Q 15 - What is the risk? Identify six possible risks that could arise in software projects. Discuss how you would manage those risks at different phases.

Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to a lack of information, control, or time. A possibility of suffering from loss in the software development process is called a software risk. Loss can be anything, an increase in production cost, development of poor quality software, not being able to complete the project on time. Software risk exists because the future is uncertain and there are many known and unknown things that cannot be incorporated in the project plan. A software risk can be of two types (a) internal risks that are within the control of the project manager and (2) external risks that are beyond the control of the project manager. Risk management is carried out to:

1. Identify the risk
2. Reduce the impact of risk
3. Reduce the probability or likelihood of risk
4. Risk monitoring

A project manager has to deal with risks arising from three possible cases:

1. **Known knowns** are software risks that are actually facts known to the team as well as to the entire project. For example not having enough developers can delay the project delivery. Such risks are described and included in the Project Management Plan.
2. **Known unknowns** are risks that the project team is aware of but it is unknown that such risk exists in the project or not. For example, if the communication with the client is not of a good level then it is not possible to capture the requirement properly. This is a fact known to the project team however whether the client has communicated all the information properly or not is unknown to the project.
3. **Unknown Unknowns** are the kind of risks about which the organization has no idea. Such risks are generally related to technology such as working with technologies or tools that you have no idea about because your client wants you to work that way suddenly exposes you to absolutely unknown risks.

Software risk management is all about risk quantification of risk. This includes:

1. Giving a precise description of risk event that can occur in the project
2. Defining risk probability that would explain what are the chances for that risk to occur

3. Defining How much loss a particular risk can cause
4. Defining the liability potential of risk

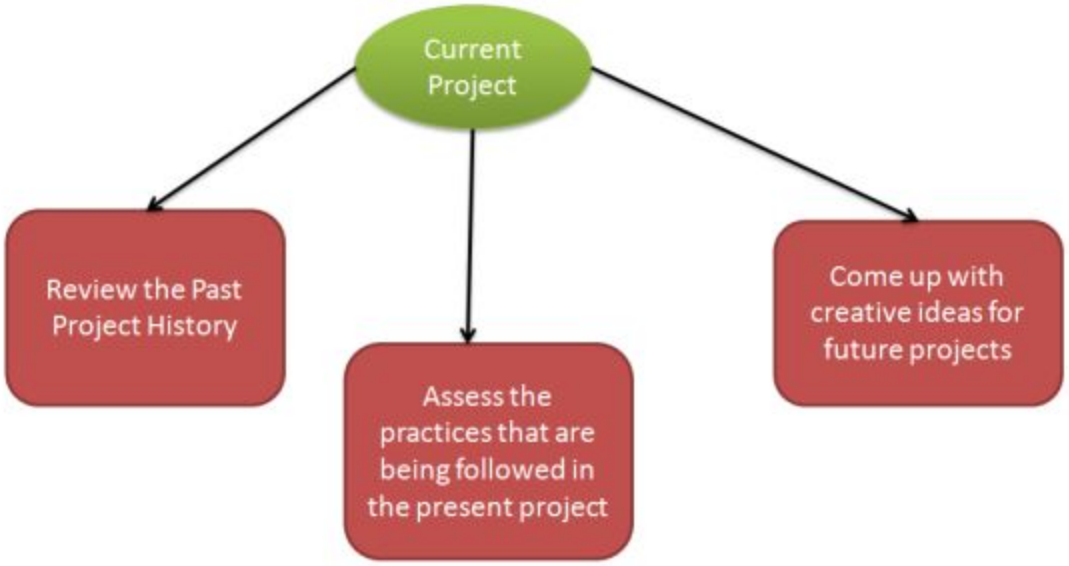
Risk Management comprises of following processes:

1. Software Risk Identification
2. Software Risk Analysis
3. Software Risk Planning
4. Software Risk Monitoring

These Processes are defined below.

