F.Y. MCA
(TWO YEARS PATTERN)
SEMESTER - II (CBCS)

USER INTERFACE LAB

SUBJECT CODE : MCAL25

Prof. Suhas Pednekar
Vice-Chancellor,
University of Mumbai,

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai,

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai,

Programme Co-ordinator : **Shri Mandar Bhanushe**
Asst. Prof. cum Asst. Director in Mathematics,
IDOL, University of Mumbai, Mumbai

Course Co-ordinator **:** **Mr. Shyam Mohan T**
Dept.of MCA,
IDOL, University of Mumbai, Mumbai

Course Writers **:** **Mr. Milind Thorat**
Lecturer,
KJSIEIT, Sion, Mumbai

: **Mrs. Laxmi B Pandya**
Assistant Professor,
Tolani College, Andheri (East) Mumbai

April 2021, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai,

CONTENTS

F.Y. MCA
SEMESTER - II(CBCS)

USER INTERFACE LAB

Module	Detailed Contents	Hrs
1.	**The UI life cycle:** Introduction to UI life cycle and UI tools. Self Learning Topics: phases and importance of UI life cycle	4
2.	**Requirement gathering:** Include the business purpose and user needs. Self Learning Topics: Understand the user, types of users, requirement gathering techniques, contextual enquiry.	2
3.	**Analysis:** User analysis, Task analysis, Domain analysis Self Learning Topics: Identifying the types of tasks, design objects model, contextual analysis.	4
4.	**Design:** Scenario, Storyboard designs. Self Learning Topics: Principles of good design, Mental model	4
5.	**Build and test the low fidelity prototype:** Build a prototype. Paper prototype, Wireframe Prepare a briefing for test users.(test the prototype) Self Learning Topics: Types of prototypes	4
6.	**Implementation:** Working implementation of the chosen project. Light weight page loading, optimal design. Self Learning Topics: Implementation tool, user friendly design.	6
7.	**Testing:** Evaluate the interface with a small user test and write a final reflection Self Learning Topics : Testing Techniques	2

Module I

1

THE UI LIFE CYCLE

The UI life cycle: Introduction to UI life cycle and UI tools. Self Learning Topics: phases and importance of UI life cycle

EXPERIMENT NO. 1

AIM: Introduction to UI life cycle and UI tools.

OBJECTIVE: Understand the importance of User Interface Design (UI) Process.

THEORY:

Introduction:

- A user interface, also called a "UI" or simply an "interface," is the means in which a person controls a software application or hardware device.
- A user interface is the point of human-computer interaction and communication on a device, webpage, or app.
- This can include display screens, keyboards, a mouse, and the appearance of a desktop. User interfaces enable users to effectively control the computer or device they are interacting with.
- User interface is important to meet user expectations and support effective functionality.
- A successful user interface should be intuitive, efficient, and user-friendly.
- Nearly all software programs have a graphical user interface, or GUI. This means the program includes graphical controls, which the user can select using a mouse or keyboard.
- A typical GUI of a software program includes a menu bar, toolbar, windows, buttons, and other controls.
- User Interface Design is the craft and process of designing what a user interacts with when communicating with software.

Types of UI are:

1. **Form-based user interface**: Used to enter data into a program or application by offering a limited selection of choices. For example, a settings menu on a device is form-based.

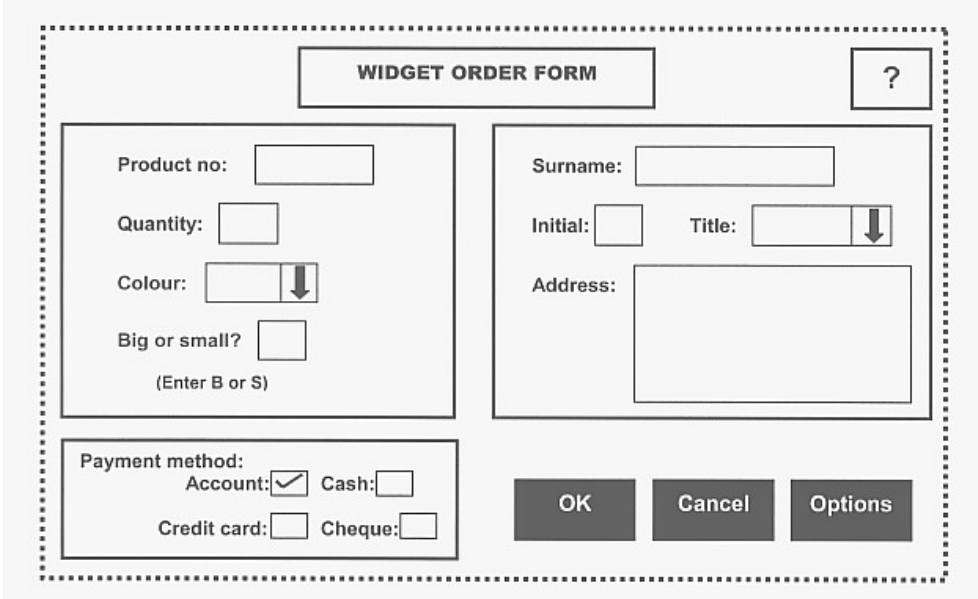


Fig. 1 Form based UI

2. **Graphical user interface**: A tactile UI input with a visual UI output (keyboard and monitor).

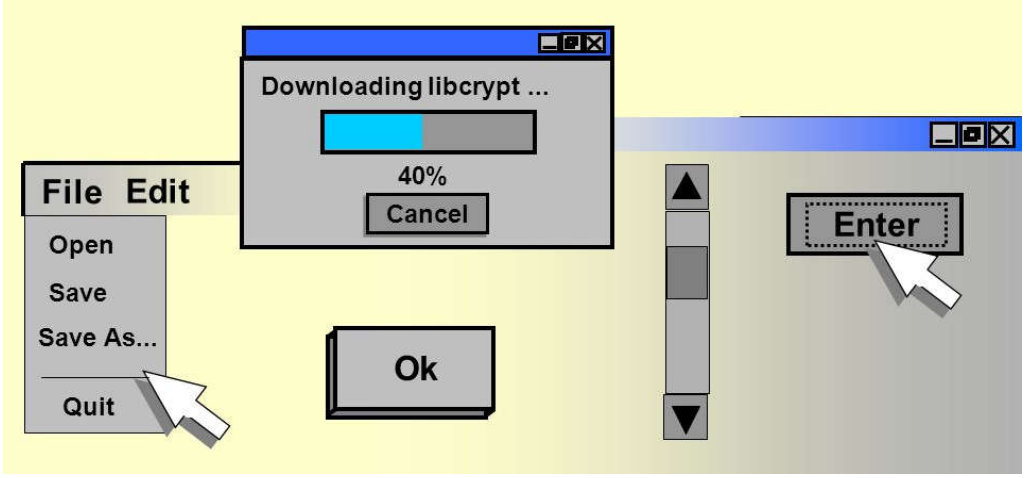


Fig. 2 GUI

3. **Menu-driven user interface**: A UI that uses a list of choices to navigate within a program or website. For example, ATMs use menu-driven UIs and are easy for anyone to use.

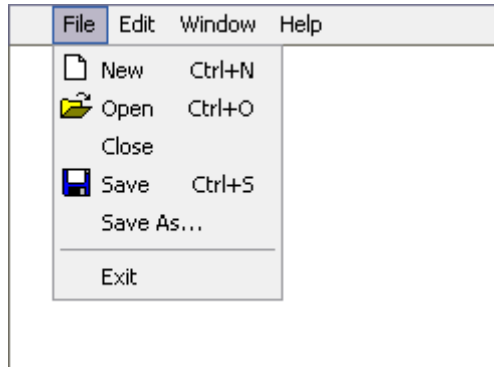


Fig. 3 Menu-driven UI

4. **Touch user interface**: User interface through haptics or touch. Most smartphones, tablets and any device that operates using a touch screen use haptic input.

Fig. 4 Touch UI

5. **Voice user interface**: Interactions between humans and machines using auditory commands. Examples include virtual assistant devices, talk-to-text, GPS and much more.

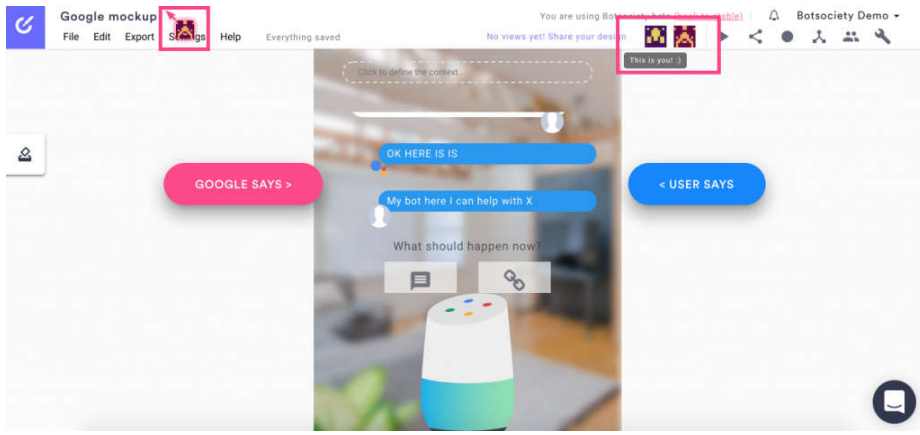


Fig. 5 Voice UI

UI LIFE CYCLE:

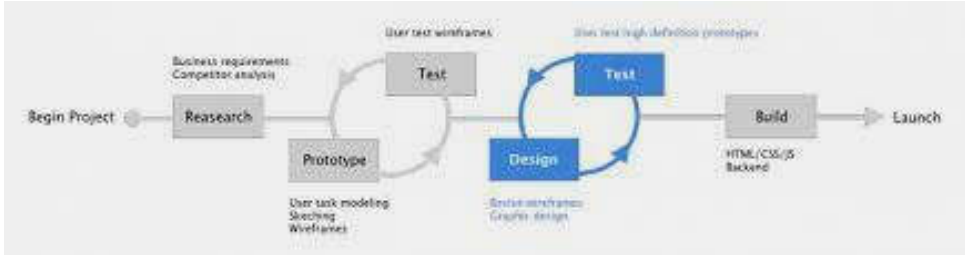


Fig. 6 Life Cycle of UI

UI life cycle can be categorized into four following parts:

A. **Research and Analysis**:

In the traditional Research & Analysis phase, two categories of information are gathered and analyzed by the user experience team.

1. Information about the users of the application
2. Information about the application itself
3. This establishes the context for User Interface design and this context informs the entire design process.

B. **Design and Branding**:

1. During the design and branding phase, User Interface design is created that addresses the specific needs identified in the research & analysis phase and creates, revise or leverage the applications brand.
2. During the design phase UI developers can work closely with the UX team to define the User interface (Wireframes, Visual design). A User Experience team may think out of the box while creating wireframes and visual design, but may not be aware of challenges, possibilities and limitations. Involving UI developers in this phase may ease the process, as UI developers understand the technologies and possibilities. This will reduce the last minute efforts from the UI developer's side and additionally both the UI and UX team, as well as client will have a clear expectations set.

C. **Prototype Development**:

1. Using the approved design document as blueprint, prototypes of the User Interface designs are created. Based on clients needs the prototypes are created using HTML or flash. Prototypes can be low fidelity or high fidelity based on user needs.
2. The scope of the prototype created during this phase is tailored to the specific application and the user testing requirements. Some applications require a comprehensive click through or working model of the entire interface, while others only require a prototype of core functionality.

D. Production:

1. A proper UI and UX team collaboration and Integration of UI design from the starting of SDLC can reduce a lot of effort and confusion. Also it can help in successful and timely delivery of the products in any company without any slippages and can increase customer satisfaction. Additionally it can help build ongoing relations with our clients.

UI TOOLS:

There are multiple UI tools present such as:

a. **MockFlow:**

1. MockFlow WireframePro is the leading tool for designing User Interface blueprints for Websites and Apps. It includes ready-made UI components for iOS, Android, Web, Bootstrap, WatchOS.. etc; and also comes with a built-in template store to quickly turn your User Interface Ideas into designs.

Fig. 7 MockFlow

b. **Balsamiq:**

1. Balsamiq Wireframes is a user interface design tool for creating wireframes (sometimes called mockups or low-fidelity prototypes).

2. You can use it to generate digital sketches of your idea or concept for an application or website, to facilitate discussion and understanding before any code is written. The completed wireframes can be used for user testing, clarifying your vision, getting feedback from stakeholders, or getting approval to start development.

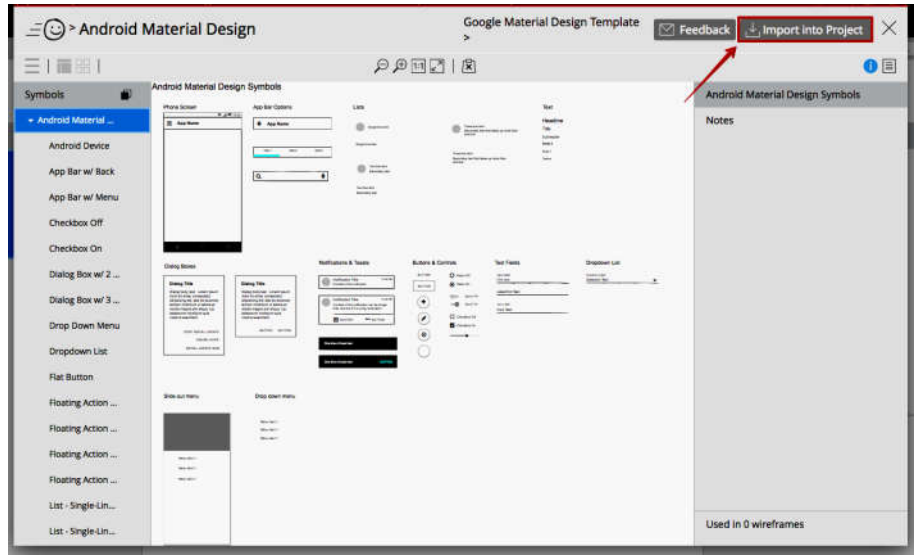


Fig. 8 Balsamiq

c. **Axure:**

1. Axure is a dedicated rapid prototyping tool that allows anyone with even a basic familiarity with the software to create simple wireframes. It uses a 'what you see is what you get' (WYSIWYG) interface that allows you to drag shapes onto a canvas and build up your design.

2. Axure is particularly well suited to low-fidelity prototyping as it has a very short learning curve. Within a few hours, anyone can be creating designs that are quick and cheap to change. A single click will render your prototype in a web browser on your desktop, tablet or phone and allow you to gain feedback from users or other stakeholders.

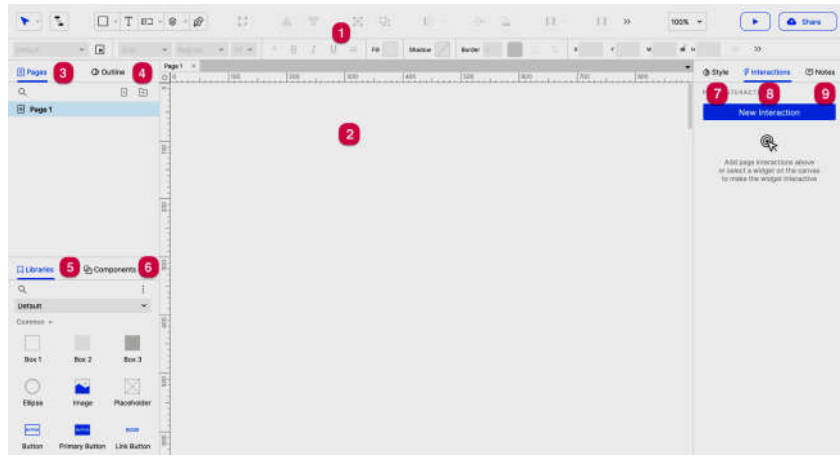


Fig. 9 Axure

d. **Adobe XD:**

1. Adobe XD (also known as Adobe Experience Design) is a vector-based user experience design tool for web apps and mobile apps, developed and published by Adobe Inc.

2. Adobe XD supports website wireframing and creating click-through prototypes.

3. Adobe XD creates user interfaces for mobile and web apps. Many features in XD were previously either hard to use or nonexistent in other Adobe applications like Illustrator or Photoshop.

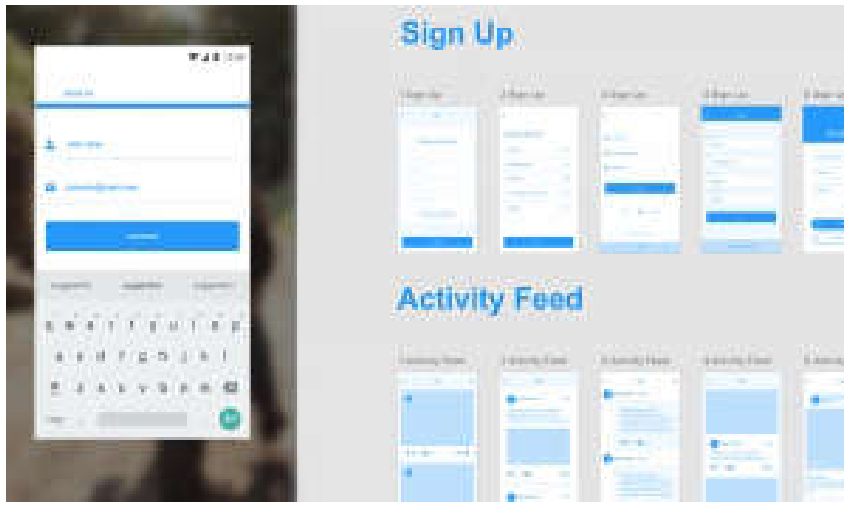


Fig. 10 Adobe XD

e. **Sketch:**

1. Sketch is a vector graphics editor for macOS developed by the Dutch company Sketch B.V.

2. It is primarily used for user interface and user experience design of websites and mobile apps and does not include print design features.

3. Sketch has more recently added features for prototyping and collaboration. Being only available for macOS, third party software and handoff tools may be used to view Sketch designs on other platforms.

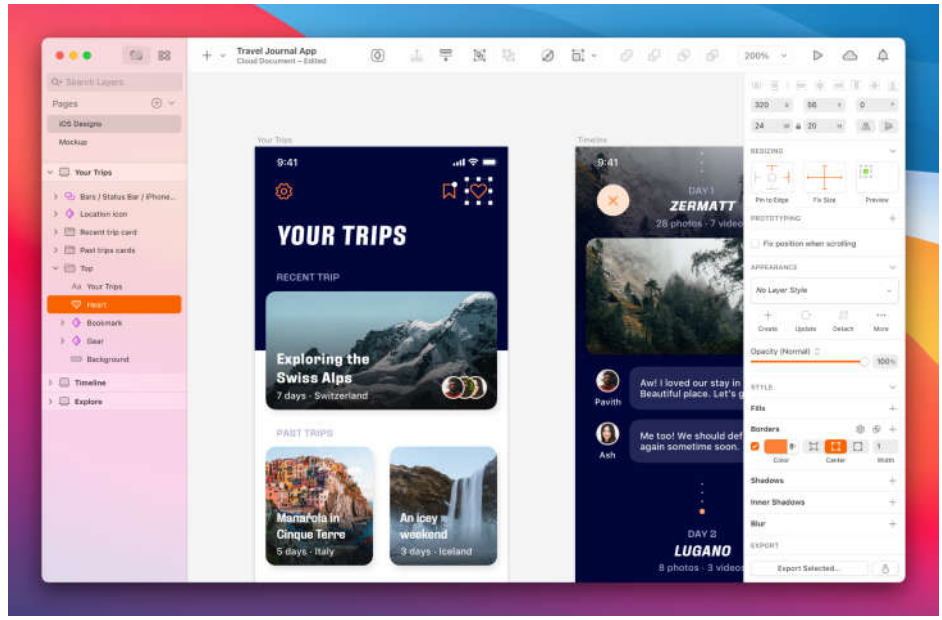


Fig. 11 Sketch

CONCLUSION:

We successfully learnt about UI life cycle and UI Tools.

❖❖❖❖

Module II

2

REQUIREMENT GATHERING

Requirement gathering: Include the business purpose and user needs.

EXPERIMENT NO. 2

Aim: **Project Proposal Digital Wallet**

Objective:

- The main objective of the project proposal is to develop a Smartphone application that enables the users to perform any sort of transaction securely through their bank accounts connected to the application over the internet.
- This system is hassle free and more convenient for current modes of transactions which are performed physically.
- Improve wallet apps with more security features like roll-backing a transaction and reporting a transaction as fraud.

Theory:

Overview:

- The project proposal is aimed to provide the user with an option to perform all sorts of transactions over the internet.
- Users can perform all transactions and the transactions will be done over users bank accounts which he/she can connect with the application.
- This will help the user to perform these transactions easily and is hassle free.
- This application allows users to make in-store payments without having to carry cash or physical credit cards.

Problem Statement:

- In today's date most of the transactions are done over the internet.
- There are very few Digital Wallets available at today's date and very less awareness.
- Normally to do any transaction one has to go to the nearest bank or ATM to perform the transaction. Or while doing any type of shopping one needs to carry the hard cash to buy or sell the goods. This is time consuming and also extremely hectic.

- The project proposal aims to develop a new payment method for all sorts of transactions over the Smartphone.
- You can perform bank transactions and also can do shopping and perform transactions over the internet. This not only saves time but also is a seamless and easy process.
- In present digital wallets there are many limitations like Number of transactions we can make in a day are very less and for each transaction the digital wallet deducts some extra processing fee which is the same for different amounts.

Purpose:

- The purpose of this project is to increase the usability of digital wallets and make transactions securely. Most of the current wallets have difficult and complex systems designs and User Interfaces. The aim of this project is to create a prototype of a mobile wallet app and make the UI/UX simpler
- A background of electronic transactions and Digital wallet-
- A number of electronic commerce applications allow end-users to purchase goods and services using digital wallets. Once a user decides to make an online purchase, a digital wallet should guide the user through the transaction by helping him or her choose a payment instrument that is acceptable to both the user and the vendor, and then hide the complexity of how the payment is executed. A number of wallet designs have recently been proposed, but we will argue they are typically targeted for particular financial instruments and operating environments. In this paper, we describe a wallet architecture that generalizes the functionality of existing wallets, and provides simple and crisp interfaces for each of its components.

What is a Digital Wallet?

- A digital wallet is a software component that provides a client with instrument management and protocol management services.
- It is a software-based system that securely stores users' payment information and passwords for numerous payment methods and websites. By using a digital wallet, users can complete purchases easily and quickly with near-field communications technology.
- They can also create stronger passwords without worrying about whether they will be able to remember them later. Digital wallets can be used in conjunction with mobile payment systems, which allow customers to pay for purchases with their smartphones.
- A digital wallet can also be used to store loyalty card information and digital coupons.
- They are capable of executing an operation using an instrument according to a protocol. A digital wallet presents its client with a standard interface of functions; in the case that the client is a human user, this standard interface of functions may be accessed through a graphical user interface (GUI).

- A digital wallet is linked into an end-user, bank, or vendor application and provides the application with instrument management and protocol management services. The digital wallets that are linked into vendor and bank applications provide these management services in the same way that end-user digital wallets do.
- A vendor's digital wallet, however, may be part of a much larger software application that is integrated with order and fulfillment systems. Furthermore, a wallet is not limited to being a plug-in or applet or some other extension of a web browser.
- A digital wallet with a graphical user interface may also run as an application on its own.
- A digital wallet may also run on computers that are not connected to the Internet such as smart cards or personal digital assistants. The user interface to the digital wallet may vary in such

Proposed System:

- There are many applications in a smartphone which have their own e-wallets. So when you make any purchase of goods or services through these applications you get various options for making payment. Generally people add money to their e-wallet and pay them. In other words, it is handling separate wallets for different applications which is an overhead. So if we have money in our e- wallet but we cannot use it for other applications Therefore, money in the e-wallet of earlier applications will be of no use until you make any payment or transaction through it.

1. In response to this problem, we propose a technique which makes the separate e- wallets of different applications centralized. One can use the money on any platform without any restrictions. There are some apps that might charge you for doing a transaction.
2. Generally, provide all the offers that are cash backs that can only be transferred to the bank after paying a fee. In all current digital wallets there are many limitations like Number of transactions we can make in a day are very less and for each transaction the digital wallet deducts some extra processing fee which is the same for different amounts Whether we transfer a hundred or thousands of money.
3. In most wallet apps there are limitations like a fixed number of transactions but in this system that won't be the case. In our app, there will be no limitations on the number of transactions.

Specifications/Modules:

1. Send/Receive Money-

This is the main task of this app. Users can send and receive money from their contacts who're using this application. The system will create a transaction for each money transfer operation.

2. Request Money-

Users can request for money to their contacts and the receiver user will get a message to send the requested amount or not. If yes then the requested amount will be deducted from the user.

3. View Past Transactions-

Users can view all the passed transactions they've made along with the transaction details like transaction id, timestamp, sender, amount etc.

4. Pay Later Function-

If the user is having insufficient money in his wallet and they're making a transaction then that transactions will be added to Pay later transactions history and the equivalent amount will be deducted after a person adds money in their the wallet

5. Rollback Transactions-

If the user sees some transactions that are not made by them or if they see any malicious activities with their account (password change emails, otps etc) they can request for a rollback transaction in the support.

6. User Profile-

User can update their profile details like email, phone number and change their password

7. Add/Remove Bank Account-

Users can add and remove new bank accounts from the app.

