Object oriented programming

**The abstraction and construction of an object**

Up until the mid 1970s the basic programming methodology in software engineering was known as structured programming. The structured paradigms approach to software engineering were either operation oriented or data oriented. This began to be seen as an oversimplification of software engineering and a new approach to programming began to emerge. One which gathered a lot of attention was object-oriented programming (OOP). In object-oriented programming the software consists of artificial objects, which are created out of abstractions of real world objects. These objects are then enabled to act upon each other. An object can contain both data and execute specific functions, therefore object-oriented programming became very successful in relation to the programming of the structured paradigm, since it provided the programmer with the possibility of combining data and operations.

When making software using object-oriented programming it is necessary for the programmer to choose how to describe the objects of the program. These objects can be either the programmer’s own invention of objects but is often objects which is inspired from real world objects. Objects in the real world have different attributes regarding their appearance and behaviour characteristics. A tree for instance can be described by having a stem and a green crown which describes its appearance, and its behaviour characteristics can be described by its ability to move in the wind. If this tree is to be described as an artificial object or a computer object, the programmer would create a class which consist of these attributes. In object-oriented programming classes is used to construct objects from. The class can be perceived as the objects blueprint. Therefore, a class often consist of two general functions a constructor function and a destructor function. In the constructor function the programmer will initialize the objects attributes. In the instance of the tree this would be the attributes that describe the object visually, and a function that can be called so that the object could potentially move in the wind. Apart from this the destructor function will be initialized so that this function can be called if the programmer wants to delete the given object at one point during the execution of the program. (importance of deleting things) In this process the programmer chooses what he/she determines to be relevant or important attributes to include in the object class. The translation that happens from a real-world object into an artificial object therefor rises some questions about whether or not the act of choosing between different attributes of an object, has consequences for the general operators use of software that is built upon object-oriented programming. Does the fact that the user is presented with a subjective simplification of an object from the real world affect the users understanding of this object in the real world.

In “software studies/ a lexicon” edited by Matthew Fuller, Cecilie Crutzen and Erna Kotkamp have written an article called “Object Orientation”. They argue that object-oriented programming is not only used for developing software but has also become a methodology and theory for analysis of “world of human interaction with which the computer interfaces”, this they call object-oriented approach. They focus on this approach instead of object-oriented programming directly. They argue that human interaction can in the ontology of OO only be represented and modelled as predefined actions, because of the use of abstraction tools as classification, separation and inheritance.

*“Abstraction activities, a fundament of most modeling methods such as generalization, classification, specialization, division, and separation, are seen as unavoidable for the projection of dynamic real-world processes into ready-to-hand modeling structures and for the production of ready-made acting. Abstractions are simplified descriptions with a limited number of accepted properties. They rely on the suppression of a lot of other aspects of the world. Hoare suggests: “Abstraction arises from recognition of similarities between certain objects, situations or processes in the real world and the decision to concentrate upon these similarities and to ignore for the time being the differences.” (p.203)*

Here the concept of an abstraction is not understood as the general conceptual process of thinking where rules and concepts are derived from a specific object. But instead the process in which an object from the real world is abstracted into an artificial object described with a class in software. What we believe the act of abstracting an objects purpose is for, is important to how an abstraction will be made. If the purpose of the abstraction, and the use of object-oriented programming is to ease our ability to understand and live within the real worlds complexity, by obfuscating the underlying processes in software, it might have an impact on how we interact with the software we use in our daily lives. The question is whether or not the fact that we are presented with a simplification of an object or predefined actions in our use of software affects the way in which we understand the object in the real world, or whether we already are unable to fully understand the attributes of an object and we ourselves simplify objects to live in the complexity of the real world.

This paper will examine the history of object-oriented programming, to describe how object-oriented programming started to emerge and what it has become today. It will examine how or if abstractions of objects have change through the progression of the object-oriented paradigm, by analysing the programming languages Simula, Sketchpad and Smalltalk, with this as a foundation the paper will discuss how abstractions is made into objects today in the software we daily use. This paper will try to analyse object-oriented programming on mac OS X and whether or not the objects a user is presented with in the interface has an impact on the user’s interaction with and understanding of the software, discussing how this could have an effect on how people understand objects and abstractions of the real world through software.

**Texts:**

* Jaque, Jason. (2016). The invention of the object – Object orientation and the philosophical development of programming languages. Springer Science+Business Media Dordrecht.

*With this text I will try to analyse how abstractions of objects has changed during the development of object-oriented programming.*

* Cecile Crutzen and Erna Kotkamp, "Object Orientation", in Fuller, op. cit., pp. 200-207.

*This text draws attention to fear, invisibility and doubt regarding object orientation. With this text I might find perspectives on whether or not abstractions of objects have an impact on how we understand the real world. And which consequences if any there is to this.*

* Lee, Roger Y. *Software Engineering: A Hands-On Approach*. Springer, 2013. 17-24, 35-37 (ch. 2 Object-Oriented concepts)

This article describes technically what and how object-oriented programming works. With this article I can begin to understand why abstractions are necessary to make, when using object-oriented programming languages.

* Matthew Fuller, “It Looks Like You’re Writing a Letter: Microsoft Word”, in Behind the Blip!, New York: Autonomedia, 2003;

*I will use this article to put my analysis of Mac OS X into perspective. In this article Matthew Fuller reflects upon how object orientated programming in Microsoft word affects user’s usage of the program.*