Software Risk Identification

In order to identify the risks that your project may be subjected to, it is important to first study the problems faced by previous projects. Study the project plan properly and check for all the possible areas that are vulnerable to some of the other types of risks. The best way of analyzing a project plan is by converting it to a flowchart and examine all essential areas. It is important to conduct a few brainstorming sessions to identify the known unknowns that can affect the project. Any decision taken related to technical, operational, political, legal, social, internal or external factors should be evaluated properly.



Software Risk Identification

In this phase of Risk management, you have to define processes that are important for risk identification. All the details of the risk such as unique Id, the date on which it was identified, description, and so on should be clearly mentioned.

Software Risk Analysis

Software Risk analysis is a very important aspect of risk management. In this phase, the risk is identified and then categorized. After the categorization of risk, the level, likelihood (percentage), and impact of the risk are analyzed. The likelihood is defined in percentage after examining what are the chances of risk to occur due to various technical conditions. These technical conditions can be:

1. The complexity of the technology
2. Technical knowledge possessed by the testing team
3. Conflicts within the team
4. Teams being distributed over a large geographical area
5. Usage of poor quality testing tools

With impact we mean the consequence of a risk in case it happens. It is important to know about the impact because it is necessary to know how a business can get affected:

1. What will be the loss to the customer
2. How would the business suffer
3. Loss of reputation or harm to society
4. Monetary losses
5. Legal actions against the company
6. Cancellation of business license

Level of risk is identified with the help of:

Qualitative Risk Analysis: Here you define risk as:

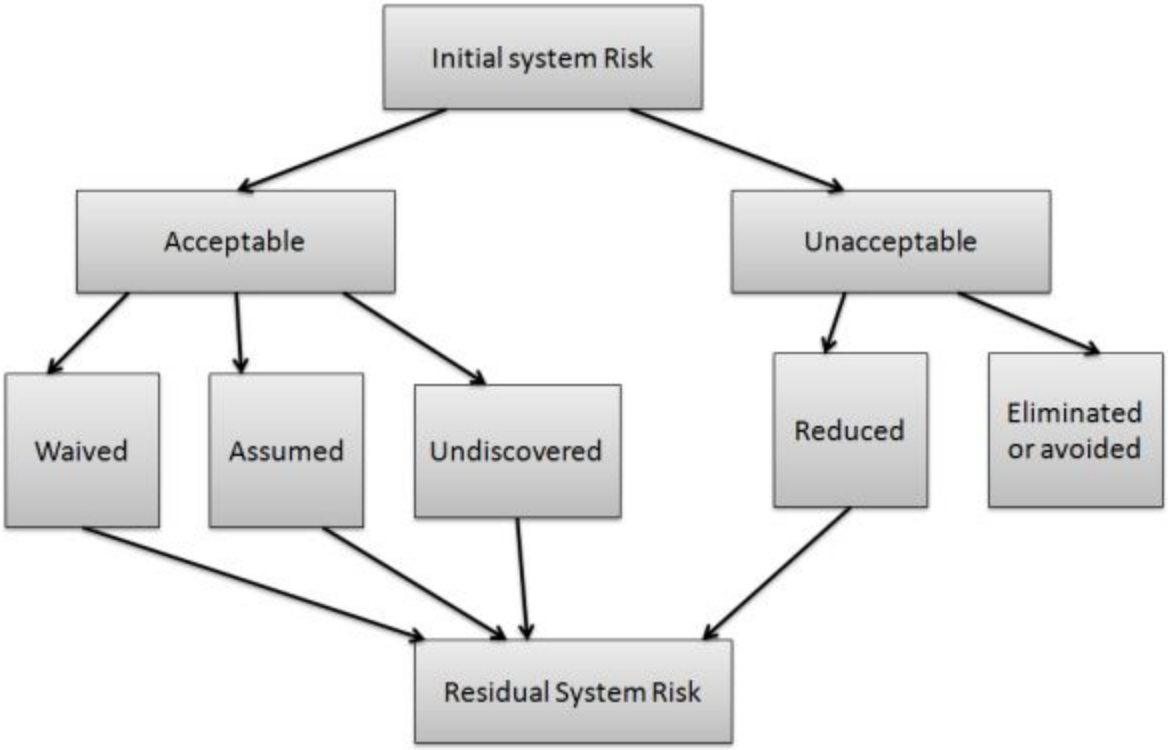
- High
- Low
- Medium

Quantitative Risk Analysis: can be used for software risk analysis but is considered inappropriate because risk level is defined in % which does not give a very clear picture.

