CMR INSTITUTE OF TECHNOLOGY	USN	

Internal Assesment Test – I, June 2021

Sub:	SOFTWARE ENGINEERING					Code:	20MCA24
Date:	23-6-2021	Duration:	90 mins	Max Marks:	50	Sem:	II	Branch:	MCA

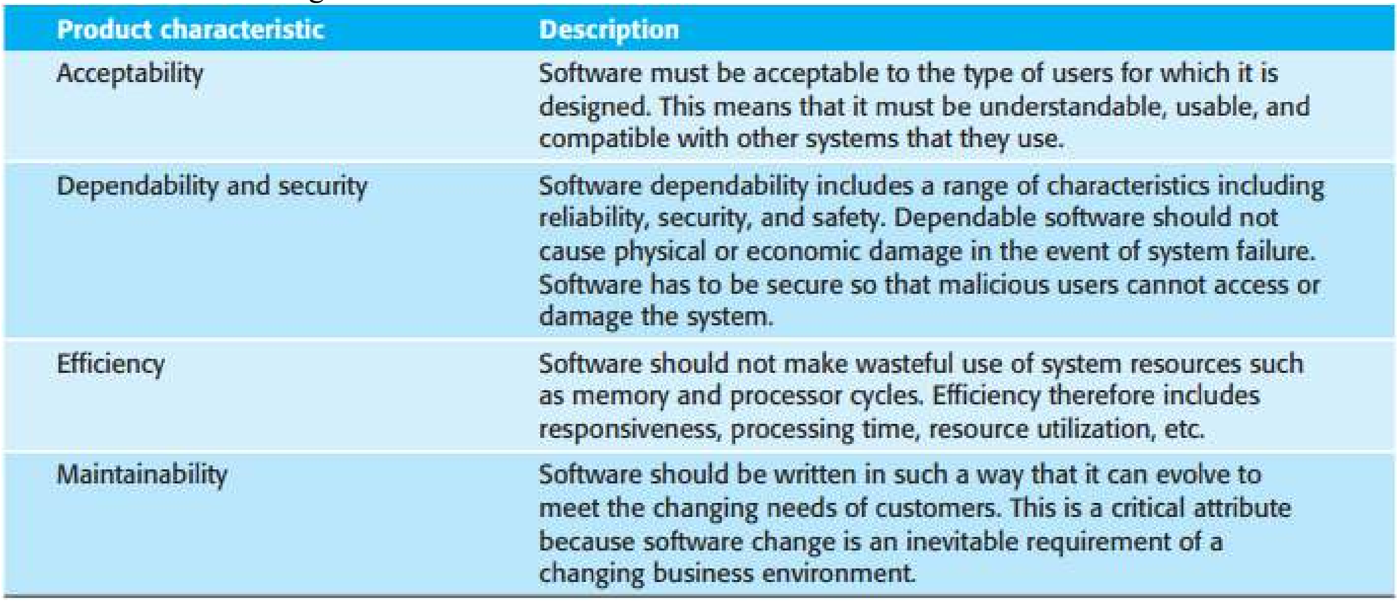
Answer **ONE FULL QUESTION** from each part

No.		Question	Marks	OBE CO	OBE RBT
		Part-I			
1	(a)	What is software process model? Explain the essential attributes of good software	[6]	CO1	L2
	(b)	Describe the difference between software engineering and system engineering product	[4]	CO1	L3
		(OR)			
2		Discuss the various issues in software engineering professional responsibility and code of ethics	[10]	CO1	L2
		Part-II			
3	(a)	Discuss the key challenges facing software engineering professionals that are common to all software processes	[4]	CO2	L2
	(b)	Explain the phases of RUP with a neat diagram.	[6]	CO2	L2
		(OR)			
4		Discuss the process involved in waterfall and incremental development model with advantages and disadvantages.	[10]	CO2	L2
		Part-III			
5	(a)	What are the various techniques used for requirements elicitation_and analysis? Explain any in detail.	[10]	CO2	L2
		(OR)			
6	(a)	Explain the characteristic features and Structure of a software requirements document.	[6]	CO2	L2
	(b)	Justify which process model will be suitable to design insulin pump system	[4]	CO2	L3
		Part-IV			
7		Explain in detail about extreme programming.	[10]	CO2	L2
		(OR)			
8		Describe Requirements Change Management Process and planning.	[10]	CO1	L2
		Part-V			
9	(a)	Define: (i) Functional requirements and (ii) Non-functional requirements	[4]	CO1	L1
	(b)	Explain the different metrics of non-functional requirements.	[6]	CO1	L2
		(OR)			
10		Write Short Notes on: a) requirement specification b)View point, stockholders , c) scenario techniques, d) Ethnography e). Feasibility study	10	CO3	L2

