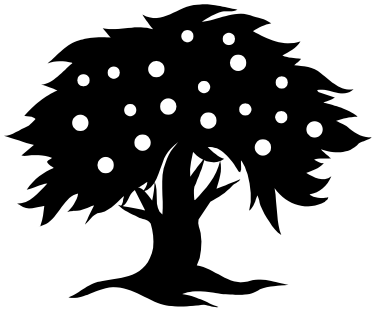
Basic concepts and terminology

There is a large and fast growing vocabulary used in the software industry. To help organize this information, it is essential to develop an understanding of the basic concepts or *abstractions* that underlie software systems.



Starting with dictionaries

Funk&Wagnall define *abstraction* as

> "… *the process of separating qualities or attributes from the individual objects to which they belong*."

Suggests a combination of *classification* and *aggregation*, the idea of a generic kind of thing with a (relevant) set of properties.

Classification

Act of creating a *class*.

Synonyms:

- *set*
- *prototypical instance*
- *type*
- *bag*
- *generic instance*
- *kind*
- *variety*

Relevant relationships:

- *instance of*
- *member of*
- *element of*
- *is a kind of*

Aggregation

Act of creating *properties*.

Relevant relationships:

- *property of*
- *part of*
- *attribute of*

An example of using classification and aggregation:
Choice of a data set, with its fields.

Another example:
Choice of a subroutine, with parameters and parts (possibly other subroutines).

Generalization and *specialization*

A kind of relationship between classes.

Synonyms:

- *isa*
- *subtype of*
- *subset of*

To assert that B *is a generalization of* A, or
that A *is a specialization of* B is to imply that:

1. Anything that is an instance of A is also an instance of B.

This in turn implies that:

2. Everything B has or does, A also has or does.

Generalization and *specialization* (cont'd)

An example of using generalization:
The class of students, each with a name and an age, is a generalization of 1) the class of graduate students, each with a supervisor, and 2) the class of undergraduates, each with a year.

An example of using specialization:
The enrollment of a graduate student is a more specialized case of the enrollment of a student; the enrollment of an undergraduate is also a more specialized case of the enrollment of a student.

Generalization and *specialization* (cont'd)

There is an additional and distinct suggestion of process or methodology.

The generalization process:
Start by thinking of specific classes, and then proceed by introducing additional classes that abstract two or more existing classes.

The specialization process:
Start by thinking of very general classes, and then proceed by introducing additional classes to abstract proper subsets of existing classes with peculiar additional properties.

Abstract data type (or *ADT*)

Another kind of abstraction in which we separate *implementation* or *realization* from a *service-focused* understanding of *functionality*.

A service-focused understanding derives from the notion of an *abstract algebra* which consists of two components.

1. An underlying *domain* of one or more "things".
2. A collection of *operations* defined over the domain.

ADT vs. *abstract algebra*

Unlike an abstract algebra, an operation in an ADT can have side effects.

1. The operation can modify the underlying domain by adding additional things or removing existing things.
2. The operation can modify the interpretation of other (not necessarily distinct) operations.

The second observation implies that the sequencing of operations can matter.

Part of the service-focused understanding can therefore include *protocols* and *timing constraints*.

ADT vs. *abstract algebra* (cont'd)

Also unlike an abstract algebra, each operation for an ADT will have an *implementation*.

An important requirement for all implementations:

> *There is no influence or side effect on the functionality of the associated ADT.*

Implementations can influence the quantitative behavior so long as this principle is not violated.

Related concept in database systems:

physical data independence.

ADT (cont'd)

Big advantage:
One is free to change implementations of the operators for an ADT without concern for external use of the ADT in implementations of operators for any other distinct ADT.

The STACK (a quintessential example):

1. Underlying domain: a stack object and a collection of stackable things.
2. Operations: `Empty`, `Push(E)`, `Pop(E)`.

Implementations for the operations might be based on the use of arrays, linked lists and so on.

Data structure

An ADT for which the service-focused understanding includes knowledge of time and/or store costs relating to one or more of its operations.

An example, the ARRAY:

1. Underlying domain: a collection of fixed sized arrays, the positive integers and a collection of possible array entries.
2. Operations: `New(A,n)`, `Access(A,i,E)`, `Update(A,i,E)`.

Operations `Access` and `Update` run in constant time.

Store cost is: (size of an element) × (sum of array sizes).

Aside: about multiple levels

All of what we have talked about so far may have more than one level.

Example:
A part of something is something else that in turn has parts.

Another example:
An implementation may use an ADT that has implementations that may in turn use other ADTs.

Quantification

An important kind of *control* abstraction in which we allow:

1. naming an object by describing its properties, and
2. *iterating over* (i.e., quantifying over) a group of things without saying how this is accomplished.

The distinction between *propositional logic* and *first order logic*.

E.g.:

“A student with GPA exceeding 90. ”

“The number of students with GPA less than 50. ”

“For each student, do …”

Process

The idea that the implementation of more than one operation can be executing at the same time.

Related phenomena:

- *concurrency*
- *multiple agents*
- *communication*

Complications:

- *deadlock*
- *mutual exclusion*
- *synchronization*

There is also an important taxonomy relating to process communication:

- *synchronous* vs. *asynchronous*
- *symmetric* vs. *asymmetric*

Delegation

An idea concerning two adjacent levels of abstraction:
An object at the higher level can be realized or implemented by more than one object at the lower level.

E.g.:

"Different information about an employee is encoded as three different records in three separate files."

"The implementation of an operation simply invokes another operation on a related object."

Additional basic concepts

- *Predicate* (and *predicative style*)
- *Function* (and *functional style*)
- *Procedure*
- *Module*

Homework:

1. Develop your own understanding of the notion of an abstract data type.
2. Think about the various facets of the C language `struct` type.