Software Risk Planning

Software risk planning is all about:

1. Defining preventive measures that would lower down the likelihood or probability of various risks.
2. Define measures that would reduce the impact in case a risk happens.
3. Constant monitoring of processes to identify risks as early as possible.



Software Risk Planning

Software Risk Monitoring
Software risk monitoring is integrated into project activities and regular checks are conducted on top risks. Software risk monitoring comprises of:

- Tracking of risk plans for any major changes in actual plan, attribute, etc.
- Preparation of status reports for project management.
- Review risks and risks whose impact or likelihood has reached the lowest possible level should be closed.
- Regularly search for new risks

Q 16 - What is software quality? Discuss software quality attributes.

Software quality product is defined in terms of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of

use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Definition by IEEE
The degree to which a system, component, or process meets specified requirements.
The degree to which a system, component, or process meets customer or user needs or expectations.

Definition by ISTQB
Quality: The degree to which a component, system, or process meets specified requirements and/or user/customer needs and expectations.
software quality: The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

Attributes of Software Quality are -

Reliability

- Measure if the product is reliable enough to sustain in any condition. Should give consistently correct results.
- Product reliability is measured in terms of working on the project under different working environments and different conditions.

Maintainability

- Different versions of the product should be easy to maintain. For development it should be easy to add code to the existing system, it should be easy to upgrade for new features and new technologies from time to time.
- Maintenance should be cost-effective and easy. The system is easy to maintain and correcting defects or making a change in the software.

Usability

- This can be measured in terms of ease of use. The application should be user-friendly. It should be easy to learn. Navigation should be simple.
- Easy to use for input preparation, operation, and interpretation of the output.

- Provide consistent user interface standards or conventions with our other frequently used systems.
- Easy for new or infrequent users to learn to use the system.

Portability

- This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioral issues related to porting.

Correctness

- The application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means the application should adhere to functional requirements.

Efficiency

- Major system quality attribute. Measured in terms of time required to complete any task given to the system. For example, the system should utilize processor capacity, disk space, and memory efficiently.
- If the system is using all the available resources then the user will get degraded performance failing the system for efficiency. If the system is not efficient then it can not be used in real-time applications.

Integrity or Security

- Integrity comes with security. System integrity or security should be sufficient to prevent unauthorized access to system functions, preventing information loss, ensure that the software is protected from virus infection, and protecting the privacy of data entered into the system.

Testability

- The system should be easy to test and find defects. If required it should be easy to divide into different modules for testing.

Flexibility

- Should be flexible enough to modify. Adaptable to other products with which it needs interaction. It should be easy to interface with other standard 3rd party components.

Reusability

- Software reuse is a good cost-efficient and time-saving development way. Different code libraries classes should be generic enough to use easily in different application modules. Dividing application into different modules so that modules can be reused across the application.

Interoperability

- The interoperability of one system to another should be easy for the product to exchange data or services with other systems. Different system modules should work on different operating system platforms, different databases, and protocols conditions.
- Applying above quality attributes standards we can determine whether the system meets the requirements of quality or not.

Q 17 - Explain why the process of project planning is iterative and why a plan must be continually reviewed during a software project.

We know that the project plan is developed based on the available information at that moment. Thus, at the beginning of the project, we develop the plan without knowing enough uncertain matters that are relevant to the project. When the project progresses, there are more and more matters relevant to the project become available. Thus, we need to revise the project plan iteratively. The progress of the project through phrases will make some changes to the plan, so the plan needs to be updated to reflect this change to help with monitoring the project.

