Q 2 (a) What do you mean by system requirement and system design?

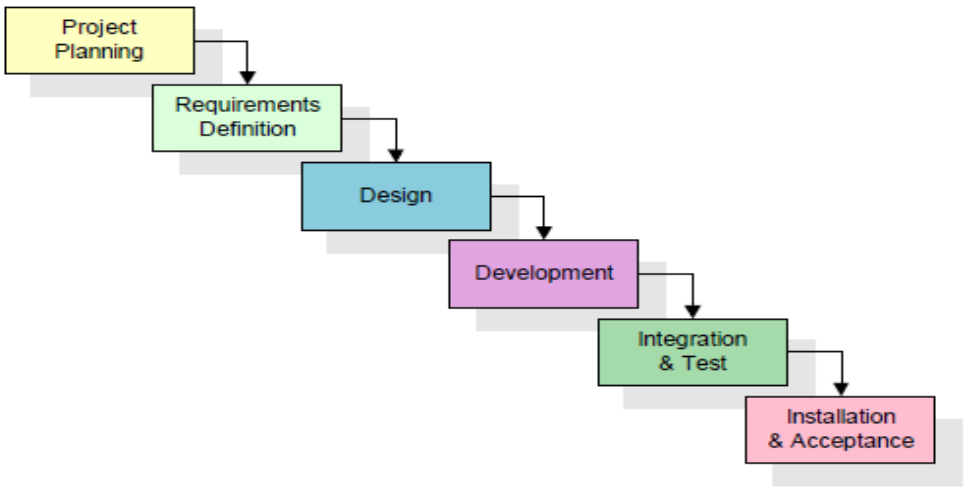
Answer

A requirement is an objective that must be met. Planners cast most requirements in functional terms, leaving design and implementation details to the developers. They may specify price, performance, and reliability objectives in fine detail, along with some aspects of the user interface. Sometimes, they describe their objectives more precisely than realistically. There are actually several kinds of requirements; the term requirement is awkward because it describes the concept of an objective or goal or necessary characteristic, but at the same time the term also describes a kind of formal documentation, namely the requirements document. Putting aside the particular document for now, requirements are instructions describing what functions the software is supposed to provide, what characteristics the software is supposed to have, and what goals the software is supposed to meet or to enable users to meet.

Design objectives assist in selecting a solution from a number that are offered to you. Only you know what is the most important feature of a new system, whether it should be fast, have large storage, be easy to use, or whatever. Unfortunately you can't have all you want; compromises have to be made. A number of teams were given an identical set of functional requirements, but each had a different design objective: some had to make the system fast, some small to use only a small amount of computer storage, some easy to use, etc. Each team delivered a system that met their top objective fully, and other objectives to a lesser degree. If you do not produce a set of design objectives, which are in a priority order, the developers will produce their own, and these might not be what you want. For the customer records example the top design objective could be that it easy for users to find customer information.

Q 2 (b) With the help of a suitable diagram, explain the software lifecycle.

Answer



The relationship of each stage to the others can be roughly described as a waterfall, wher the outputs from a specific stage serve as the initial inputs for the following stage.
During each stage, additional information is gathered or developed, combined with the inputs, and used to produce the stage deliverables. It is important to note that the additional information is restricted in scope; "new ideas" that would take the project in directions not anticipated by the initial set of high-level requirements are not incorporated into the project. Rather, ideas for new capabilities or features that are out-of-scope are preserved for later consideration.
After the project is completed, the Primary Developer Representative (PDR) and
Primary End-User Representative (PER), in concert with other customer and development team personnel develop a list of recommendations for enhancement of the current software.

Q 2 (c) Explain the risk management process. What are the different categories of risk?