System Design:

Created using Framer

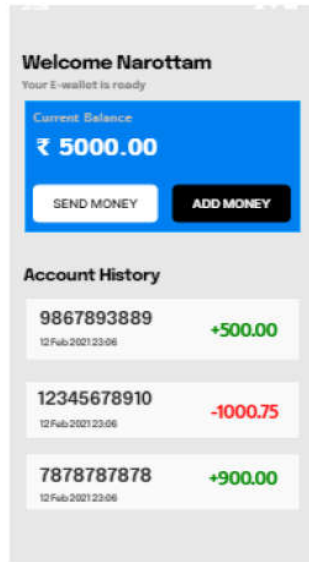
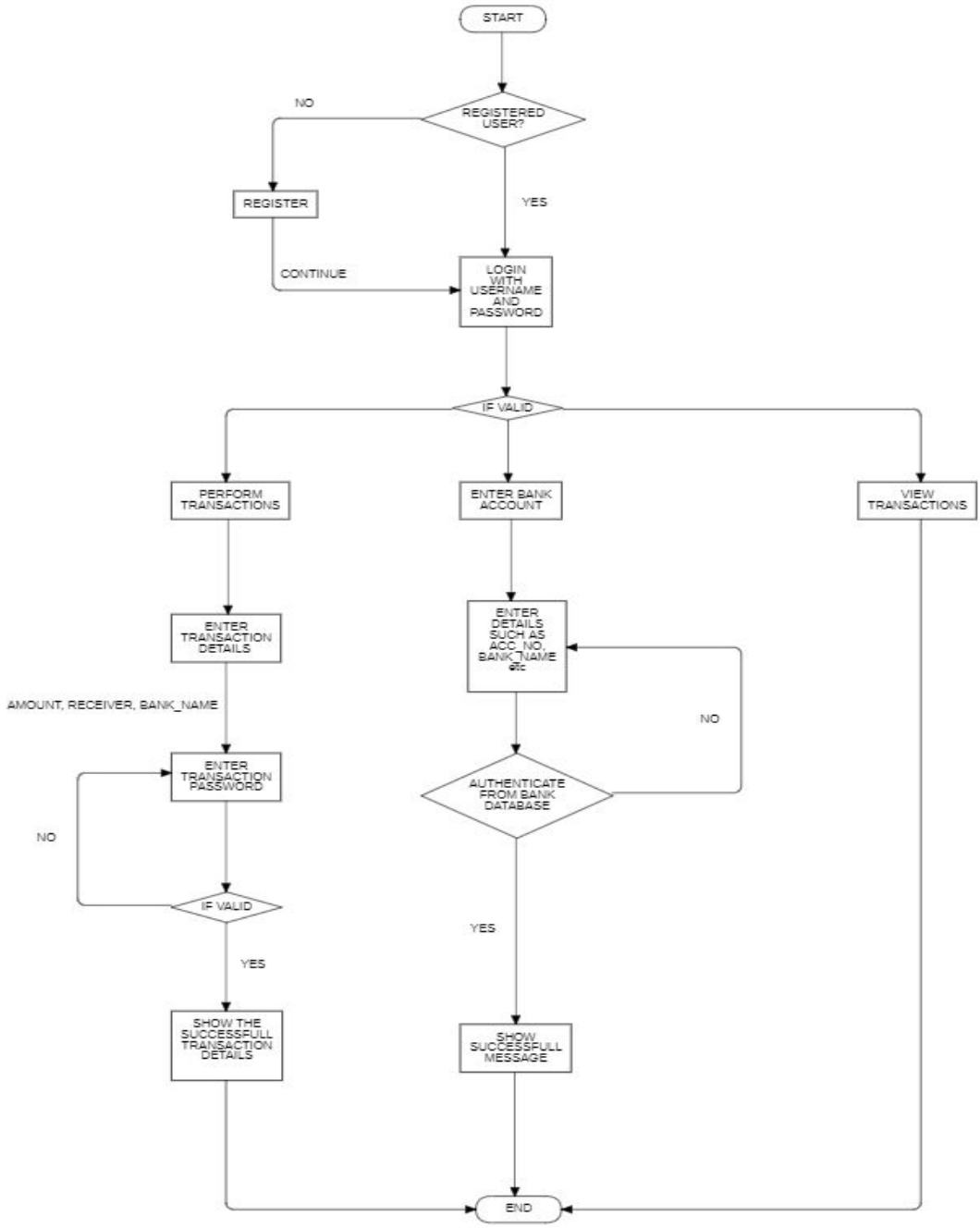


Fig. 1 Home Screen

Flowchart:



Conclusion: We have successfully created and submitted the project proposal.

Self Learning Topics: Understand the user, types of users, requirement gathering techniques, contextual enquiry Include the business purpose and user needs.

❖❖❖❖

Module III

3

ANALYSIS

Learning Objectives:

To make the learners familiar with the importance of requirement and user analysis for a particular project and the techniques to be used.

Learning Outcomes:

The learners will acquire the knowledge to analyze a software project using the given techniques.

CHAPTER 1- INTRODUCTION TO ANALYSIS

Requirement Analysis, also known as Requirement Engineering, is the process of understanding the users' requirements either for a completely new system or the existing system under modification. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous.

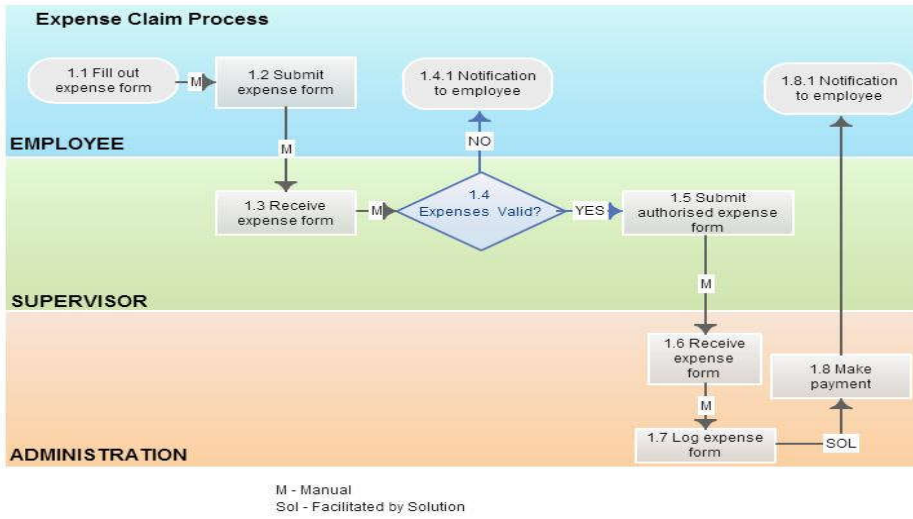
There are different techniques for requirement analysis viz Business Process Modeling, Unified Modeling language (UML), Data flow diagram (DFD), Flow chart, Entity relationship diagram (ERD), Gantt charts, Petri nets, Gap analysis etc.

1) **Business process modeling** is the graphical representation of a company's business processes or workflows, as a means of identifying potential improvements. This is usually done through different graphing methods, such as the flowchart, data-flow diagram, etc. It is mainly used to **map a workflow** so you can **understand, analyses and make positive changes** to that workflow or process.

Flow chart Notations

Symbol	Symbol Name	Description
	Flow Lines	Used to connect symbols
	Terminal	Used to start, pause or halt in the program logic
	Input/output	Represents the information entering or leaving the system
	Processing	Represents arithmetic and logical instructions
	Decision	Represents a decision to be made
	Connector	Used to Join different flow lines
	Sub function	used to call function

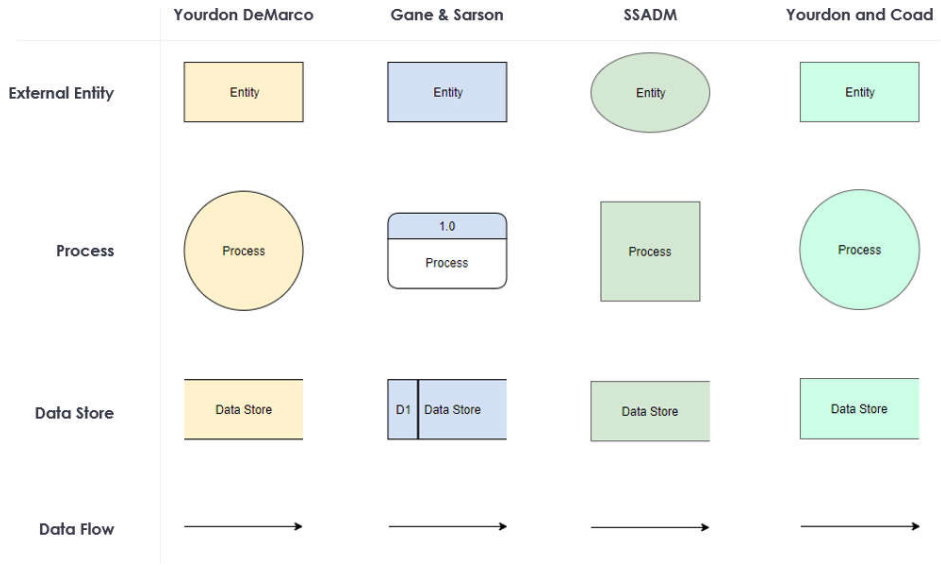
Example of Business process model using Flow chart to depict the organisation Claim Process.



2) Data flow Diagram: -

Data flow diagram is to visualize the working processes within the organization viz manual or automated. It specifies the relationship between processes, data stores and external entities in business information system.

Notation



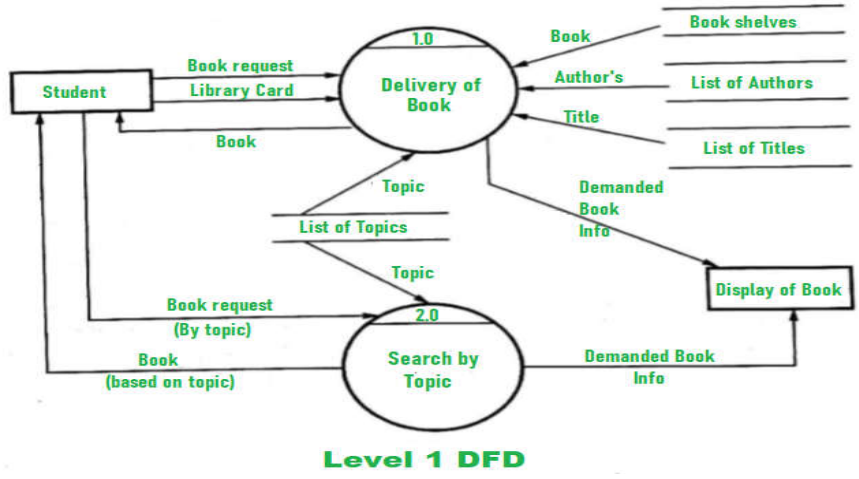
Example of Data Flow Diagram for Library Management System

- Student entity will request for book , provide his library card and after authentication the library system will provide him with the book.

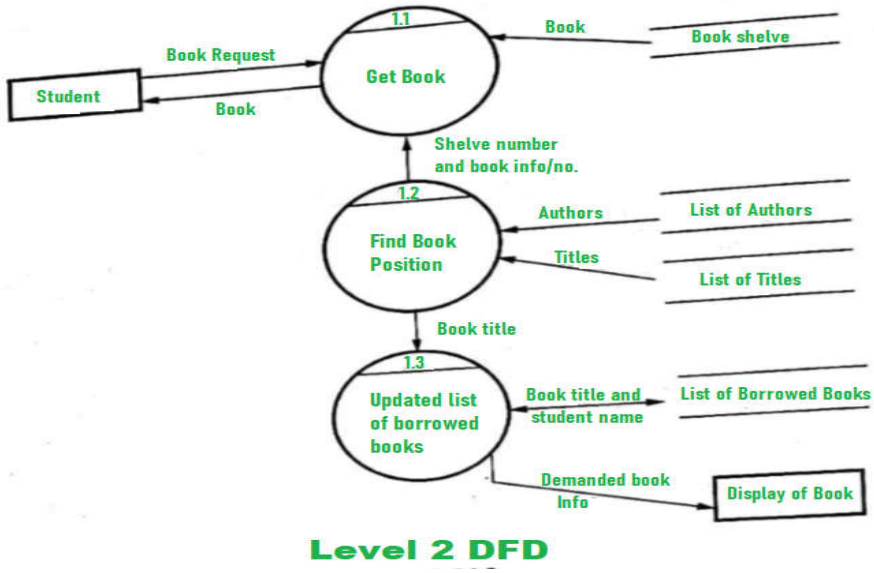
i. Level 0 DFD –

ii. Level 1 DFD –

Book request process including delivery of book and book search process is being demonstrated here. **Data store** is used to represent this type of information.

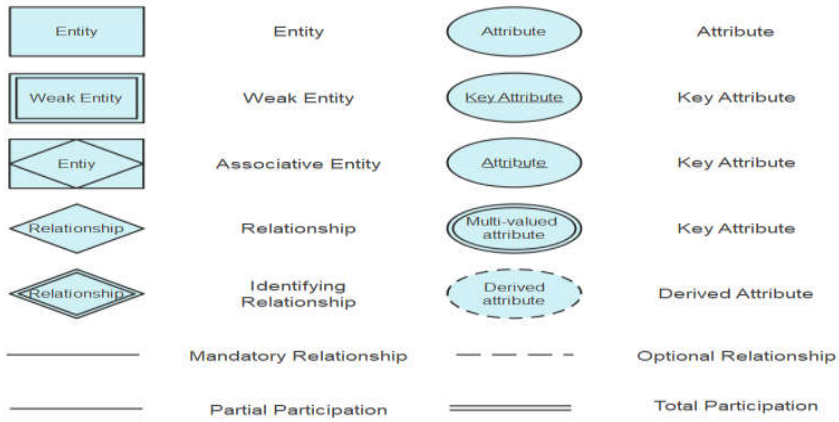


iii. Level 2 DFD – Detailing of Level 1 DFD

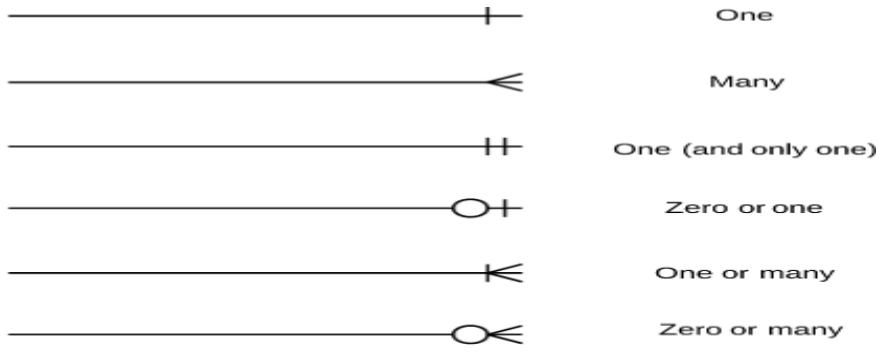


Level 2 DFD

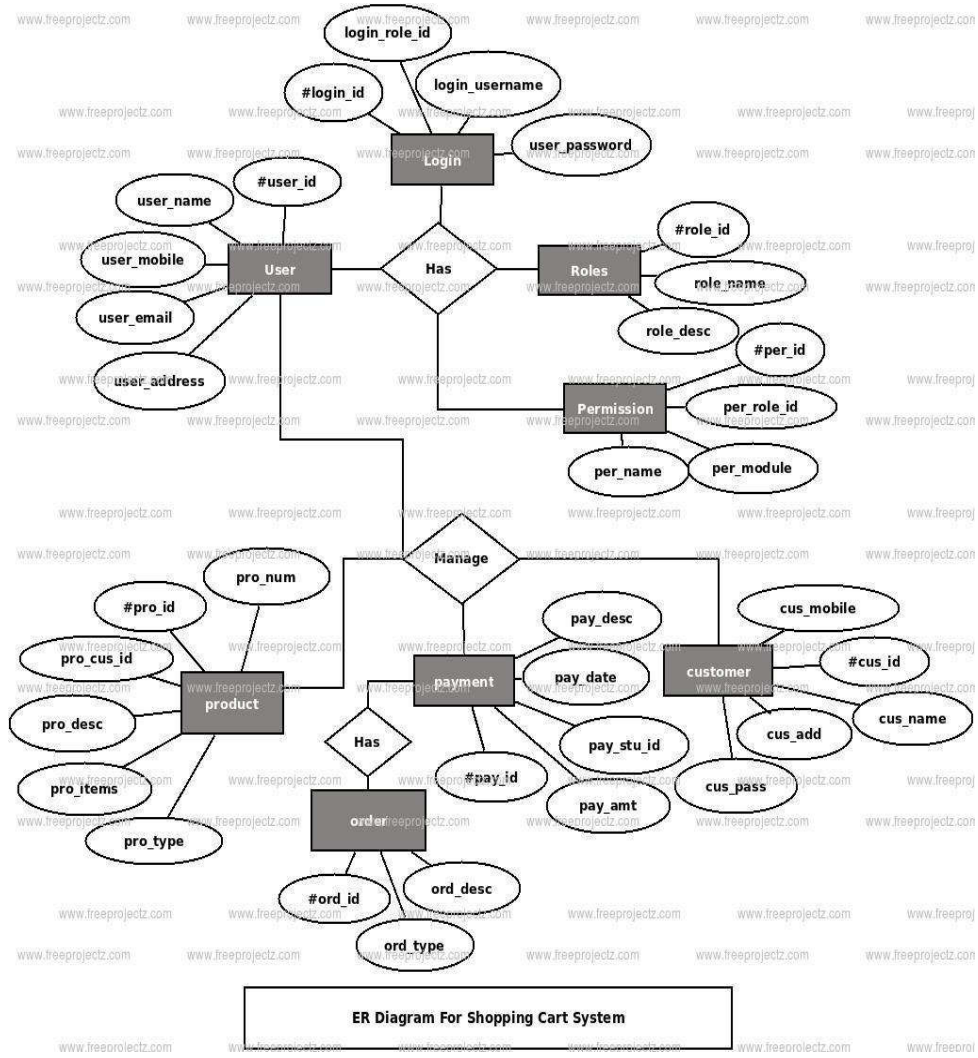
3) Entity Relationship Diagram : An entity relationship diagram (ERD) isa graphical diagram that **depicts the relationships among people, objects, places, concepts or events within an information technology (IT) system**.



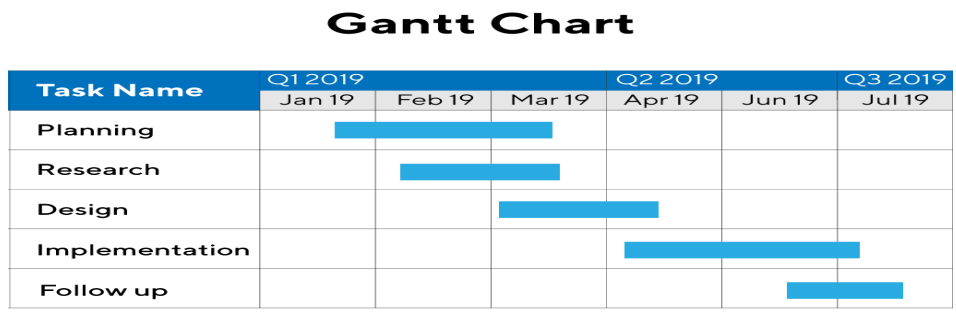
Cardinality in Entity Relationship Diagram



Example of Entity Relationship Diagram (ERD) for Shopping Cart System

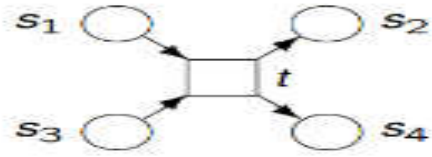


4) Gantt Chart: A Gantt chart is a bar chart that displays a detailed schedule of tasks defining linear schedule and task dependencies for a project.



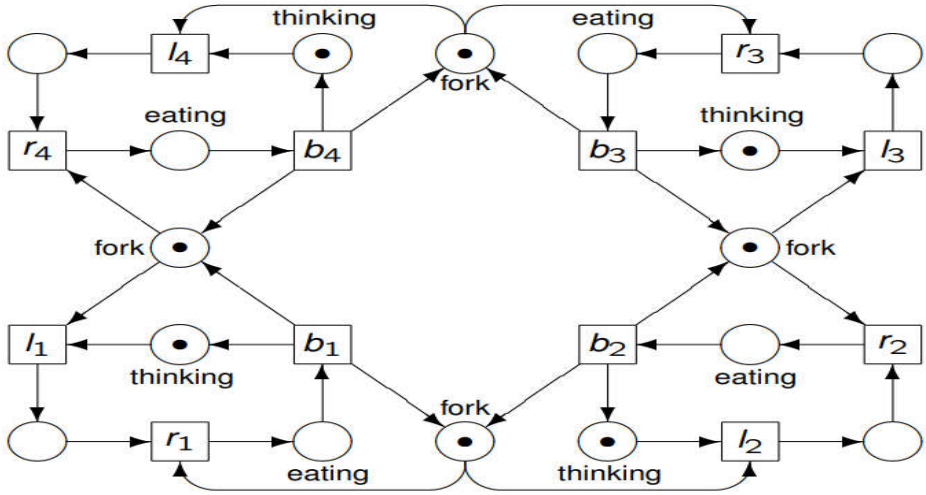
5) Petri Nets : . It is a technique for the description and analysis of concurrent /distributed systems. It is also known as a place/transition net. It is a mathematical model that describes the distributed system. It is a

class of discrete event dynamic system .The basic idea is to describe state changes in a system with transitions.

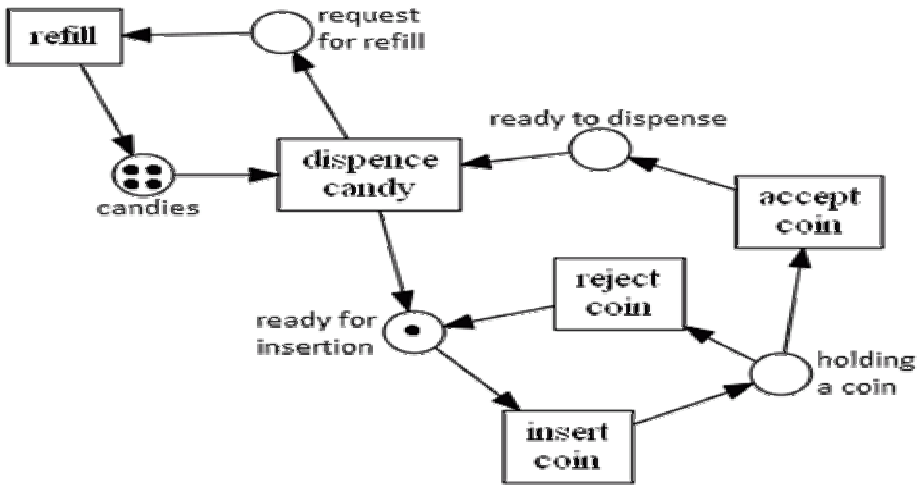


Petri nets contain **Places**  and T**ransitions**  that may be connected by directed arcs. ***Places symbolize states, conditions, or resources that need to be met/be available before an action can be carried out. Transitions symbolize actions***

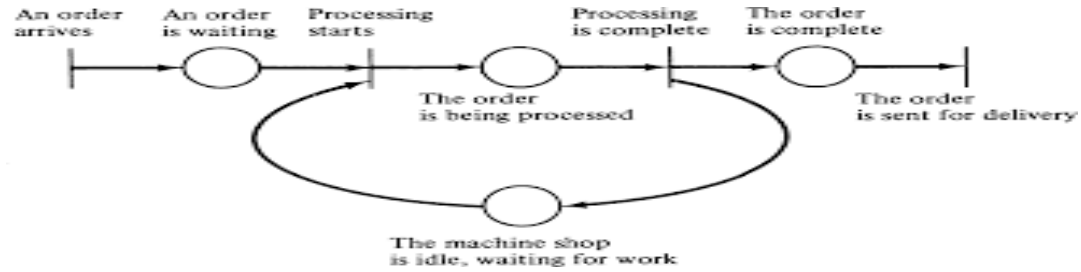
Example: Petri Net for Dinning Philosopher Problem



Petri Net diagram for Candy dispensing machine



Petri Net for Order Processing System



OBJECTIVE OF REQUIREMENT ANALYSIS:

a) To ensure all requirements are elicited.

b) To ensure requirements are complete, unambiguous, and non-conflicting.

c) To ensure its operational, financial, technical, and legal viability.

d) To establish a requirements baseline that serves as the basis for defining the further needed work products viz design, code, test cases, help manual etc.

CHAPTER 2- USER INTERFACE ANALYSIS TYPES AND TECHNIQUES

UI design analysis analyses users, tasks, content, and user work environment. This includes User Analysis, Task Analysis and Domain Analysis

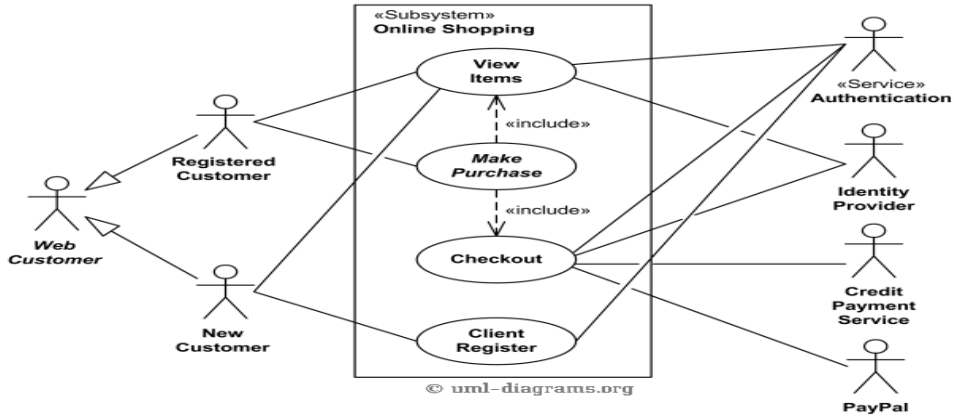
(a) User analysis

It is the process of studying the physical type, role and capability of the person using the proposed system. Whether the user is physically fit or physically challenged. For example, a physically fit user can type in the search box for performing the search functionality, however, a person physically challenged by hand may use the voice recognition software. The roles of the user in terms of the modules and features they will access should be known. User objectives and goals to access the system should be known. **Use cases and User stories can be used to analyse different user access scenario.**

USE CASE SCENARIO-1

Customer on the Web- An actor uses some web site to make purchases online. Top level use cases for the same are **View Items**, **Make Purchase** and **Client Register**. View Items use case is used when customer only wants to find and see some products. This use case could also be used as a part of Make Purchase use case. Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales. Note, that **Checkout** use case is included use case not available by itself - checkout is part of making purchase.

SOLUTION

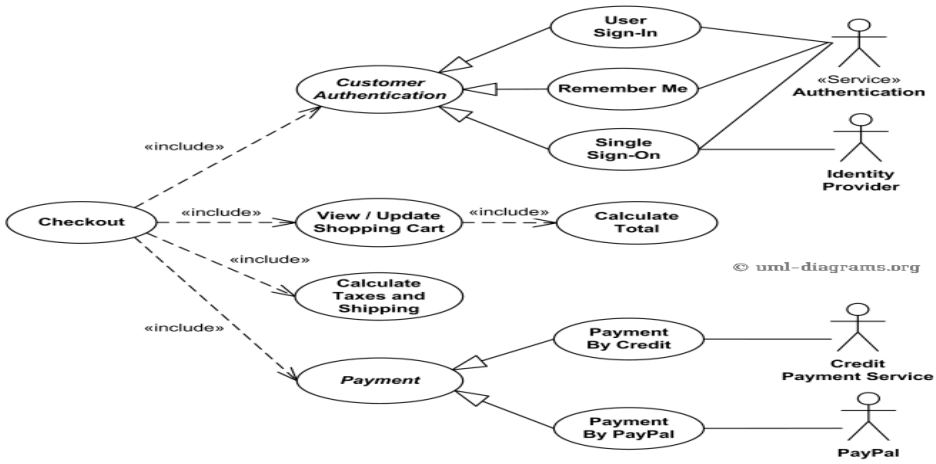


Online shopping UML use case diagram example - top level use cases.

