SWE 212 Analysis of Programming Languages

Week 4 Summary of 'Programming with Abstract Data Types' by Barbara Liskov and Stephen Zilles

Samyak Jhaveri

The authors of the paper describe an approach to abstraction in a new structured programming language which can also be implemented in high-level languages. First, they draw the reader's attention towards the similarities between high-level languages and structured programing languages as they discuss how the goal of both the types of languages is to use abstractions that are correct for the problem being solved and how abstractions are essential to help the programmer develop code without concerning himself with nitty-gritty details. Next, they mention how both the types differ: in a high level language, the set of useful abstractions are identified in advance whereas in a structured programming language, it is more open to the programmer which abstractions to use and a mechanism is set in place for the user to extend the languages to inculcate the abstractions as needed.

The authors note, when working at a certain level you are concerned with the way a program makes use of abstractions provided at lower levels, but not with the details of how they are realized. Higher-level programming languages could provide increasing numbers of abstractions through their built-in types, but could never hope to provide all of the useful abstractions for building programs.

The authors bring these distinguished ideas together in describing Abstract Data Types as an approach to abstractions that permit the pre-defined built-in abstractions to be augmented as new abstractions when needed.

Abstraction is the mechanism that permits the expression of relevant information while the suppression of irrelevant information to the programmer. Typically, this is achieved using procedures (here used interchangeably with functions). Abstraction allows the programmer to be productive with code but only knowing what operations he wants to implement instead of knowing how to implement those operations. While procedures fulfill most of the features of abstraction for the programmer, they are not sufficiently rich in their vocabulary.

Accounting for abstraction in structural programming, the authors introduce the concept of abstract data types. An ADT is a class of abstract objects characterized by the operations available on those objects. This supports abstraction for the user as with the use of ADTs, he is only concerned with the characteristic operations that have been defined for that particular object. Comparing built-in data types with abstract ones - for using built-in data types, the user treats them to be atomic in nature and does not have to think about how memory is allocated to it or whether it can store any different type of data in it. All of these predefined rules and the set of possible operations are baked into the compiler of the language at the lower level. However, in the case of ADTs, although the realization happens at a lower level it is not embedded into the language itself. This means that ADTs provide a way to implement custom sets of operations for objects as per the need of the programmer;s solution code. This set of (usually) user-defined operations is called an operation cluster. For the remaining operations not associated with an ADT the authors use the term "functional abstraction." By programming in terms of ADTs and the operations they expose it is possible to defer the implementation representation decision

until a later point. And thus it is possible to program according to one of Dijkstra's programming principles: build the program one decision at a time. Resounds of the paper on Silver Bullets from SWE 211 Introduction to software Engineering that suggested that a possible solution to addressing the essences of programming is to grow software with the analogy of germinating a seed with a single objective of growing into a tree that is capable of a myriad of operations. The concept of ADTs is elaborated on by the use of an example of writing a program to convert infix expressions to polish notation.

The proposed programming language has syntax for declaring abstract data types variables that is the same as that for built in types. Declaring variables only requires the type of the variable, or it might also have to indicate the creation of an object during declaration. Being a strongly typed language, abstract objects can only be used either by calling one of its characteristic operations, by passing it as a parameter to a procedure, or by assigning it to a variable of corresponding type. Furthermore, it is possible to create objects outside the variable declarations. Operation clusters are defined in a module supporting independent compilation. A cluster definition contains 3 components: the object representation (its internal state), the code used to create the objects (constructor), and the operations themselves. This looks very similar to classes in modern day programming languages.

One of the biggest distinguishing concepts presented in this paper is the focus on information hiding. With Simula 67(the prevalent and famous language of the time), the classes were designed to represent and provide full accessibility to the objects i.e. every function and attribute in which the class is embedded is accessible to the user. Therefore, the actual form of the representation is always known to the user. In contrast, the representation of an operation cluster is not accessible outside the cluster. Operations in the cluster provide the only way to access the contents of the representation, and even then only a subset of the operations defined in the cluster may be accessible.

Finally, the authors believe that the abstraction presented in this paper can be used to augment existing high level languages as well as be implemented in their new structured programming language since any high level language no matter how advanced, can always be improved by giving its user the ability to build their own abstractions.