1. (a) What is software process model? Explain the essential attributes of good software

A software process model (sometimes called a Software Development Life Cycle or SDLC model) is a simplified representation of a software process. Each process model represents a process from a particular perspective and thus only provides partial information about that process. For example, a process activity model shows the activities and their sequence but may not show the roles of the people involved in these activities. These generic models are high-level, abstract descriptions of software processes that can be used to explain different approaches to software development. You can think of them as process frameworks that may be extended and adapted to create more specific software engineering processes.

Essential attributes of a good software:



Product characteristic	Description
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

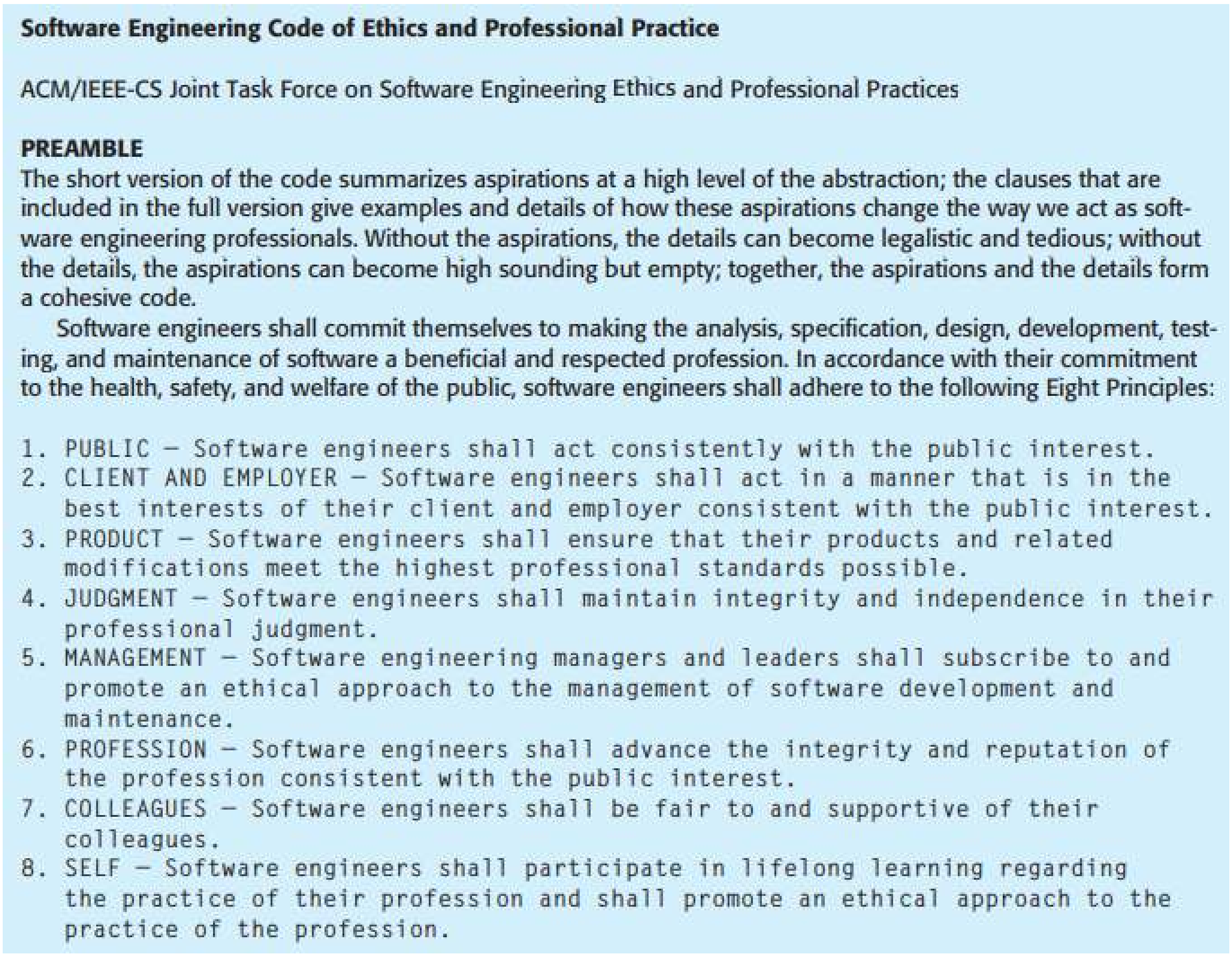
1. (b) Describe the difference between software engineering and system engineering product

