SOFTWARE ENGINEERING

(MC – 310)

ASSIGNMENT – 2

AIMAN SIDDIQUA **2K18/MC/008**

Ques. Suggest why it is important to make a distinction between developing the user requirements and developing system requirements in the requirements engineering process.

Solution. There is a fundamental difference between the user and the system requirements that mean they should be considered separately.

- The user requirements are intended to describe the system's functions and features from a user perspective and it is essential that users understand these requirements. They should be expressed in natural language and may not be expressed in great detail, to allow some implementation flexibility. The people involved in the process must be able to understand the user's environment and application domain.
- The system requirements are much more detailed than the user requirements and are intended to be a precise specification of the system that may be part of a system contract. They may also be used in situations where development is outsourced and the development team need a complete specification of what should be developed. The system requirements are developed after user requirements have been established.

Ques. Explain why the process of project planning is iterative and why a plan must be continually reviewed during a software project.

Solution. Project planning can only be based on available information. At the beginning of a project, there are many uncertainties in the available information and some information about the project and the product may not be available. As the project develops, more and more information become available and uncertainties are resolved. The project plan therefore must be reviewed and updated regularly to reflect this changing information environment.

Ques. What are the characteristics of a good design? Describe the main activities in the software design process and the outputs of these activities. Describe different types of coupling and cohesion.

Solution. Characteristics of a good design:

- Correctness
- Understandability
- Efficiency
- Maintainability

Different types of coupling:

- **Data coupling:** The dependency between module A and B is said to be data coupled if their dependency is based on the fact, they communicate by only passing of data. Other than communicating through data, the two modules are independent.

- **Stamp coupling:** Stamp coupling occurs between module A and B when complete data structure is passed from one module to another.

- **Control coupling:** Module A and B are said to be control coupled if they communicate by passing of control information. This is usually accomplished by means of flags that are set by one module and reacted upon by the dependent module.

- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.

- **Common coupling:** With common coupling, module A and module B have shared data. Global data areas are commonly found in programming languages. Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of changes.

- **Content coupling:** Content coupling occurs when module A changes data of module B or when control is passed from one module to the middle of another.

Different types of cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer.

- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.

- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at initializing time.

- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.

- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

Ques. Identify six possible risks that could arise in software projects.

Solution. Different risks that could arise in software projects are:

Dependencies on outside agencies or factors

- Availability of trained, experienced persons
- Inter group dependencies
- Customer-Furnished items or information
- Internal & external subcontractor relationships

Requirement issues

- Lack of clear product vision

- Unprioritized requirements
- Lack of agreement on product requirements
- New market with uncertain needs
- Rapidly changing requirements
- Inadequate Impact analysis of requirements changes

Management Issues

- Inadequate planning
- Inadequate visibility into actual project status
- Unclear project ownership and decision making
- Staff personality conflicts
- Unrealistic expectation
- Poor communication

Lack of knowledge

- Inadequate training
- Poor understanding of methods, tools, and techniques
- Inadequate application domain experience
- New Technologies
- Ineffective, poorly documented or neglected processes

Other risk categories

- Unavailability of adequate testing facilities
- Turnover of essential personnel
- Unachievable performance requirements
- Technical approaches that may not work

Ques. Discuss the differences of object-oriented and function-oriented design.

Solution.

FUNCTION ORIENTED DESIGN

- Function Oriented design is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function. Thus, system is designed from a functional viewpoint.
- It is a top-down approach.

OBJECT ORIENTED DESIGN

- Object oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to do manipulated by the program. Thus, it is orthogonal to function oriented design.
- It is a bottom-up approach.

Ques. Write short notes on:

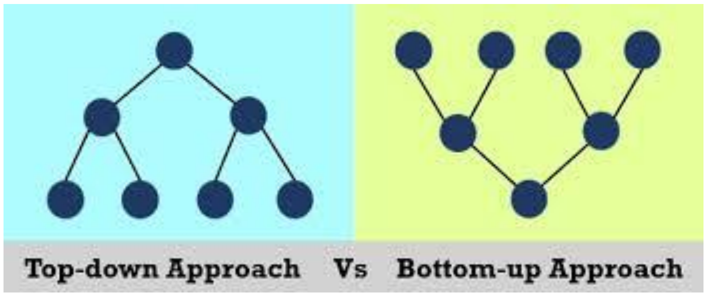
(a)Top-down and bottom-up design

TOP-DOWN DESIGN

A top-down design approach starts by identifying the major modules of the system, decomposing them into their lower level modules and iterating until the desired level of detail is achieved. This is stepwise refinement; starting from an abstract design, in each step the design is refined to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly.

BOTTOM-UP DESIGN

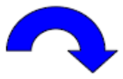
Bottom-Up Design is a system design approach where parts of the system are defined in details. Once these parts are designed and developed, then these parts or components are linked together to prepare a bigger component. This approach is repeated until the complete system is built. Advantage of Bottom-Up Model is in making decisions at very low level and to decide the re-usability of components.

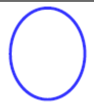
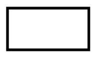
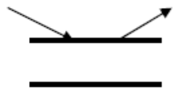


(b) Data flow diagrams

Data Flow Diagrams show the flow of data through the system.

- All names should be unique
- It is not a flow chart
- Suppress logical decisions
- Defer error conditions & handling until the end of the analysis

SYMBOL	NAME	FUNCTION
	Data Flow	Connect Process

	Process	Perform some transformation of its input data to yield output data.
	Source or sink	A source of system inputs or sink of system outputs
	Data Store	A repository of data, the arrowhead indicates net input and net outputs to store

(c) LOC

A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declaration, and executable and non-executable statements.

(d) Token Count

In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

The basic measures are

n1 = count of unique operators.
n2 = count of unique operands.
N1 = count of total occurrences of operators.
N2 = count of total occurrence of operands.

In terms of the total tokens used, the size of the program can be expressed as N = N1 + N2.

(e) Function count

Functional Point Analysis is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types:

1.Number of External Inputs(EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.