QP code: 22530

Q1

1.

- Prototyping doesn't contribute to development goals
- Prototyping delays time to market
- Schedule is too tight to enable integrating results from prototype
- Excessive cost
- It is a slow process.
  Too much involvement of client, is not always preferred by the developer.
  Too many changes can disturb the rhythm of the development team.

2.

- Consider the effect of a risk on the total project rather than on just part of it.
- Consider the combined effect of related risks. The likelihood that your schedule will slip is greater if three activities on the same critical path have a significant risk of delay rather than just one
- Decision trees: These diagrams illustrate different situations that may occur as your project unfolds, the likelihood of each situation's occurrence, and the consequences of that occurrence to your project.
- Risk-assessment questionnaires: These formal data-collection instruments elicit expert opinion about the likelihood of different situations occurring and their associated effects.
- Automated impact assessments: These computerized spreadsheets consider — in combination — the likelihood that different situations will occur and the consequences if they do.

3.

Cohesion of a module represents how tightly bound the internal elements of the module are to one another. Coupling between modules is the strength of interconnection between modules or a measure of independence among modules.
Two modules are considered independent if one can function completely without the presence of other. Obviously, if two modules are independent, they are solvable and modifiable separately. However, all the modules in a system cannot be independent of each other, as they must interact so that together they produce the desired external behavior of the system.
Benefits of low coupling are

- maintainability – changes are confined in a single module
- testability – modules involved in unit testing can be limited to a minimum
- readability – classes that need to be analyzed are kept at a minimum

The benefits of high cohesion are

- Readability – (closely) related functions are contained in a single module
- Maintainability – debugging tends to be contained in a single module
- Reusability – classes that have concentrated functionalities are not polluted with useless functions

4.

Verification	Validation
Are we building the system right?	Are we building the right system?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	**Validation** is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.
Following activities are involved in **Verification**: Reviews, Meetings and Inspections.	Following activities are involved in **Validation**: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check whether implementation software is as per specification document or not.	**Validation** is carried out by testing team.
Execution of code is not comes under **Verification**.	Execution of code is comes under **Validation**.
Verification process explains whether the outputs are according to inputs or not.	**Validation** process describes whether the software is accepted by the user or not.
Verification is carried out before the Validation.	**Validation** activity is carried out just after the Verification.
Following items are evaluated during **Verification**: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during **Validation**: Actual product or Software under test.
Cost of errors caught in **Verification** is less than errors found in Validation.	Cost of errors caught in **Validation** is more than errors found in Verification.
It is basically manually checking the of documents and files like requirement specifications etc.	It is basically checking of developed program based on the requirement specifications documents & files.

Q2.

Requirements Gathering

1. Brainstorming
2. Document Analysis
3. Focus Group
4. Interface Analysis
5. Interview
6. Observation
7. Prototyping
8. Requirements Workshop
9. Reverse Engineering
10. Survey

Requirements Elicitation

Elicitation techniques

- Stakeholder analysis
- Analysis of existing systems or documentation, background reading
- Discourse analysis
- Task observation, ethnography
- Questionnaires
- Interviewing
- Brainstorming
- Joint Application Design (JAD)
- Prototyping
- Pilot system

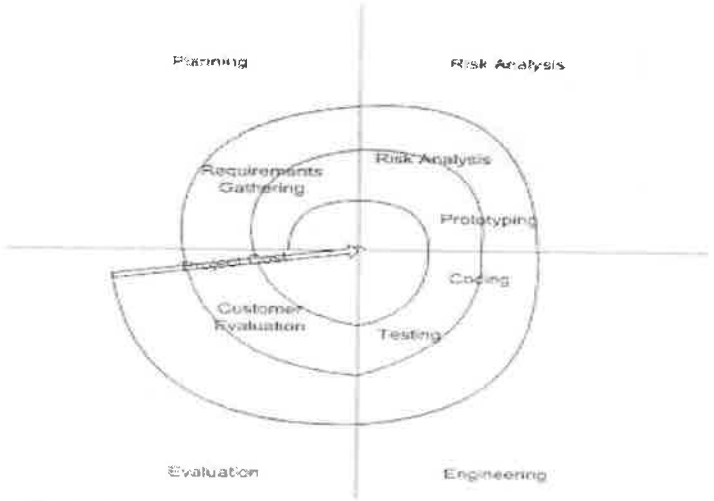
		• Use cases and scenarios • Risk analysis It may also involve a different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

b) Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Bussiness Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is **developed**, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.