01.	A System Engineer is a person who deals with the overall management of engineering projects during their life cycle (focusing more on physical aspects).	A Software Engineer is a person who deals with the designing and developing good quality of software applications/software products.
02.	System Engineers follows an interdisciplinary approach governing the total technical and managerial effort required to transform requirements into solutions.	Software Engineers follows a systematic and disciplined approach for software design, development, deployment and maintenance of software applications.
03.	In general they are concerned with all aspects of computer based system development including hardware, software and process engineering.	In general they are concerned with all aspects of software development, infrastructure, control, applications and databases in the system.
04.	One thing software engineering can learn from system engineering i.e Consideration of trade-offs and use of framework methods.	One thing system engineering can learn from software engineering i.e Disciplined approach to cost estimation.

05.	System engineers mostly focus on users and domains.	Software engineers mostly focus on developing good software.
06.	Systems Engineering Methods are Stakeholder Analysis, Interface Specification, Design Tradeoffs, Configuration Management, Systematic Verification and Validation, Requirements Engineering etc.	Software Engineering Methods are Modeling, Incremental Verification and Validation, Process Improvement, Model-Driven Development, Agile Methods, Continuous Integration etc.
07.	It ensures correct external interfaces, interfaces among subsystems and software.	It makes interfaces among software module, data and communication path work.
08.	System Engineers requires a broader education background like Engineering, Mathematics and Computer science etc.	While Software Engineers requires Computer Science or Computer Engineering background.

2. Discuss the various issues in software engineering professional responsibility and code of ethics.

Software Engineering Professional Responsibility:
1. **Confidentiality** You should normally respect the confidentiality of your employers or clients regardless of whether or not a formal confidentiality agreement has been signed.
2. **Competence** You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. **Intellectual property rights** You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. **Computer misuse** You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses or other malware).



Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety, and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC — Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER — Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT — Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT — Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION — Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES — Software engineers shall be fair to and supportive of their colleagues.
8. SELF — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

3. (a) Discuss the key challenges facing software engineering professionals that are common to all software processes

Software engineering employs a well-defined and systematic approach to develop software. This approach is considered to be the most effective way of producing high-quality software. However, despite this systematic approach in software development, there are still some serious challenges faced by software engineering. Some of these challenges are listed below.

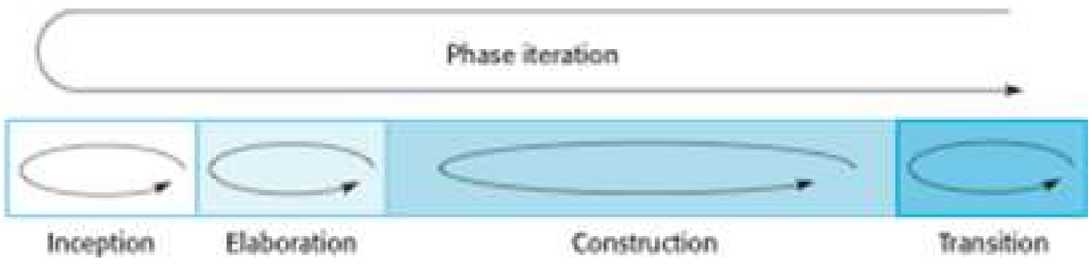
1. The methods used to develop small or medium-scale projects are not suitable when it comes to the development of large-scale or complex systems.
2. Changes in software development are unavoidable. In today's world, changes occur rapidly and accommodating these changes to develop complete software is one of the major challenges faced by the software engineers.
3. The advancement in computer and software technology has necessitated for the changes in nature of software systems. The software systems that cannot accommodate changes are not of much use. Thus, one of the challenges of software engineering is to produce high quality software adapting to the changing needs within acceptable schedules. To meet this challenge, the object-oriented approach is preferred, but accommodating changes to software and its maintenance within acceptable cost is still a challenge.
4. Informal communications take up a considerable portion of the time spent on software projects. Such wastage of time delays the completion of projects in the specified time.
5. The user generally has only a vague idea about the scope and requirements of the software system. This usually results in the development of software, which does not meet the user's requirements.

