SOFTWARE DEVELOPMENT TECHNIQUES

The term ***software engineering*** is composed of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

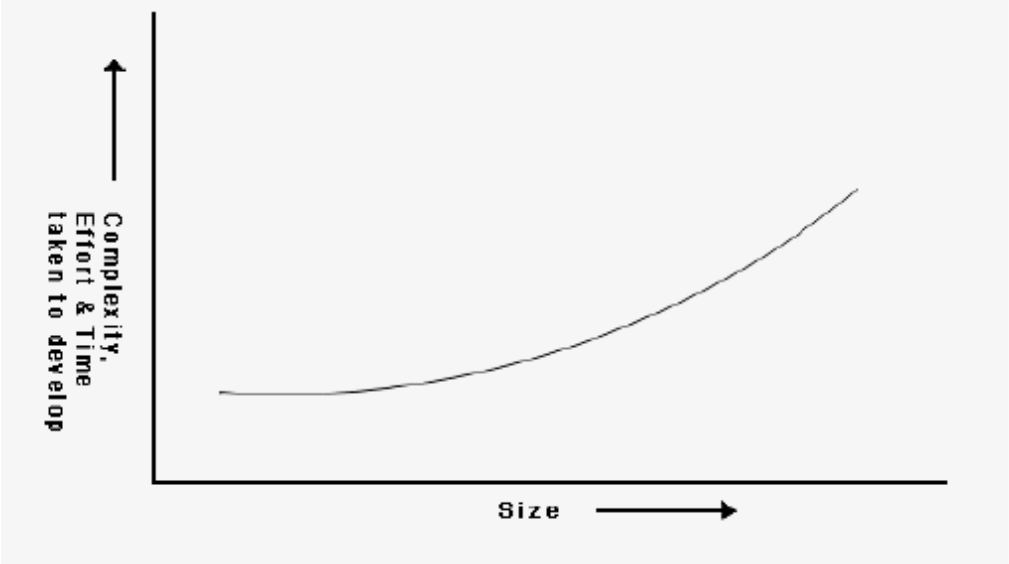
For a program, the user interface may not be very important, because the programmer is the sole user. On the other hand, for a software product, user interface must be carefully designed and implemented because developers of that product and users of that product are totally different. In case of a program, very little documentation is expected, but a software product must be well documented. A program can be developed according to the programmer's individual style of development, but a software product must be developed using the accepted software engineering principles.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

So, we can define ***software engineering*** as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Without using software engineering principles it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions. A problem with developing

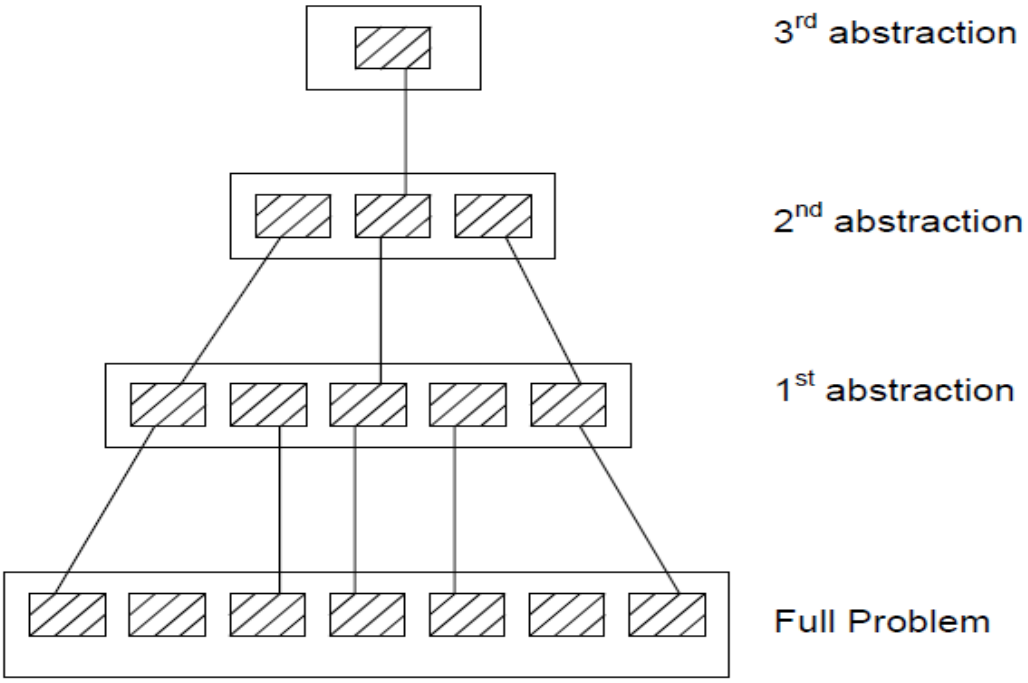
such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes. Software engineering helps to reduce this programming complexity. Software engineering principles use two important techniques to reduce problem complexity: ***abstraction*** and ***decomposition*.**