Every project plan needs to be monitored and updated on a regular basis for the simple reason it is impossible to predict the future with certainty, therefore the plan will be wrong. The degree of uncertainty varies depending on the project's typology and many other factors which indicates the best approach to maintaining the plan.

In traditional – linear - project management, the approach is to implement the activities under the assumption that all events affecting the project are predictable, that activities are well understood by everybody, and there is no need to revisit the plans. Unfortunately, this approach proves to be not very effective, given the level of uncertainty on many development projects. What happens is that the original assumptions under which the project plan was built change and in some cases in dramatic ways. What was originally assumed to be true is no longer valid. Assumptions about approvals, additional funding, economic and social conditions change dynamically

and the project needs to have the flexibility to adapt to these changes.

The project should find these opportunities to review the original assumptions and make the appropriate changes to the plans, specifically in the areas of scheduling, risks, and stakeholders. This approach consists of a series of iterative planning and development cycles, allowing a project team to constantly evaluate the implementation and results of the project and obtain immediate feedback from beneficiaries, or stakeholders.

Q 18 - Some very large software projects involve writing millions of lines of code. Explain why the effort estimation models, such as COCOMO, might not work well when applied to very large systems.

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort**: Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule**: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

The effort estimation models such as COCOMO estimate the software size based on the application and function points, and lines of code. For smaller projects, the estimation based on these factors will be effective. But in the case of large systems, where there will be millions of lines of codes, coding is only a small fraction of the total effort. LOC becomes a minor factor in project estimation. The estimation models may not work well when there is ambiguity in counting codes. Effort estimation depends on various factors such as requirements, plans, people, etc. So for very large systems, effort estimation models such as COCOMO might not work well.

Q 19 - What are the objectives of testing? Explain why testing can only detect the presence of errors, not their absence.

Software testing is a crucial element in the software development life cycle (SDLC),

which can help software engineers save time & money for organizations by finding errors and defects during the early stages of software development. With the assistance of this process, one can examine various components associated with the application and guarantee its appropriateness.

The goals and objectives of software testing are numerous, which when achieved help developers build a defectless and error-free software and application that has exceptional performance, quality, effectiveness, security, among other things. Though the objective of testing can vary from company to company and project to project, there are some goals that are similar for all. These objectives are:

Verification: A prominent objective of testing is verification, which allows testers to confirm that the software meets the various business and technical requirements stated by the client before the inception of the whole project. These requirements and specifications guide the design and development of the software, hence they are required to be followed rigorously. Moreover, compliance with these requirements and specifications is important for the success of the project as well as to satisfy the client.

Validation: Confirms that the software performs as expected and as per the requirements of the clients. Validation involves checking the comparing the final output with the expected output and then making necessary changes if there is a difference between the two.

Defects: The most important purpose of testing is to find different defects in the software to prevent its failure or crash during implementation or go-live of the project. Defects if left undetected or unattended can harm the functioning of the software and can lead to loss of resources, money, and reputation of the client. Therefore, software testing is executed regularly during each stage of software development to find defects of various kinds. The ultimate source of these defects can be traced back to a fault introduced during the specification, design, development, or programming phases of the software.

Providing Information: With the assistance of reports generated during the process of software testing, testers can accumulate a variety of information related to the software and the steps taken to prevent its failure. These then can be shared with all the stakeholders of the project for a better understanding of the project as well as to establish transparency between members.

Preventing Defects: During the process of testing the aim of testes to identify defects

and prevent them from occurring aging in the future. To accomplish this goal, the software is tested rigorously by independent testers, who are not responsible for software development.

Quality Analysis: Testing helps improve the quality of the software by constantly measuring and verifying its design and coding. Additionally, various types of testing techniques are used by testers, which help them achieve the desired software quality.

Compatibility: It helps validate the application's compatibility with the implementation environment, various devices, Operating Systems, user requirements, among other things.

For Optimum User Experience: Easy software and application accessibility and optimum user experience are two important requirements that need to be accomplished for the success of any project as well as to increase the revenue of the client. Therefore, to ensure this software is tested again and again by the testers with the assistance of stress testing, load testing, spike testing, etc.

