**Analysis modelling**

In summary, analysis modelling is essential to the software development process, as it helps to ensure that the software meets the needs and requirements of the stakeholders, identifies potential issues early, improves communication and collaboration, reduces cost and time, and ensures the quality of the software.

**Design modelling**

Design modelling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level detail. Design modelling provides a variety of different views of the system like architecture plan for home or building. Different methods like data-driven, pattern-driven, or object-oriented methods are used for constructing the design model. All these methods use set of design principles for designing a model.

**Data Flow Diagram**

DFD stands for Data Flow Diagram. It is a visual representation of a system's flow of data and how it is processed or transformed. A DFD is typically used to describe a system at a high level of abstraction, showing the major processes, data stores, data flows, and external entities involved.

DFDs are made up of four main components:

1.Processes: These represent the activities or operations that are performed on data. A process transforms input data into output data.

2. Data stores: These represent the places where data is stored within the system. Data stores can be physical or virtual.

3. Data flows: These represent the movement of data between processes and data stores. Data flows are represented by arrows that show the direction of data flow.

4. External entities: These represent the sources or destinations of data that are outside the system being modelled. External entities can be people, other systems, or organizations.

DFDs can be used for a variety of purposes, including requirements analysis, system design, and documentation. They provide a high-level view of the system being modelled, which can help stakeholders understand how the system works and identify potential problems or opportunities for improvement.

**Explain**

Cardinality:

In the context of data modelling, cardinality refers to the number of occurrences of one entity that are associated with another entity in a relationship. Cardinality can be one-to-one, one-to-many, or many-to-many. One-to-one cardinality means that one occurrence of an entity is related to exactly one occurrence of another entity. One-to-many cardinality means that one occurrence of an entity is related to many occurrences of another entity, but each occurrence of the other entity is related to only one occurrence of the first entity. Many-to-many cardinality means that many occurrences of one entity are related to many occurrences of another entity.

Relationships:

Relationships describe the associations or connections between entities in a data model. Relationships can be one-to-one, one-to-many, or many-to-many, and they can be optional or mandatory. Optional relationships mean that an occurrence of one entity can be related to zero or more occurrences of another entity, while mandatory relationships mean that an occurrence of one entity must be related to at least one occurrence of another entity.

Data objects:

Data objects represent the entities or objects that are part of a system or process. Data objects can be physical or abstract and can be represented in a data model as rectangles with their name at the top.

Attributes:

Attributes are the properties or characteristics of a data object. They describe the specific details or information associated with an entity. For example, the attributes of a customer entity might include name, address, and phone number. Attributes are represented as ovals or ellipses in a data model, and they are usually connected to the entity they describe by a line. Attributes can also have data types, such as string, integer, or date.

**Decision table**

A decision table is a structured representation of a set of rules or conditions and the corresponding actions or outcomes that result from those rules or conditions. Decision tables are often used to model complex business logic or decision-making processes.

Here's an example of a decision table for determining the appropriate level of customer support based on a customer's account type and their support request:

|  |  |  |
| --- | --- | --- |
| Account Type | Support Request | Customer Support Level |
| Basic | Technical | Level 1 |
| Basic |  | Billing |
| Standard | Technical | Level 2 |
| Standard | Billing | Level 3 |
| Premium | Technical | Level 3 |
| Premium | Billing | Level 4 |
|  |  |  |

In this example, the decision table has three columns: Account Type, Support Request, and Customer Support Level. The rows represent all the possible combinations of account type and support request.

For each combination, the decision table specifies the appropriate customer support level. For example, if a customer has a Basic account and submits a technical support request, they will be provided with Level 1 customer support. If a customer has a Premium account and submits a billing support request, they will be provided with Level 4 customer support.

Decision tables can be used to simplify complex decision-making processes and ensure consistency in decision-making across different scenarios. They can also be used to identify gaps or errors in a decision-making process by highlighting cases where rules or conditions are not well-defined or ambiguous.

**The primary objectives of testing are:**

Validating software functionality: The main goal of testing is to validate that the software functions correctly and meets the specified requirements. Testing ensures that the software behaves as intended and performs its intended functions.

Finding defects and issues: Testing helps to identify defects, errors, and other issues in the software. By finding and fixing these issues, testing helps to improve the quality and reliability of the software.

Enhancing software usability: Testing helps to identify usability issues that may impact user satisfaction and experience. By addressing these issues, testing can help to enhance the usability of the software and improve user satisfaction.