View Items use case is extended by several optional use cases - customer may search for items, browse catalog, view items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item. **Customer Authentication** use case is included in **View Recommended Items** and **Add to Wish List** because both require customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.

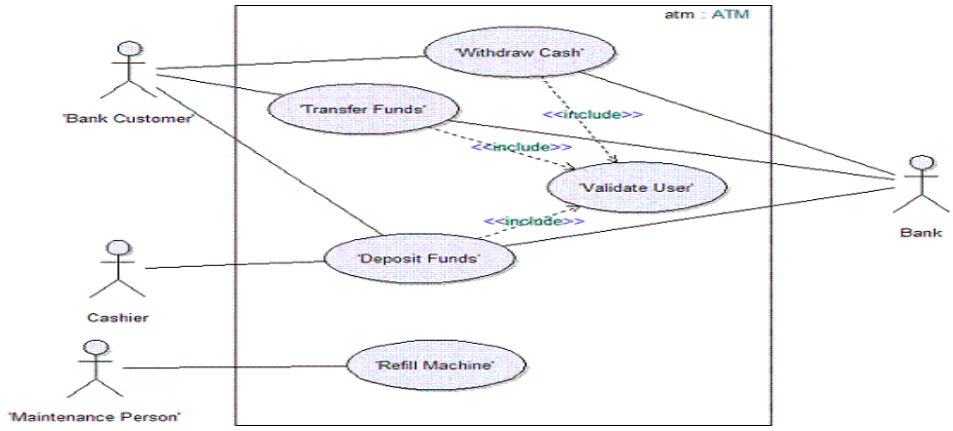
Checkout use case includes several required uses cases. Customer on web should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider.

Checkout use case also includes **Payment** use case which could be done either by using credit card and external credit payment service or with PayPal.



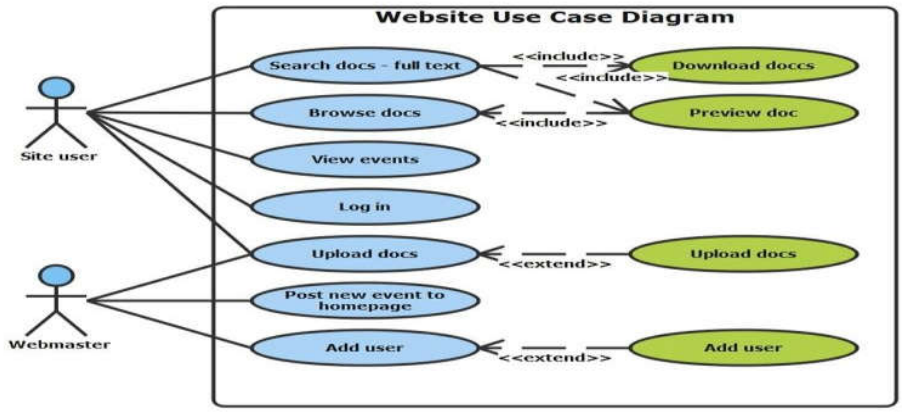
USE CASE SCENARIO 2

Use Case Diagram Demonstrating the Atm Transaction



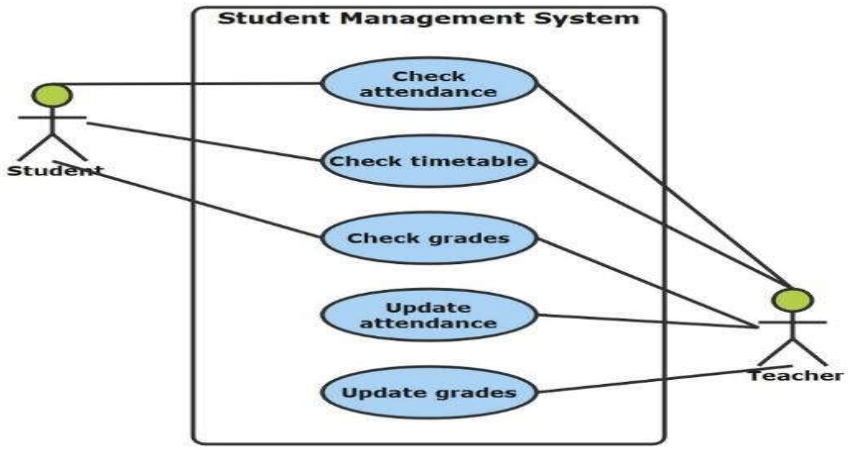
USE CASE SCENARIO EXAMPLE 3: -Website use diagram is another example of the use case diagram examples. This illustrates how a web system can be used, and the following are the various UML use case diagrams that have two key elements the site user and webmaster.

Site User: this is a key entity in the website use case diagram. The site user can extend to the following use cases that are; search docs-full text, browse docs, view events, log in or even upload docs. Under the search docs use case, the site user can download docs and the browse doc extension. The site user also has options to preview the doc. Webmaster: under this element, the webmaster has the use case extension to upload docs, post new events to the homepage and add a user.

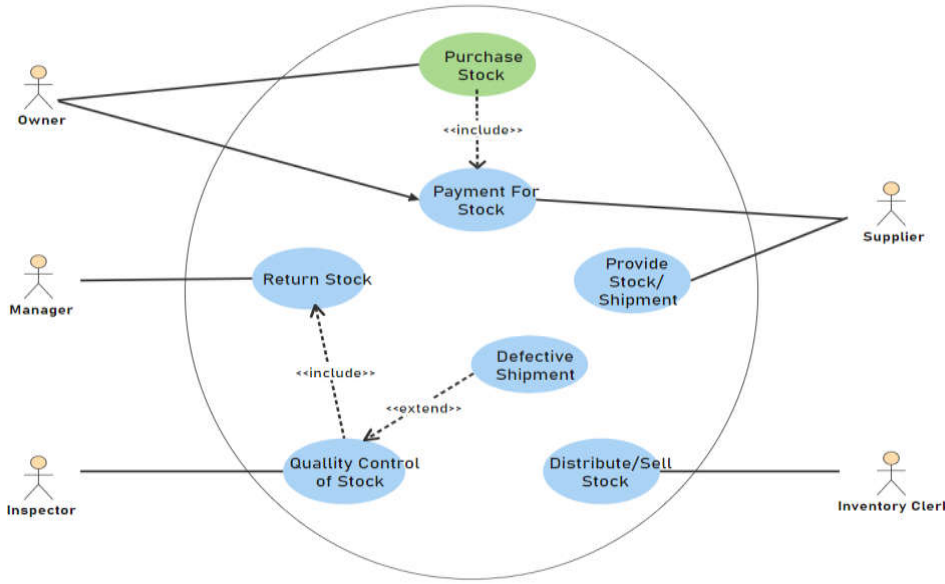


USE CASE SCENARIO EXAMPLE 4:-The following UML use case diagram demonstrates the working of a student management system. It has a teacher and student as the two key actors. The five use cases are; check attendance, check time table, check test score, update attendance and update the score.

A teacher can check attendance, check timetable, check test scores, update attendance and update the score. The student, on the other hand, can check attendance, timetable, and test score. The interactions of the student and the teacher are what sum up the student management use case diagram example.



USE CASE SCENARIO EXAMPLE 5- INVENTORY MANAGEMENT SYSTEM



This use case diagram example shows the various interaction by the inventory management UML case diagram. The main elements in these inventory management systems are the owner, manager, inspector, supplier, and inventory clerk.

The other use cases includes purchase stock, payment for stock, return stock, quality control of stock, defective shipment, distribute/sell stock and provide stock shipment. The use case has several interactions and builds up the inventory management system.

The owner can purchase stock and make payments for the stock. The manager has the return stock use case. The inspector is concerned with the quality control of stock and defective shipment. The inventory clerk distributes/ sells the stock, and the supplier has the payment for stock and provides stock/ shipment use cases.

b) Task Analysis

Task analysis is the process of interpreting how users accomplish their tasks through a software system / application or other product. This type of analysis focuses on:

- What the users' goals are
- What the users do to achieve those goals
- The workflow the users follow to accomplish their tasks
- What the users' level of experience is
- How the users are affected by their physical environment

It's important to perform a task analysis early in your process, in particular prior to design work. Task analysis helps support several other aspects of the user-centred design process, including:

- Website requirements gathering
- Developing your content strategy and site structure
- Wireframing and Prototyping
- Performing usability testing

Types of Task Analysis

a) Cognitive Task Analysis is focused on understanding tasks that require decision-making, problem-solving, memory, attention, and judgement.

Cognitive task analysis has its application in knowing:-

- Performance differences between novices and experts
- Mental workload associated with complex controls and displays
- Decision-making of experts
- The development and evolution of mental models.
- Information requirements for command-and-control systems
- Troubleshooting, fault isolation, and diagnostic procedures

Process used for Cognitive Task Analysis

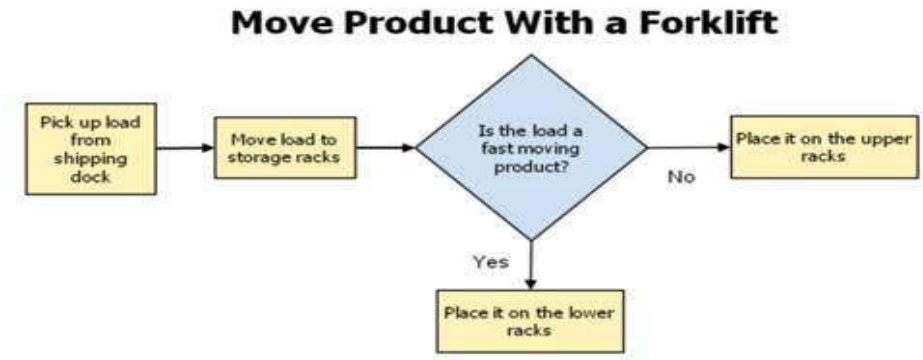
- Interviewing Methods
- Team Communication Methods
- Diagramming Methods
- Verbal Report Methods
- Psychological Scaling Methods

Task Analysis can be depicted using

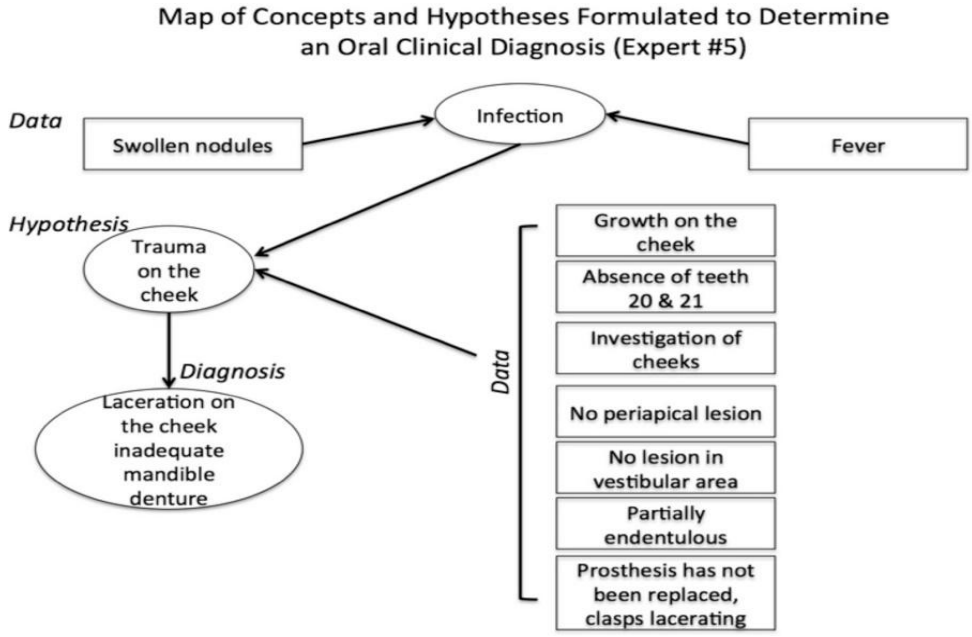
- Flowchart,
- Hierarchical chart,
- Work breakdown structure
- Concur Task Trees

Example on Cognitive Task Analysis

Flowchart demonstrating the decision-making process that a forklift operator must make when moving material from the receiving dock to a storage area:



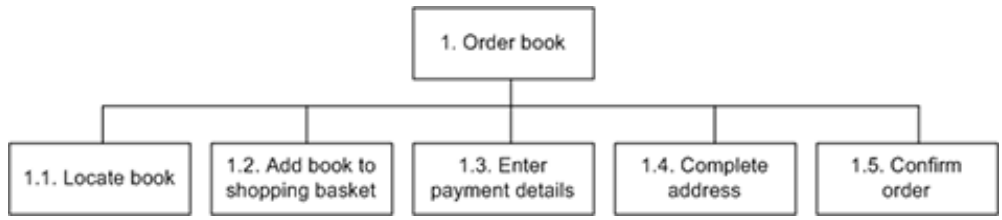
Cognitive Task Analysis using flow chart to determine an oral clinical diagnosis



Cite:-Journal of Dental Education- Wiley Online Library

b) Hierarchical Task Analysis Hierarchical task analysis (HTA) also referred as hierarchical decomposition is a widely used type of task analysis where a high-level task is decomposed into a hierarchy of subtasks.

Figure below —Hierarchical task analysis for ordering a book

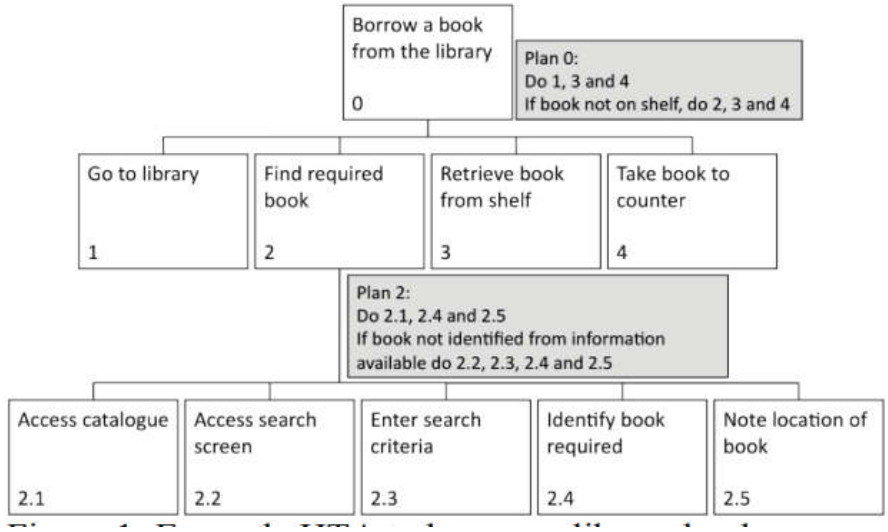


In this hierarchical task analysis, task is broken into subtasks, expressing the relationships between the parent task and its subtasks through a numbering scheme. This hierarchical task analysis is very coarse from a user experience standpoint. It does *not* communicate anything about what is happening at the level of a user's interaction with the system. However, it *does* give a clear understanding of the task's high-level steps. A more complete task analysis would ultimately get down to the level of user interactions. To illustrate, Subtask 1.4, "Complete address," would break down as follows:

1. Locate the **Full Name** field.	1. Locate the **County** field.
2. Move the insertion point to the field.	2. Move the insertion point to the field.
3. Type the full name.	3. Type the county.
4. Locate the **Address Line 1** field.	4. Locate the **Postcode** field.
5. Move the insertion point to the field.	5. Move the insertion point to the field.
6. Type the address.	6. Type the postal code.
7. Optional: Locate the **Address Line 2** field.	7. Locate the **Country** field.
8. Move the insertion point to the field.	8. Move the insertion point to the field.
9. Type the address.	9. Select the country from the drop-down list.
10. Locate the **Town/City** field.	10. Locate the **Phone Number** field.
11. Move the insertion point to the field.	11. Move the insertion point to the field.
12. Type the town or city.	12. Type the phone number.

Optionally, you can provide an illustration of the screen on which a user performs this task, helping to put this interaction in context. Figure below shows the screen for the "Complete address" task.

Example of Hierarchical Task Analysis to borrow a Library book

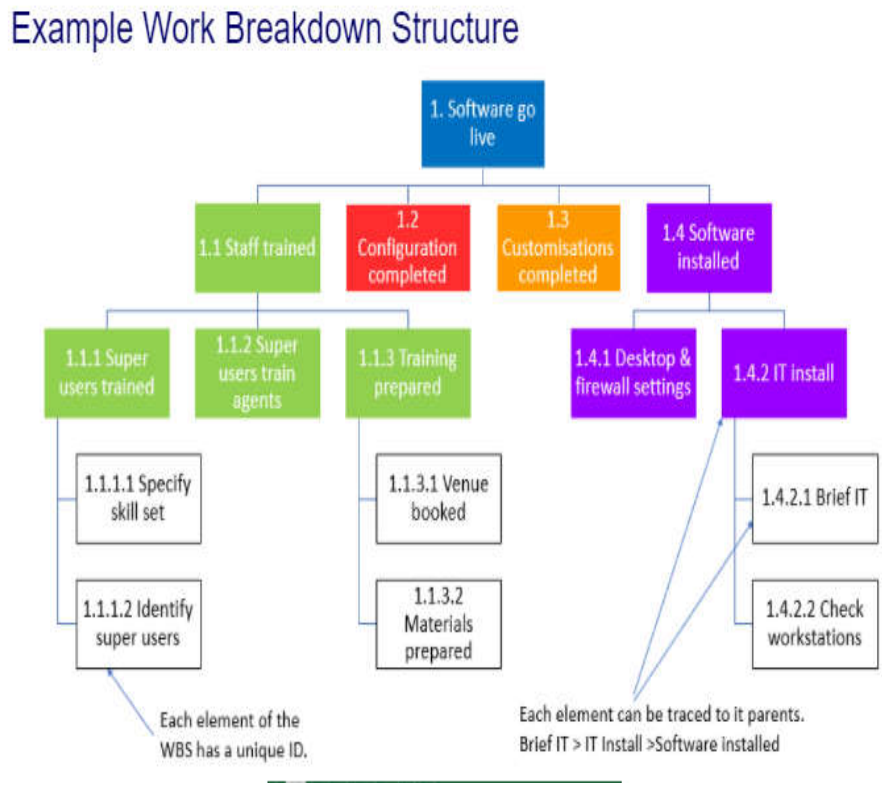


(c)Work breakdown Structure: A Work Breakdown Structure (WBS) is a hierarchical outline of the tasks that exhibits the project subtask and deliverables. The WBS "breaks down" the structure of a project into manageable deliverables. Each deliverable is assigned a task, or series of tasks that can be further broken down into subtasks to meet the needs of the project.

WBS benefits: -

- it defines and organizes the work required
- it facilitates the quick development of a schedule by allocating effort estimates to specific sections of the WBS
- it can be used to identify potential scope risks if it has a branch that is not well defined
- it provides a visual of entire scope
- it can be used to identify communication points
- it provides a visual of impacts when deliverables are falling behind
- it can be used to show and assign accountabilities and responsibilities
- it can show control points and milestones

Example: Work breakdown structure-(WBS) demonstrating the prerequisite before the software goes live



(d) Concur Task Trees for Task Analysis: is a notation for task model specifications useful to support design of interactive applications specifically tailored for user interface model-based design.

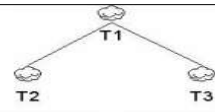
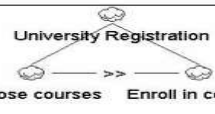
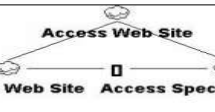
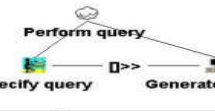
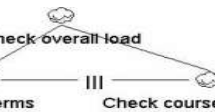
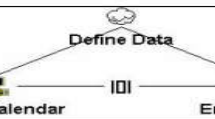
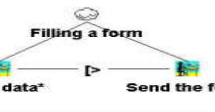
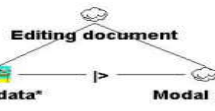
Notation: - Type of Task

Abstraction	Interaction	Application	User	Co-operation
				

Temporal operators used in Concur Task Trees

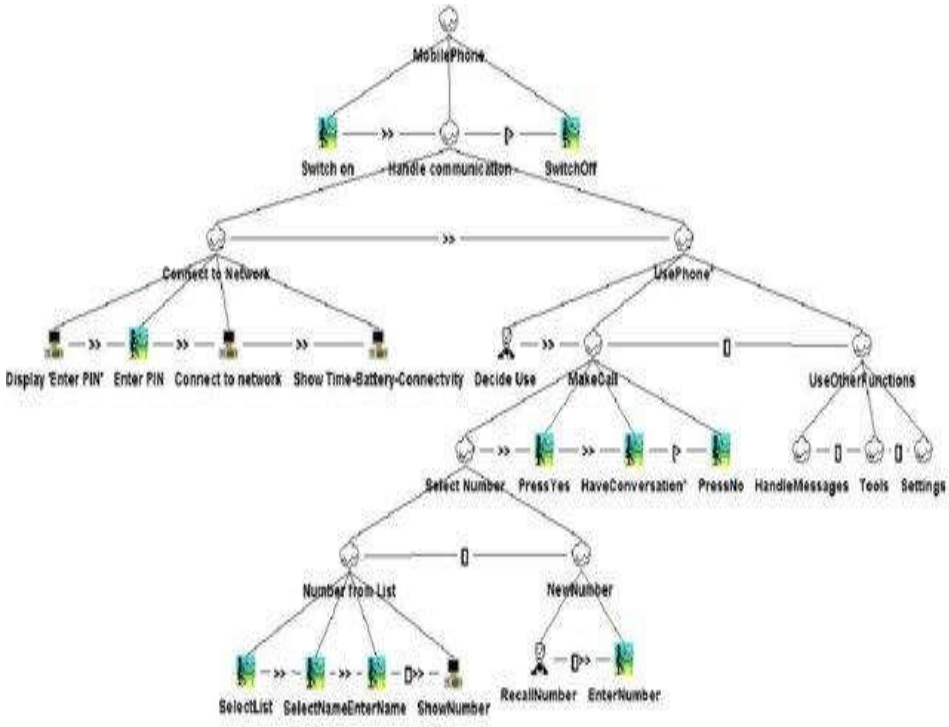
Choice	T1 [] T2
Order Independency	T1 |=| T2
Interleaving	T1 ||| T2
Synchronisation	T1 |[]| T2
Sequential composition (enabling)	T1 >> T2
Sequential composition with information passing	T1 []>> T2
Disabling	T1 [> T2
Infinite Iteration (unary operator)	T1*
Optional Execution (unary operator)	[T1]
Suspend/Resume	T1 |> T2

To avoid confusion, the temporal relationship priority in decreasing order is: unary operators, [], |=|, |||, |[]|, [> and |>, >>, []>>. There are some restrictions in combining unary operators with others: • T1*>>T2 is wrong, T2 is never reachable 5 . left and right side of the operator |> , [> and [] cannot be optional

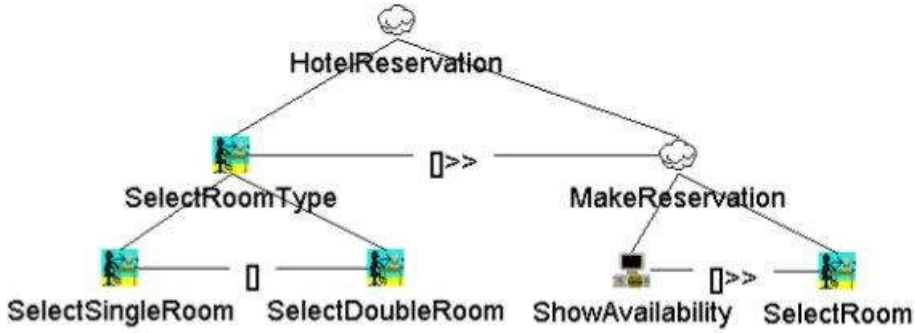
	**Hierarchy** Tasks at same level represent different options or different tasks at the same abstraction level that have to be performed. Read levels as "In order to do T1, I need to do T2 and T3", or "In order to do T1, I need to do T2 or T3"
	**Enabling** Specifies second task cannot begin until first task performed. Example: I cannot enroll at university before I have chosen which courses to take.
	**Choice** Specifies two tasks enabled, then once one has started the other one is no longer enabled. Example: When accessing a web site it is possible either to browse it or to access some detailed information.
	**Enabling with information passing** Specifies second task cannot be performed until first task is performed, and that information produced in first task is used as input for the second one. Example: The system generates results only after that the user specifies a query and the results will depend on the query specified.
	**Concurrent tasks** Tasks can be performed in any order, or at same time, including the possibility of starting a task before the other one has been completed. Example: In order to check the load of a set of courses, I need to consider what terms they fall in and to consider how much work each course represents
	**Concurrent Communicating Tasks** Tasks that can exchange information while performed concurrently Example: An application where the system displays a calendar where it is highlighted the data that is entered in the meantime by the user.
	**Task independence** Tasks can be performed in any order, but when one starts then it has to finish before the other one can start. Example: When people install new software they can start by either registering or implementing the installation but if they start one task they have to finish it before moving to the other one.
	**Disabling** The first task (usually an iterative task) is completely interrupted by the second task. Example: A user can iteratively input data in a form until the form is sent.
	**Suspende-Resume** First task can be interrupted by the second one. When the second terminates then the first one can be reactivated from the state reached before Example: Editing some data and then enabling the possibility of printing them in an environment where when printing is performed then it is no possible to edit.