Answer
Risk management identifies a new type of a risk that has a 100% probability of occurring but is ignored by the organization due to a lack of identification ability. For example, when deficient knowledge is applied to a situation, a knowledge risk materializes. Relationship risk appears when ineffective collaboration occurs. Process-engagement risk may be an issue when ineffective operational procedures are applied. These risks directly reduce the productivity of knowledge workers, decrease cost effectiveness, profitability, service, quality, reputation, brand value, and earnings quality. Intangible risk management allows risk management to create immediate value from the identification and reduction of risks that reduce productivity.
Risk management also faces difficulties in allocating resources. This is the idea of opportunity cost. Resources spent on risk management could have been spent on more profitable activities. Again, ideal risk management minimizes spending and minimizes the negative effects of risks.
"Risk are future uncertain events with a probability of occurrence and a potential for loss"
Risk identification and management are the main concerns in every software project. Effective analysis of software risks will help to effective planning and assignments of work.

Types of risks:

ScheduleRisk:
Project schedule get slip when project tasks and schedule release risks are not addressed properly. Schedule risks mainly affect on project and finally on company economy and may lead to project failure.
Schedules often slip due to following reasons:
Budget Risk:
Wrong budget estimation.
Cost overruns

Project scope expansion

Operational Risks:

Risks of loss due to improper process implementation, failed system or some external events risks.

Technical risks:Programmatic Risks:

These are the external risks beyond the operational limits. These are all uncertain risks are outside the control of the program.

Q 3 (a) Differentiate between the functional and non functional requirements.

Answer

Basically, functional requirements directly support the user requirements by describing the "processing" of the information or materials as inputs or outputs. Nonfunctional requirements generally support all users in that they describe the business standards and the business environment, as well as the overall user's experience (user attributes).

Functional	**Nonfunctional**
Product features	Product properties
Describe the work that is done	Describe the character of the work
Describe the actions with which the work is concerned	Describe the experience of the user while doing the work
Characterized by verbs	Characterized by adjectives

The functional requirements specify what the product must do. They relate to the actions.

Q 3 (b) What are the various activities performed during the requirement engineering process.

Answer

Requirements engineering can be divided into discrete chronological steps:

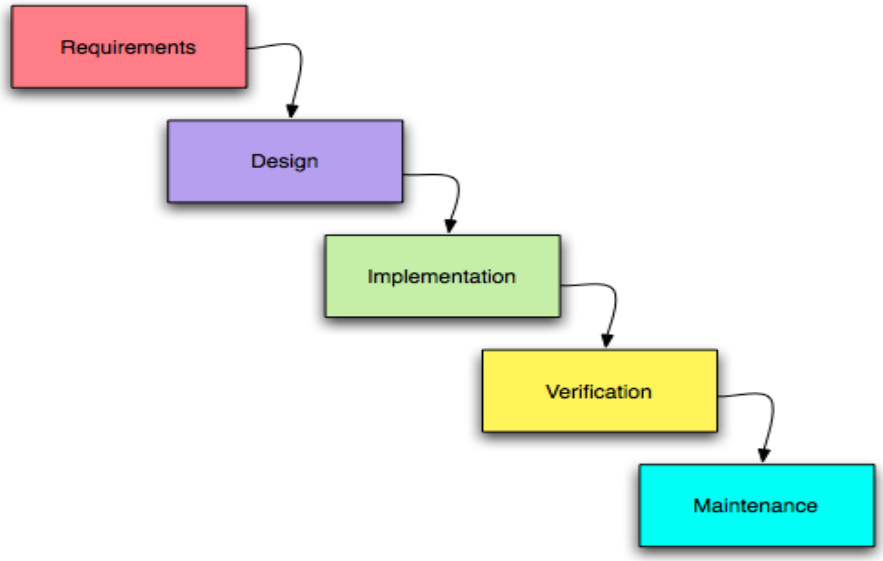
1. Requirements elicitation,
2. Requirements analysis and negotiation,
3. Requirements specification,
4. System modeling,
5. Requirements validation,
6. Requirements management.

Requirement engineering is "a sub discipline of systems engineering and software engineering that is concerned with determining the goals, functions, and constraints of hardware and software systems." In some life cycle models, the requirement engineering process begins with a feasibility study activity, which leads to a feasibility report. If the feasibility study suggests that the product should be developed, then requirement analysis can begin. If requirement analysis precedes feasibility studies, which may foster outside the box thinking, then feasibility should be determined before requirements are finalized.

