|  |  |  |
| --- | --- | --- |
|  | **Software Design Principles** |  |
|  | Software design principles are concerned with providing means to handle the complexity of the design process effectively. The following are some of the design principles that can be adopted. |  |
|  | 1. **Problem partitioning** |  |
|  | It involvesdivision of a problem into manageable pieces. The goal of problem partitioning is as follows.   1. Software becomes easy to understand. 2. Software becomes easy to test. 3. Software is easy to modify. 4. Software is easy to scale. |  |
|  | 1. **Modularity** |  |
|  | Modularity specifies to the division of software into separate modules which are named and addressed differently and are integrated later to obtain the completely functional software. A module should have the following properties.   1. Each module should have a single specified objective. 2. Each module can be separately compiled and saved in the library for reusability. |  |
|  | 1. **Functional Independence** |  |
|  | Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. This is also known as decoupling. |  |
|  | **Strategy of Design** |  |
|  | In system design, there are two possible approaches:   1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components. This approach is suitable for new systems. 2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy. This approach is suitable for existing systems. |  |
|  | 1. **The Open-Closed Principle** |  |
|  | It states that classes should be open for extension and closed to modification. This means we should be able to add new functionality without touching the existing code in a class. This is made possible with the help of interfaces and abstract classes. |  |
|  | **Relational Database Modelling** |  |
|  | A database model shows the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. There are mainly three different types of data models: conceptual data models, logical data models, and physical data models. |  |
|  | 1. **Conceptual Data Model:** This Data Model defines **what** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules. 2. **Logical Data Model:** Defines how the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to develop technical map of rules and data structures. 3. **Physical Data Model**:This Data Model describes how the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database. During the physical design process, you translate the expected schemas into actual database structures. So, you must map the logical to the physical as follows: 4. Entities to tables 5. Relationships to foreign key constraints 6. Attributes to columns 7. Primary unique identifiers to primary key constraints 8. Unique identifiers to unique key constraints |  |
|  | **Entity Relationship Diagram** |  |
|  | An Entity Relationship Diagram is a type of flowchart that illustrates how entities such as people, objects or concepts relate to each other within a system. ERDs use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships, and their attributes. The entity relationships can be categorized into; many to many, one to many and one to one.  In an ERD diagram, the rectangles showcase the entities, the ovals show the attributes of the entities, the diamonds show the action binding two entities, and the lines show the relationship shared between two or more entities. |  |
|  |  |  |
|  | **Software Requirements Document** |  |
|  | A system requirement document comprises of user requirements for a system as well as detailed specifications of the system requirements. The following are examples of requirements documents.   1. **User Requirements:** This document represents what the user expects the product to do. 2. **Product Requirements:** Used interchangeably with a market requirements document, this document details the purpose of a product. 3. **Business Process Documents:** This document details a business process. 4. **Business Needs Assessments:** This document describes the gaps between current conditions and desired business conditions. 5. **Technical Design Specifications:** This document describes the programming elements required for the proposed design. 6. **Validation Documents:** Validation documents can include a traceability matrix which tracks features throughout the development process, test plans, and operation requirements. 7. **Systems Requirements:** This document sketches a high-level expectation for a system or product. 8. **Business Requirements:** This document describes the high-level reasons for creating a product or update. 9. **Use Cases:** This document offers functional details and context for features from a user perspective. 10. **User Stories:** This document is used for mainly for Agile development. It communicates the intent of the product by detailing what the user will do with it.   These requirements can be categorized as functional and non-functional specification.   1. **Functional Requirement:** This describes a behaviour, activity, or expected result from a product or system. Common functional requirements include inputs to the system, authorization and authentication, audit tracking and reporting. 2. **Non-functional Requirement:** This describes how something works. If the English word describing the process ends in ‘ity’, it’s non-functional. Examples include: 3. Usability-easy navigation 4. Reliability-check number of critical failures and their intervals 5. Security-account locking, security questions answering. 6. Availability-measures the uptime of a system |  |
|  | **Functional Requirement Document (FRD)** |  |