Example of Concur Task trees on Mobile phone

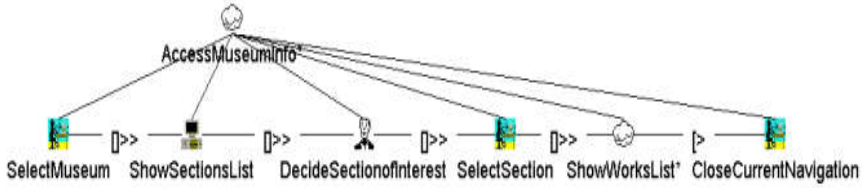
It is important to note that the task model depends on the features of one specific type of cellular phone. This example shows that task models can also be useful to analyses the design of more recent mobile systems. The model is structured into two sequential parts: one dedicated to connecting to the phone the other one to handle the communication. At any time, it is possible to switch off the phone. The first part is a sequence of interactive and system tasks. Then, there is the part dedicated to the actual use of the phone, which is iterative because a user can perform multiple uses in a session. First, users have to decide what they want to do. Next, there are two main choices: either make a call (the most frequent function) or use other functions. In case of a call, the user can either select the number from a list or recall and enter it. The other possible uses are not detailed for sake of brevity.



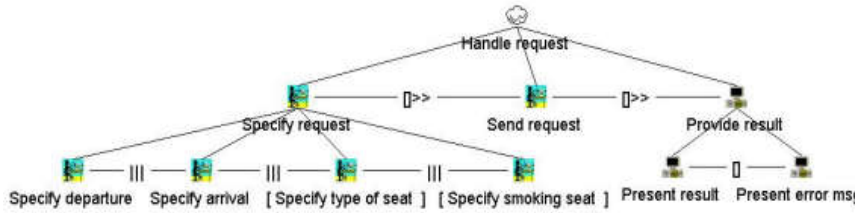
Example of applying the CTT notation to a hotel booking task:



The symbols on the arcs indicate the kind of temporal operator. In the above diagram a choice is given between booking a single or double room. However, it is necessary to select the room type before making the reservation. The choice of room type is an input to the reservation task. CTT has a rich visual vocabulary for its temporal operators. The three kinds of node symbols differentiate between tasks requiring user input, tasks involving the user reading output presented by the application and grouped tasks.

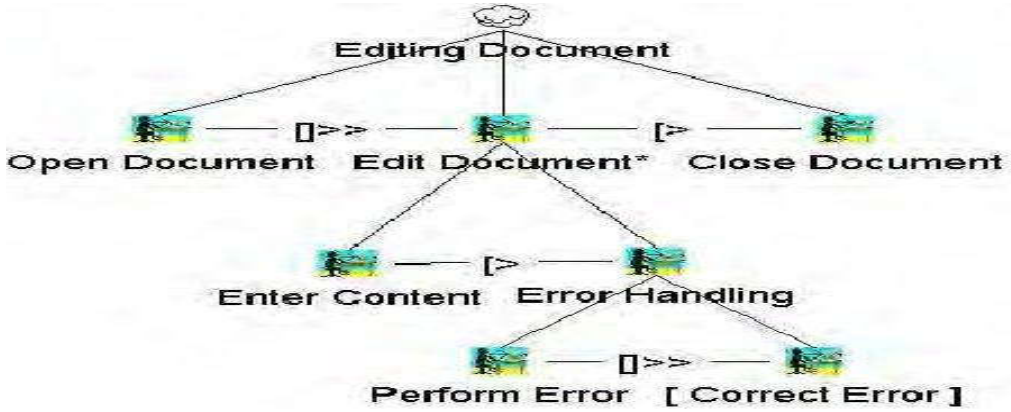


Example of optional task :- In the above diagram the once the task CloseCurrentNavigation is performed then the task Select Museum is enabled again because the parent task is iterative. This means that it can be performed multiple times and so its first subtask is enabled once the last one has been completed.



Optional tasks have a subtle semantics in CTT. They can be used only with concurrent and sequential operators. Their name is closed between squared brackets. For example, in Figure 4 Specify type of seat and Specify smoking seat are optional tasks. This means that once the mandatory sibling tasks (Specify departure and Specify arrival) are performed then both the optional tasks and the task after the enabling operator (Send request) are enabled. If the task after the enabling operator is performed, then the optional tasks are no longer available.

In the task model, tasks inherit the temporal constraints of the ancestors. So, for example in Figure below ShowAvailability is a sub-task of MakeReservation and since Make Reservation should be performed after SelectRoomType then this constraint will apply also to ShowAvailability.



For editing a document, you will open the document and then edit. Opening the document is a prerequisite for editing the document and close document.

c) DOMAIN ANALYSIS: It refers to understanding the domain requirements of the proposed software. It refers to understanding of the users work environment that will be automated using the proposed system. A domain expert is a person who has experience and knowledge of the given domain. Domain analysis is used to identify design patterns in software and data

The "specific application domain" can be for organic, semidetached or embedded systems which are systems that differ in terms of its complexity. For example Domain can range from College, banking to avionics, multimedia video games to applications within an MRI device. The goal of domain analysis is to find or create those classes that are broadly applicable, so that they may be reused.

Domain analysis is an ongoing umbrella software engineering activity that is not connected to any one software project. In a way, the role of a domain analyst is like the role of a master tool smith in a heavy manufacturing environment. The job of the tool smith is to design and build tools that may be used by many people doing similar but not necessarily the same jobs. The role of the domain analyst is to design and build reusable components that may be used by many people working on similar but not necessarily the same applications.

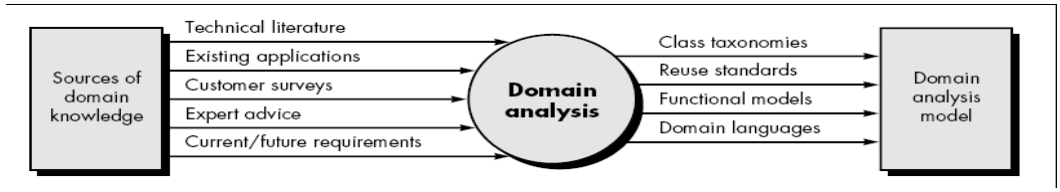
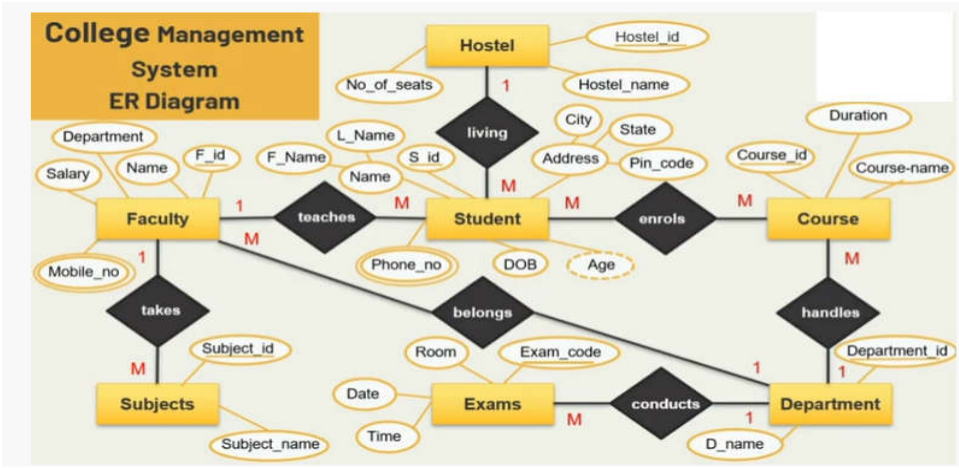


Figure illustrates key inputs and outputs for the domain analysis process.

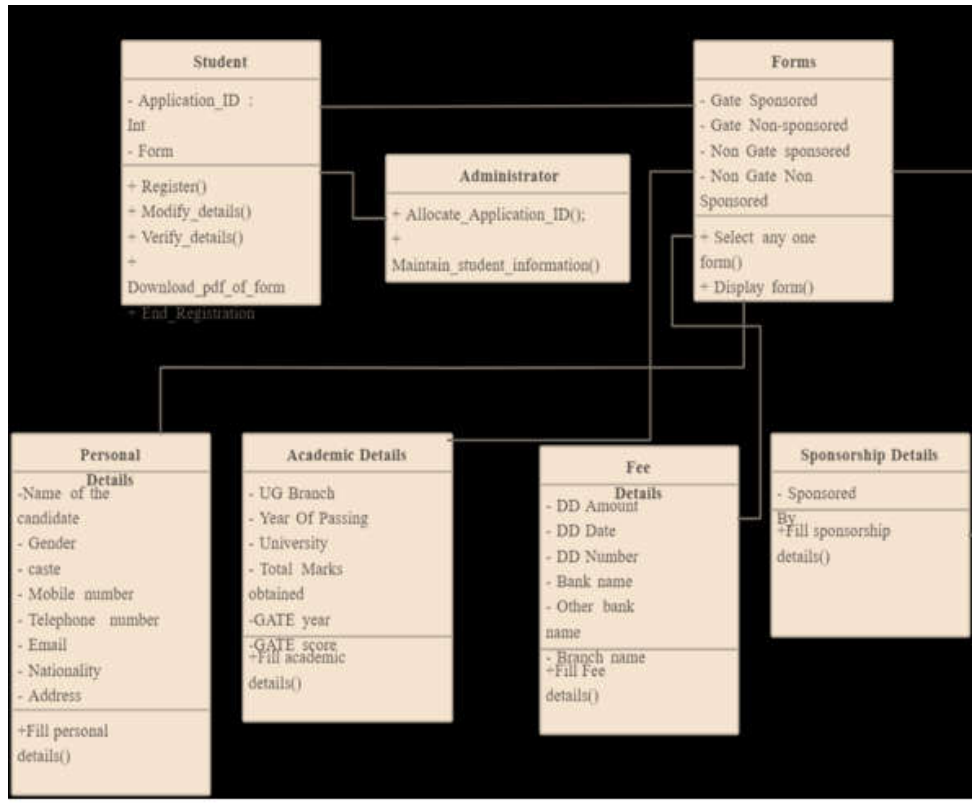
Domain Analysis methodologies include Unified modelling language diagrams Class diagram, Component diagram, Package diagram, etc. and Entity relationship diagrams

Examples

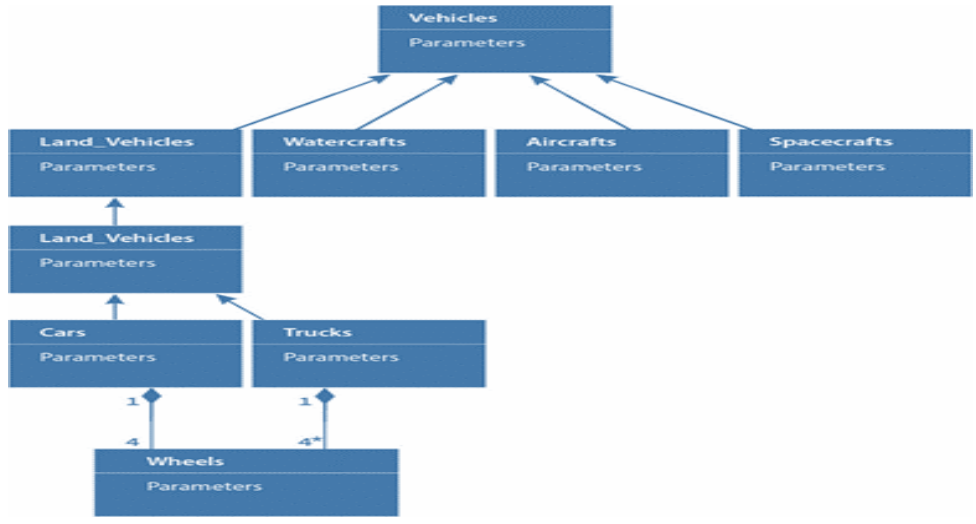
Entity Relationship Diagram for College Management System for understanding the college domain



Class diagram for Admission System



Class diagram for Vehicle domain

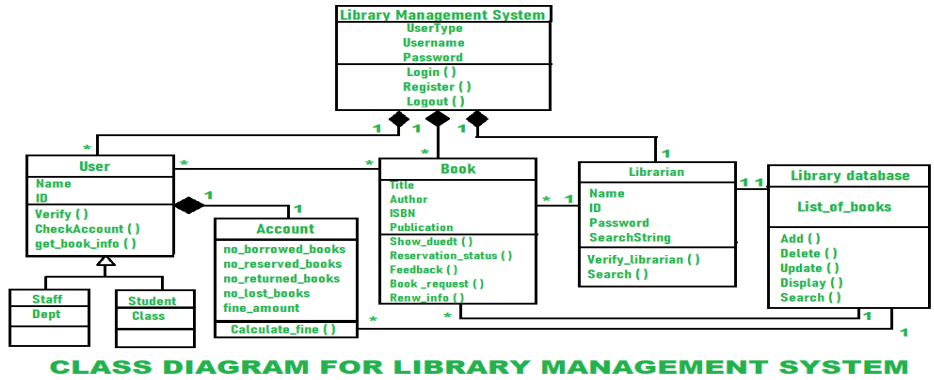


Class Diagram for Library Management System simply describes structure of Library Management System class, attributes, methods or operations, relationship among objects.

Classes of Library Management System:

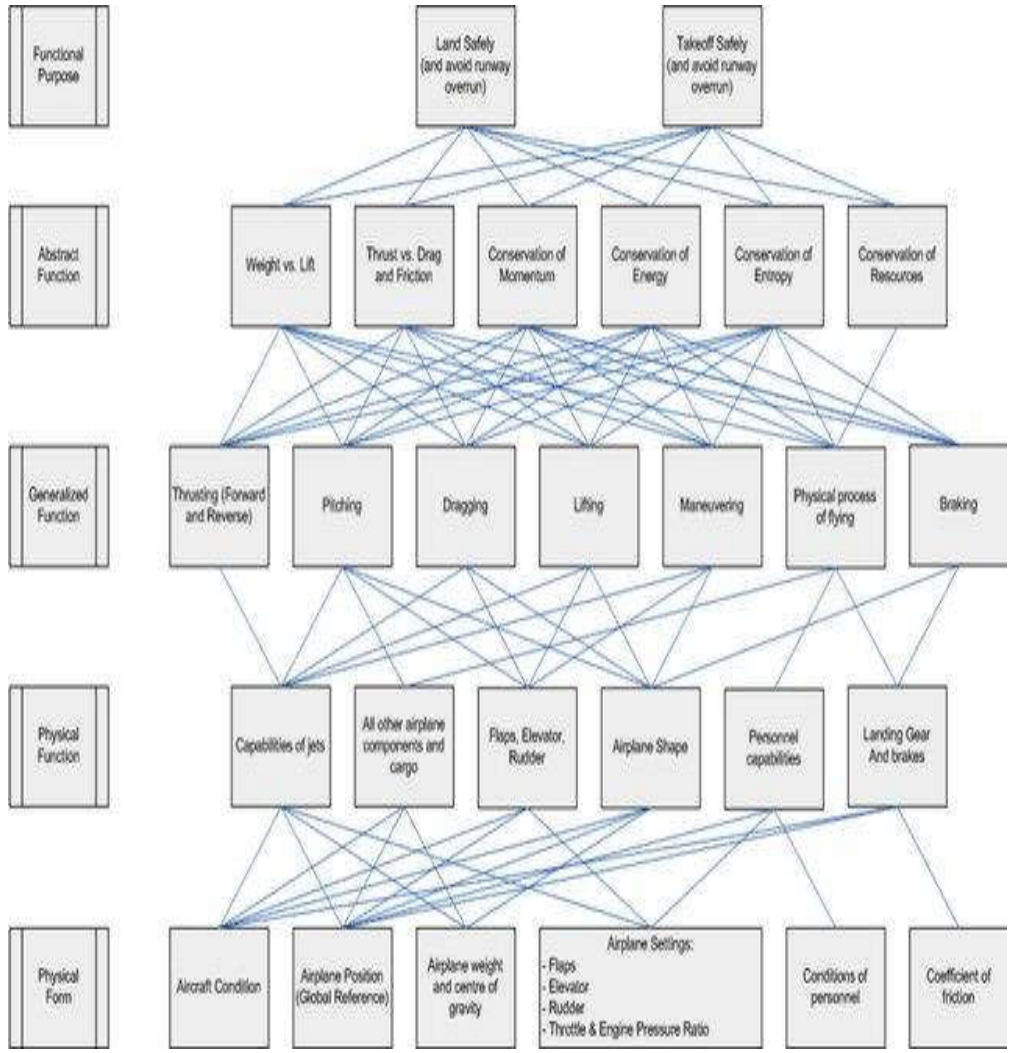
CLASS	ATTRIBUTES	METHODS
User Class –It manages all operations of user.	UserType, Username, Password	**Library Management System Methods –**Login(), Register(), Logout()
Librarian Class – It manages all operations of Librarian.	**User Attributes –** Name, Id	**User Methods –** Verify(), CheckAccount(), get_book_info()
Book Class –It manages all operations of books. It is basic building block of system.	**Librarian Attributes –**Name, Id, Password, SearchString	**Librarian Methods –** Verify_librarian(), Search()
Account Class –It manages all operations of account.	**Book Attributes –** Title, Author, ISBN, Publication	**Book Methods –** Show_duedt(), Reservation_status(), Feedback(), Book_request(), Renew_info()
Library database Class –It manages all operations of library database.	**Account Attributes –** no_borrowed_books, no_reserved_books, no_returned_books, no_lost_books fine_amount	**Account Methods –** Calculate_fine()
Staff Class –It manages all operations of staff.	**Library database Attributes –** List_of_books	**Library database Methods –**Add(), Delete(), Update(), Display(), Search()
Student Class –It manages all operations of student.	**Staff Class Attributes –**Dept	
	Student Class Attributes –Class	

Class diagram for Library Management System

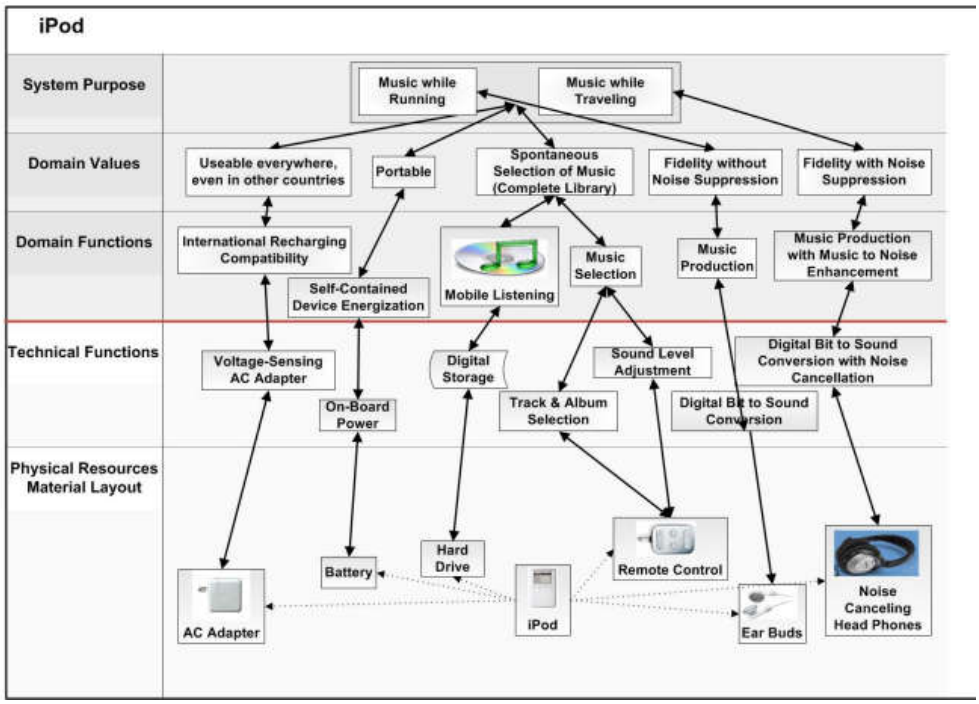


CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

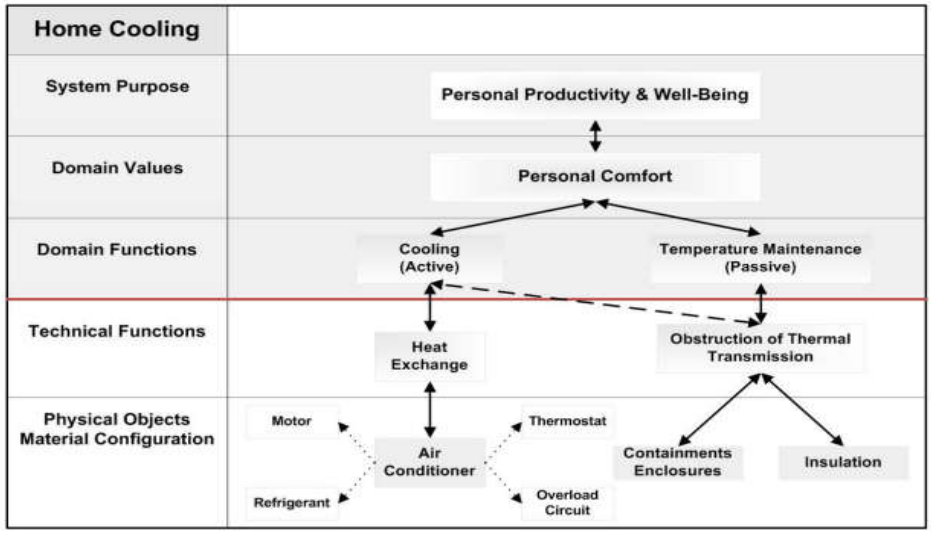
Work Domain Analysis : Aircraft Domain



Technical Domain Analysis of IPOD

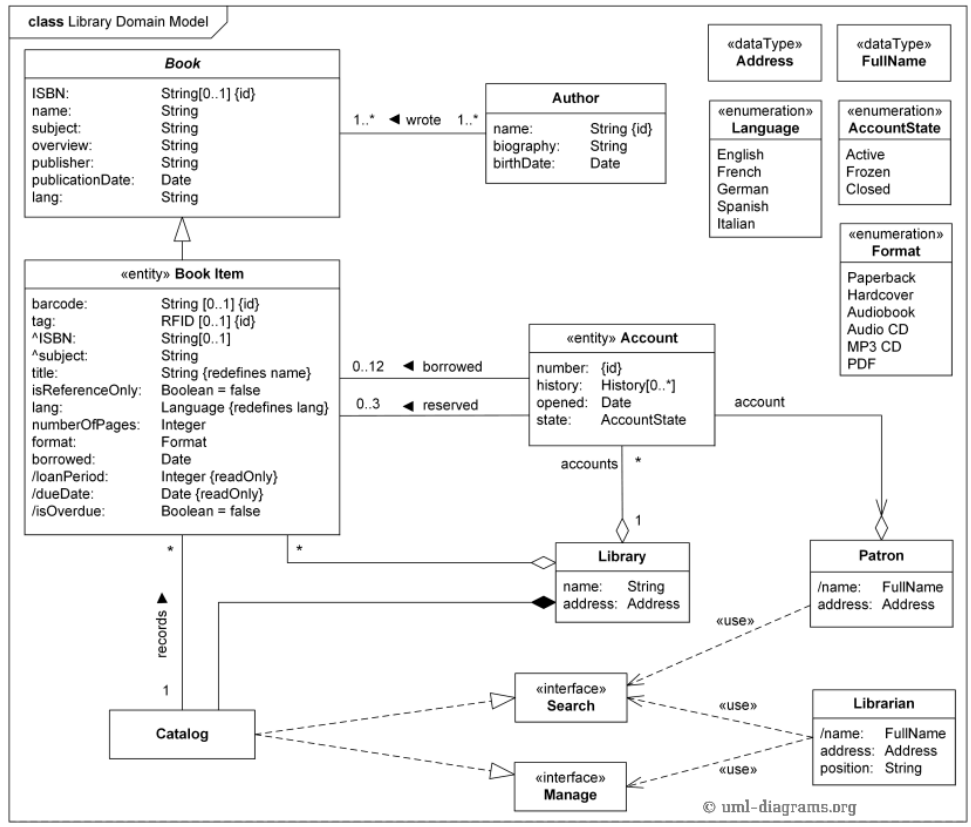


Technical Work Domain Analysis of Home Cooling system



Library Management System Domain Model

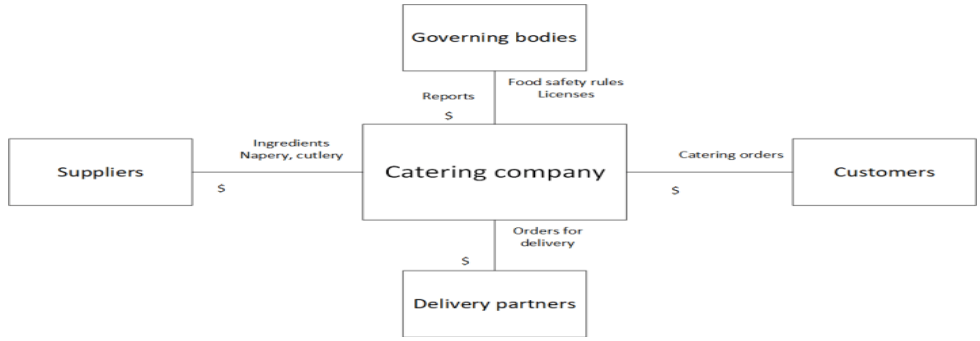
Library Domain Model describes the library management system that comprises of several classes, Attributes and functions of classes are depicted. Also, the relationship between classes is demonstrated.



Each physical library item - book, tape cassette, CD, DVD, etc. could have its own item number. To support it, the items may be **barcoded**. The purpose of barcoding is to provide a unique and scannable identifier that links the barcoded physical item to the electronic record in the **catalogue**. Barcode must be physically attached to the item, and barcode number is entered into the corresponding field in the electronic item record.

Barcodes on library items could be replaced by **RFID tags**. The RFID tag can contain item's identifier, title, material type, etc. It is read by an RFID reader, without the need to open a book cover or CD/DVD case to scan it with barcode reader. Book item class is inherited from book class.