6. Changes are usually incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.
7. The development ofl1igh-quality and reliable software requires the software to be thoroughly tested. Though thorough testing of software consumes the majority of resources, underestimating it because of any reasons deteriorates the software quality.

In addition to the above-mentioned key challenges, the responsibilities of the system analyst, designers, and programmers are usually not well defined. Also, if the user requirements are not precisely defined, software developers can misinterpret the meaning. All these challenges need to be addressed in order to ensure that the software is developed within the specified time and estimated costs and also meets the requirements specified by the user.

3.(b) Explain the phases of RUP with a neat diagram

The Rational Unified Process:
- ✧ A modern generic process derived from the work on the UML and associated process.
- ✧ Brings together aspects of the 3 generic process models discussed previously.
- ✧ Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practive perspective that suggests good practice.

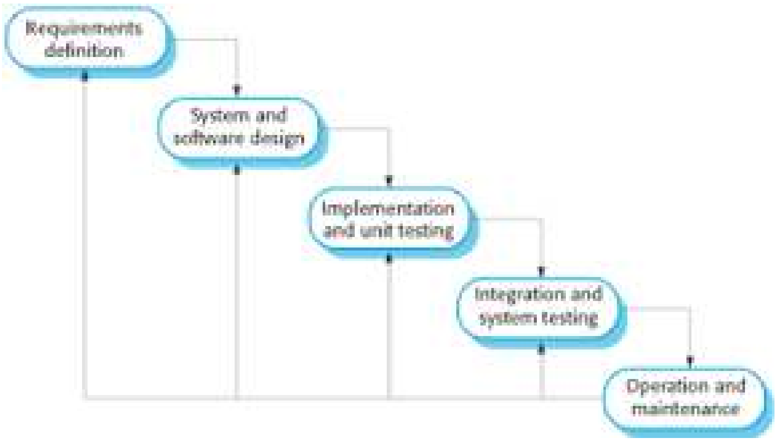


4. Discuss the process involved in waterfall and incremental development model with advantages and disadvantages

Waterfall Model:
- ✧ There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- ✧ The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Problems:
- ✧ Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- ✧ The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

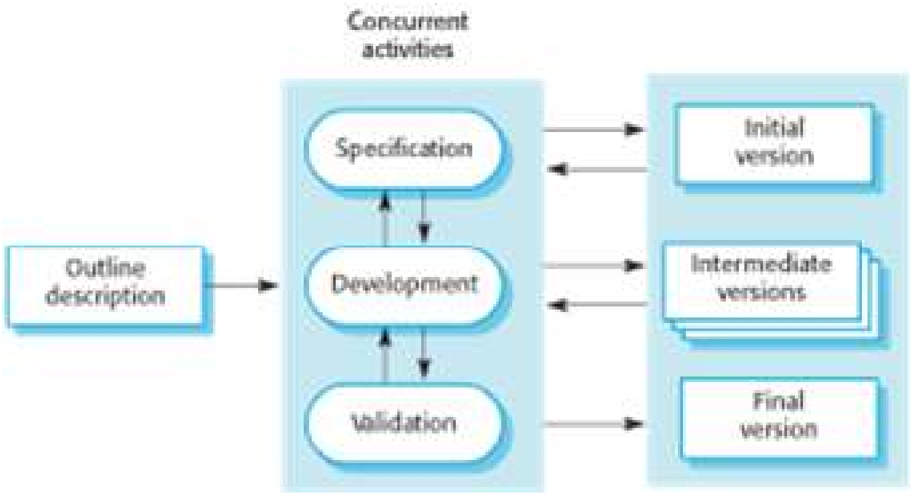

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Incremental Development Model:

- ✧ The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- ✧ It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- ✧ More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Problems:

- ✧ The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- ✧ System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.



