Is Extension Complexity a Fundamental Software Metric?

E. Kantorowitz

Computer Science Dept. Technion-israel Institute of Technology
32000 Haifa, Israel
kantor@cs.technion.ac.il

The concepts of implementation and extension complexities were developed in connection with an object oriented reengineering of a legacy CAD system [KA97]. These concepts consider a software system as an implementation of a number of different algorithms. Ideally, an algorithm in an object oriented (O-O) system is implemented by a single method that only processes data in its own object. Such an algorithm is desirable because it cannot cause coupling between different objects. It is possible to design and test the method that implements such an algorithm without knowing anything about the other object types (classes). The costs of implementing and testing such an algorithm are therefore usually modest. Higher implementation and testing costs may be expected in algorithms that process data in a number of objects belonging to different object types. The reason is that the implementers must have the data structures of these different object types in their mind while implementing the algorithm. Furthermore, an algorithm involving different object types will typically employ a number of different methods in these different object types. The implementers must therefore also consider all of these different methods during the implementation. It may therefore be assumed that the cognitive load on the mind of the implementers is generally higher with algorithms involving a number of different object types. The costs of implementing and testing an algorithm are therefore assumed to be related to the number of different object types involved. In order to express this the following definitions were introduced:

The *domain of an algorithm* is the set of all the object types employed by the algorithm.

The *size of the domain of an algorithm* n is the number of different object types in the domain of the algorithm.

It is recalled that one of the basic principles of OOP is to model the problem domain as precisely as possible. The construction of such a model is typically based on an analysis of all the envisioned use cases [JA92]. The set of object types identified in this analysis represents therefore a model of the domain that is expected not to be biased toward any one of single use case. The definition of the size of the domain of an algorithm, that is based on a count of such object types, is therefore expected not be biased toward any one of the compared algorithms. We define:

The *implementation complexity of an algorithm* is an indicator of the number of code segments required to implement the algorithm as function of the size of the domain of the algorithm. A code segment may be a class, a function, or any other unit of code. We do not look for an accurate estimate of the required amount of code, as may be achieved by an elaborate time consuming software metrics analysis [FE91]. Similar to space and time complexities of algorithms the implementation complexity may be considered as a kind of a classification for implementation costs.

The *extension complexity of an algorithm* is a measure for the number of code segments required in order to extend the domain of an algorithm with a single new object type.

The *extension complexity of a software system* is the highest algorithm extension complexity found among the algorithms that the system implements.

The chang propagation algorithm of the legacy system of [EK97] had an extension complexity of O(N), and it was therefore not practical possible to extend it beyond N=9. Extending it to N=10 required 9 new code segments and so on. The new algorithm in the new system had extension algorithm O(1), i.e. the effort required to extend the domain of the algorithm with one object type is roughly constant and independent of N. The system could therefore be quite easily extended to the required N=75. The above example illustrates that an extension complexity of O(1) is desirable, while an extension complexity of O(N) is usually undesirable as it means that it becomes increasingly more difficult to extend the algorithm as N grows. We employ therefore the notion:

A *simple algorithm* as an algorithm whose extension complexity is O(1).

It has been proved that an algorithm having an extension complexity O(1) (a simple algorithm) has an implementation complexity of O(n). It can on the other hand be shown that an implementation complexity of O(N) does not imply that the extension complexity is O(1). The extension complexity is in this sense more fundamental concept than the implementation complexity. This theoretical result supports the feeling that the ability to be easily extended (to be simple in the above sense) is possibly the most important single characteristic of software quality.

References:

[FE91] N.E. Fenton, 'Software Metrics a Rigorous Approach', Chapman Hall, New York, 1991

[JA92] I. Jacobson, M.Christerson, P. Jonsson, and G. Overgaard, 'Object Oriented Software Engineering', Addison Wesley, Reading. Massachusetts, 1992

[KA97] E. Kantorowitz, 'Algorithm Simplification Through Object Orientation', Software Practice and Experience, 27(2), (Feb. 1997), 173