1. What are the software process activities?

Ans: **Software process activities**

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

**Software specification**

The process of establishing what services are required and the constraints on the system’s operation and development.

Requirements engineering process:

* **Feasibility study**: is it technically and financially feasible to build the system?
* Requirements **elicitation and analysis**: what do the system stakeholders require or expect from the system?
* Requirements **specification**: defining the requirements in detail
* Requirements **validation**: checking the validity of the requirements

**Software design and implementation**

The process of converting the system specification into an executable system.

* **Software design**: design a software structure that realizes the specification;
* **Implementation**: translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

* **Architectural design**: identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
* **Interface design**: define the interfaces between system components.
* **Component design**: take each system component and design how it will operate.
* **Database design**: design the system data structures and how these are to be represented in a database.

**Software validation**

Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

* **Validation**: are we building the right product (what the customer wants)?
* **Verification**: are we building the product right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most commonly used V & V activity and includes the following stages:

* **Development or component testing**: individual components are tested independently; components may be functions or objects or coherent groupings of these entities.
* **System testing**: testing of the system as a whole, testing of emergent properties is particularly important.
* **Acceptance testing**: testing with customer data to check that the system meets the customer's needs.

**Software evolution**

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

1. **What in inception and elicitation?**

**ANS:** 1. Inception, in which the nature and scope of the system is defined.

 2. **Elicitation,** in which the requirements for the software are initially gathered.

3. **Elaboration**, in which the gathered requirements are refined.

4. **Negotiation,** in which the priorities of each requirement is determined, the essential requirements are noted, and, importantly, conflicts between the requirements are resolved.

 5. **Specification**, in which the requirements are gathered into a single product, being the result of the requirements engineering.

 6. **Validation**, in which the quality of the requirements (i.e., are they unambiguous, consistent, complete, etc.), and the developer's interpretation of them, are assessed.

7. **Management**, in which the changes that the requirements must undergo during the project's lifetime are managed.

**Inception**

 The requirements engineering process begins by examining the problem which the software should solve and gaining an understanding of both the problem's nature and the nature of the desired solution. This should be done in a context-free manner, that is, a manner which does not presume to know anything concerning the problem, the customers, the users, and the requested solution. The following questions may be asked:

 • Who is requesting the software?

• Who will use the software?

 • What is the benefit that the software will bring? It is important to identify the stakeholders in the project. Stakeholders are the people who will find benefit in the project and the software being developed. They may include:

 • Customers

• End users

 • Business operations managers

 • Product managers

• Advertising and marketing staff

 • Software engineers

 • Support engineers Each of the stakeholders will have a different view on what the software product should do and on what the software engineers should focus on. This might be on creating “sexy” features (from the marketing department), staying within budgets and deadlines (from managers), maintainability (from support engineers) and so on. Out of these views, the requirements engineer should determine which requirements there are a consensus on, and on which requirements the stakeholders disagree. Resolving disagreements between stakeholders makes up the negotiation step. Something to consider during inception is the effectiveness of the communication between the requirements engineer and the customers. This may be done by, for example, asking the customer if they feel that they have been asked appropriate questions concerning their problems, and if the person communicating with the requirements engineer feels that they are (or are not) the person who should be answering the engineer's questions.

**Elicitation**

This step is concerned with identifying the overall problem the software is attempting to solve, proposing solutions, negotiating between the differing approaches to solving the problem, and finally specifying a basic set of requirements. This can be done by calling a meeting between all of the stakeholders. It is important to nominate someone to act as a facilitator, who will guide the meeting. Each of the attendees should bring to the meeting a list of:

• objects that make up the system's operating environment

 • objects used by the system (such as those things which make up the input to the system) • objects produced by the system

 • the services that interact with these objects

• various constraints, such as time and budget constraints, interoperability constraints, performance restraints, usability constraints, and so on.