Context models provides birds eye-view of a system . They depict the context of a business, a system, or a process. Here is a simple context model of a catering company:



Suggest some key requirements in each category (functional, data, Environmental, user characteristics, and usability goals) for each of the

following systems. In addition, what factors (environmental, user, usability) would affect the following systems most? (Where 1=Most important and 3= Least important)

- Self-service cafeteria in a university's - paying using credit system

Functional Data Environmental	Calculate total cost of purchases Access to price of products in cafeteria Noisy and busy environment, users maybe talking while using the system
User characteristics	Majority of users under 25, comfortable dealing with technology
Usability goals	The system should be easy to learn so that new users can use it immediately, and memorable for frequent users. Users would not want to wait around for the system to finish processing, so system needs to be efficient
Factors	1. Environmental 2. Usability 3. User characteristics

Practical Questions

1) Prepare a UML Class diagram for Online Shopping

2) Create a Work break down structure for the task of website designing based on the following inputs

1.Gather Requirements	1.3.2 Page views	2.2.3 Navigation layout	4.3 Integrate back end and front end
1.1 Technical specifications	1.3.3Session length	2.3 Content elements	5. Create Content
1.1.1 Expected bandwidth	2. Establish Design	2.3.1 About page	5.1 Create content summary
1.1.2 User registration	2.1. Design elements	2.3.2 Contact page	5.2 Establish content details
1.1.3 Restricted areas	2.1.1 Banner	2.3.3 Services page	5.3 Assign content creation
1.2 User requirements	2.1.2 Footer	2.3.4 FAQ page	5.4 Create detailed content
1.2.1 Menu navigation	2.1.3 Logo	2.3.5 Photo Gallery	6. Load Content 7. Test Site
1.2.2 Interactive modules	2.1.4 Color scheme	3. Select Technical Framework	7.1 Navigation

1.2.3 Static pages	2.1.5 Font usage	3.1 Evaluate options against requirements	7.2 Interactive elements
1.2.4 Flash elements	2.2 Overall layout	3.2 Evaluate cost and time to develop	7.2.1 Contact form
1.3 Reporting requirements	2.2.1 Column setup	3.3 Make decision	7.2.2 User registration
1.3.1 Bandwidth & usage	2.2.2 Optional modules	4. Implement Technical Framework	7.3 Browser compatibility
		4.1 Build or acquire back end	8. Roll Out Site
		4.2 Build or acquire front end (user interface)	8.1 Establish target date
			8.2 Create communication plan
			8.3 Make site live

3) For the below given hierarchy description draw the concur task tree model

Hierarchy description ...
0. in order to clean the house
1. get the vacuum cleaner out
2. get the appropriate attachment
3. clean the rooms
3.1. clean the hall
3.2. clean the living rooms
3.3. clean the bedrooms
4. empty the dust bag
5. put vacuum cleaner and attachments away

... and plans
Plan 0: do 1 - 2 - 3 - 5 in that order. when the dust bag gets full do 4
Plan 3: do any of 3.1, 3.2 or 3.3 in any order depending on which rooms need cleaning

N.B. only the plans denote order

4) Prepare an ERD Diagram for Payroll Management System

5) Prepare a domain model for Hospital Management System

6) Prepare an ATM Work flowchart.

7) Prepare an UML class diagram for Airline Reservation System

8) From what you have learned about cognitive psychology, devise appropriate guidelines for use by designers. You may find it helpful to group these under key headings: for example, Functionality, Visual perception, Memory, etc, although some may overlap such groupings. **For e.g.**

9) For an ATM System Suggest some key requirements in each category(functional,data, environmental, user characteristics and usability goals)

Module IV

4

DESIGN

Learning Objectives: To make the learners familiar with the different levels of designs to be created in a software project and the technique used for designing.

Learning Outcomes: The learners would be enabled to design a software project.

CHAPTER 1- INTRODUCTION TO DESIGN

Software design is **a pictorial representation of user requirement to better understand the same and also to know if missing or conflicting requirements if any.** It helps the programmer in software coding and implementation. ... Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain.

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

The software design process can be divided into the following three levels of phases of design:

1. User Interface Design
2. Architectural Design
3. Detailed Design

User interface (UI) design is about focusing on the right use of user interface elements like radio button, check boxes, drop down list , text box, push button etc. T**he process designers use to build interfaces in software or computerized devices, focusing on looks or style**. Designers aim to create interfaces which users find easy to use and pleasurable. UI design refers to graphical user interfaces and other forms—e.g., voice-controlled interfaces.

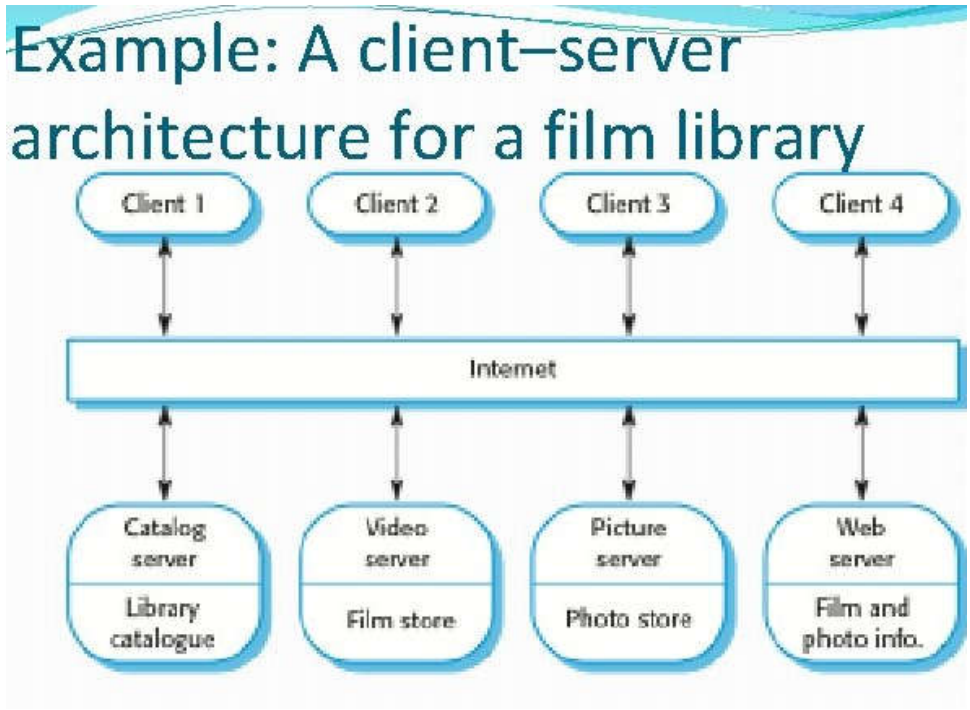
UI and UX are used interchangeably these days. UX design is the process of enhancing user satisfaction by improving the usability and accessibility of a product. On the other hand, UI designers are responsible for the product's look and feel. The below is the difference between UX and UI

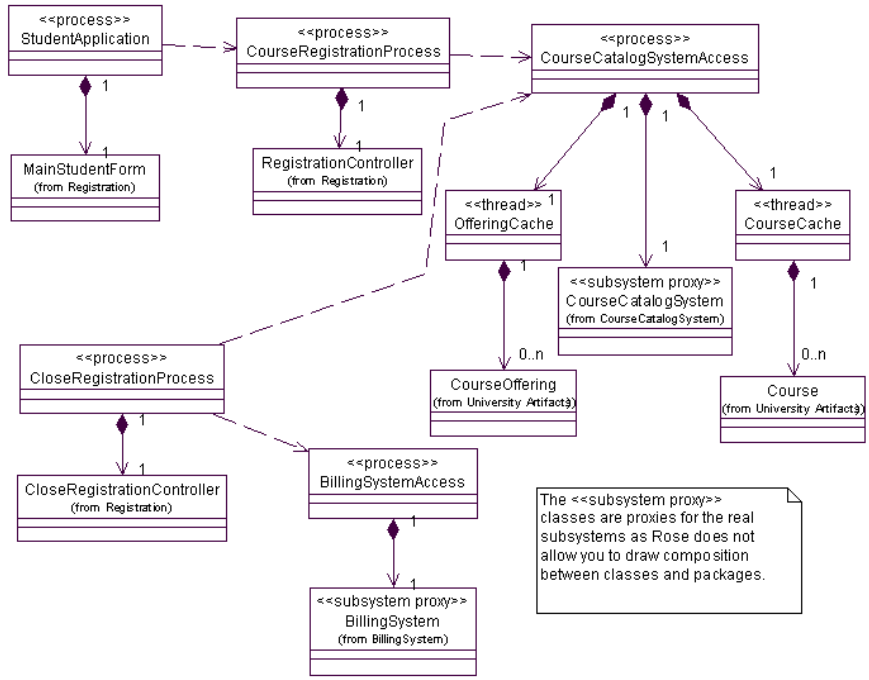
User Interface(UI)	User Experience(UX)
It is a human first approach to designing the aesthetic experience of a product.	It is a human first approach to product design
Its application is only for digital products.	It has its application to both physical and digital product
Focus: Visual touch points that allows to interact with the product.	The focus is on the full experience from the first contact till last.
Creates combination of typography , color palatte,buttons, animation, imagery etc.	Offers Structural design solutions to pain points that user encounters during the use of a product.
Result is product that delights users aesthetically.	Result is the product that delights users with its effectiveness.

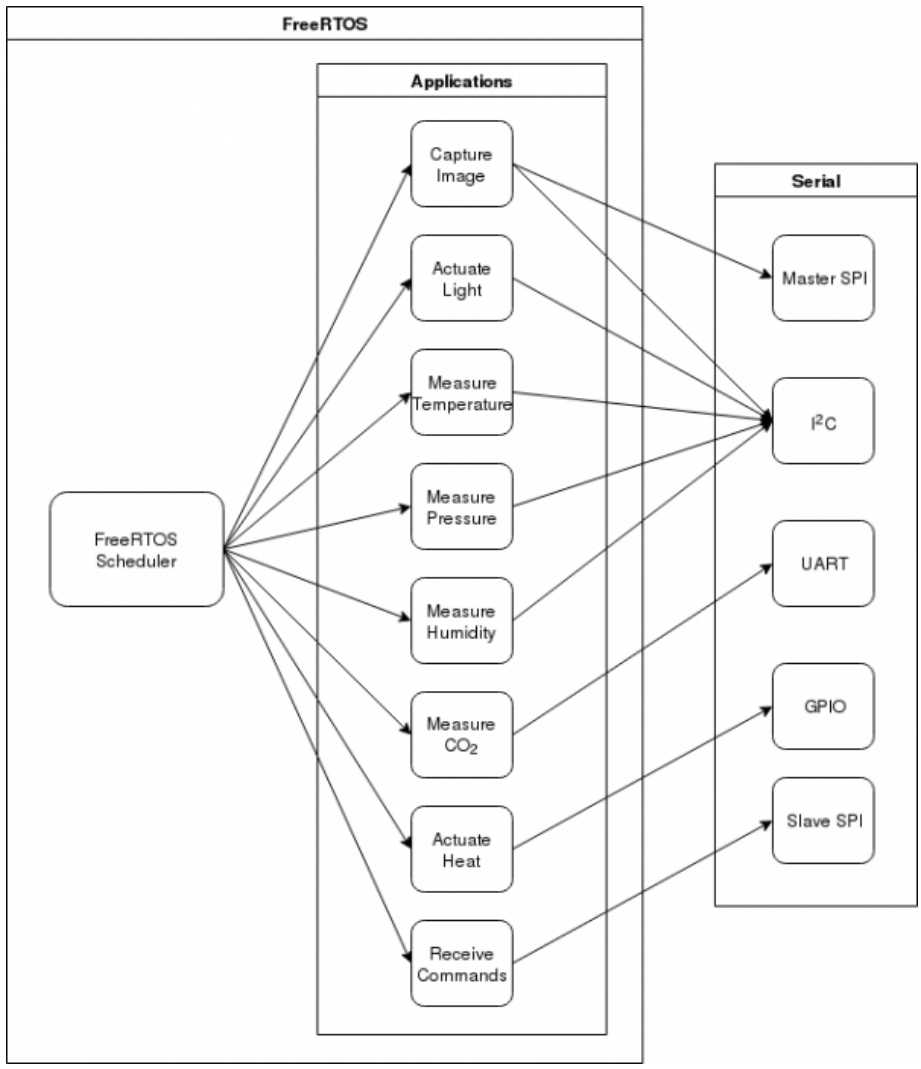
User Interface Design Example

Architectural Design is the designing of overall components and their relationships. focuses on components or elements of a structure.UML component diagram, deployment diagram exhibits architectural design.

Architectural Design Example







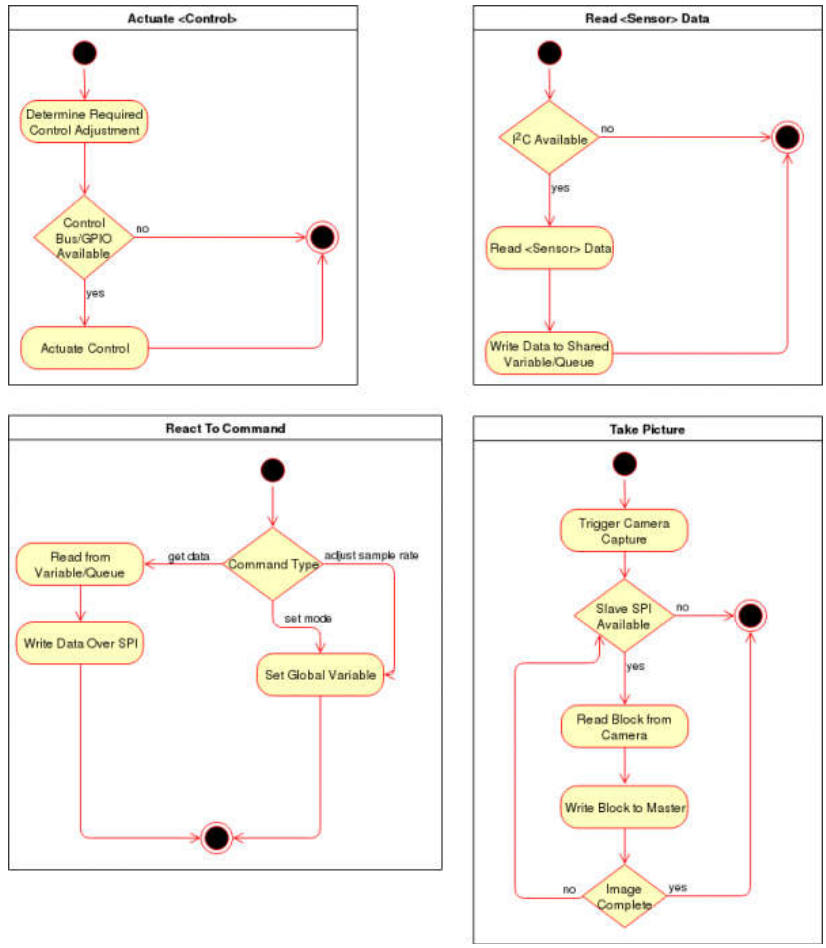
The above is the architectural diagram of an embedded system software BioCell, which consists of a MSP432, a collection of sensors, and some peripheral devices for controlling the cell.

The software on the MSP432 will be running FreeRTOS, which will handle the scheduling of all the different tasks that the BioCell needs to perform. Between these different tasks communication will be carried out through the use of shared variables. Shared variables are effective in this application because all of the shared data is only updated by a single process since most of it will be either sensor data or sample rates.

Detailed Designing includes depicting the interaction among component, sequence of activities, internal structure etc. Detailed designing includes the following: -

- Each module's responsibilities should be specified as precisely as possible
- Constraints on the use of its interface should be specified
- Pre and Post conditions can be identified module-wide invariants can be specified
- Internal data structures and algorithms can be suggested

Below is the detailed diagram on BIOCELL



Objectives of Software Design

Following are the Objectives of Software design:

Objectives of Software Design

1. **Correctness:** Software design should be correct , non-conflicting and unambiguous as per requirement.
2. **Completeness:** The design should have all components like data structures, modules, and external interfaces, attributes, relationships etc.
3. **Efficiency:** Program should efficiently use all the resources like CPU, Memory, Storage, Program must be optimized for efficiency.
4. **Flexibility/Agility:** Software must have the ability to embrace changes.

5. **Consistency:** There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

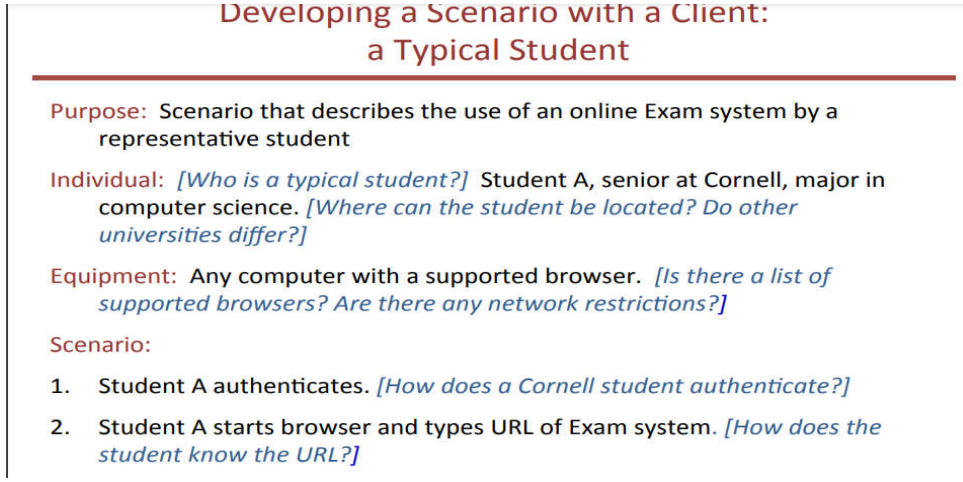
CHAPTER 2- INTRODUCTION TO USER SCENARIO AND USER STORYBOARD

USER SCENARIO

A scenario is **a description of different business situation that is used to understand and test the proposed system**. Scenarios capture the system, as viewed from the outside, e.g., by a user, using specific examples. A scenario is a scene that illustrates some interaction with a proposed system.

Example: Create a scenario for how a typical student interacts with the system based on the requirements given that the system should enable university students to take exams online from their residence using a web browser.

Developing a Scenario with a Client: A Typical Student



3. Student enters his credentials. [How will his credentials be checked]
4. Student A selects CS 1234 Exam 1.
5. A list of questions is displayed, each marked to indicate whether completed or not. [Can the questions be answered in any order?]
6. Student A selects a question and chooses whether to submit a new answer or edit a previous answer. [Is it always possible to edit a previous answer? Are there other options?]
7. [What types of question are there: text, multiple choice, etc.?] The first question requires a written answer. Student A is submitting a new answer. The student has a choice whether to type the solution into the browser or to attach a separate file. Student A decides to attach a file. [What types of file are accepted?]

8. For the second question, the student chooses to edit a previous answer. Student A chooses to delete a solution previously typed into the browser, and to replace it with an attached file. *[Can the student edit a previous answer, or must it always be replaced with a new answer?]*
9. As an alternative to completing the entire exam in a single session, Student A decides to saves the completed questions work to continue later. *[Is this always permitted?]*
10. Student A logs off.
11. Later Student A log in, finishes the exam, submits the answers, and logs out. *[Is this process any different from the initial work on this exam?]*
12. The Student A has now completed the exam. The student selects an option that submits the exam to the grading system. *[What if the student has not attempted every question? Is the grader notified?]*
13. Student A now wishes to change a solution. The system does not permit changes once the solution has been submitted. *[Can the student still see the solutions?]*
14. Later Student A logins in to check the grades. *[When are grades made available? How does the student know?]*
15. Student A requests a regrade. *[What are the policies? What are the procedures?]*

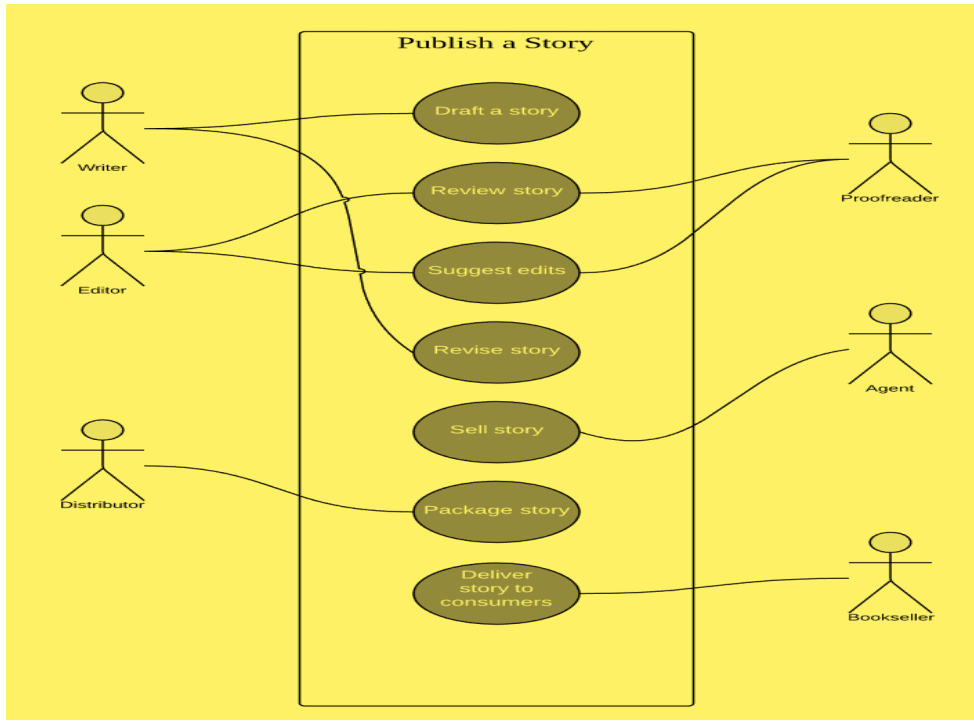
Developing a scenario with a client clarifies many functional requirements that must be agreed before a system can be built, e.g., policies, procedures, etc. Scenarios are very useful for analyzing special requirements. Examples

- Reversals. In a financial system, a transaction is credited to the wrong account. The sequence of steps to be used to reverse the transaction?
- Errors. A mail order company has several copies of its inventory database. Considering if they become inconsistent?
- Malfeasance. In a voting system, a voter has houses in two cities. Considering the scenario if he attempts to vote from both locations.

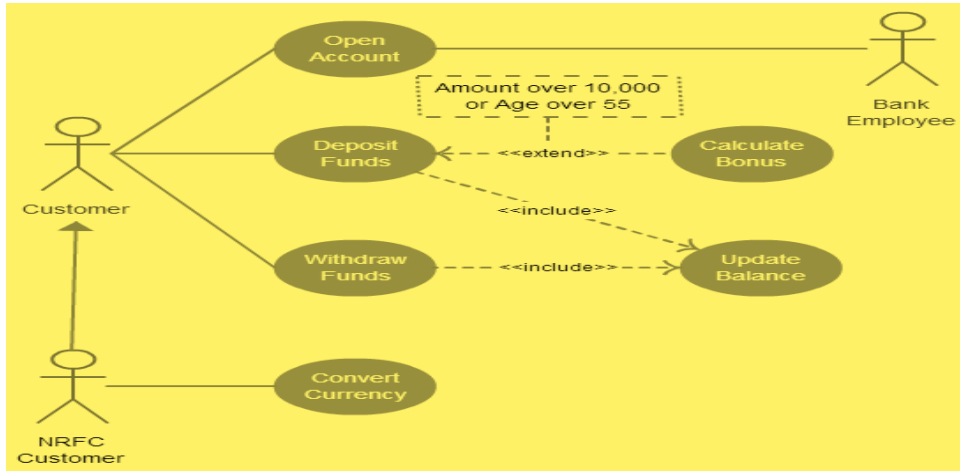
As per Murphy's Law: "If anything can go wrong, it will". One must anticipate and consider creating a scenario for what can go wrong and how it can be handled e.g., Exception Handling.

Scenarios can be modeled using a use case and process flow diagram

Example: In a book publishing system the writer will draft a story, editor and proofreader will review and suggest editing. Based on their inputs writer will revise the story. Agent will send the story to the distributor. Distributor will add this story to a particular book. (Package it) and then sell it to the customer. This scenario modelling can be done using the UML use case diagram.



Example : Banking System Use Case Diagram



User Case vs Use Case Scenario

A use case is a set of steps that are required to accomplish a specific task or goal. A use case can have multiple paths to reach the goal; each of them is considered a use case scenario. In simple words, a use case is a goal with various processes, and a case scenario represents a linear and straight path through one of the operations.

The use case it answers the questions like:

- What is the context of various task or what are the different scenarios?
- What preconditions does the process have?

- What exceptions can the task encounter?
- How will the job be accomplished?
- What errors can we encounter in the process?

How many ways do we have to accomplish the task? (Basic paths and alternative paths)

A **scenario** is a sequence of steps describing an interaction between a user and a system. So, if we have a Web-based on-line store, we might have a Buy a Product scenario that would say this:

The customer browses the catalogue and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale. The system checks the authorization on the credit card and confirms the sale both immediately and with a follow-up email.

This scenario is one thing that can happen. However, the credit card authorization might fail. This would be a separate scenario.

A **use case**, then, is a set of scenarios tied together by a common user goal. In the current situation, you would have a Buy a Product use case with the successful purchase and the authorization failure as two of the use case's scenarios. Other, alternative paths for the use case would be further scenarios. Often, you find that a use case has a common all-goes-well case, and many alternatives that may include things going wrong and also alternative ways that things go well.

A simple format for capturing a use case involves describing its primary scenario as a sequence of numbered steps and the alternatives as variations on that sequence, as shown below

Buy a Product

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Alternative: *Authorization Failure*
At step 6, system fails to authorize credit purchase
Allow customer to re-enter credit card information and re-try

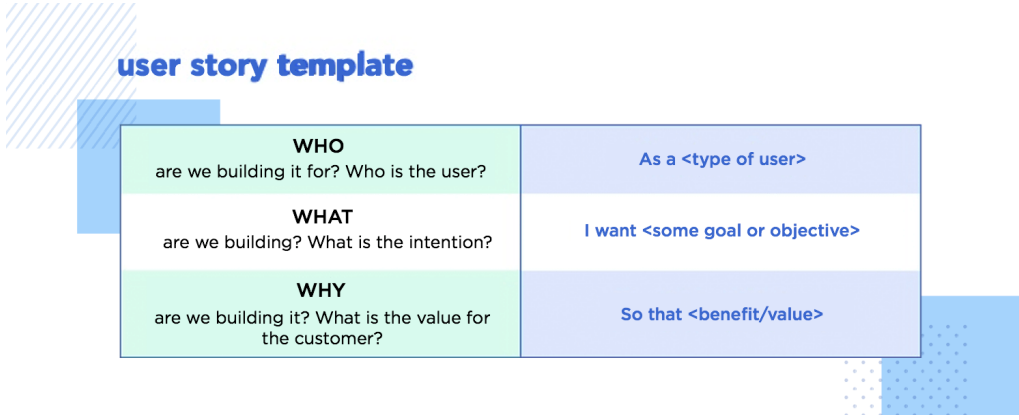
Alternative: *Regular Customer*
3a. System displays current shipping information, pricing information, and last four digits of credit card information
3b. Customer may accept or override these defaults
Return to primary scenario at step 6

USER STORY/STORYBOARD

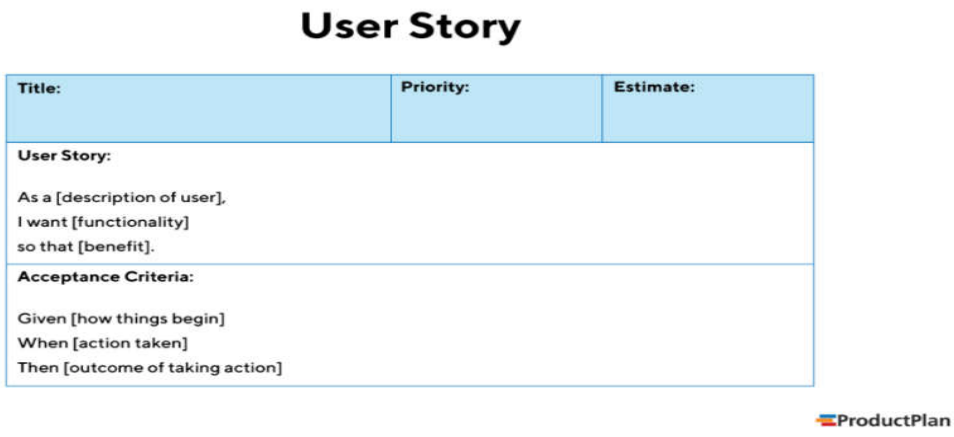
The user story is a faster and less specific tool typically used in agile methodologies. The focus of the user story is on developing short descriptions of user-cantered needs. User stories are simplified versions of requirements. A user story should also focus on being valuable to the end-user. A user story should be short, estimable, and testable. It is essential to mention that user stories do not replace requirement documentation since requirements deal with the product's specifications and technical aspects.