5. What are the various techniques used for requirements elicitation_and analysis? Explain any in detail

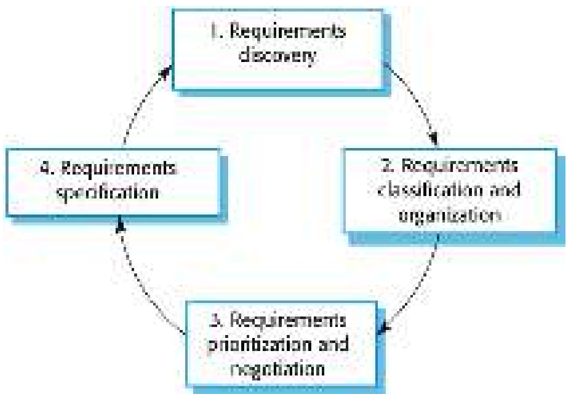
Requirements elicitation_and analysis:

- ✧ Sometimes called *requirements elicitation* or *requirements discovery*.
- ✧ Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- ✧ May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders*

- ✧ Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- ✧ Stages include:
  - Requirements discovery,
  - Requirements classification and organization,
  - Requirements prioritization and negotiation,
  - Requirements specification.

Process Activities:

- ✧ Requirements discovery
  - ▪ Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
- ✧ Requirements classification and organization
  - ▪ Groups related requirements and organizes them into coherent clusters.
- ✧ Prioritization and negotiation
  - ▪ Prioritizing requirements and resolving requirements conflicts.
- ✧ Requirements specification
  - ▪ Requirements are documented and input into the next round of the spiral.


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

6. (a) Explain the characteristic features and Structure of a software requirements document

Following are the characteristics of a good SRS document:

1. **Correctness:**
   User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.
2. **Completeness:**
   Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.
3. **Consistency:**
   Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

4. **Unambiguousness:**
   A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.
5. **Ranking for importance and stability:**
   There should a criterion to classify the requirements as less or more important or more specifically

as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

6. **Modifiability:**
   SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.
7. **Verifiability:**
   A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.
8. **Traceability:**
   One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.
9. **Design Independence:**
   There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.
10. **Testability:**
    A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.
11. **Understandable by the customer:**
    An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.
12. **Right level of abstraction:**
    If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

Structure of SRS:

1. **Introduction :**
   - **(i) Purpose of this Document –**
     At first, main aim of why this document is necessary and what's purpose of document is explained and described.
   - **(ii) Scope of this document –**
     In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
   - **(iii) Overview –**
     In this, description of product is explained. It's simply summary or overall review of product.
2. **General description :**
   In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.
3. **Functional Requirements :**
   In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.
4. **Interface Requirements :**
   In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.
5. **Performance Requirements :**
   In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.

6. **Design Constraints :**
   In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc.
7. **Non-Functional Attributes :**
   In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.
8. **Preliminary Schedule and Budget :**
   In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
9. **Appendices :**
   In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

