**User Interface Design**

User Interface (UI) Design focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions.

**Choosing Interface Elements**

Users have become familiar with interface elements acting in a certain way, so try to be consistent and predictable in your choices and your layout. Doing so will help with task completion, efficiency, and satisfaction.

**Interface elements include:**

**Input Controls:** button, text fields, checkboxes, radio buttons, dropdown lists, list boxes, toggles, data field

**Navigation components:** breadcrumb, slider, search field, pagination, tags, icons

**Informational components:** tooltips, icons, progress bar, notifications, message boxes, modal windows

**Containers:** accordion

**Best Practices for Designing an Interface:**

Everything stems from knowing your users, including, understanding their goals, skills, preferences, and tendencies. Once you know your users, consider the following:

**Keep the interface simple.**

**Create consistency and use common UI elements.**

**Be purposeful in page layout.**

**Strategically use colour and texture.**

**Use typography to create hierarchy and clarity.**

**Make sure the system communicates what’s happening.**

**Think about the defaults.**

**Entity Relationship Model**

An entity relationship model, also called an entity relationship diagram (ERD), is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data in databases or information systems. An entity is a piece of data an object or concept about which data is stored.

**Relationship between entities**

A relationship is how the data is shared between entities. There are three types of relationships between entities:

**One-To-One**

**One-To-Many**

**Many-To\_Many**

**Data Design**

Data design is the first of four design activities that are conducted in software engineering.

Transforms the information domain model created during analysis into the data structures that will be required to implement the software.

The data objects and relationships defined in the Entity Relationship diagram (ERD) and the detailed data content depicted in the data dictionary provide the basis for the data design activity.

The primary activity during data design is to select logical representations of data objects (data structures) identified during the requirements definition and specification phase.

The selection process may include algorithmic analysis of alternative structures in order to determine the most efficient design or may simply involve the use of a set of modules( a ‘package’) that provide the desired operations upon some representation of an object.

**Data Dictionary**

Designing tables defines the efficiency of your software. Most software developers agree that the database design is the first step to engineering software. The way you define your tables determines how you design your software. After you’ve designed your tables you then create what is known as a ‘data dictionary’. A data dictionary is a document that outlines your table designs, the data for each column and a brief explanation of each field.

A data dictionary is a collection of data about data. It maintains information about the definition, structure, and use of each data element that an organization uses.

There are many attributes that may be stored about a data element. Examples are:

* Name
* Aliases or synonyms
* Default label
* Description
* Source(s)
* Date of origin
* Users
* Programs in which used
* Change authorizations
* Access authorization
* Data type
* Length
* Units(cm., degrees C, etc.)
* Range of values
* Frequency of use
* Input/output/local
* Conditional values
* Parent structure
* Subsidiary structures
* Repetitive structures
* Physical location: record, file, database

A data dictionary is invaluable for documentation purposes, for keeping control of information on corporate data, for ensuring consistency between organizational systems, and for use in developing databases.

**Identify the characteristics of good program design**

**Functionality**

While we all know that getting our software to work correctly is important, the functional qualities of our software are often not emphasized as much as they should be.

**Reliability**

Even if our software does what it is supposed to do, if it does not do so reliably, it will not do a good job of keeping the users happy.

**Usability**

Once we have code that does it’s job correctly and does it well, we still need to think about how pleasant an experience we create for our users.

**Efficiency**

Ruby has had a reputation for being slow, resource intensive programming language. As a result, we need to rely on some special tricks to make sure that our code is fast enough to meet the needs of our users.

**Maintainability**

No matter how good our software is, it will ultimately be judged by how well it can change and grow over time. This is the area we tend to spend most of our time studying, because difficult to maintain projects make us miserable as programmers.

**Portability**

One thing we don’t think about often in ruby, perhaps not often enough, is how easy it is for folks to get our software up and running in environments other than our own. While writing code in a high level language does get us away from some of the problems that system programmers need to consider, there are still platform and environment issues that deserve our attention.

**Functional Requirements**

**A functional requirement** specifies something a system should do. Typically a functional requirement will specify a behaviour or function, for example: “Display the name, total size, available space and format of a flash drive connected to a USB port. Other examples are “add customer” and “print invoice”.

* Business Rules
* Transaction corrections, adjustments and cancellations
* Administrative functions
* Authentication
* Authorization levels
* Audit Tracking
* External Interfaces
* Certification Requirements
* Reporting Requirements
* Historical Data
* Legal or Regulatory Requirements

**A non-functional requirement** essentially specifies how the system should behave and that it is a constraint upon the system behaviour. One could also think of non-functional requirements as quality attributes for of a system.

Non- functional requirements cover all the remaining requirements which are not covered by the functional requirements. They specify criteria that judge the operation of a system, rather than the specific behaviours, for example: “Modified data in a database should be updated for all users accessing it within two seconds”.

* Performance – for example Response Time, Throughput, Utilization, Static Volumetric
* Scalability
* Capacity
* Availability
* Reliability
* Recoverability
* Maintainability
* Serviceability
* Security
* Regulatory
* Manageability
* Environmental
* Data Integrity
* Usability
* Interoperability