- The user story answers questions like:
- Who will perform the task?
- What does that user need to do?
- Why does the user need to accomplish the task?

User Story Template: -



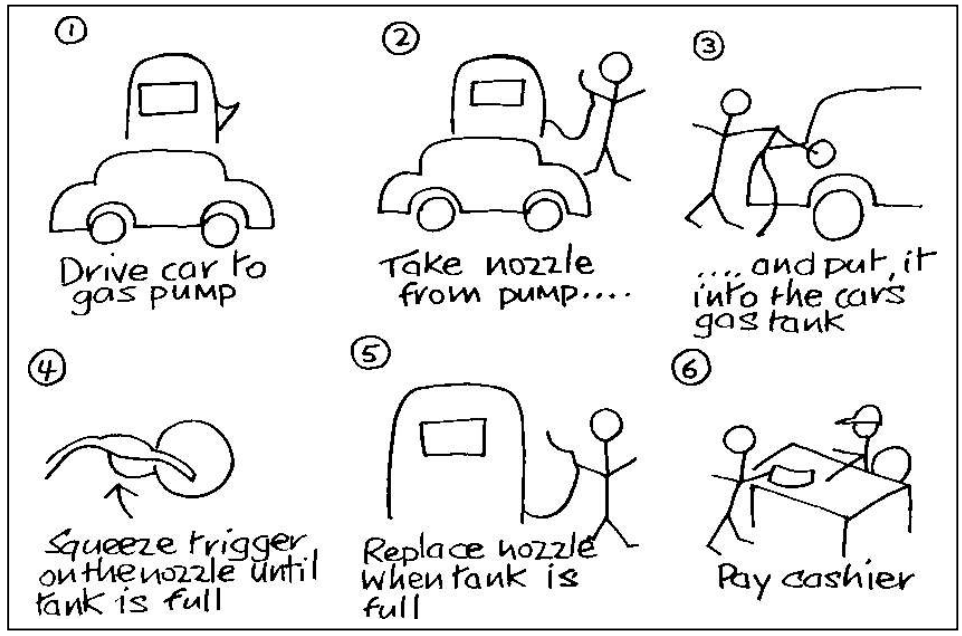
User stories are generally captured in the **following format: -**



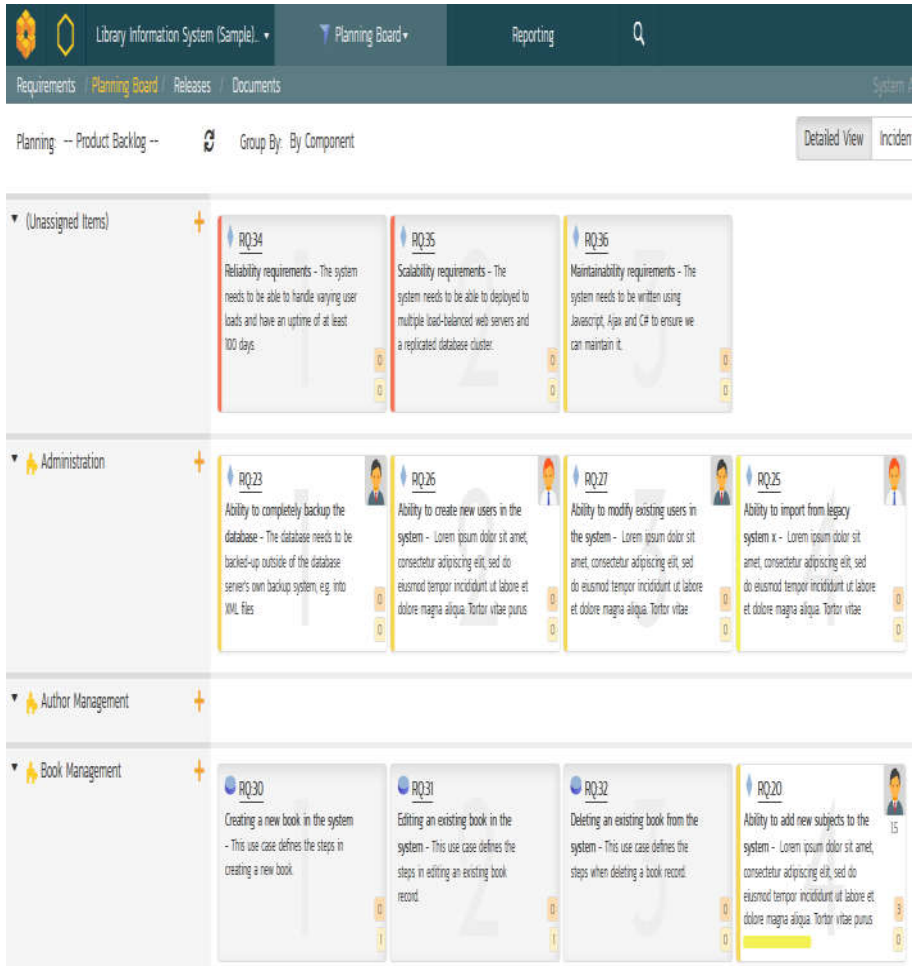
As a customer, I want to be able to search for flights between two cities to see which ones have the best price and route.
Estimate: 1.0 points
Priority: 2 - High

Example Story Card

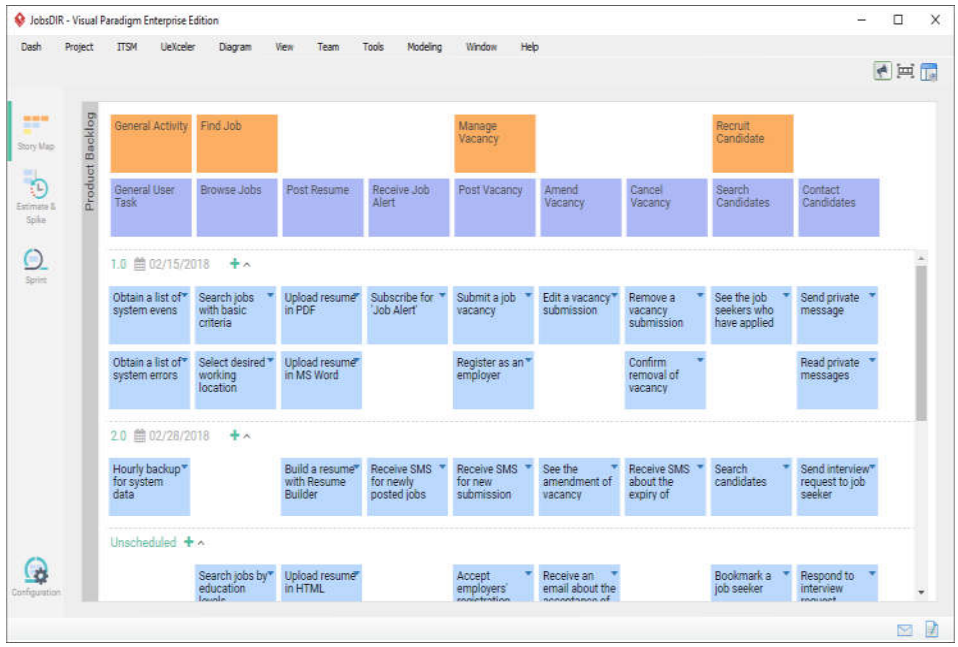
Storyboard for Car Gas filling

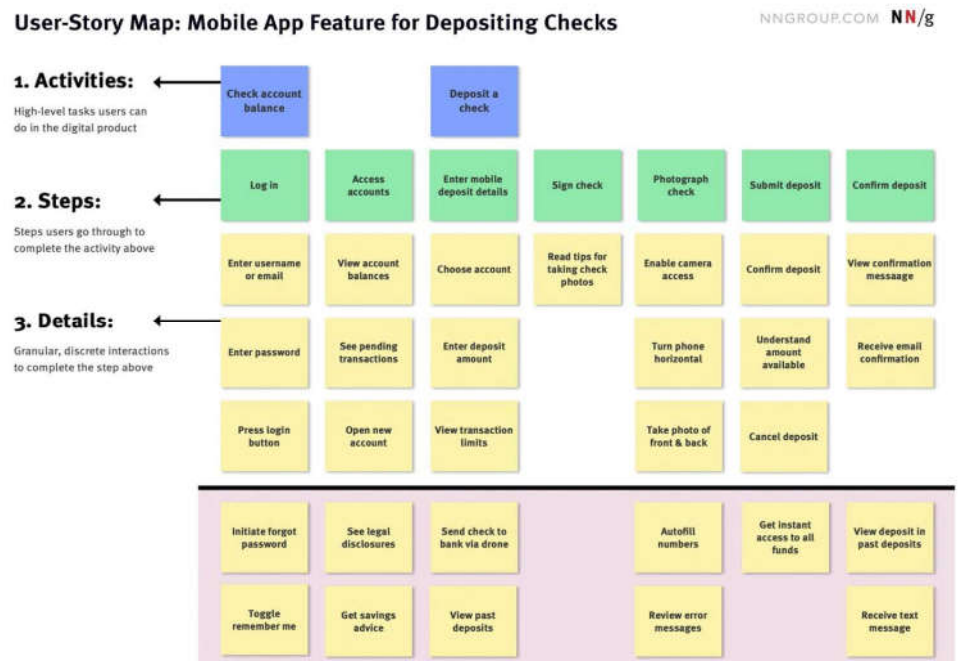


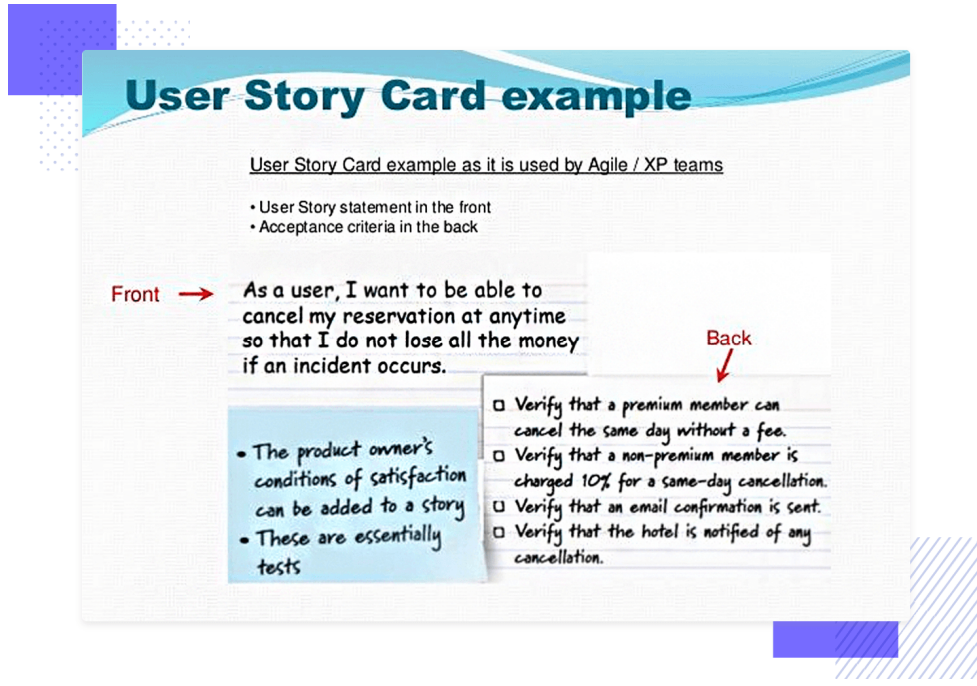
Different user stories for Library Management System



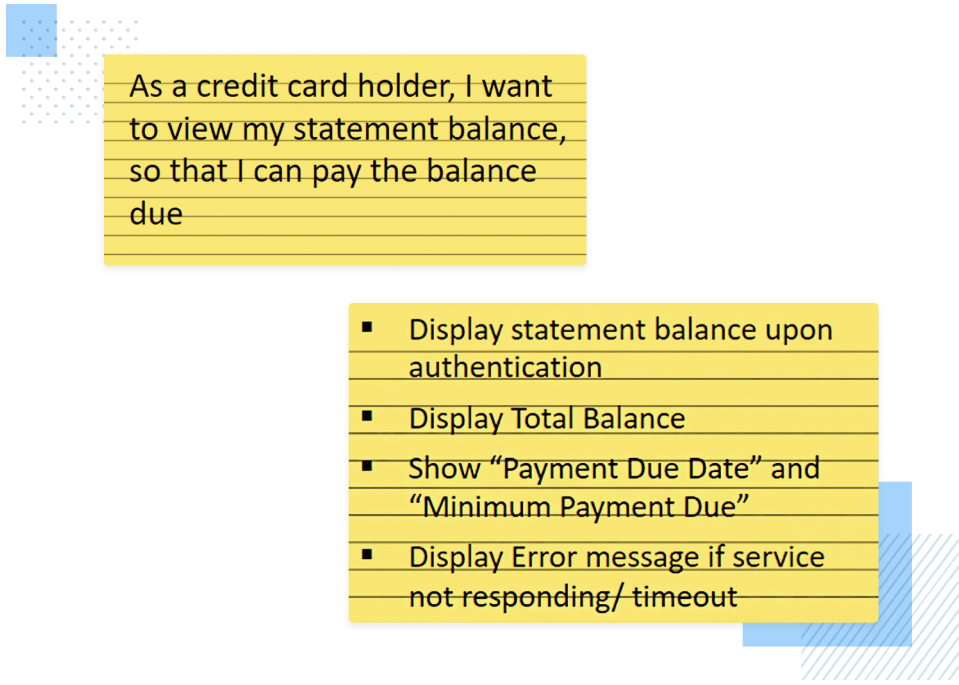
Example on User Story Mapping in Agile Environment




User-Story Map: Mobile App Feature for Depositing Checks
NNGROUP.COM NN/g
1. Activities:
High-level tasks users can do in the digital product
2. Steps:
Steps users go through to complete the activity above
3. Details:
Granular, discrete interactions to complete the step above
Check account balance
Deposit a check
Log in
Access accounts
Enter mobile deposit details
Sign check
Photograph check
Submit deposit
Confirm deposit
Enter username or email
View account balances
Choose account
Read tips for taking check photos
Enable camera access
Confirm deposit
View confirmation messaage
Enter password
See pending transactions
Enter deposit amount
Turn phone horizontal
Understand amount available
Receive email confirmation
Press login button
Open new account
View transaction limits
Take photo of front & back
Cancel deposit
Initiate forgot password
See legal disclosures
Send check to bank via drone
Autofill numbers
Get instant access to all funds
View deposit in past deposits
Toggle remember me
Get savings advice
View past deposits
Review error messages
Receive text message


User Story Card example
User Story Card example as it is used by Agile / XP teams
• User Story statement in the front
• Acceptance criteria in the back
Front
As a user, I want to be able to cancel my reservation at anytime so that I do not lose all the money if an incident occurs.
Back
Verify that a premium member can cancel the same day without a fee.
Verify that a non-premium member is charged 10% for a same-day cancellation.
Verify that an email confirmation is sent.
Verify that the hotel is notified of any cancellation.
• The product owner's conditions of satisfaction can be added to a story
• These are essentially tests

Banking app User Story Example



The user in this user story example wants to pay their balance, so the first thing the design team might do is start working on a solution that gives them more or less instant access to their credit card balance. Either it should be the first thing they see when they open the app or there should be a clear option to see the balance of that card that's just a tap away.

BBC Sports User Story Example

In this example, the BBC were thinking of adding a share button to their sports articles, with the idea that readers can share sports related news and also get their friend's opinions. It's complete, conveys a lot of information and the logic is simple. It definitely justifies the need for a share button and points at someone having done quite a bit of research into their user base prior to penning this user story.

User Story

As a <role>
I want <goal>
so that <benefit>

Acceptance criteria:
(Conditions of Satisfaction)
…
…

As an Account Manager
I want a sales report of my account to be sent to my inbox daily
So that I can monitor the sales progress of my customer portfolio

Acceptance criteria:
1. The report is sent daily to my inbox
2. The report contains the following sales details: …
3. The report is in csv format.

Samples from a travel website

As a user, I want to reserve a hotel room.

As a vacation traveler, I want to see photos of the hotels.

As a user, I want to cancel a reservation.

As a frequent flyer, I want to rebook a past trip so that I save time booking trips I take often.

User Stories and Technical stories for a Banking system

ID	User Stories
E2F1U1	As a Banking Customer, I want to access/view summary of my savings account, so that I know my balance and other details
E2F2U1	As a Banking Customer, I want to Login to Net banking so that I can view credit card details
E2F4U1	As a Banking Customer, I want to transfer funds within my own accounts so that I can move some balance across my accounts
E2F4U2	As a Banking Customer, I want to transfer funds from my account to another account in another bank, so that I can send money to my family and friends who have accounts in other banks
E2F4U3	As a Banking Customer, I want to add beneficiary to my account, so that I can transfer funds to the beneficiary
	Technical Stories
E2TU1	As a Net Banking Administrator, I want to have the customer's data backed up so that I can restore it any time in case of issues
E2TU2	As a Net Banking application, I want to shake hands with another bank using a specific formatted XML so that funds can be transferred based on the customers' needs

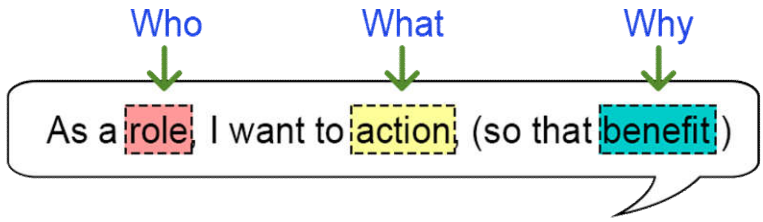
Tableau(Free) ,Excel,Jira, Featmap, Miro user ,Visual paradigm ,Avion are some of the tool for user story mapping

User personas- Part of User Story

A user persona is an archetype or character that represents a potential user of your website or app. In user centered-design, personas help the design team to target their designs around users.

For example: "As a UX Manager, John oversees all the design projects, including assets creation and prototyping efforts, at the design consultancy where he works. He needs easy access to a design tool that allows him to centralize UI libraries so that multiple designers to work simultaneously on a prototype."

User stories are generally created using this format



- As a [**customer**], I want [**shopping cart feature**] so that [**I can easily purchase items online**].
- As an [**manager**], I want to [**generate a report**] so that [**I can understand which departments need more resources**].
- As a [**customer**], I want to [**receive an SMS when the item is arrived**] so that [**I can go pick it up right away**]
- As an employer I want to post job vacancies.

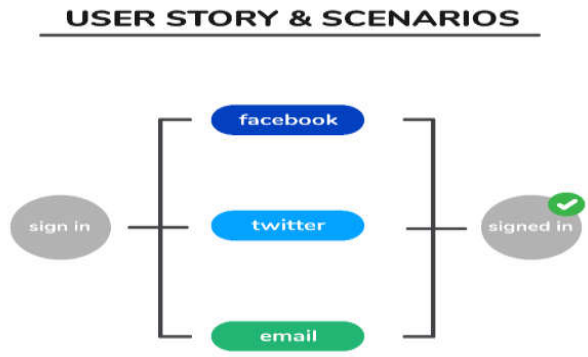
Use case story vs Use case vs Use case scenario

Example 1 :-

For a mobile app sign-in; the **user story will be** User A needs to SIGN IN to access the application . Lets consider the (Who, What, Why) of the same

we developed **the use case:** first, we need to open the app, verify connectivity in the device, and then present the user the options that they must accomplish the task.

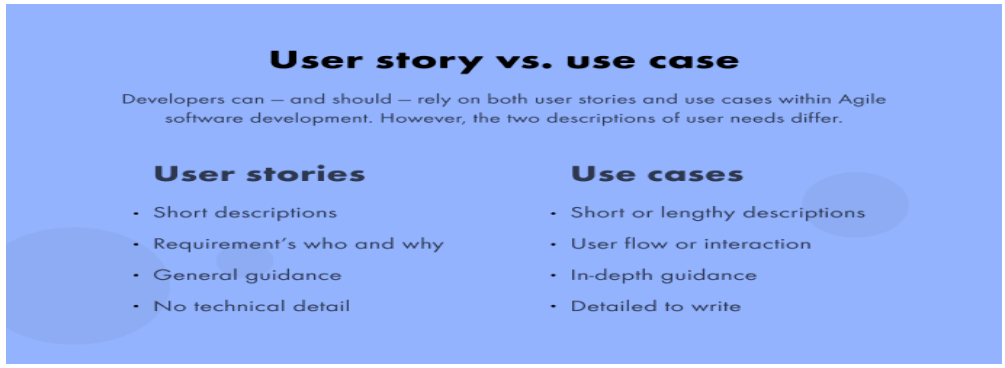
The user can sign in using multiple ways. He can sign-in using our Facebook account, Twitter account, or email. **The use case is sign-in; each of the options to sign in is a use case scenario.** All the situations lead to the same outcome, but as users, we will encounter different interfaces depending on our choice of sign-in.



Example 2: For example, you are a carpenter planning to craft a door. The use case for this scenario would consist of all the steps taken by the carpenter to achieve the goal. This whole documentation would help study the flaws and errors of the process. Collection of Use case scenarios will make a use case for eg use case scenario to open the door, close the door, view through the peephole. User stories would be the door must have an easy interface to open and close, It must be secured, It must look good, It must be durable etc.

Example 3 :- Use Case Scenario for warehouseOnFire

Scenario name	warehouseOnFire
Participating actor instances	bob, alice:FieldOfficer john:Dispatcher
Flow of events	1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her FRIEND laptop. 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appears to be relatively busy. She confirms her input and waits for an acknowledgment. 3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgment and the ETA.



User Scenarios are created for different purposes. It is being created by user researchers to help communicate with the design team.

User stories are created by project/product managers to define the requirements prior to a sprint in agile development.

Use cases are created for developers to help with testing. The difference in target audience means that the structure and information contained in the three approaches also differs.

Scenarios are stories that capture the goals, motivations, and tasks of a persona in a given context.

A scenario will include pictures of the persona, the context, and anything else that contributes to the story. Example :- **Use case scenario**

"Jim, a second-year internal medicine intern at Mount Pleasant Hospital, walks into the room of his patient, Andrew Ross. Since Andrew stayed the night in the hospital, Jim needs to review Andrew's medical records to see if the nurses on the night shift had checked in and recorded any changes in Andrew's condition."

Generated at the beginning of an agile sprint, user stories are brief statements regarding the requirements of the system. in a format **"As a <User Role>, I <<want/need>> <<goal/desire>> so that <<benefit>>."** Use of **"need"** versus **"want"** indicates if a requirement is required or a

"nice to have" when building the system. An example of a **user story** is shown below.

As a doctor, I need to get up to medical date records so that I know how to proceed with my patients' treatment

Example of Use Case

Use case: *Review Records*

Actor: *Doctor*

Steps: *Doctor walks into room*

Doctor sees patient in bed

Doctor identifies patient in bed

Doctor sees medical charts on foot of bed

Doctor gets medical charts from foot of bed

Doctor opens medical charts

Doctor reads medical charts

Doctor changes pages to continue reading

Doctor closes medical chart

In summary, the scenarios, user stories, and user cases are not the same thing although they are used interchangeably. While the context may help determine which of the three approaches is best (scenarios when dealing with researchers, stories in an agile environment, cases for developers), make sure everyone in a design sprint and project group understands the differences to avoid confusion around deliverables.

Practical Questions :-

1) Create a User Scenarios and User Story for Airline Reservation/Railway Reservation system.
2) Create a Storyboard for
   a) University Online Admission Management System.
   b) Steps of Purchasing some items from a shopping center.
   c) Steps of Installing any software on computer machine.
3) For the given list of different scenarios, draw **Series of Sketches** and **A story Board prototypes** for each scenario on the paper by using pen or pencil or can use any tool to accomplish the same
   a) Steps of using ATM Machine for withdraw or deposit of money.
   b) Steps for purchasing online
   c) Architectural Design of Health App.