Improving software performance: Testing helps to identify performance issues and bottlenecks that may impact the speed and efficiency of the software. By addressing these issues, testing can help to improve the performance of the software and ensure that it meets the required performance criteria.

In summary, testing is critical to ensuring that software is functional, defect-free, usable, and performs well. By achieving these objectives, testing helps to ensure that software is of high quality, reliable, and meets the needs of its users.

| **Black Box Testing** | **White Box Testing** |
| --- | --- |
| It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. |
| Implementation of code is not needed for black box testing. | Code implementation is necessary for white box testing. |
| It is mostly done by software testers. | It is mostly done by software developers. |
| No knowledge of implementation is needed. | Knowledge of implementation is required. |
| It can be referred to as outer or external software testing. | It is the inner or the internal software testing. |
| It is a functional test of the software. | It is a structural test of the software. |
| This testing can be initiated based on the requirement specifications document. | This type of testing of software is started after a detail design document. |
| No knowledge of programming is required. | It is mandatory to have knowledge of programming. |
| It is the behaviour testing of the software. | It is the logic testing of the software. |
| It is applicable to the higher levels of testing of software. | It is generally applicable to the lower levels of software testing. |
| It is also called closed testing. | It is also called as clear box testing. |
| It is least time consuming. | It is most time consuming. |
| It is not suitable or preferred for algorithm testing. | It is suitable for algorithm testing. |
| Can be done by trial and error ways and methods. | Data domains along with inner or internal boundaries can be better tested. |
| **Example:** Search something on google by using keywords | **Example:** By input to check and verify loops |
| **Black-box test design techniques-**   * Decision table testing * All-pairs testing * Equivalence partitioning * Error guessing | **White-box test design techniques-**   * Control flow testing * Data flow testing * Branch testing |
| **Types of Black Box Testing:**   * Functional Testing * Non-functional testing * Regression Testing | **Types of White Box Testing:**   * Path Testing * Loop Testing * Condition testing |
| It is less exhaustive as compared to white box testing. | It is comparatively more exhaustive than black box testing. |

**Test Plan**

A test plan is a comprehensive document that outlines the testing strategy, objectives, scope, approach, and schedule for a software project. It helps to ensure that the testing effort is well-planned, executed, and documented, resulting in high-quality software that meets the specified requirements.

**Test Case**

A test case design is the process of creating detailed test cases that cover all possible scenarios and use cases of the software. It involves understanding the requirements, identifying test scenarios, defining and writing test cases, prioritizing them, reviewing and refining them, validating them, and maintaining them. By following these steps, testers can ensure that the software is thoroughly tested and meets the specified requirements.

**Risk assessment**

Risk assessment is the process of identifying, evaluating, and prioritizing potential risks or threats that may affect an organization or a specific project. It involves analysing the likelihood and potential impact of each risk, and developing strategies to mitigate or manage those risks. Risk assessments can be performed for a variety of purposes, including compliance with regulatory requirements, project planning, and business continuity planning.

**Risk prioritization**

Risk prioritization is the process of ranking potential risks based on their likelihood and potential impact, in order to focus resources on the most critical risks. By prioritizing risks, organizations can ensure that they are focusing their efforts and resources on the areas that pose the greatest threat to their business.

**The 4 P's of the software project spectrum are:**

People: The people involved in a software project can have a significant impact on its success. This includes the project team, stakeholders, customers, and end-users. The skills and experience of team members, as well as their ability to communicate and collaborate effectively, can greatly influence the outcome of a project.

Process: The process of software development refers to the steps and activities involved in planning, designing, building, testing, and delivering the software product. Effective processes are critical to ensure that the project stays on track, meets its goals, and produces a high-quality product. This includes the use of methodologies such as Agile or Waterfall, as well as tools and techniques to manage project scope, schedule, and budget.

Product: The software product itself is another key aspect of the software project spectrum. This includes the features and functionality of the product, as well as its quality, usability, and reliability. The product must meet the needs of the stakeholders and end-users, and should be delivered on time and within budget.

Project environment: The project environment encompasses the context in which the software project is developed. This includes the organizational culture, external factors such as regulations and industry standards, and the available resources such as hardware, software, and infrastructure. The project environment can greatly influence the success of the project, and must be carefully managed to ensure that the project stays on track and meets its objectives.

**Test Case Template**

A test case template is a standardized format for documenting the steps to be taken to verify a particular functionality or behaviour of a system or software application. The template serves as a guideline to ensure that each test case is structured in a consistent manner and contains all necessary information required to execute the test case.