Q3	Some of the main Characteristics of Agile development Methodology are as follows : **Early identification and resolution of issues** **Frequent Delivery** **Quality** **Visibility** **Iterative releases, Communication, continuous integration** **Quick & Good Return on your investments** **Frequent Testing** **Collaborative approach**

	There are 2 popular Agile methods available – Scrum and Extreme programming	
	b) **Resource Risks** *Organization* • Is there sufficient commitment to this project (including management, testers, QA, and other external but involved parties)? • Is this the largest project this organization has ever attempted? • Is there a well-defined process for software engineering? Requirements capture and management? *Funding* • Is the funding in place to complete project? • Has funding been allocated for training and mentoring? • Are there budget limitations such that the system must be delivered at a fixed cost or be subject to cancellation? • Are cost estimates accurate? *People* • Are enough people available? • Do they have appropriate skills and experience? • Have they worked together before? • Do they believe the project can succeed? • Are user representatives available for reviews? • Are domain experts available? *Time* • Is the schedule realistic? • Can functionality be scope-managed to meet schedules? • How critical is the delivery date? • Is there time to "do it right"? **Business Risks** • What if project funding is jeopardized (the other way to look at this is to ask "what can assure adequate funding")? • Is the projected value of the system greater than the projected cost? (be sure to account for the time-value of money and the cost of capital). • What if contracts cannot be made with key suppliers? **Technical Risks**	

Scope risks

- Can success be measured?
- Is there agreement on how to measure success?
- Are the requirements fairly stable and well understood?
- Is the project scope firm or does the scope keep expanding?
- Are the project development time scales short and inflexible?

Technological risks

- Has the technology been proven?
- Are reuse objectives reasonable?
    - An artifact must be used once before it can be re-used.
    - It may take several releases of a component before it is stable enough to reuse without significant changes.
- Are the transaction volumes in the requirements reasonable?
- Are the transaction rate estimates credible? Are they too optimistic?
- Are the data volumes reasonable? Can they be held on currently available mainframes, or, if the requirements lead you to believe a workstation or departmental system will be part of the design, can the data reasonably be held there?
- Are there unusual or challenging technical requirements that require the project team to tackle problems with which they are unfamiliar?
- Is success dependent on new or untried products, services or technologies, new or unproven hardware, software, or techniques?
- Are there external dependencies on interfaces to other systems, including those outside the enterprise? Do the required interfaces exist or must they be created?
- Are there availability and security requirements which are extremely inflexible (for example, "the system must never fail")?
- Are the users of the system inexperienced with the type of system being developed?
- Is there increased risk due to the size or complexity of the application or the newness of the technology?
- Is there a requirement for national language support?
- Is it possible to design, implement, and run this system? Some systems are just too huge or complex to ever work properly.

External dependency risk

- Does the project depend on other (parallel) development projects?
- Is success dependent on off the shelf products or externally-developed components? Is success dependent on the successful integration of development tools (design tools, compilers, and so on), implementation technologies (operating systems, databases, inter-process communication mechanisms, and so on). Do you have a back-up plan for delivering the project without these technologies?

Schedule Risks

85% of the risks have a direct or indirect impact on the schedule, and therefore implicitly

	on cost. Maybe 5% have only a cost impact. The rest have no direct impact on cost or schedule, but on quality for example. If a deadline is the enemy, approach it smoothly with incremental deliveries. Avoid having one massive delivery in an attempt to make the schedule. Delays mostly affect cost. In general, make your schedule commitment equal to your best estimate, plus some reasonable contingency. Where schedule is not negoatiable, try to reduce the number of requirements, by moving some low-priority requirements to evolution, to meet the project schedule.	
Q4	• Key concepts in discussing reliability: – Fault – Failure – Time – Three kinds of time intervals: MTTR, MTTF, MTBF **McCall's Software Quality Factors** ▸ Correctness: ◦ defects per KLOC ◦ maintainability ◦ mean time to change (MTTC) the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users ◦ spoilage = (cost of change / total cost of system) ▸ integrity ◦ threat = probability of attack (that causes failure) ◦ security = probability attack is repelled Integrity = Σ [1 - threat * (1 - security)]	
Q5	Configuration Management • Software changes are inevitable • One goal of software engineering is to improve how easy it is to change software • Configuration management is about change control. • Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project. • To ensure quality is maintained the change process must be audited. Software Configuration Items • Computer programs – source – executable • Documentation – technical – user • Data – contained within the program – external data (e.g. files and databases)	

Change Control

- Submission of Change Request (CR)
- Technical and business evaluation and impact analysis
- Approval by Change Control Board (CCB)
- Engineering Change Order (ECO) is generated stating
  - changes to be made
  - criteria for reviewing the changed CI
  - CI's checked out
- Changes made and reviewed
- CI's checked in

b) Test Cases and the Class Hierarchy
Scenario-Based Test Design
Testing Surface Structure and Deep Structure
Random Testing for OO Classes
Partition Testing at the Class Level
Multiple Class Testing
Tests Derived from Behavior Models