❖❖❖❖

Module V

5

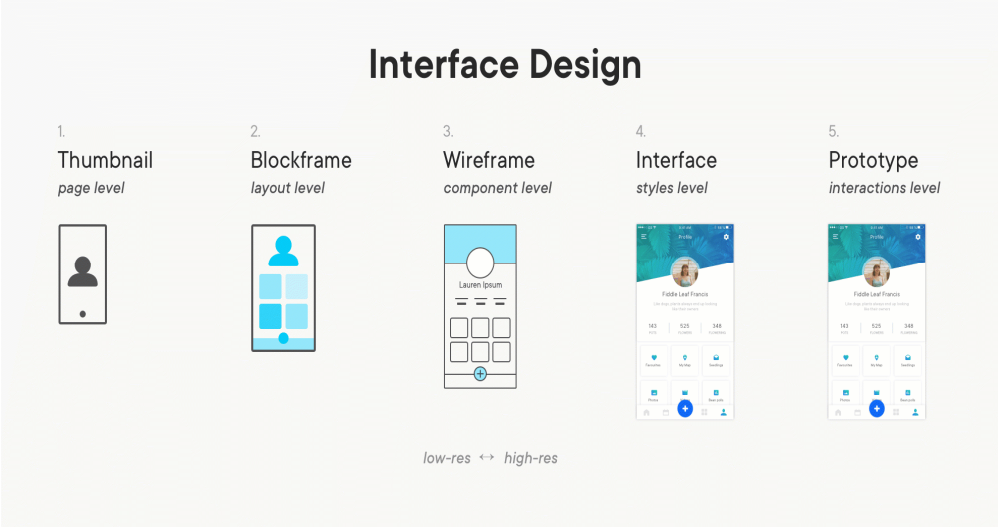
BUILD AND TEST THE LOW FIDELITY PROTOTYPE

Learning Objectives:
To familiarise the learner with different levels of design and their applications.
To introduce them to Figma tool- a tool for prototyping

Learning Outcomes:
The learners would know the applications of different levels of design and accordingly could create it for their project.

UNIT 5- CHAPTER 1- INTRODUCTION TO PROTOTYPE

User Interface Design can be at different level of details as given below: -



Thumbnail: A thumbnail is a miniature version of a larger picture. The thumbnail can illustrate anything graphical: a picture, movie or even a screenshot of a webpage.

Thumbnail for an Audio Mixer Thumbnail for a blog

Blockframe :- Block framing is a minimalistic approach in user interface designing wherein design can be quickly created as block containing various user interface elements. It is essentially a "light wireframe." It doesn't replace wireframing or other highly detailed, functional prototypes—instead, block framing is an effective way to organize and present ideas quickly. Rather than focusing on the details and the data necessary to create your design, blocks and other shapes give you a quick idea of where the data will be and how the data will flow. It follows a minimalist approach in displaying the details.

Wireframe: - A wireframe is a rough sketch about how a website/app will look like. It is usually presented with gray lines, boxes, colours, and placeholders. It is like the blueprint of a building the blueprint of a building which involves a lot of work from many participants to be converted into, So, **it is perfect to be used at the brainstorming or very early design stage.** Wireframes act like the building blocks of each screen and will act as a skeleton for your everything that comes after including visuals, interactions and content Goals of a wireframe includes: -

- To represent the main page contents
- To outline and sketch the page structure and layout
- Display basic website/app UIs

<u>Benefits of Wireframe</u>

Easy to communicate. A wireframe is a good way to communicate. It can reduce the communication barriers between you and your team members.

Sometimes, it is also a good way to demonstrate your design ideas to customers and other stakeholders.

Fast to build. You can easily express your ideas by drawing a wireframe with paper and pens. For better and faster communication, you need not pay much attention to details. It's OK to use placeholders and simple texts to showcase everything.

Low cost. The cost of creating a wireframe with paper and pen can be zero. Even when you use a free wireframe tool, the cost is very low.

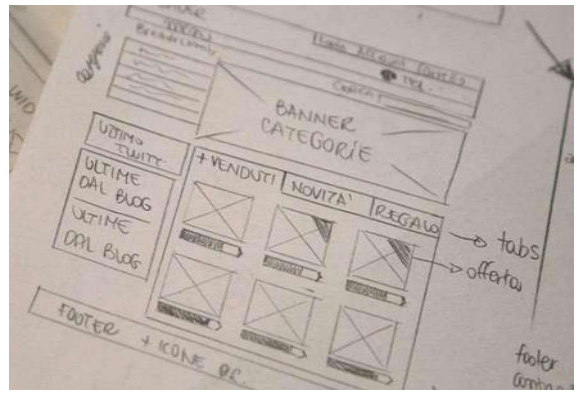
Wireframe is used during the brainstorming session of the project to understand the macro elements of the proposed project.

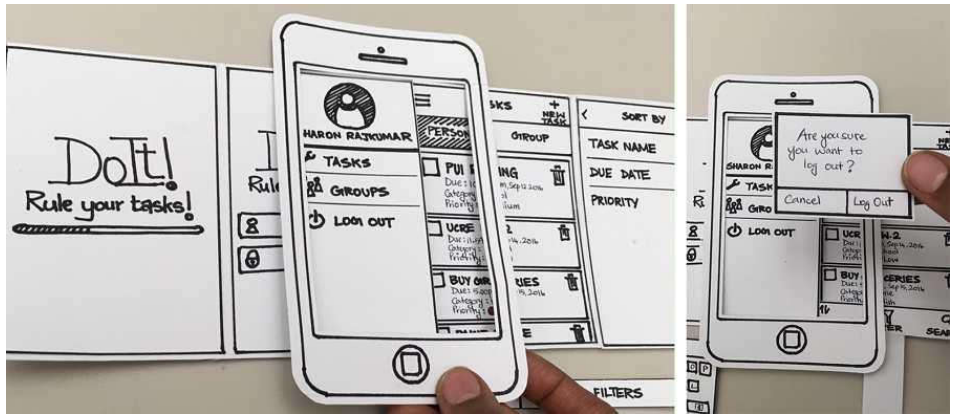
Types of Wireframes

a) Low Fidelity Wireframes

a) High Fidelity Wireframes

- **Low-fi wireframes refer to paper wireframes**
- At the brainstorming stage, many designers love to draw out their web/app interfaces directly on paper, creating a paper wireframe for discussion and interaction.
- Since these sketches usually showcase a part or major parts of a website/app roughly, without many UI details, they are low-fidelity wireframes.



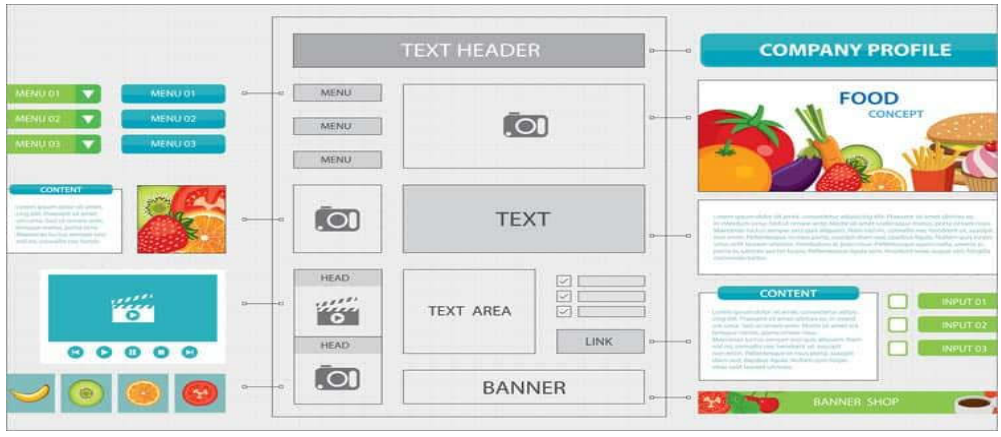


Low-fi wireframes refer to static wireframes made on paper or through the use of wireframing tools . Low fidelity designs are static and does not include any interactions/animations in their design.

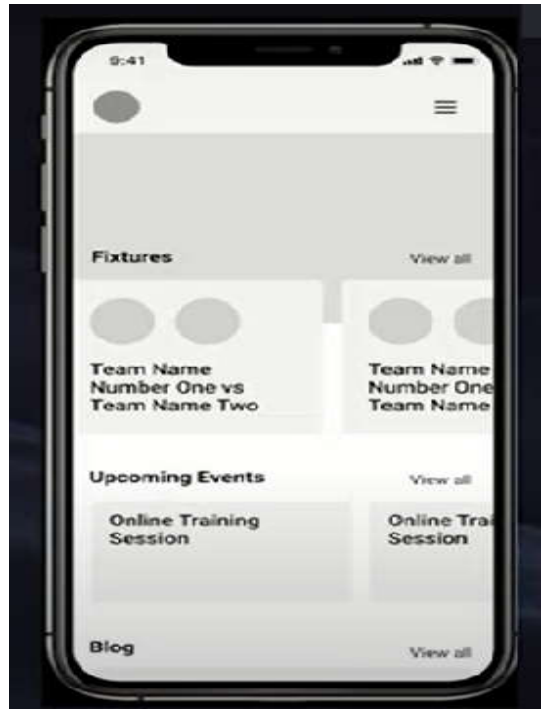
<u>High Fidelity Wireframes :-</u> High-fi wireframes refer to wireframes with more details and simple interactions

To convey the design ideas clearly, many designers add more UI details and simple interactions to their low-fidelity wireframes with a professional wireframe tool, creating high-fidelity wireframes.

- High-fidelity wireframes are usually clickable and simulate a simple interaction of a website/app section.
- For example, many designers use high-fidelity wireframes to quickly visualize the simple interaction flows of the logging-in, checking-out or navigating process.



This high-fidelity wireframe is created to showcase the navigation bar and its simple interaction flow.



Low Fidelity Design
High Fidelity Design

(Template – Field names)
(Actual Values)

Methods to create Wireframe Design

Wireframes can be created on paper or using various tools like Balsamiq, Pencil, Wireframe CC, Figma(Free tool)

Other Wireframe Example:- Table Grid Wireframes

Table name

seq	Product	Category	Period	Sales
1	Tea	cat1	Jan-11	20K
2	Coffee	cat1	Jan-11	15K
3	Milk	cat2	Jan-11	5K
4	Cereal	cat3	Jan-11	30K
5	Chocolate	cat4	Jan-11	25K

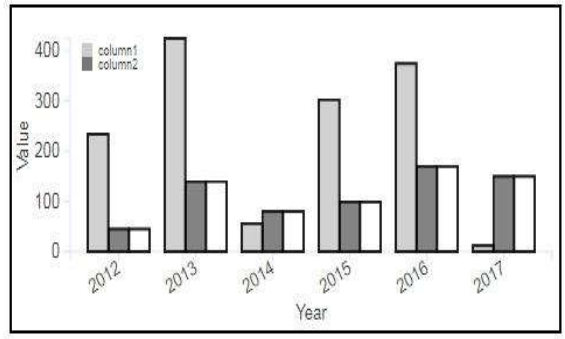
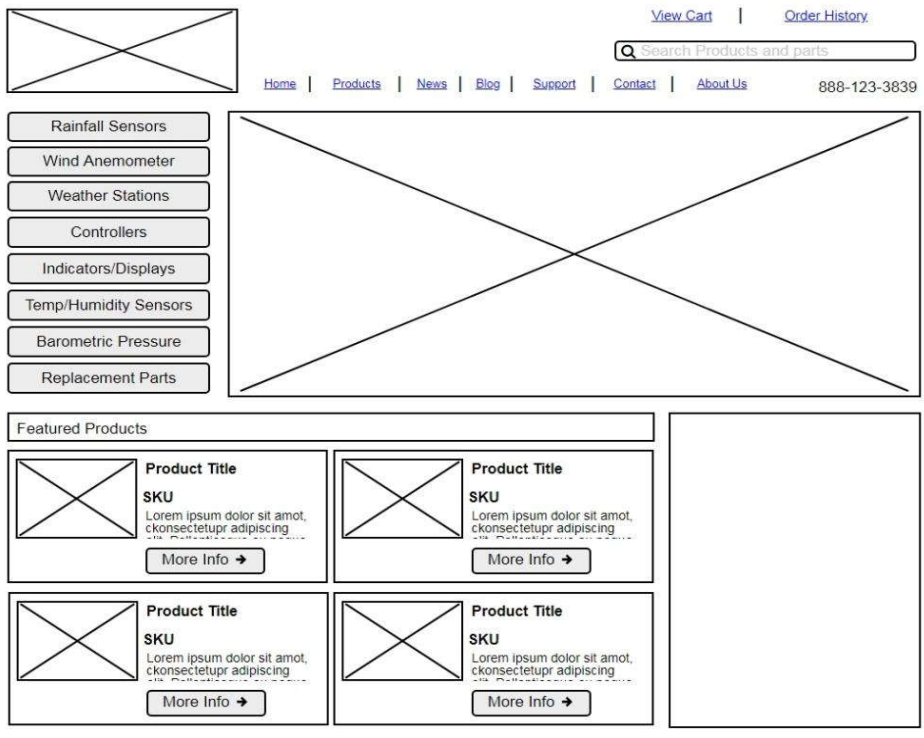


Table name

seq	Product	Category	Period	Sales
1	Tea	cat1	Jan-11	20K
2	Coffee	$23,234.56	Jan-11	15K
3	Milk	cat2	Jan-11	5K
4	Cereal	cat3	Jan-11	30K
5	Chocolate	cat4	Jan-11	25K

Wireframe for Weather condition application

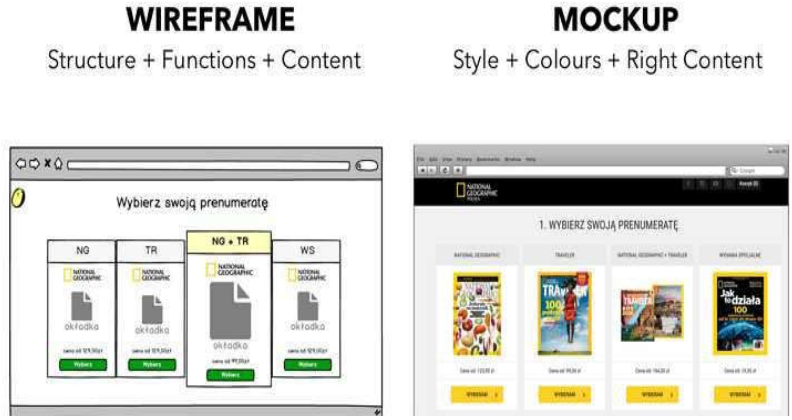


Interface/Mockups

A mockup is a static wireframe with much more UI and visual details. If a wireframe is considered as the blueprint of a building, a mockup is similar to a real-life building model. It gives viewers a more realistic impression of how the final website/app will look like, So, **it is good for communicating, discussing, collaborating and iterating projects with your team members at a later design stage.**

Briefly, unlike wireframes with gray lines, boxes and placeholders, mockups are built with more visual details of the final web/app:

- Rich colors, styles, graphics, and typography
- Actual buttons and texts
- Content layouts and component spacing
- Navigation graphics



Mockup Benefits

1. Showcase rich project details for better communication

Mockups showcase rich project details. It is easy for you and your team to communicate and discuss about a specific detail.

2. Easy to understand for clients and stakeholders

A wireframe may require viewers to use their imagination. However, with a better visual appearance, a mockup makes it easy for anyone, including your clients and stakeholders, to understand and know more about the actual product well.

3. Easy to Preview, test and iterate

Unlike skeletal wireframes, mockups are much closer to the final product.

They are good models for you and your team to preview, test, find errors and iterate them early on.

4. Easy to create

With a good mockup tool, it is easy to create a realistic mockup.

So, a mockup is absolutely good to be used when you need to communicate, discuss, collaborate and iterate projects with your team members

Tools for Mockup Creation

Since mock-ups have included many visual details, you need professional tools to create your desired website/app mockups. Some popular ones are Sketch/Photoshop, Mockflow, Mockplus RP, Balsamiq(free), Marvel app, Mocking bird, Lucidchart(free), Canva(free)

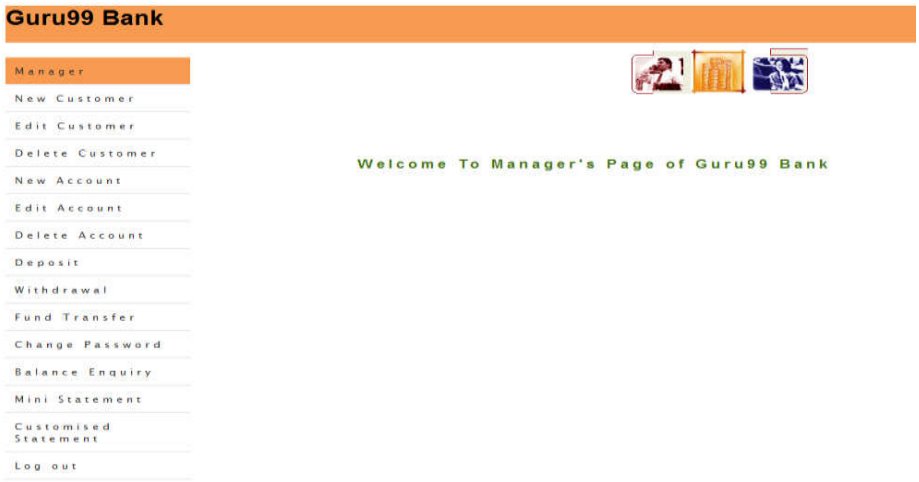
Example : Mockups for Guru99 Bank

LOGIN PAGE

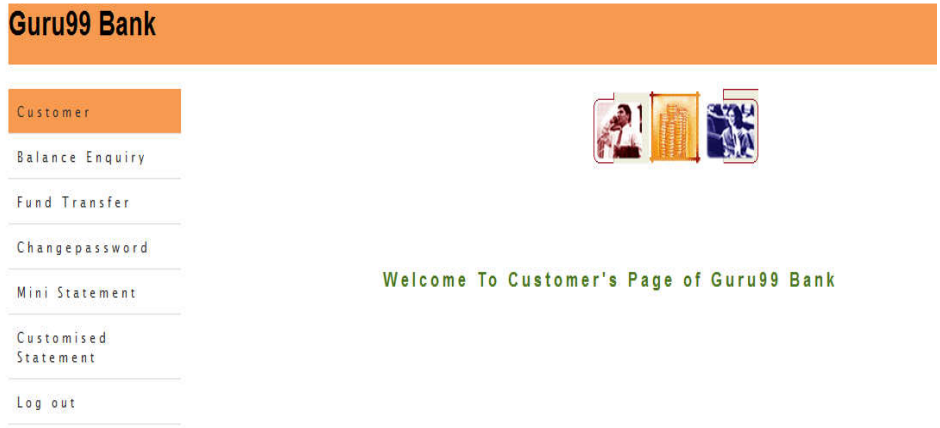
This page will be used for login by Manager as well as Customer

HOME PAGE

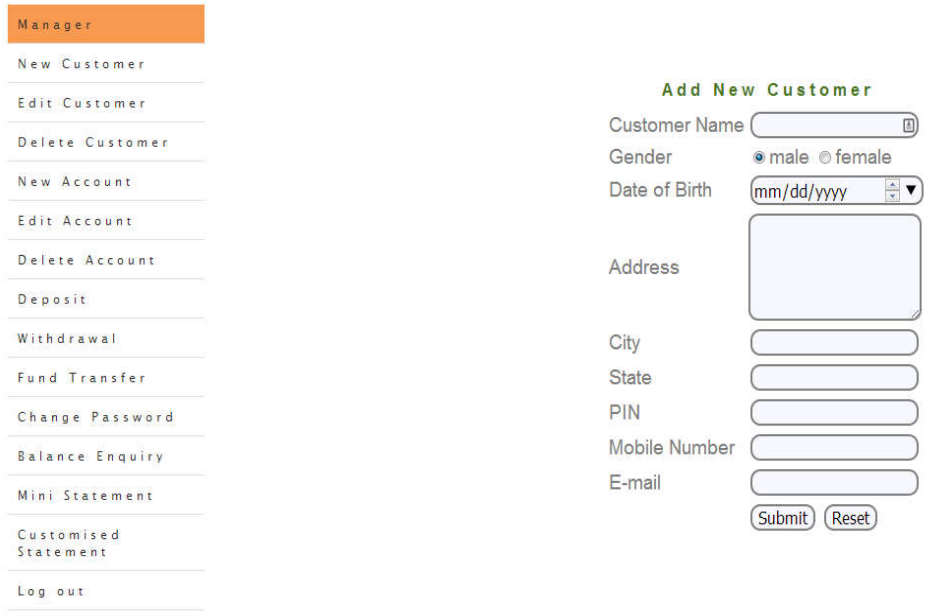
Manager



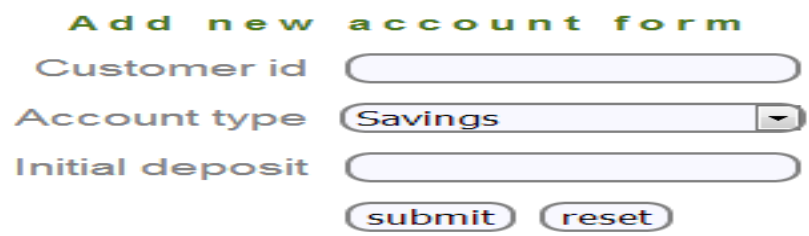
Customer



ADD CUSTOMER FUNCTIONALITY FOR MANAGER



ADD ACCOUNT FUNCTIONALITY FOR MANAGER



Mini Statement Form

Account No

Submit Reset

CUSTOMIZED STATEMENT FOR MANAGER/CUSTOMER

Customized Statement Form

Account No

From Date mm/dd/yyyy

To Date mm/dd/yyyy

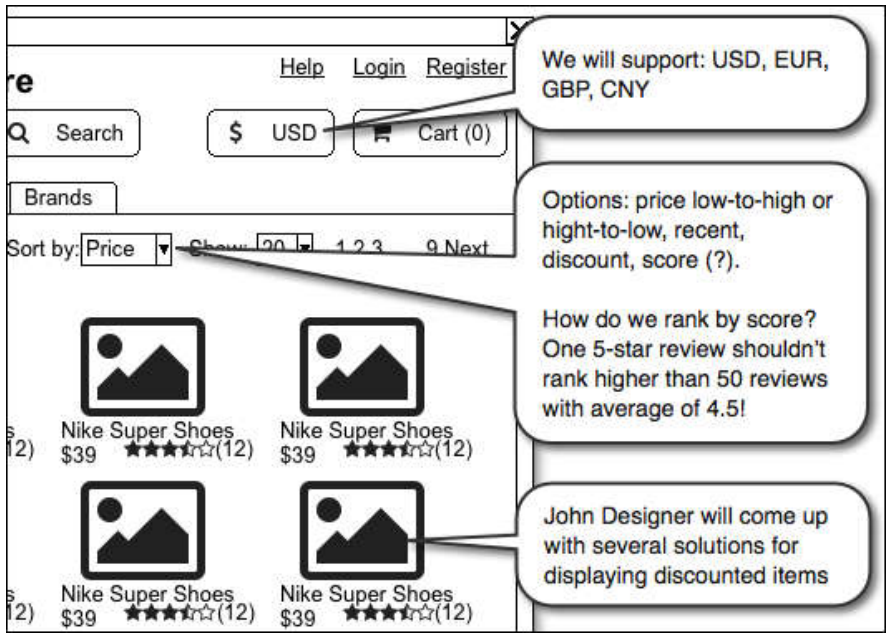
Amount Lower Limit

Number of Transaction

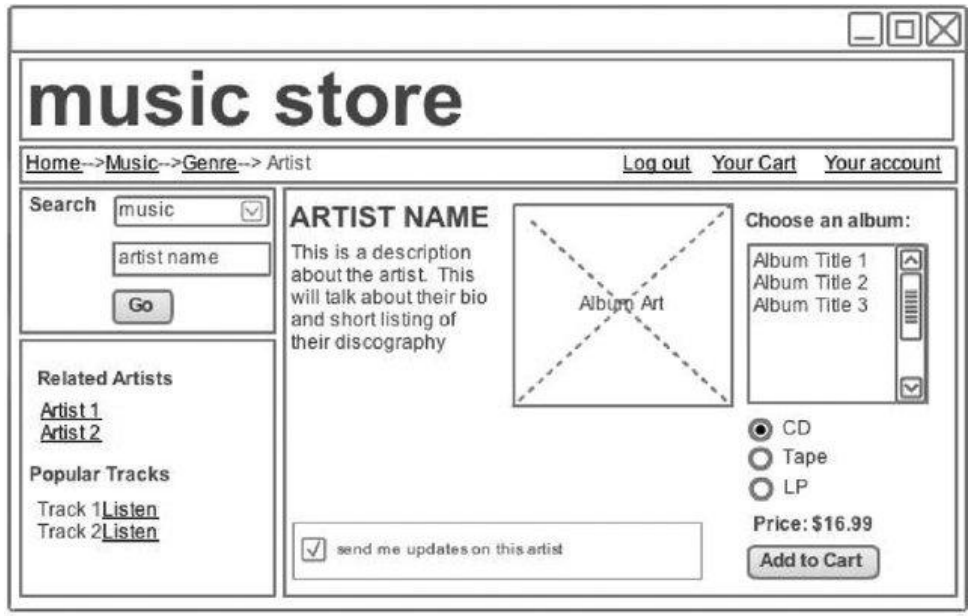
Submit Reset

Mockups are non-interactive. Mockups of every screen page can be created.

Mockup for Shoe shopping page



Mockup for music store portal



CHAPTER 2- INTRODUCTION TO PROTOTYPE

Software prototyping is very useful in the project of developing online application systems that require a high level of user interaction. Systems that require users to fill out forms or view various screens before data is processed can benefit greatly from prototyping to convey the exact look and feel even before the software is developed. Prototyping is usually not an option for software that involves a large amount of information processing, and most of the functionality is internal with little programme. Prototype development can be an additional overhead in such projects and should necessitate a significant amount of additional effort.

Advantages of Software Prototyping:

- Provides a feel to the users on how the proposed system would look like
- Helps in identifying errors if any.
- Prototyping is also considered a risk reduction function because it allows non-existent performance to be seen, lowering the risk of failure.
- Assists team members in effectively communicating.
- Customer satisfaction exists, and he can feel the product from the start.
- There will be no risk of software loss.
- Quick user feedback aids in the development of better software solutions.

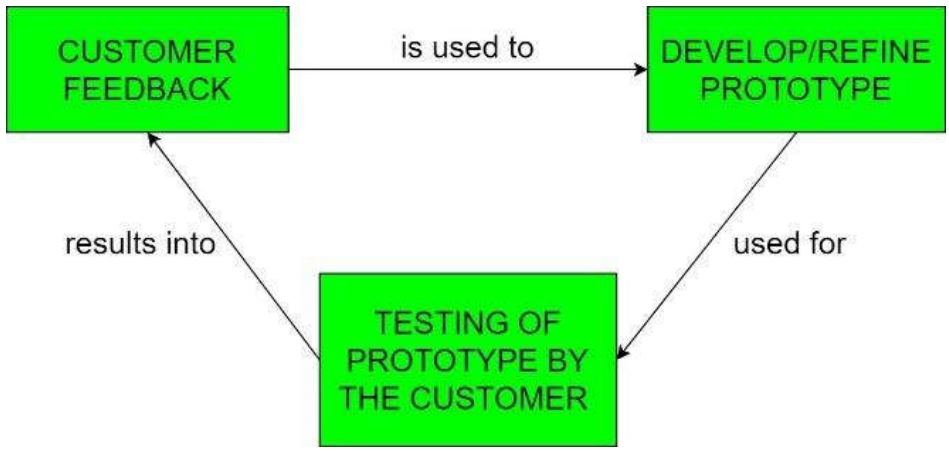
Disadvantages of Software Prototyping:

- Prototyping is a time-consuming and labor-intensive process.