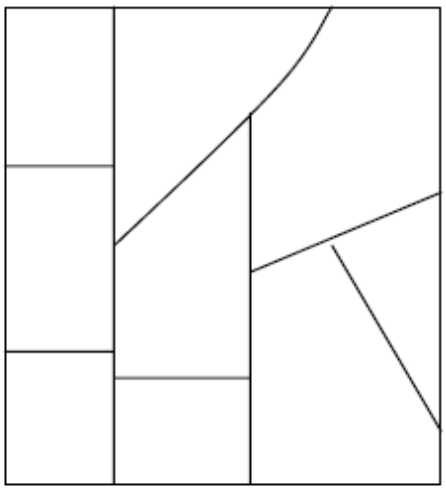
Increase in development time and effort with problem size

The principle of abstraction implies that a problem can be simplified by omitting irrelevant details. In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose. Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on. Abstraction is a powerful way of reducing the complexity of the problem. The other approach to tackle problem complexity is decomposition. In this technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one. However, in this technique any random decomposition of a problem into smaller parts will not help. The problem has to be

decomposed such that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution. A good decomposition of a problem should minimize interactions among various components. If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.



A hierarchy of abstraction



Decomposition of a large problem into a set of smaller problems.

CHARACTERESTICS OF GOOD SOFTWARE

While developing any kind of software product, the first question in any developer's mind is, “What are the qualities that good software should have? First, a software product must meet all the requirements of the customer or end-user. Also, the cost of developing and maintaining the software should be low.

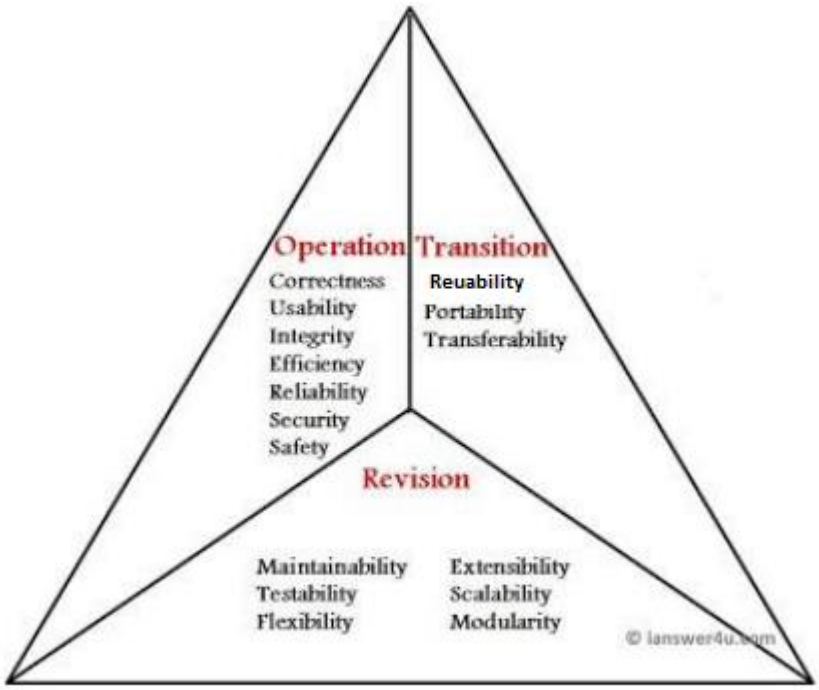
The development of software should be completed in the specified time-frame. Well these were the obvious things which are expected from any project.

Now lets take a look at Software Quality factors. These set of factors can be easily explained by Software Quality Triangle. The three characteristics of good application software are :-

1) Operational Characteristics

2) Transition Characteristics

3) Revision Characteristics



Software Quality Triangle with characteristics

Maintainability: This is a measure of how easy a system is to maintain during its deployed life. This cannot be measured directly before the system goes into operation because it depends on many features of the software and on what changes the systems will be expected to undergo. At this stage the maintainability of a system is assessed qualitatively on the basis of inspections and measures of the quality of the structure of the code. In operation, Maintainability can be measured (usually as mean time to repair). This measure is only significant if the product is used for a long time and/or it has a large number of installations.

Dependability: This is a measure of how "trustworthy" the software is. Usually this is a combined measure of the safety, reliability, availability and security of a system. The issues of measurement of Dependability are similar to those of Maintainability.

Efficiency: For some systems it is important to keep the use of system resources (time, memory, bandwidth) to a minimum. This is often at the cost of added complexity in the software. Improving the efficiency of a system often involves a detailed analysis of the interactions between different modules making up the system as a consequence the cost of improving efficiency often grows non-linearly in the size of the system and the required efficiency.

Usability: Usability is a measure of how easy the system is to use. Again this is hard to measure since it arises from many factors. Often this is approximated by very rough measures like learning time to carry out some operation.

Attributes can make conflicting demands on the software product. For example, improving efficiency may lead to more complex interactions between software modules and to more interactions between formerly independent modules. Changes like these can have a serious effect on the Maintainability of a system because more interaction between modules can make tracking down and fixing errors much more difficult.