6.(b) Justify which process model will be suitable to design insulin pump system

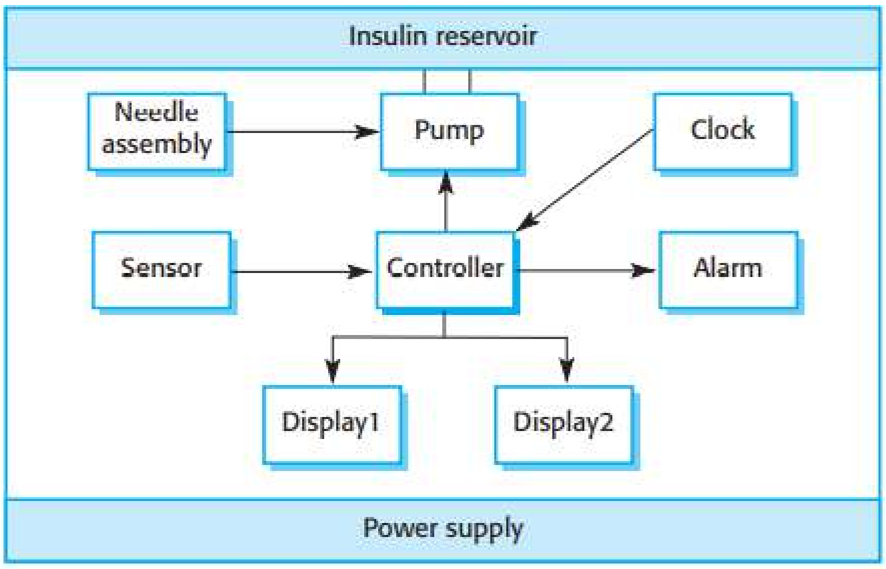
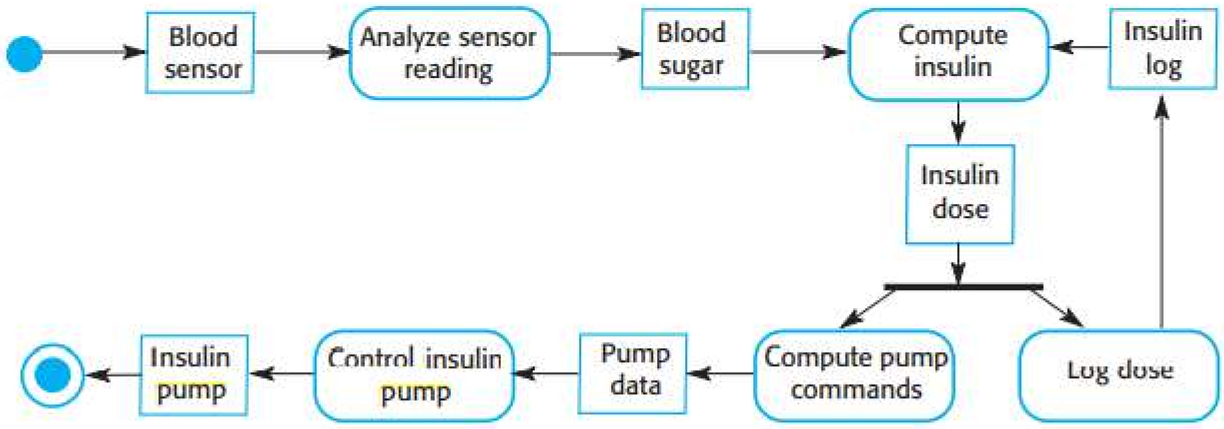


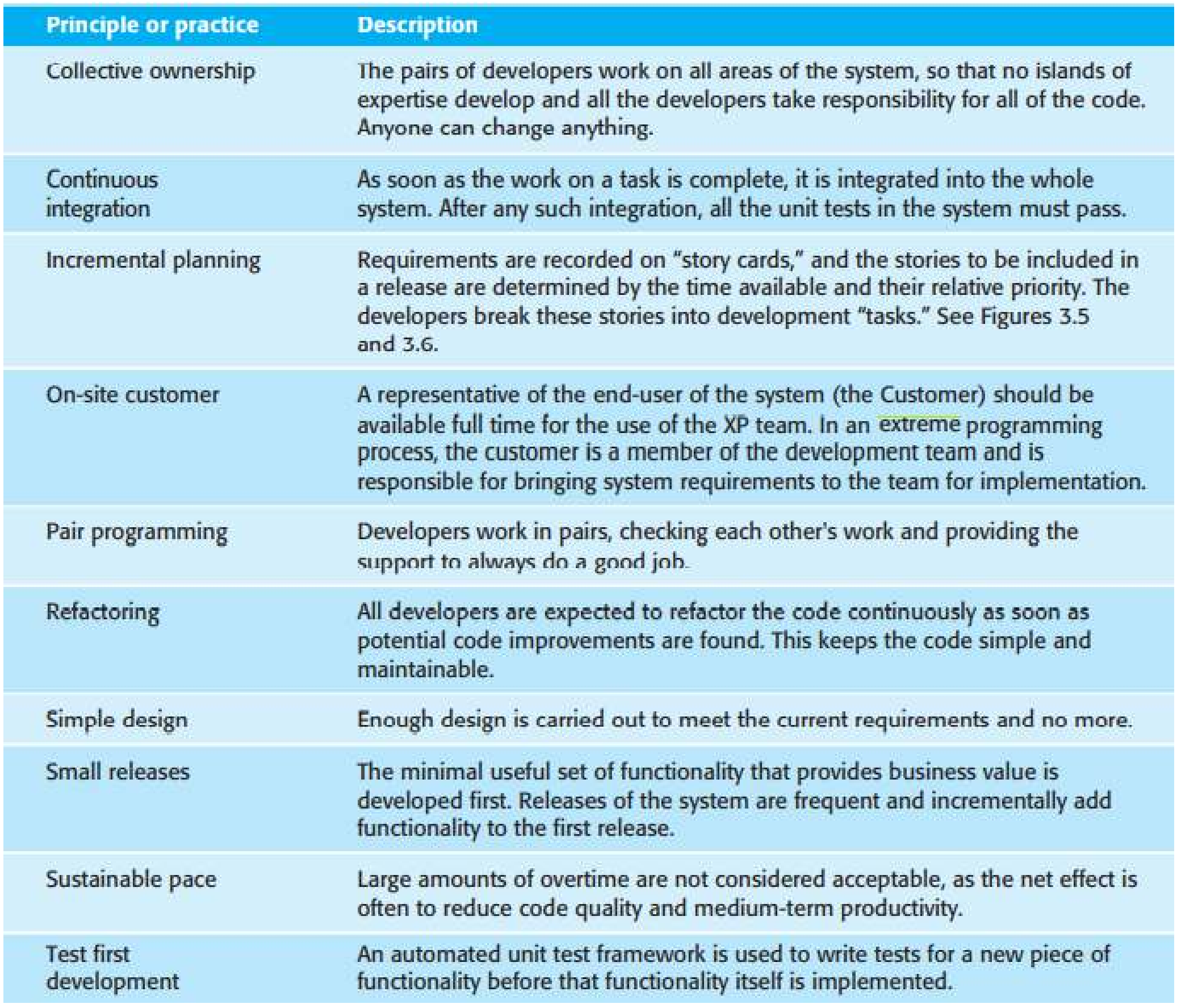
Figure 1.4 Insulin pump hardware architecture