Verifying Performance & Functionality: It ensures that the software has superior performance and functionality. This is mainly verified by placing the software under extreme stress to identify and measure its all plausible failure modes. To ensure this, performance testing, usability testing, functionality testing, etc. are executed by the testers.

Testing can detect only the presence of errors, not their absence because the main goal of the testing is: to observe the behavior of the particular software and to check whether it meets its requirement expectation or not. ... It is always possible that a test overlooked could discover a further problem with the system.

Q 20 - What are the different levels of testing and the goals of the different levels? For each level, specify the most suitable testing approach.

There are generally four recognized levels of testing: unit/component testing, integration testing, system testing, and acceptance testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the

test. Below you can see 4 different levels of testing:

1. Unit/Component Testing
2. Integration testing
3. System testing
4. Acceptance testing

1. Unit/component testing - Unit testing aims to verify each part of the software by isolating it and then perform tests to demonstrate that each individual component is correct in terms of fulfilling requirements and the desired functionality. This type of testing is performed at the earliest stages of the development process, and in many cases, it is executed by the developers themselves before handing the software over to the testing team. The advantage of detecting any errors in the software early in the day is that by doing so the team minimizes software development risks, as well as time and money wasted in having to go back and undo fundamental problems in the program once it is nearly completed.

2. Integration testing - Integration testing aims to test different parts of the system in combination in order to assess if they work correctly together. By testing the units in groups, any faults in the way they interact together can be identified. There are many ways to test how different components of the system function at their interface; testers can adopt either a bottom-up or a top-down integration method. In bottom-up integration testing, testing builds on the results of unit testing by testing a higher-level combination of units (called modules) in successively more complex scenarios. It is recommended that testers start with this approach first, before applying the top-down approach which tests higher-level modules first and studies simpler ones later.

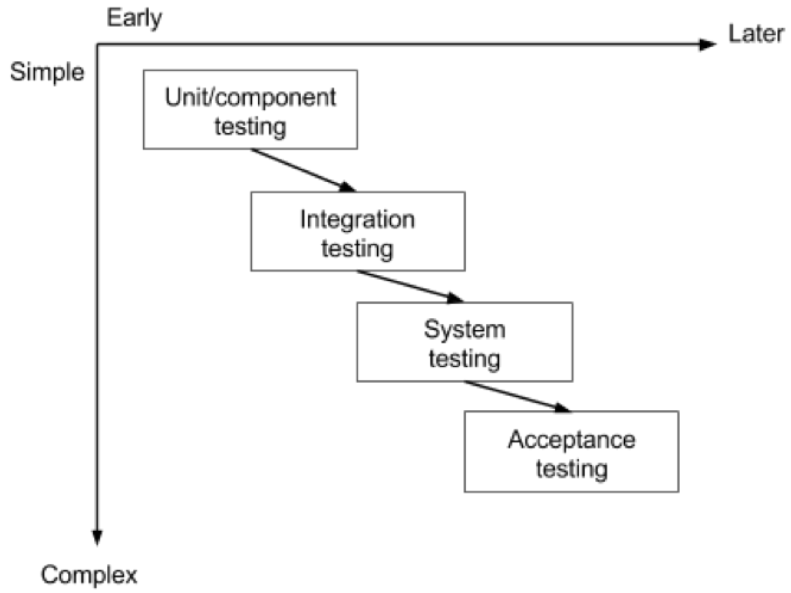
3. System testing - The next level of testing is system testing. As the name implies, all the components of the software are tested as a whole in order to ensure that the overall product meets the requirements specified. System testing is a very important step as the software is almost ready to ship and it can be tested in an environment that is very close to that which the user will experience once it is deployed. System testing enables testers to ensure that the product meets business requirements, as well as determine that it runs smoothly within its operating environment. This type of testing is typically

performed by a specialized testing team.

