WHAT AND WHY OF SOFTWARE MODELING AND DESIGN

When we say software we mean, a set of instruction, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspect of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device. It can be thought of as the variable part of a computer, while hardware is the invariable part

Modeling is used in many walks of life, going to early civilization such as ancient Egypt, Rome and Greece, where modeling was used to provide small-scale plans in art and architecture. Modeling is widely used in science and engineering to provide abstractions of a system at some level of precision and detail. The model is then analyzed in order to obtain a better understanding of the system being developed. According to the Object Modeling Group (OMG), modeling is the design of software application before coding.

A software architecture design can be described at different levels of detail. At a high level, it can describe the decomposition of the software system into subsystems. At lower level, it can describe the decomposition of subsystems into modules or components. In each case the emphasis is on the external view of the subsystem/tions with other subsystems/components.

For good quality software to be produced, the software design must also be of good quality. Now the matter of concern is how the quality of good software design is measured? This is done by observing certain factors in software design. These factors are: **correctness, understandability, efficiency, maintainability.**

Firstly, for all the design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and considered.

Secondly, the software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self-explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer.

Thirdly, the software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software that is not efficiency and useful, then the developed software would also stand on the same level of efficiency.

Lastly, the software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes.

Moving on software modeling principles there are many different principles which I'm going to list below.

The first principle, the primary goal of the software team is to build software, not create models.

The second principle, the travel light – don't create more models than you need.

The third principle, strive to produce the simplest model that will describes the problem of the software.

The fourth principle, build models in a way that makes them amenable to change.

The fifth principle, be able to state an explicit purpose for each model that is created.

The sixth principle, don't become dogmatic about the syntax of the model, if it communicates content successfully, representation is secondary.

The seventh principle, if your instincts tell you a model isn't right even though it seems OK on paper, your probably have reason to be concerned.

In addition, to know the software modeling principles in design part

Principle one, design should be traceable to the requirements model.

Principle two, always consider the architecture of the system to be built.

Principle three, design of data is as important as design of processing.

Principle four, interface must be designed.

Principle five, user interface design should be tuned to the needs of the end user. However, in every case, it should stress ease of use.

As follows, we are going to continue with the software modeling examples

Firstly, class model, class diagrams are fundamental to object-oriented analysis and design. These diagrams show the static structure of object classes and important relationships between them.

Secondly, process model, also called data flow diagrams (DFDs) start with a top-level context diagram for a system. The system is represented as a named process with data flows in and out to the external world.

Thirdly, data model, an entity-relation diagram, called an ERD illustrates the data structure of an information system. A database ca ne designed using logical and physical data models that highlight primary and foreign keys.

Fourthly, state model, the essential behavior of systems can often be expressed with a state model. State diagrams show events, states and actions in various notations including Mealy, Moore and UML/Harel.

Fifthly, structure model, structure chat diagrams illustrate the organization of procedural programs. Each thread of execution begins with a root module at the top of an inverted tree of called modules

Lastly, object model, an object model shows object instances, their operations and messages between object to document the mechanisms within an object-oriented design. each diagram illustrates part of the design with a collection of communicating objects.

As conclusion it would be better if we implement software modeling and design for making good and full system which is clear to both side.