1. State five symptoms of the present software crisis.

Ans: Software engineering appears to be among the few options available to tackle the present software crisis. To explain the present software crisis in simple words, it is considered the following that are being faced. The expenses that organizations all around the world are incurring on software purchases compared to those on hardware purchases have been showing a worrying trend over the years (as shown in fig.)

![Diagram showing the cost trend of hardware and software over years](image)

Organizations are spending larger and larger portions of their budget on software. Not only are the software products turning out to be more expensive than hardware, but they also present a host of other problems to the customers: software products are difficult to alter, debug, and enhance; use resources non-optimally; often fail to meet the user requirements; are far from being reliable; frequently crash; and are often delivered late. Among these, the trend of increasing software costs is probably the most important symptom of the present software crisis.

2. Identify the problem one would face, if he tries to develop a large software product without using software engineering principles.

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 at various levels. The problem is that the complexity and the difficulty levels of the programs increase exponentially with their sizes as shown in fig.
For example, a program of size 1,000 lines of code has some complexity. But a program with 10,000 LOC is not 10 times more difficult to develop, but may be 100 times more difficult unless software engineering principles are used. Software engineering helps to reduce the programming complexity.

3. **Explain the problems that might be faced by an organization if it does not follow any software life cycle model.**

The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated by using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure.
A software life cycle model defines entry and exit criteria for every phase. A phase can start only if its phase-entry criteria have been satisfied. So without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle models (such as classical waterfall model, iterative waterfall model, prototyping model, evolutionary model, spiral model etc.) it becomes difficult for software project managers to monitor the progress of the project.

4. **Identify six different phases of a classical waterfall model.**
   The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it cannot be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model.
5. **Mention at least two reasons as to why classical waterfall model can be considered impractical and cannot be used in real projects.**

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

6. **Identify when does a prototype need to develop.**

   A prototype can be developed when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

7. **Identify three reasons for the necessity of developing a prototype during software development.**

   There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer’s needs:

   - how screens might look like
   - how the user interface would behave
   - how the system would produce outputs

   This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.
Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

8. Identify at least two activities carried out during each phase of a spiral model.

The Spiral model of software development is shown in fig. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. The following activities are carried out during each phase of a spiral model.

**First quadrant (Objective Setting)**

1. During the first quadrant, it is needed to identify the objectives of the phase.
Examine the risks associated with these objectives.

- **Second Quadrant (Risk Assessment and Reduction)**
  - A detailed analysis is carried out for each identified project risk.
  - Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

- **Third Quadrant (Development and Validation)**
  - Develop and validate the next level of the product after resolving the identified risks.

- **Fourth Quadrant (Review and Planning)**
  - Review the results achieved so far with the customer and plan the next iteration around the spiral.
  - Progressively more complete version of the software gets built with each iteration around the spiral.

9. **Without developing an SRS document an organization might face severe problems. Identify those problems.**

   The important problems that an organization would face if it does not develop an SRS document are as follows:

   - Without developing the SRS document, the system would not be implemented according to customer needs.
   - Software developers would not know whether what they are developing is what exactly required by the customer.
   - Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
   - It will be very much difficult for user document writers to write the users’ manuals properly without understanding the SRS document.

10. **Identify at least five important items developed during software design phase.**

    For a design to be easily implementable in a conventional programming language, the following items must be designed during this phase.

    - Different modules required to implement the design solution.
    - Control relationship among the identified modules. The relationship is also known as the call relationship or invocation relationship among modules.
    - Interface among different modules. The interface among different modules identifies the exact data items exchanged among the modules.
• Data structures of the individual modules.
• Algorithms required to implement the individual modules.

11. Represent the following relations among classes using UML diagram.

1. Students credit 5 courses each semester. Each course is taught by one or more teachers.

2. Bill contains number of items. Each item describes some commodity, the price of unit, and total on this price.
12. What is the necessity for developing use case diagram? How to identify use cases of a system?

From use case diagram, it is obvious that the utility of the use cases are represented by ellipses. They along with the accompanying text description serve as a type of requirements specification of the system and form the core model to which all other models must conform. But, what about the actors (stick person icons)? One possible use of identifying the different types of users (actors) is in identifying and implementing a security mechanism through a login system, so that each actor can involve only those functionalities to which he is entitled to. Another possible use is in preparing the documentation (e.g. users’ manual) targeted at each category of user. Further, actors help in identifying the use cases and understanding the exact functioning of the system.

The use case model for any system consists of a set of “use cases”. Intuitively, use cases represent the different ways in which a system can be used by the users. A simple way to find all the use cases of a system is to ask the question: “What the users can do using the system?” Thus for the Library Information System (LIS), the use cases could be:

- issue-book
- query-book
- return-book
- create-member
- add-book, etc.

Use cases correspond to the high-level functional requirements. The use cases partition the system behavior into transactions, such that each transaction performs some useful action from the user’s point of view. To complete each transaction may involve either a single message or multiple message exchanges between the user and the system to complete.