4. Acceptance testing - acceptance testing is the level in the software testing process where a product is given the green light or not. The aim of this type of testing is to evaluate whether the system complies with the end-user requirements and if it is ready for deployment. The testing team will utilize a variety of methods, such as pre-written scenarios and test cases to test the software and use the results obtained from these tools to find ways in which the system can be improved. The scope of acceptance testing ranges from simply finding spelling mistakes and cosmetic errors to uncovering bugs that could cause a major error in the application. By performing acceptance tests, the testing team can find out how the product will perform when it is installed on the user's system.

The testing sequence

These four testing types cannot be applied haphazardly during development. There is a logical sequence that should be adhered to in order to minimize the risk of bugs cropping up just before the launch date. Any testing team should know that testing is important at every phase of the development cycle. By progressively testing the simpler components of the system and moving on the bigger, more complex groupings, the testers can rest assured they are thoroughly examining the software in the most efficient way possible.



The four levels of testing shouldn't only be seen as a hierarchy that extends from simple to complex, but also as a sequence that spans the whole development process from the early to the later stages. Note however that later does not imply that acceptance testing is done only after say 6 months of development work.

Q 21 - What is software reliability? Discuss the following models of software reliability (a) Basic Execution Time model (b) Jelinski-Moranda model.

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The high complexity of software is the major contributing factor to Software Reliability problems. Software Reliability is not a function of time - although researchers have come up with models relating the two. The modeling technique for Software Reliability is reaching its prosperity, but before using the technique, we must carefully select the appropriate model that can best suit our case. Measurement in software is still in its infancy. No good quantitative methods have been developed to represent Software Reliability without excessive limitations. Various approaches can be used to improve the reliability of software, however, it is hard to balance development time and budget with software reliability.

Reliability Growth Models -The exponential model can be regarded as the basic form of software reliability growth models. The basic execution model is the most popular and widely used reliability growth model, mainly because:

- It is practical, simple and easy to understand;
- Its parameters clearly relate to the physical world.
- It can be used for accurate reliability prediction. -The basic execution model specifies failure behavior initially using execution time. Execution time may later be converted in calendar time.
- The failure behavior is a nonhomogeneous Poisson process, which means the associated probability distribution is a Poisson process whose characteristics vary in time.
- It is equivalent to the M-O logarithmic Poisson execution time model, with different mean value functions.
- The mean value function, in this case, is based on an exponential distribution.

Variables involved in the basic execution model:

- Failure intensity (lambda): number of failures per time unit.
- Execution time (tau): time since the program is running.
- Mean failures experienced (μ): mean failures experienced in a time interval

- In the basic execution model, the mean failures experienced m is expressed in terms of the execution time t as

$$m(t) = n_0 \times \left(1 - e^{-\frac{I_0}{n_0}t}\right)$$

The Jelinski-Moranda (JM) model, which is also a Markov process model, has strongly affected many later models which are in fact modifications of this simple model.

Characteristics of JM Model
Following are the characteristics of JM-Model

- It is a Binomial type model
- It is certainly the earliest and certainly one of the most well-known black-box models.
- The J-M model always yields an over-optimistic reliability prediction.
- JM Model follows a perfect debugging step, i.e., the detected fault is removed with certainty in a simple model.
- The constant software failure rate of the J?M model at the i^th failure interval is given by:

$\lambda(ti) = \phi\,[N-(i-1)]$, i=1, 2... N _____ equation 1

Where

ϕ=a constant of proportionality indicating the failure rate provided by each fault

N=the initial number of errors in the software

ti=the time between (i-1)th and (i)th failure.

The mean value and the failure intensity methods for this model which belongs to the binominal type can be obtained by multiplying the inherent number of faults by the cumulative failure and probability density functions (pdf) respectively:

$\mu(ti) = N(1-e-\phi ti)$ _____equation 2

And

€(ti)=Nϕe-ϕti _____equation 3

The assumptions made in the J-M model contains the following:

- The number of initial software errors is unknown but fixed and constant.
- Each error in the software is independent and equally likely to cause a failure during a test.
- Time intervals between occurrences of failure are separate, exponentially distributed random variables.
- The software failure rate remains fixed over the ranges among fault occurrences.
- The failure rate is corresponding to the number of faults that remain in the software.
- A detected error is removed immediately, and no new mistakes are introduced during the removal of the detected defect.
- Whenever a failure appears, the corresponding fault is reduced with certainty.