It is important to note that part of software engineering includes choosing the right method for the type of system being built. For example, the insulin pump control system, which analyzes information from the blood collected by a sensor, and then controls the dose of insulin being administered to an individual with diabetes. This is considered a safety-critical system, since the patient's life depends on the reliability of the system. In this case, the design process should be determined by the requirements of the system, which include delivering insulin when required, and correctly administering differing levels of insulin, depending on readings from the sensor. This system would benefit most from the waterfall model of development, since safety is critical, and all specifications should be thoroughly documented and tested before being used on a human.

7. Explain in detail about extreme programming

The most significant approach to changing software development culture was the development of Extreme Programming (XP). The name was coined by Kent Beck (Beck 1998) because the approach was developed by pushing recognized good practice, such as iterative development, to "extreme" levels. For example, in XP, several new versions of a system may be developed by different programmers, integrated, and tested in a day. In XP, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system. Extreme programming was controversial as it introduced a number of agile practices that were quite different from the development practice of that time.

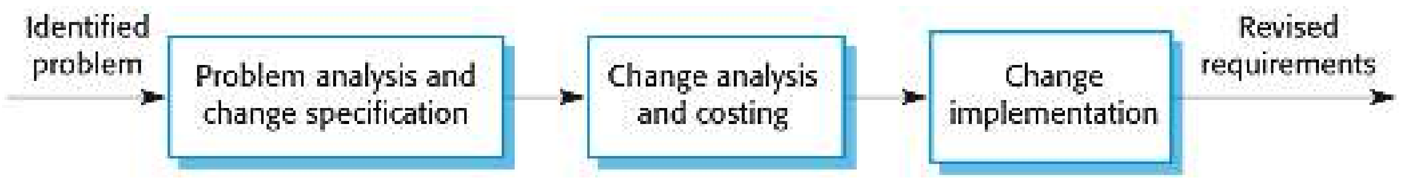


Principle or practice	Description
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Incremental planning	Requirements are recorded on "story cards," and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "tasks." See Figures 3.5 and 3.6.
On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Refactoring	All developers are expected to refactor the code continuously as soon as potential code improvements are found. This keeps the code simple and maintainable.
Simple design	Enough design is carried out to meet the current requirements and no more.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Sustainable pace	Large amounts of overtime are not considered acceptable, as the net effect is often to reduce code quality and medium-term productivity.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

8. Describe Requirements Change Management Process and planning.

- ✧ Deciding if a requirements change should be accepted
    - ▪ *Problem analysis and change specification*

        - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
    - *Change analysis and costing*
        - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
    - *Change implementation*
        - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.


Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

9. (a) Define: (i) Functional requirements and (ii) Non-functional requirements

- ✧ Functional requirements
    - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
    - May state what the system should not do.
- ✧ Non-functional requirements
    - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
    - Often apply to the system as a whole rather than individual features or services.

