DESIGN CONSIDERATIONS

There is really no universal design method. The design process is largely a creative one and the quality and expertise of the designers are a critical determinant for its success.

Over the years ideas and guidelines have emerged to serve us in designing software.

1. INFORMATION HIDING

This is the single most important principle of software design. It exemplifies how to apply abstraction in software design. Abstraction means that we concentrate on essential issues and ignore abstract form that is to say the details that are irrelevant at this stage.

The principle of information hiding is that each module has a secret which it hides to other modules. It uses a guiding principal in design which is aptly illustrated in the KWIC-index example.

If we are to consider a second decomposition, for example module “STORE” hides how lines are a sorted and module “SORT” hides how and when shifts are sorted.

Design also involves a sequence of decisions, such as how to represent certain information or in which order to accomplish tasks.

Information hiding is closely related to the notions of abstraction, cohesion and coupling.

Explanation:

If a module hides some design decision, the user of that module may abstract from(ignore) the outcome of that decision. Since the outcome is hidden, it can possibly interfere wit the use of that module.

Also, if a module hides a secret, that secret does not permeate the module’s boundary, thereby decreasing the coupling between the module and its environment.

Information hiding increases cohesion, since the module’s secret is what binds the constituents of the module together. It should be noted that in order to maximize cohesion only secret should be hidden by the module.

It depends on the programming language used whether the separation of concerns obtained during the design stage will be identifiable in the ultimate code.

1. MODULARITY

During design, the system is decomposed into a number of modules and the relationships between the modules are indicated. In another design of the same system, different modules

may show up and there may be different relationships between the modules.

We may try to compare the designs by considering a typology for the individual modules and the types of connection between them. This leads us to two structural design criteria: cohesion and coupling.

Cohesion may be viewed as the glue that keeps the module together. It is a measure of the mutual affinity of the elements of a module. In general we will wish to make the cohesion as strong as possible.

Coincidental cohesion. With coincidental cohesion, elements are grouped into modules in a haphazard way. There is no significant relation between the elements.

Logical cohesion With logical cohesion, the elements realize tasks that are logically related. One example is a module that contains all input routines. These routines do not call one another and they do not pass information to each other. Their function is just very similar.

Temporal cohesion is A typical example of this type of cohesion is an initialization module. The various elements of it are independent but they are activated at about the same point in time.

Procedural cohesion A module exhibits procedural cohesion if it consists of a number of elements that have to be executed in some given order. For instance, a module may have to first read some datum, then search a table, and finally print a result.

Communicational cohesion This type of cohesion occurs if the elements of a module operate on the same (external) data. For instance, a module may read some data from a disk, perform certain computations on those data, and print the result.

Sequential cohesion Sequential cohesion occurs if the module consists of a sequence of elements where the output of one element serves as input to the next element.

Functional cohesion. In a module exhibiting functional cohesion all elements contribute to the one single function of that module. Such a module often transforms a single input datum into a single output datum. The well-known mathematical subroutines are a typical example of this. Less trivial examples are modules like ‘execute the next edit command’ and ‘translate the program given’.