Q 23 - If the primary goal is to make software maintainable, list some of the things you will do and some of the things you will not do during coding and testing

Some points that a software engineer must keep in mind while working on any software are as follows

- Design for maintainability from the outset
- Iterative development and regular reviews help to improve quality - see the section below
- Readable code is easy to understand ("write programs for people")
- Refactor code to improve its understandability
- Relevant documentation helps developers understand the software
- Automated build make the code easy to compile
- Automated tests make it easy to validate changes

- Continuous integration makes the code easier to build and test
- Version control helps keep code, tests, and documentation up to date and synchronized
- Change the way you work to make maintainability a key goal

Q 25 - Discuss reverse engineering and re-engineering

RE Engineering:

• Restructuring or rewriting part or all of a system without changing its functionality

• Applicable when some (but not all) subsystems of a larger system require frequent maintenance

• Reengineering involves putting in the effort to make it easier to maintain

• The re-engineered system may also be restructured and should be re-documented

When do you decide to re-engineer?

• When system changes are confined to one subsystem, the subsystem needs to be reengineered

• When hardware or software support becomes obsolete

• When tools to support restructuring are readily available

Re-engineering advantages:

Reduced risk

There is a high risk of new software development. There may be development problems, staffing problems and specification problems

Reduced cost

The cost of re-engineering is often significantly less than the costs of developing new software

The complete Software Reengineering life cycle includes:

- **Product Management**: Risks analysis, root cause analysis, business analysis, requirements elicitation and management, product planning and scoping,

competitive analysis

- **Research and Innovation**: Definition of a problem, data gathering, and analysis identifying a solution and developing best-of-breed or innovative algorithms, verification of quality for data and results, patent preparation
- **Product Development**: Technology analysis and selection, software architecture and design, data architecture, deployment architecture, prototyping and production code development, comprehensive software testing, data quality testing, and product packaging and deployment preparation
- **Product Delivery and Support**: Hardware/Platform analysis and selection, deployment and release procedures definition, installations and upgrades, tracking support issues, organizing maintenance releases.
- **Project Management**: Brings efficiency and productivity to your software re-engineering project by utilizing modern, practical software project management, software quality assurance, data quality assurance, and advanced risk management techniques.

Reverse Engineering:

Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object. The practice, taken from older industries, is now frequently used on computer hardware and software. Software reverse engineering involves reversing a program's machine code (the string of 0s and 1s that are sent to the logic processor) back into the source code that it was written in, using program language statements.

Reverse-engineering is used for many purposes: as a learning tool; as a way to make new, compatible products that are cheaper than what's currently on the market; for making software interoperate more effectively or to bridge data between different operating systems or databases, and to uncover the undocumented features of commercial products.

Following are reasons for reverse engineering a part or product:

1. The original manufacturer of a product no longer produces a product
2. There is inadequate documentation of the original design
3. The original manufacturer no longer exists, but a customer needs the product
4. The original design documentation has been lost or never existed
5. Some bad features of a product need to be designed out. For example, excessive wear might indicate where a product should be improved
6. To strengthen the good features of a product based on long-term usage of the product

7. To analyze the good and bad features of competitors' product
8. To explore new avenues to improve product performance and features
9. To gain competitive benchmarking methods to understand the competitor's products and develop better products
10. The original CAD model is not sufficient to support modifications or current manufacturing methods
11. The original supplier is unable or unwilling to provide additional parts
12. The original equipment manufacturers are either unwilling or unable to supply replacement parts or demand inflated costs for sole-source parts
13. To update obsolete materials or antiquated manufacturing processes with more current, less-expensive technologies.