Rationale for the Use of Object Oriented Programming

This report evaluates the Object-Oriented Programming paradigm (OOP henceforth) and describes why it should be adopted. This will be done by first considering the problem of maintenance that many software projects will have to face. The report will then give an overview of what OOP is and how it can be used to address the problem effectively. The report will also describe some of the drawbacks that OOP has before concluding that the benefit of using OOP can outweigh its disadvantages.

One of the biggest problem software projects have to face is the maintenance process. It was estimated that the maintenance cost may be as high as 67% in the overall software development life-cycle whereas the actual cost of programming can be as low as 12% (Schach, 1999; Henry and Humphrey, 1988). Projects getting pushed for premature release due to tight time constraints will face even bigger challenges when it comes to maintaining since less measures will be made to ensure that the code can be easily fixed and/or extended at a later date. OOP can and should be adopted to overcome this problem.

OOP is a programming paradigm that revolves around the interaction of “objects”. First, this allows us to talk to different objects, executing various methods related to them without having to know how the objects actually perform those operations. This forms an encapsulation. Furthermore, we can factor out common functionalities that may be present in two different objects and form an abstraction that may be implemented by more objects in the future. This allows us to generalise aspects of the code in a very natural way. OOP also allows us to specialise part of the codes via the use of inheritance whereby one object can inherit behaviours from another, gaining the properties that the parent object has, as well as having its own specialised behaviours that only the child(ren) have access to. Different objects may also perform similar operations, but instead of creating multiple functions that have slightly different names which may cause confusion later on in the development process, objects can share methods with the same name, but implemented in a different way according to how each of the objects should execute that same operation. This is called polymorphism. These facets of OOP together supports a very powerful concept of the separation of concerns. This report will consider how this concept can help OOP reduce maintenance costs.

We achieve separation of concerns when different objects only concern themselves and nothing else. This turns out to be a very effective way to organise one’s thoughts (Dijkstra, 1982). It allows an object to focus sorely on its raison d’etre, any other objects can then fully rely on this object for the tasks it offers. Furthermore, separation of concerns meant that should one aspect of the code break, we will know immediately the whereabouts of the piece of code that is responsible for breakage since there will only be one object that is dedicated to the task that causes it. This reduces the cost of corrective maintenance. In addition, should we also wish to extend or add new features into the software, we will also know where the new piece of code should go depending on its functionality that identifies which object we will be dealing with. This reduces the cost of adaptive maintenance.

OOP can also help reduce not only the maintenance cost in particular, but also the whole development cycle in general through code re-use. If different objects concerns only a particular set of related tasks and nothing more, then in particular, they are also separate from the rest of the interface of the software project they were originally created in (objects dealing with business logic

should be independent of objects dealing with the presentation layer). This means that we can extract these objects that performs useful operations that we've already written and use them in other projects as well. The amount of work is then reduced to only providing the suitable interface specific to the project that will interact with the objects we already have. This is code re-use and OOP allows us to carry it out in a natural way. Furthermore, if each objects concerning different tasks were built effectively, we will gradually accumulate libraries of classes that can also be shared within the community and everyone will benefit from the code re-usability each of us obtain from using OOP.

Having looked at how OOP can address the software maintenance problems, we should also consider its disadvantages. One of the disadvantages of OOP is that it introduces an extra overhead of having to model the problem using objects and classes. A particular experiment have been conducted and the results showed that the hours spent debugging problems in OOP is higher than analogous problem in non-OO languages (Turner, 2004). So in this case, using OOP does not necessarily make problem solving easier, especially if one is still not proficient in OOP and the paradigm shift it requires. However, this problem is relatively minor compared to the positive change OOP can have towards the maintenance process in the long-run.

Another point to note is that OOP favours design over performance. Developing OO programs shifts the focus towards coming up with a good model and abstraction of the problems involved, while less attention is paid to how fast it runs or which implementation is the most suitable for the task the objects have to perform. However, we should also note that it is also relatively easy in OOP to change the implementation of a particular functions of an object, hence somewhat negating this problem.

In conclusion, we've considered 2 disadvantages of OOP, namely that it does not necessarily make the maintenance task easier compared to other programming paradigms, and that its performance may not be as fast due to more overheads. However, we also see that good OOP can allow us to remedy these problems in the long-run. In particular, OOP allows us to easily re-use time-tested codes and integrate them into different projects by having separation of concerns for each objects. This helps reduce maintenance cost as it allows us to identify problems relatively easily as well as allowing us to enhance and/or add new features to existing objects without having to worry about the concerns of other objects. These are the reasons that we should turn to Object-Oriented Programming.

References:

- Dijkstra, W. (1982). *On the Role of Scientific Thought*. Selected Writings on Computing: A Personal Perspective. New York, USA: Springer-Verlag. p. 60-66.
- Henry, S. and Humphrey, M. (1988). *Comparison of an Object-Oriented Programming Language to a Procedural Programming Language for Effectiveness in Program Maintenance*.
- Schach, R. (1999). *Software Engineering* (4th Ed). McGraw-Hill; Boston.
- Turner, K. (2004). *The Effects of Object-Oriented Programming on Hours Spent in Software Maintenance*.