Q 4 (a) Explain the various stages of software specification and its interface with the design process.

Answer

A software development process, also known as a software development lifecycle, is a structure imposed on the development of a software product. Similar terms include software life cycle and *software process*. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a lifecycle model a more general term and a software development process a more specific term. For example, there are many specific software development processes that 'fit' the spiral lifecycle model.



Q 4 (b) With the help of a neat diagram, draw and explain the RAD model.

Answer

Rapid application development (RAD) refers to a type of software development methodology that uses minimal planning in favor of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements.

"Rapid Application

Development (RAD) is a development lifecycle designed to give much faster development and higher-quality results than those achieved with the traditional lifecycle. It is designed to take the maximum advantage of powerful development software that has evolved recently."

RAD (rapid application development) is a concept that products can be developed faster and of higher quality through:

Gathering requirements using workshops or focus groups

Prototyping and early, reiterative user testing of designs

The re-use of software components
A rigidly paced schedule that defers design improvements to the next product version
Less formality in reviews and other team communication
Rapid Application Development encouraged the creation of quick-and-dirty prototype-style software which fulfilled most of the user's requirements but not necessarily all. Development would take place in a series of short cycles, called time boxes, each of which would deepen the functionality of the application a little more. Features to be implemented in each time box were agreed in advance and this game plan rigidly adhered to. The strong emphasis on this point came from unhappy experience with other development practices in which new requirements would tend to be added as the project was evolving, caused massive chaos and disrupting the already carefully prepared plans and development schedules. Rapid Application Development methodology advocated that development be undertaken by small, experienced teams using CASE (Computer Aided Software Engineering) tools to enhance their productivity.

Q 5 (a) Explain Modular Decomposition Style. Also write various properties of a modular system.

Answer Page Number 276 of Text Book

Q 5 (b) Differentiate between two-tier Client Server approach and three-tier Client Server architecture.

Answer

2-tier architecture

In 2-tier, the application logic is either buried inside the User Interface on the client or within the database on the server (or both). With two tier client/server architectures, the user system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients

3-tier architecture

In 3-tier, the application logic (or) process lives in the middle-tier, it is separated from the data and the user interface. 3-tier systems are more scalable, robust and flexible. In addition, they can integrate data from multiple sources. In the three tier architecture, a middle tier was added between the user system interface client environment and the database management server environment.
There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client
The most basic type of three tier architecture has a middle layer consisting of Transaction Processing (TP) monitor technology. The TP monitor technology is a type of message queuing, transaction scheduling, and prioritization service where the client connects to

the TP monitor (middle tier) instead of the database server. The transaction is accepted by the monitor, which queues it and then takes responsibility for managing it to completion, thus freeing up the client.

Q 6 (a) Explain different stages of an object oriented design process. Use suitable examples wherever necessary.

Answer

Object-Oriented Design is concerned with developing an object-oriented model of a software system to implement the identified requirements. The objects in an object oriented design are related to the solution to the problem that is being solved. There may be close relationships between some problem objects and some solution objects but the designer inevitably has to add new objects and to transform problem objects to implement the solution.

1. Understand and define the context and the modes of use of the system.
2. Design the system architecture
3. Identify the principal objects in the system
4. Develop design models
5. Specify object interfaces

Q 6 (b) Discuss the benefits and problems of software reuse.

Answer

Benefits of reuse

Increased dependability

Reused software, that has been tried and tested in working systems, should be m ore dependable than new software. The initial use of the software reveals any design and implementation faults. These are then fixed, thus reducing the number of failures when the software is reused.

Reduced process risk

If software exists, there is less uncertainty in the costs of reusing that software than in the costs of development. This is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub-systems are reused.

Effective use of specialists

Instead of application specialists doing the same work on different projects, these specialists can develop reusable software that encapsulate their knowledge.

Standards compliance

Some standards, such as user interface standards, can be implemented as a set of standard reusable components. For example, if menus in a user interfaces are implemented using reusable components, all applications present the same menu formats to users. The use of

standard user interfaces improves dependability as users are less likely to make mistake when presented with a familiar interface.

Accelerated development
Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time should be reduced.

