functionality. In other coding techniques, for example procedural, a developer may need to understand the majority, if not all the code within the project before adding additional functionality.

A good example to represent how object oriented programming provides easier expansion is the setup of a car system. Say for example a car was developed with no display for the speed, to add this functionality within a functional programming language, many parts of the system that deal with the cars speed may have to be modified. If the same addition is introduced into a car coded in an object oriented language, one additional object, the display would need to be introduced through inheritance to the speed object. [1]

Object oriented expansion is attractive to companies, as large projects can be split into smaller sub-projects, which can be designated to several different teams of developers. This reduces the completion time for a project and makes it easier to delegate sections for developers to produce. The final system will combine all the teams sub-projects, through inheritance, and will be less likely to create conflicts within the code as the state of each object can only be modified by itself. This resulted in object oriented languages being used widely throughout large companies, and with many programmers employed to code using the technique, its popularity accelerated at an exponential rate.

Encapsulation within object oriented programming results in secure code. The field variables which describe the state of an object can only be modified through the use of methods stated within that object. Therefore methods outside of an object can not modify the state of the object directly, making the state of an object secure to outside modification. This is a feature that has to be programmed into other programming languages by the developer. This could result in potentially vulnerable code due to developer errors, whereas within object oriented programming this feature is part of the structure of the code and therefore can't be overlooked. These securities mean future code will be less likely to break past code, therefore developers will spend less time fixing errors created through conflicts between sections of code. Software attacks from outside

sources are also less likely to find vulnerabilities within the code to exploit, as only methods implemented by the developer can influence the state of each object.

Remaining Issues

The assumption that everything in the real world can be represented as an object that can be related to another object is not always true. In a simple example such as Animals object, with a relation to Cat objects, Dog objects etc. there is a clear relation that can be easily implemented within a object oriented language, but if for example you have a Factory object, related to Factory Manager object should that Factory Manager really be related to the Factory object or should they be related to a Person object. If they should be related to both, this suggests that Factory should be related to Person. This within the real world doesn't make sense. Within small applications, objects are fairly easy to relate, but at a larger scale it may become difficult for the developer to determine where within the tree of inheritance an object should be placed. This can actual create greater complexity within the system as many objects have to be related through additional abstraction objects.

Object oriented programming is a technique that prioritises data before functions/ methods. In reality there is no link between functions and methods, a function acts on data but does not necessarily need to be linked to the data type in the real world. The use of encapsulation, combines data types and functions into one each object. This means many functions which have no relation to one another are encapsulated into one object, due to their interaction with the data structures within that object. There is an argument that data types and methods should be completely separate, as the same function can act on many different types of objects. Within object oriented programming objects using the same functions have to be linked through inheritance, this causes many objects to rely upon one another to run the final system. If a developer was writing a new piece of software and decided to implement something from their previous code, they may assume they need to just implement one class, but because each class relies on many other parent class' the developer eventually copies over half the previous system, resulting in complex and abstracted code. The idea that object oriented coding results in simplistic and elegant code, is simply not true in large scale systems.

Another issue with object oriented programming is assuming everything in the real world can be represented to as an object. For example time has to be represented as an object, within other languages time is represented as a data type and can be manipulated by any function within the system. As only the time object can manipulate the time data type, the object will need to have all possible functions within the object and relate to all objects that would like to interact with it through inheritance. This again creates large levels of complexity within the heir-achy of the system that need not be there.

Conclusion

An object oriented language creates a good basis to represent real world scenarios within a system. An object oriented system is easier to maintain and expand than previous systems relying on coding techniques, such as procedural language, although at a large scale the systems structure can become increasingly complex with the reliance on class' needing to be linked through inheritance. Object oriented programming makes it easier for developers to expand previous systems without introducing new conflictions with previous code. Overall I believe object oriented programming is a progression to procedural, as it allows developers to extend systems more easily but the limitations within object orientation for the developer means I believe it is not the final progression for programming techniques.

References:

[1] IEEE Transactions On Software Engineering, Vol. SE-12, No.2
"Object-Oriented Development", Grady Booch, (1986)

[2] Object-Oriented System Analysis: Modelling the World in Data
Sally Shlaer & Stephen J. Mellor, (1988)

[3] Java How to Program: Early Objects, 10th Edition
Paul Deitel & Harvey Deitel, (2014)

[4] C Traps and Pit falls falls
Andrew Koenig (1989)