**Elaboration**

This step involves expanding on the requirements defined in the previous two steps, and from these requirements producing an analysis model, which is a technical model of the software and its functions. The construction of analysis models will be discussed in detail in the following two chapters.

**Negotiation**

This step involves negotiating between the various stakeholders in order to remove any conflict in the requirements. A useful technique for resolving these conflicting requirements is to provide each of the stakeholders with a finite number of priority points. They may then allocate points between the conflicting requirements as they see fit. The overall importance of any requirement can then be determined by the number of priority points that it has received.

**Specification**

The specification step produces the final product of the requirements engineering process. It describes the software, both its functions and constraints. The specification need not be a written document, but could also be a graphical model (such as those produced using the UML), software prototype or formal model, or a collection of these.

**Validation**

This step is concerned with ensuring that the gathered requirements in the software specification meet certain standards of quality. For example, have the requirements been written to the proper level of abstraction, or do they provide too much technical detail for the given stage of development? Is the requirement necessary, or something not essential to the software? Are the requirements unambiguous? Do requirements contradict other requirements? A useful action during validation is to ensure that each requirement has a source attributed to it. In this way, if more information is required, the requirements engineers know who to contact.

 **Management**

Requirements change over time; requirements management is concerned with controlling and tracking change in the requirements. Requirements management proceeds by associating requirements with various aspects of the software engineering process. As these aspects are changed, the requirements associated with them can be easily identified and changed. As these requirements are changed, all aspects of development associated with the modified requirements can be examined, and in this way the changes can more easily be propagated through the project. Such an association between requirements and aspects of the project can be done using a table: each row in the table represents a specific requirement, each column an aspect of the software project. The entries mark whether a requirement is associated with that aspect.

Q 3) What is quality function requirement in terms of elicitation?

Ans : Quality requirements are an important class of non-functional requirements They concern software system be attributes such as functional suitability, performance, reliability, usability, security, and portability that are important for achieving stakeholder goals the satisfaction of these quality attributes determines whether the software system meets the goals of its stakeholders or whether the system has a negative impact for these stakeholdersMeeting the right level of quality is important to balance benefits and cost The quality of a software system needs to be at least as good as to make the software useful and competitive, but should not be excessive to avoid cost and unnecessary use of resources.Insufficient quality leads to disappointment and consequent churn when stakeholders decide to abandon the software solution and adopt alternatives instead Excessive quality may lead to an unnecessarily expensive design of the software system to unnecessary consumption of resources needed for operating the systemand to trade-offs where other quality attributes suffer To address the problem of finding the level of good-enough quality, the relationship between software quality and the impacts of such quality for the stakeholders of the software system needs to be understood. As demonstrated for the Quality of Service (QoS) of a telecommunication network and the Quality of Experience (QoE) of the network users, a quality–impact relationship can be developed empirically by setting quality levels of a given quality attribute and measuring the reaction of the stakeholders that were exposed to these quality levels This paper describes how to use quality–impact analysis for eliciting requirements about good-enough quality of a software system. The proposed method guides the elicitation of the quality–impact relationships and explains how to use the gained insights to specify quality requirements. The method delivers empirical evidence for a specific software system that is more reliable than generic expert opinion. The evidence pertains to the features that were investigated and the stakeholders that were participating in the requirements inquiry, thus is adequate and relevant for decision-making about that software system’s quality requirements.The paper describes the proposed quality–impact elicitation method in depth. It gives details about the key ideas of the method and explains how to tailor the method depending on the investigated quality characteristics, the stakeholder goals impacted by these quality characteristics, and the instruments

that the investigator is able to apply. The paper provides an example of how the method is applied in practice by reporting about its use in a real-world software development project.The paper is structured as follows. Section II reviews existing work and motivates quality requirements elicitation based on quality–impact relationship inquiry. Section III describes the method in-depth. Section IV describes how the method is applied and reports the lessons-learned from such method application. Section V compares the method and the obtained results with related work. Section VI concludes.