Reuse problems

Increased maintenance Costs
If the source code of a reused software system or component is not available then maintenance costs may be increased as the reused elements of the system may become increasingly incompatible with system changes.

Lack of tool support
CASE toolsets may not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account.

Not-invented-here Syndrome
Some software engineers sometimes prefer to re-write components as they believe that they can improve on the reusable component. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.

Creating and maintaining a component library

Populating a reusable component library and ensuring the software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature.

Q 7 (a) Explain the general principles of user interface design.

Answer

structure The principle. Your design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with your overall user interface architecture.

1. **The simplicity principle**. Your design should make simple, common tasks simple to do, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

2. **The visibility principle**. Your design should keep all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with too many alternatives or confuse them with unneeded information.
3. **The feedback principle**. Your design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
4. **The tolerance principle**. Your design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonable.
5. **The reuse principle**. Your design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

Q 7 (b) Explain the basic elements of a component model. Write a brief note on CBSE.

Answer

Software component technology, which is based on building software systems from reusable components, has attracted attention because it is capable of reducing developmental costs. In a narrow sense, a software component is defined as a unit of composition, and can be independently exchanged in the form of an object code without source codes. The internal structure of the component is not available to the public.
The characteristics of the component-based development are the following:

- Black-box reuse
- Reactive-control and component's granularity
- Using RAD (rapid application development) tools
- Contractually specified interfaces
- Introspection mechanism provided by the component systems
- Software component market (CALS)

Q 8 (a) Differentiate between Verification and Validation.

Answer

Verification is quality control in which we only take corrective actions and Verification is quality assurance which involves preventive actions. Software companies combine both verification and validation in their software quality assurance departments to produce high quality software. Software Quality Assurance is an essential part of Software Development Life Cycle and Software Testing comes under Software Quality Assurance. Validation and Verification are part of Software Quality Assurance and Software Testing.
To understand the basic concept of Verification and Validation, let's

discuss them separately.
Verification is a process of ensuring that every product is designed according to the user requirements. Reviews and meetings are conducted to evaluate different documents. These include requirements, specifications, plans and code. Evaluation of all these documents is done via checklists, walkthroughs and inspections. After completion of verification we move one step further and start validation.
Validation involves the actual testing where we check the program design to see if it is according to the intended design. Validation is performed both by software developers and software testers. Unit testing comes under validation and it is performed by Software developers in order to check that their code is working correctly. Alpha and beta testing also comes under validation.

Q 8 (b) What are the different levels of testing? Explain briefly.

Answer
1. Levels of testing
•Unit Testing
Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.
These type of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Unit testing is also called *component testing*

•Integration Testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localised more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.[

•System Testing

System testing tests a completely integrated system to verify that it meets its requirements.

•Acceptance Testing

Acceptance testing can mean one of two things:

1. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression.
2. Acceptance testing is performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

•Regression testing

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, or old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, to very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk

Q 9 (b) What is SQA? Discuss different software quality factors.

Answer

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI

A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program. Note that none of these factors are binary; that is, they are not “either you have it or you don’t” traits. Rather, they are characteristics that one seeks to maximize in one’s software to optimize its quality. So rather than asking whether a software product “has” factor *x*, ask instead the *degree* to which it does (or does not).

Some software quality factors are listed here:

Understandability

Clarity of purpose. This goes further than just a statement of purpose; all of the design and user documentation must be clearly written so that it is easily

understandable. This is obviously subjective in that the user context must be taken into account: for instance, if the software product is to be used by software engineers it is not required to be understandable to the layman.

Completeness
Presence of all constituent parts, with each part fully developed. This means that if the code calls a subroutine from an external library, the software package must provide reference to that library and all required parameters must be passed. All required input data must also be available.

Conciseness
Minimization of excessive or redundant information or processing. This is important where memory capacity is limited, and it is generally considered good practice to keep lines of code to a minimum. It can be improved by replacing repeated functionality by one subroutine or function which achieves that functionality. It also applies to documents.

Text Book

Software Engineering, Ian Sommerville, 7th Edition, Pearson Education, 2004