|  | There are various ways of creating an FRD, below are some of those ways and their shortcomings.   1. **Use cases -** offer context and detail. The disadvantages of this method are: the scope can creep as real user requirements become clearer and that smaller requirements may become lost among use cases. 2. **User stories** offer the advantage of describing user needs in the context of business requirements. However, they may require extra effort (i.e., researching adequate implementation). |  |
|  | **Composition of an FRD** |  |
|  | Here’s a list that you can use as a guide when preparing functional requirements:   1. **Front Section**    1. Metadata Page: This summarizes everything about the document.    2. Instructions to Authors: This explains the information required by your organization in a specs document. These instructions may appear in the introduction or throughout the template.    3. Version Number    4. Change Record/Revision Page: In the template and published requirement document, you should include all amendments, details, dates, and approver initials.    5. Approval Block: This entails signing off on each revision and approving each requirement with a signature.    6. Distribution List: Certain team members may be required to review the document. Alternatively, viewing may be restricted to only a few team members. 2. **Overview**    1. Product description    2. Business requirements overview    3. Scope of work (what will and will not be covered)    4. Description of the current system    5. Document conventions    6. Terminology (including acronyms)    7. References    8. General constraints and assumptions 3. **Functionality**    1. Business process    2. Inputs, outputs, and processing    3. User interfaces    4. Use cases.    5. User stories 4. **Other Software**    1. Inputs, outputs, and processing    2. External interfaces    3. Hardware interfaces    4. Software interfaces    5. Database 5. **Attributes**    1. Security    2. Reliability, availability, maintainability, compatibility    3. Regulatory requirements 6. **Appendix** |  |
|  | **Technical Requirement Document (TRD)** |  |
|  | A TRD includes the main architectural decisions made by a [solution architect](https://www.altexsoft.com/blog/engineering/solution-architect-role/). It describes in what way each product component will contribute to and meet the requirements, including solutions, strategies, and methods to achieve that. Going into too much detail and listing all the solutions to be used is not advisable rather focus on the most relevant and challenging ones. |  |
|  | **Composition of a TRD** |  |
|  | An effective technical requirement document comprises the following information sections:  **Overview and background-**Briefly describes the main goals of the project, what problems you are trying to solve, and the results you want to achieve.  **Architecture & Design Principles-** Underline the guiding architecture and design principles with which you will engineer the product. For instance, if you plan to structure your solution using [microservices architecture](https://www.altexsoft.com/blog/engineering/using-microservices-for-legacy-system-modernization/) or MVC.  **User Story description.** Connect user stories with associated business processes and related scenarios. You should try to avoid technical details in this section.  **Solution details.** Describe the contemplated solution by listing planned services, modules, components, and their importance.  **Diagrammatic representation of the solution.** Provide the diagrams and/or other graphic materials to help understand and communicate the structure and design principles.  **Milestones-** Includes the overall timeline, deadlines for completion, and/or functional milestones. |  |
|  | **Software design and UML diagramming** |  |
|  | UML stands for Unified modelling language. It is a standardized general-purpose visual modelling language in the field of Software Engineering. It is used for specifying, visualizing, constructing, and documenting the primary artifacts of the software system. The UML diagrams are categorized into structural diagrams and behavioural diagrams. |  |
|  | **Use case Diagram** |  |
|  | A use case diagram is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and tells how the user handles a system. |  |
|  | **Activity Diagram** |  |
|  | Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system.  Activity diagrams model the behaviour of users and systems as they follow a process. Below is an example of an activity diagram.  [Diagram  Description automatically generated](https://drawio-app.com/create-uml-activity-diagrams-in-draw-io/) |  |
|  | **Backend Frameworks** |  |
|  | The languages identified for building backend solutions are three namely.   1. Java 2. C# 3. PHP   These languages run on popular frameworks i.e., Spring boot, .net and Laravel respectively. Frameworks makes it easy for standardised cording and continuity. These languages are all object oriented and therefore they have the following advantages when it comes to building enterprise grade applications.   1. [Modularity for easier troubleshooting](https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming#toc1) 2. [Reuse of code through inheritance](https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming#toc2) 3. [Flexibility through polymorphism](https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming#toc3) 4. [Effective problem solving](https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming#toc4) |  |