SE Assignment II

1. **What is the difference between functional and non-functional requirements? Give one example for each type of requirement for a bank ATM software.**

Functional requirements define a function of a system or its component. Function is described as a set of inputs, behaviors and outputs.

Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish.

Non functional requirements are non negotiable obligations that must be supported by the software.

Non functional requirements are in the form "System shall be <requirement>", an overall property of the system as a whole and not a specific function.

For a bank ATM software, **withdrawal of cash** is a functional requirement; sub-requirements may include an option to select amount of cash, account type, and a module to perform the transaction.

And **overall interfacing** is a non functional requirement; sub requirements may include interfacing of display, input and in-process views.

2.

A. **What do you understand by requirements gathering? Explain different requirement gathering techniques.**

Requirements gathering is a process of collecting all the necessary information about the project from all of the clients and their documents and notes.

Different requirements gathering techniques include:

- **Studying existing documentation** — You get a ~~butt load~~ lot of information from the documentation itself, provided it exists, cause the client is not all times able to explain ~~shit~~ properly.
- **Interviews** — Interviews with the clients may provide you with extra information that is missing from the PRDs.
- **Task analysis** — Analysis of the tasks and breaking them down into various sub-tasks.
- **Scenario analysis** — Details of each requirements are gathered from various users under multiple scenarios.
- **Form analysis** — Analyzing the format of the product can be used to identify the various components of the input and output modules.

B. **List the characteristics of a good software design.**

Characteristics of a good software design include:

- **Accurate** — A good software should implement all the requirements accurately.
- **Comprehensible** — A good software should be easy to grasp by all sorts of ~~dumb~~ users.
- **Efficient** — A good software design should be efficient in doing what it's supposed to; it should not lag or provide time consuming solutions.
- **Maintainable** — A good software should be easily modifiable by future developers, as requirements change overtime.

3.

A. **How are the abstraction and decomposition principles used in development of good SRS?**

The principle of **Abstraction** means that an existing problem can be simplified by omitting all the irrelevant details of the project in the SRS document.

By showing only the important details about the project like "*how it will look*" instead of the "*internal functionality to implement it*" in the SRS document, it becomes comprehensible to a wider range of clients, including the ones in non-technical streams, as it appeals to users and not developers.

Decomposition means breakdown of a large module into various smaller modules.

By breaking down a large problem statement into separate smaller statements, we can explain the modules in a more easily to the clients, and reducing the cluttering in the SRS document.

B. **What do you mean by terms cohesion and coupling in the context of software design?**

Cohesion is the measure of the functional strength of a module.

Good decomposition is indicated by high cohesion rate; a good cohesion is the one where the individual modules co-operate with each other performing a single objective.

Coupling between two modules is the measure of degree of interaction (interdependence) between the two modules.

Good coupling is sharing of resources between the modules; for example if a function call requires passing large amounts of data between two modules, the modules are tightly coupled.

4. **What do you understand by the problems of over specification, forward reference, noise in SRS document. Explain each of these with suitable examples.**

Over specification

It occurs when the analyst tries to address the "how to" aspects in the SRS document. It limits the imagination of the developers/designers to come up with a a good solution.

For example, in a movie database application, you don't need to specify how the movies are stored in the database, and which algorithm you're using to fetch them.

Forward references

It happens when the analyst refers to the aspects which are discussed much later in the SRS, this causes readability issues in the document.

Noise

Noise refers to the presence of material not directly relevant to the software development process. It is hardly of any use to the software developers and would unnecessarily clutter the SRS document, diverting the attention from the crucial points.

For example, In the register customer function, information like, who mans the customer registration department and at what time do they work, etc., are considered as noise.

5. **Explain the main differences between architectural design, high level design, and detailed design of a software system.**

   Architectural Design is the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

   High Level Design (Preliminary) is the process of analyzing design alternatives and defining the architecture, components, interfaces, and timing/sizing estimates for a system or components. The outcome of the high-level design is called the program structure of the software architecture.

   Detailed Design is the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to begin implementation. The outcome of the detailed design stage is usually documented in the form of a module specification (MSPEC) document.