9. (b) Explain the different metrics of non-functional requirements

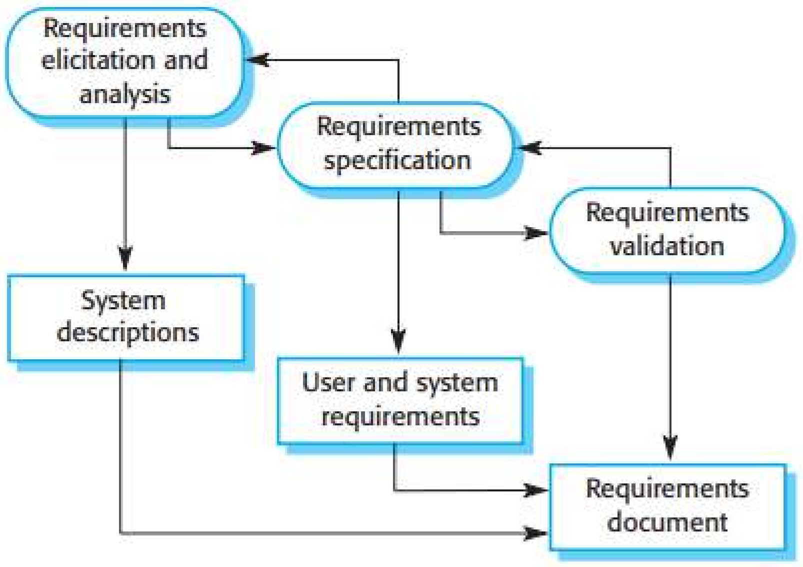
Property	**Measure**
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability

Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

10. Write Short Notes on: a) requirement specification b)View point, stockholders , c) scenario techniques, d) Ethnography e). Feasibility study

(a). Requirement Specification

Requirements specification The initial requirements for the system are proposed. These do not have to be elaborated in detail but should include brief descriptions of essential requirements and desirable system features. Requirements specification is the activity of translating the information gathered during requirements analysis into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided. In agile methods, requirements specification is not a separate activity but is seen as part of system development. Requirements are informally specified for each increment of the system just before that increment is developed. Requirements are specified according to user priorities. The elicitation of requirements comes from users who are part of or work closely with the development team.


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

(b) Viewpoint

A viewpoint is a way of organising the requirements for a software system, based on some perspective such as an end-user perspective or a manager's perspective.

A key strength of viewpoint-oriented analysis is that it recognises the existence of multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders. Viewpoints can be used as a way of classifying different types of stakeholder and other sources of requirements.

There are three generic types of viewpoint:

1. Interactor viewpoints that represent people or other systems that interact directly with the system. In the bank ATM system, examples of interactor viewpoints are the bank's customers and the bank's account database.
2. Indirect viewpoints that represent stakeholders who do not use the system themselves but who influence the requirements in some way. In the bank ATM system, examples of indirect viewpoints are the management of the bank and the bank security staff.
3. Domain viewpoints that represent domain characteristics and constraints that influence the system requirements. In the bank ATM system, an example of a domain viewpoint would be the standards that have been developed for inter-bank communications.

(c) Scenario

A scenario is a scene that illustrates some interaction with a proposed system. A scenario is a tool used during requirements analysis to describe a specific use of a proposed system. Scenarios capture the system, as viewed from the outside, e.g., by a user, using specific examples.
Some organizations have complex documentation standards for describing a scenario. At the very least, the description should include:
• A statement of the purpose of the scenario
• The individual user or transaction that is being followed through the scenario
• Assumptions about equipment or software
• The steps of the scenario

(d) Ethnography

✧ A social scientist spends a considerable time observing and analysing how people actually work.
✧ People do not have to explain or articulate their work.
✧ Social and organisational factors of importance may be observed.
✧ Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

Scope

✧ Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.
✧ Requirements that are derived from cooperation and awareness of other people's activities.
    ▪ Awareness of what other people are doing leads to changes in the ways in which we do things.
✧ Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Focused Ethnography

✧ Developed in a project studying the air traffic control process
✧ Combines ethnography with prototyping
✧ Prototype development results in unanswered questions which focus the ethnographic analysis.
✧ The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant

(e). Feasibility study

A feasibility study is a short, focused study that should take place early in the RE process. It should answer three key questions:
(1) Does the system contribute to the overall objectives of the organization?
(2) Can the system be implemented within schedule and budget using current technology? and
(3) Can the system be integrated with other systems that are used?
If the answer to any of these questions is no, you should probably not go ahead with the project.