- The cost of creating a specific type of waste is completely wasted because the prototype is eventually discarded.
- Prototyping may result in an overabundance of change requests.
- Customers may be unwilling to commit to the iteration cycle for an extended period of time.
- During each customer test, there may be too many variations in software requirements.

Prototypes are generally interactive

Prototyping offers a small-scale working model of the end product that enables taking early customer feedback.



Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which user are specific and may not have been considered by the developer during product design.

Prototyping Objectives

1) To provide the user basic understanding of how the proposed system would look and behave.
2) It triggers thinking of the end user on what more to expect from the software . This is especially true when a user moves from a manual system to automated. In the initial phases he does not know what to expect from the new system.
3) To have clarity in understanding the requirements
4) The objective of evolutionary prototyping is to deliver a working system to end users. The development starts with those requirements which are best understood.

5) The objective of throw away prototyping is to validate or derive the system requirements where it is difficult to capture.

Types of Prototyping Models

Four types of Prototyping models are:

1. Rapid Throwaway prototypes
2. Evolutionary prototype
3. Incremental prototype
4. Extreme prototype

Rapid Throwaway Prototype

Rapid throwaway is based on the preliminary requirement. It is rapidly developed to show how the requirement will appear visually. The customer feedback is taken and the enhancement of prototype continues till the baseline requirements are captured. In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

Evolutionary Prototyping

Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted. It helps you to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process is time consuming. This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage. The same can then be used with enhancement as an actual system.

Incremental Prototyping

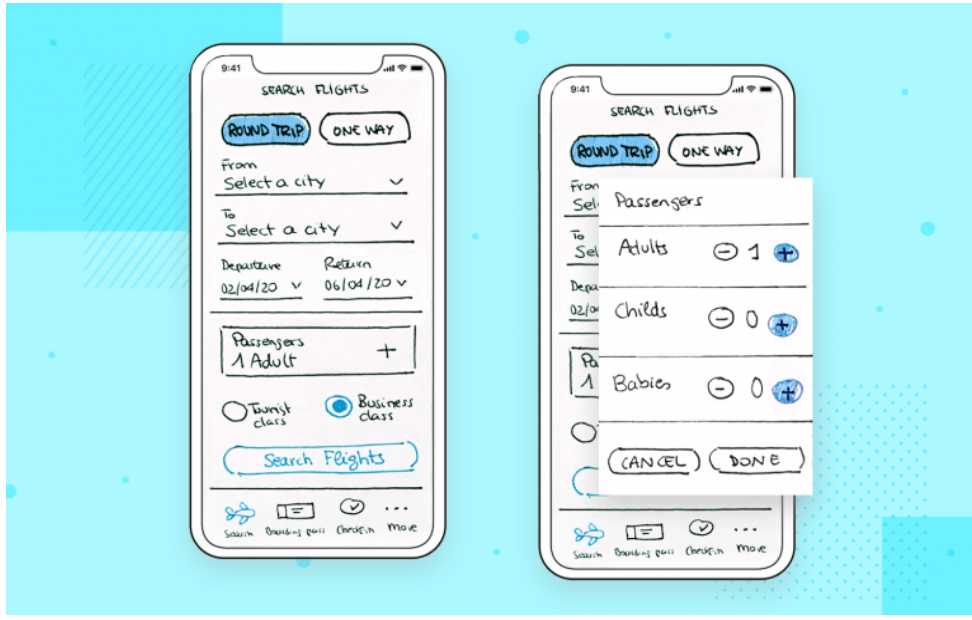
In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is helpful to reduce the feedback time between the user and the application development team.

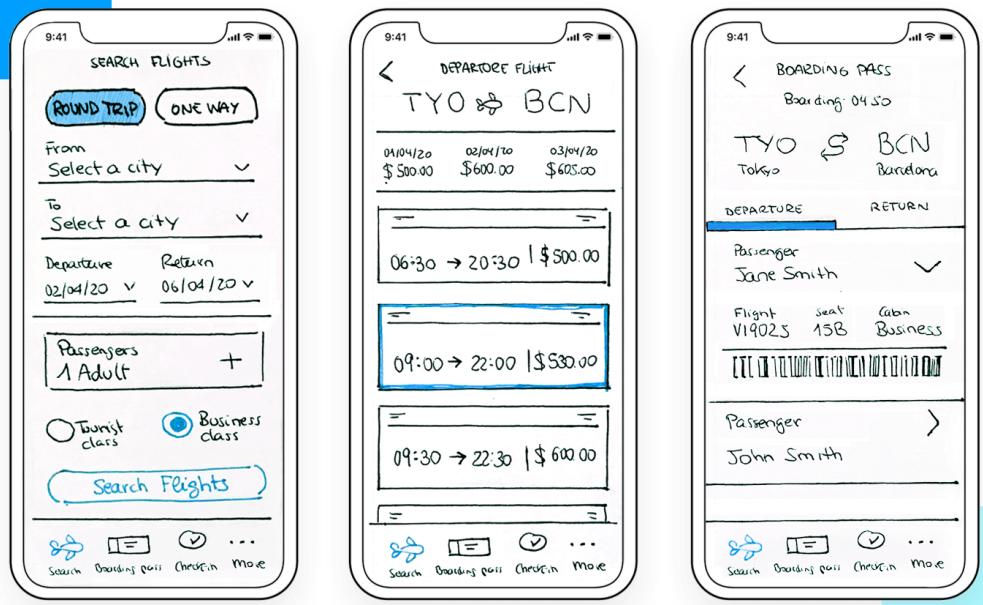
Extreme Prototyping:

Extreme prototyping method is mostly used for web development. It is consisting of three sequential phases.

1. Basic prototype with all the existing page is present in the HTML format.
2. You can simulate data process using a prototype services layer.
3. The services are implemented and integrated into the final prototype.

Paper prototype example- Describing the interaction flow in a Flight Reservation app.





Wireframe vs Prototype vs Mockup

Prototypes not necessarily will be the final product — they have different fidelity. The fidelity of a prototype refers to the level of details exhibited in product design. Fidelity- Low or High differs in the areas of:

- Visual design
- Content
- Interactivity

The prototype's fidelity based on the goals of prototyping, completeness of design, and available resources.

Levels of Prototype

- Low Fidelity prototype
- High Fidelity prototype

Low-fidelity prototyping

Low-fidelity (lo-fi) prototyping is a quick and easy way to translate high-level design concepts into tangible and testable artifacts. The first and most important role of lo-fi prototypes is to check and test functionality rather than the visual appearance of the product.

Here are the basic characteristics of low-fidelity prototyping:

- **Visual design**: Only some of the visual attributes of the final product are presented (such as shapes of elements, basic visual hierarchy, etc.).
- **Content**: Only key elements of the content are included.
- **Interactivity**: The prototype can be simulated as required. During a testing session, a particular person who is familiar with design acts as a computer and manually changes the design's state in real-time. Interactivity can also be created from wireframes, also known as "connected wireframes." This type of prototype is basically wireframes linked to each other inside an application like PowerPoint or Keynote, or by using a special digital prototyping tool such as Adobe XD.

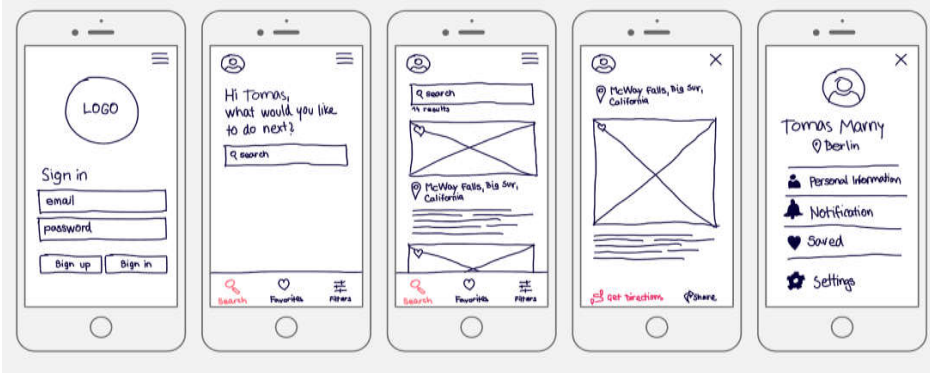
Pros

- **Inexpensive**. The clear advantage of low-fidelity prototyping is its extremely low cost.
- **Fast**. It's possible to create a lo-fi paper prototype in just five to ten minutes. This allows product teams to explore different ideas without too much effort.
- **Collaborative**. This type of prototyping stimulates group work. Since lo-fi prototyping doesn't require special skills, more people can be involved in the design process. Even non-designers can play an active part in the idea-formulation process.
- **Clarifying.** Both team members and stakeholders will have a much clearer expectation about an upcoming project.

Cons

- **Uncertainty during testing**. With a lo-fi prototype, it might be unclear to test participants what is supposed to work and what isn't. A low-fidelity prototype requires a lot of imagination from the user, limiting the outcome of user testing.
- **Limited interactivity**. It's impossible to convey complex animations or transitions using this type of prototype.

Low fidelity prototyping can be done through paper, or any wireframe tool, PowerPoint slides, Adobe XD, Sketch. Paper prototyping is popular with low fidelity prototypes.



High-fidelity prototyping: -

High-fidelity (hi-fi) prototypes appear and function as similar as possible to the actual product that will ship. Teams usually create high-fidelity prototypes when they have a solid understanding of what they are going to build and they need to either test it with real users or get final-design approval from stakeholders.

The basic characteristics of high-fidelity prototyping include:

- **Visual design_: _** Realistic and detailed design — all interface elements, spacing, and graphics look just like a real app or website.
- **Content:** Designers use real or similar-to-real content. The prototype includes most or all of the content that will appear in the final design.
- **Interactivity:** Prototypes are highly realistic in their interactions.

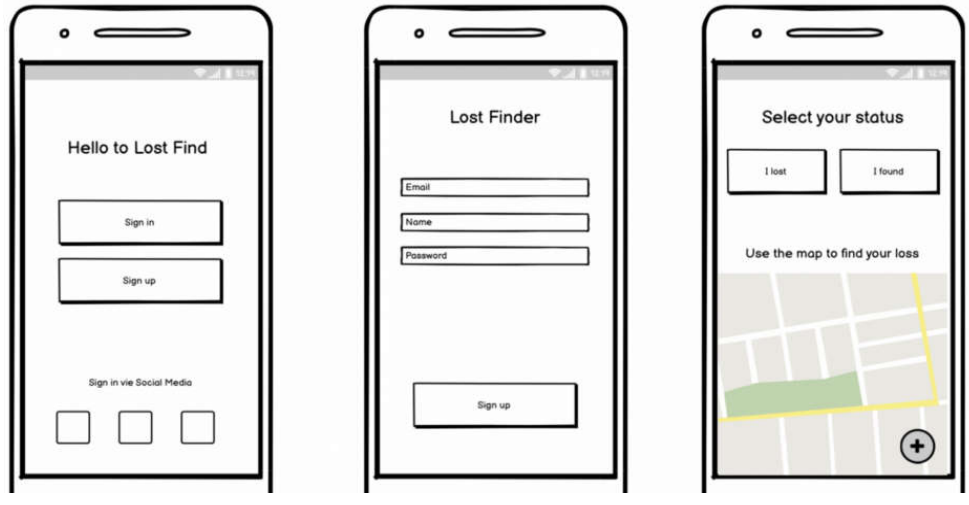
Pros

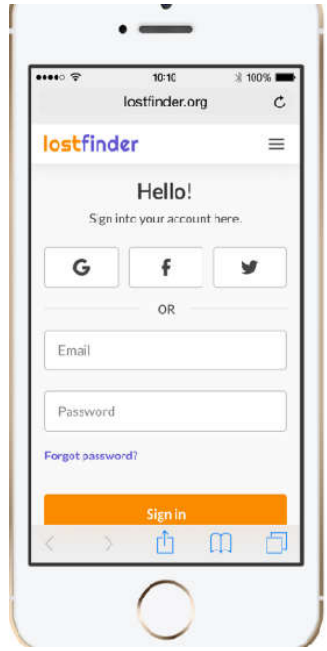
- **Meaningful feedback during usability testing**. High-fidelity prototypes often look like real products to users. This means that during usability testing sessions, test participants will be more likely to behave naturally — as if they were interacting with the real product.
- **Testability of specific UI elements or interactions**. With hi-fi interactivity, it's possible to test graphical elements like affordance or specific interactions, such as animated transitions and micro interactions.
- **Easy buy-in from clients and stakeholders**. This type of prototype is also good for demonstrations to stakeholders. It gives clients and potential investors a clear idea of how a product is supposed to work. An excellent high-fidelity prototype gets people excited about your design in ways a lo-fi, bare-bones prototype can't.

Cons

- **Higher costs**. In comparison with low-fidelity prototypes, creating high-fidelity prototypes implies higher costs, both temporal and financial.

High fidelity prototype involves interactions so they can be created using HTML, Addobe XD, Figm(Free tool), InVision Studio, Webflow, Origami Studio, Sketch ,Marvel etc.





CHAPTER 3- TESTING PROTOTYPE AND USER INTERFACE FUNCTIONALITY

Steps involved in Testing Prototype

a) Creation of Prototype

b) Selecting target users for testing

c) Deciding the location for testing- Office, Remotely

d) Preparing the test scenarios and questions for testing

e) Making the Target users test the system

f) Provide feedback on their experience through questionnaire.

g) Analysis of the feedback

h) Revising the prototype and repeating steps d to f.

Best Practices to achieve maximum feedback on the prototype

1) Solicit Feedback by demonstrating to them the old and the new version of the product if any. Give them the freedom to evaluate and provide their feedback without influencing them. The testing technique will depend on the type of prototype. For instance, if your prototype were a role-playing session, the experience of acting out the roles would be a valuable source of observations and feedback in itself. On the other hand, paper interfaces and physical models might require additional interviews with users to get them to talk about their thinking process while using the prototype.

2) Select the right audience. Try taking feedback from extreme users. If you are working on an idea related to a supermarket, for example, your extreme users could be people who shop at supermarkets every day, and — at the other end of the scale — people who *never* shop at supermarkets. Testing your prototypes on extreme users will often help you uncover some problems and relevant issues that affect regular users, because the extreme users tend to be more vocal about their love (or dislike) of doing things related to your prototype. If your product or service is cross-regional or international, you should also test your prototypes across **regions and countries**. Towards the final stages of your project, you should also get feedback on your prototypes from **stakeholders** other than your users. Internal stakeholders in your company, manufacturers, retailers and distributors will each have their own criteria for building, making or shipping a product or service, and can have an impact on the success of your idea.

3) Ask the right questions :- Every prototype is created with some objective . Frame your questions based on it For For instance, if you have built your prototype to gather feedback about the usability of your product, then you should gear your testing session towards teasing out *how usable* the prototype is to the user. Subsequently, in a post-testing interview session with your user, you should then focus on finding out the positive and negative feedback relating to usability.

4) Be Neutral during testing :- Do not react to the users or defend if negative feedback is received. Stay calm and accept all feedback.

5) Adapt testing:- When you conduct tests on your prototypes, try to adopt a **flexible mindset**. For instance, when you realise that certain components of your prototype are drawing attention away from the core functions of the prototype, you can remove these or change them in order to bring the focus back to the key elements of your idea. In addition, if you think that your planned script for the testing session does not work well, feel free to deviate from it and **improvise during the testing session** in order to get the best feedback from your users.

6) Be open to ideas from stakeholders: - Stakeholders are important to us and thus their feedback . Testers should be open to the ideas received from stakeholders.

TESTING THE FUNCTIONALITY OF THE USER INTERFACE ELEMENTS

User Interface Elements- Text box, Radio Button, Check Box, Drop down list etc.

Demonstration of Test Steps, Test Scenarios and Test Cases from User Scenarios

Test Case Template

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/ Fail
1	Check response when valid email and password is entered	Email: guru99@email.com Password: lNf9^Oti7^2h	Login should be successful	Login was successful	Pass

Examples: -

Test case Id	Scenario Name and description	Input 1 Login id	Input 2 Password	Input 3 Old password	Input 4 New password	Input 5 Confirm password	Expected output	Remarks (if any)
TC1	Scenario 1- Login	Valid input	Valid input	n/a	n/a	n/a	User is allowed to login	–
TC2	Scenario 2- Login alternative flow: Invalid Entry	Invalid input	Valid input	n/a	n/a	n/a	Login id invalid	Login Id is not in the specified format
TC3		Valid input	Valid input	n/a	n/a	n/a	Login id invalid	Login id does not exist in database
TC4		Valid input	Invalid input	n/a	n/a	n/a	Password invalid	Password is not in the specified format
TC5		Valid input	Valid input	n/a	n/a	n/a	Password invalid	Password does not exist in database
TC6		Invalid input	Invalid input	n/a	n/a	n/a	Login id and password invalid	Login id and Password are not in the specified format
TC7	Scenario 3- Login alternative flow: Exit	Valid /Invalid input	Valid /Invalid input	n/a	n/a	n/a	User comes out of the system	–
TC8	Scenario 4- Change password	Valid input	n/a	Valid input	Valid input	Valid input	User is allowed to change password	Password is changed in the database
TC9	Scenario 5- Change password alternative flow: Invalid entry	Invalid input	n/a	Valid /Invalid input	Valid /Invalid input	Valid /Invalid input	Old password invalid	Login Id is not in the specified format
TC10		Valid input	n/a	Invalid input	Valid /Invalid input	Valid /Invalid input	Old password invalid	If old password is not valid, other entries become 'do not care' entries
TC11		Valid input	n/a	Valid input	Invalid input	Valid /Invalid input	New password invalid	Password is not in the specified format
TC12		Valid input	n/a	Valid input	Valid input	Valid input	Confirm password does not match new password	New and confirm password entries are different
TC13	Scenario 6- Change password alternative flow: Exit	Valid /Invalid input	n/a	Valid /Invalid input	Valid /Invalid input	Valid /Invalid input	User is allowed to exit and returns to login screen	–

The use case description of 'maintain school details' use case is given below:

1 Introduction

Allow the administrator to maintain details of schools in the university. This includes adding, updating, deleting and viewing school information.

2 Actors

Administrator

3 Pre-Conditions

The administrator must be logged onto the system before this use case begins.

4 Post-Conditions

If the use case is successful, the school information is added/updated/deleted/viewed from the system. Otherwise, the system state is unchanged.

5 Basic Flow

This use case starts when the administrator wishes to add/edit/delete/view school information.

(i) The system requests that the administrator specify the function he/she would like to perform (either Add a school, Edit a school, Delete a school or View a school).

(ii) Once the administrator provides the requested information, one of the flows is executed.

- If the administrator selects 'Add a School', the **Add a School** flow is executed.
- If the administrator selects 'Edit a School', the **Edit a School** flow is executed.
- If the administrator selects 'Delete a School', the **Delete a School** flow is executed.
- If the administrator selects 'View a School', the **View a School** flow is executed.

Basic Flow 1: Add a School

The system requests that the administrator enter the school information. This includes:

(i) The system requests the administrator to enter the:

1. School name
2. School code

(ii) Once the administrator provides the requested information, the school is added to the system.

Basic Flow 2: Edit a School

(i) The system requests the administrator to enter the school code.

(ii) The administrator enters the code of the school. The system retrieves and displays the school name information.

(iii) The administrator makes the desired changes to the school information. This includes any of the information specified in the 'Add a School' flow.

(iv) The system prompts the administrator to confirm the updation of the school.

(v) After confirming the changes, the system updates the school record with the updated information.

(Contd.)

Basic Flow 3: Delete a School

(i) The system requests the administrator to specify the code of the school.

(ii) The administrator enters the code of the school. The system retrieves and displays the school information.

(iii) The system prompts the administrator to confirm the deletion of the school.

(iv) The administrator confirms the deletion.

(v) The system deletes the school record.

Basic Flow 4: View a School

(i) The system requests that the administrator specify the school code.

(ii) The system retrieves and displays the school information.

6 Alternative Flows

Alternative Flow 1: Invalid Entry

If in the **Add a School or Edit a School** flows, the actor enters an invalid school name/code or the actor leaves the school name/code blank, the system displays an error message. The actor returns to the basic flow and may re-enter the invalid entry.

Alternative Flow 2: School Code Already Exists

If in the **Add a School** flow, a specified school code already exists, the system displays an error message. The administrator returns to the basic flow and may re-enter the school code.

Alternative Flow 3: School Not Found

If in the **Edit a School or Delete a School or View a School** flows, a school with the specified school code does not exist, the system displays an error message. The administrator returns to the basic flow and may re-enter the school code.

Alternative Flow 4: Edit Cancelled

If in the **Edit a School** flow, the administrator decides not to edit the school, the edit is cancelled and the **Basic Flow** is re-started at the beginning.

Alternative Flow 5: Delete Cancelled

If in the **Delete a School** flow, the administrator decides not to delete the school, the delete is cancelled and the **Basic Flow** is re-started at the beginning.

Alternative Flow 6: Deletion not allowed

If in the **Delete a School** flow, a programme detail of the school code exists then the system displays an error message. The administrator returns to the basic flow.

Alternative Flow 7: User Exits

This allows the user to exit during the use case. The use case ends.

7 Special Requirements

None.

8 Associated Use cases

Login

Table 6.9. Test case matrix for the 'maintain school details' use case

Test case Id	Scenario and description	Input 1 School code	Input 2 School name	Edit confirmed	Deletion confirmed	Expected result	Remarks (if any)
TC1	**Scenario 1- Add a school**	Valid input	Valid input	n/a	n/a	School is added successfully	–
TC2	**Scenario 2- Add a school alternative flow: Invalid entry**	Invalid input	Valid/invalid input	n/a	n/a	Invalid school code	School code is not in the specified format. School name becomes do not care.
TC3	**Scenario 2- Add a school alternative flow: Invalid entry**	Valid input	Invalid input	n/a	n/a	Invalid school name	School name is not in the specified format
TC4	**Scenario 3- Add a school alternative flow: School code already exists**	Valid input	Valid input	n/a	n/a	School code already exist	The school with the same code is already present in the database
TC5	**Scenario 4- Add a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	–
TC6	**Scenario 5- Edit a school**	Valid input	Valid input	Yes	n/a	School is updated successfully	–
TC7	**Scenario 6- Edit a school alternative flow: Invalid entry**	Invalid input	Valid/invalid input	n/a	n/a	Invalid school code	School code is not in the specified format
TC8	**Scenario 7- Edit a school alternative flow: School not found**	Valid input	n/a	n/a	n/a	School not found	School with the specified code does not exist in the database
TC9	**Scenario 8- Edit cancelled**	Valid input	Valid input	No	n/a	Main screen of school appears	–
TC10	**Scenario 9- Edit a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	–
TC11	**Scenario 10- Delete a school**	Valid input	n/a	n/a	Yes	School is deleted successfully	–
TC12	**Scenario 11- Delete a school alternative flow: School not found**	Valid input	n/a	n/a	n/a	School not found	School with the specified code does not exist in the database
TC13	**Scenario 12- Delete a school alternative flow: Delete cancelled**	Valid input	n/a	n/a	No	Main screen of school appears	User does not confirm the delete operation
TC14	**Scenario 13- Delete a school alternative flow: Deletion not allowed**	Valid input	n/a	n/a	n/a	Deletion not allowed	Programme of the school exists
TC15	**Scenario 14- Delete a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	–
TC16	**Scenario 15- View a school**	Valid input	n/a	n/a	n/a	School is displayed successfully	The school name with the specified code is displayed on the screen
TC17	**Scenario 16- View a school alternative flow: School not found**	Valid input	n/a	n/a	n/a	School not found	The school with the specified code does not exist in the database
TC18	**Scenario 17- View a school alternative flow: User exits**	Valid / Invalid input	Valid /Invalid input	n/a	n/a	User is allowed to exit and returns to Main menu	–

Test case Id	Scenario and description	School ID	School Name	Edit confirmed	Deletion confirmed	Expected result	Remarks (if any)
TC1	**Scenario 1- Add a school**	101	University School of Information technology	n/a	n/a	School is added successfully	–
TC2	**Scenario 2- Add a school alternative flow: Invalid entry**	1001	*	n/a	n/a	Invalid school code	School code is not of specified length
TC3	**Scenario 2- Add a school alternative flow : Invalid entry**	101	12univ	n/a	n/a	Invalid school name	School name is not in the specified format i.e. it contains digits in the beginning
TC4	**Scenario 3- Add a school alternative flow: School code already exists**	102	University School of Management Studies	n/a	n/a	School code already exists	Entry with the same school code already exists in the database
TC5	**Scenario 4- Add a school alternative flow: User exits**	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	–
TC6	**Scenario 5- Edit a school**	102	University School of Management Studies	Yes	n/a	School is updated successfully	–
TC7	**Scenario 6- Edit a school alternative flow: Invalid entry**	101	univ	n/a	n/a	Invalid school name	School name is not in the specified format which is less than 10 characters
TC8	**Scenario 7- Edit a school alternative flow: School not found**	103	n/a	n/a	n/a	School not found	School code does not exist in the database
TC9	**Scenario 8- Edit cancelled**	101	University School of Information technology	No	n/a	Main screen of school appears	User does not confirm the edit operation
TC10	**Scenario 9- Edit a school alternative flow: User exits**	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	–
TC11	**Scenario 10- Delete a school**	101	n/a	n/a	Yes	School is deleted successfully	–
TC12	**Scenario 11- Delete a school alternative flow: School not found**	103	n/a	n/a	n/a	School not found	School code does not exist in the database
TC13	**Scenario 12- Delete a school alternative flow: Delete cancelled**	102	n/a	n/a	No	Main screen of school appears	–
TC14	**Scenario 13- Delete a school alternative flow: Deletion not allowed**	102	n/a	n/a	n/a	Deletion not allowed	Programme of the school exists

	allowed						
TC15	Scenario 14- Delete a school alter-native flow: User exits	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	–
TC16	Scenario 15- View a school	101	n/a	n/a	n/a	School is displayed suc-cessfully	–
TC17	Scenario 16- View a school alternative flow: School not found	103	n/a	n/a	n/a	School not found	School code does not exist in the database
TC18	Scenario 17- View a school alternative flow: User exits	*	*	n/a	n/a	User is allowed to exit and returns to Main menu	–

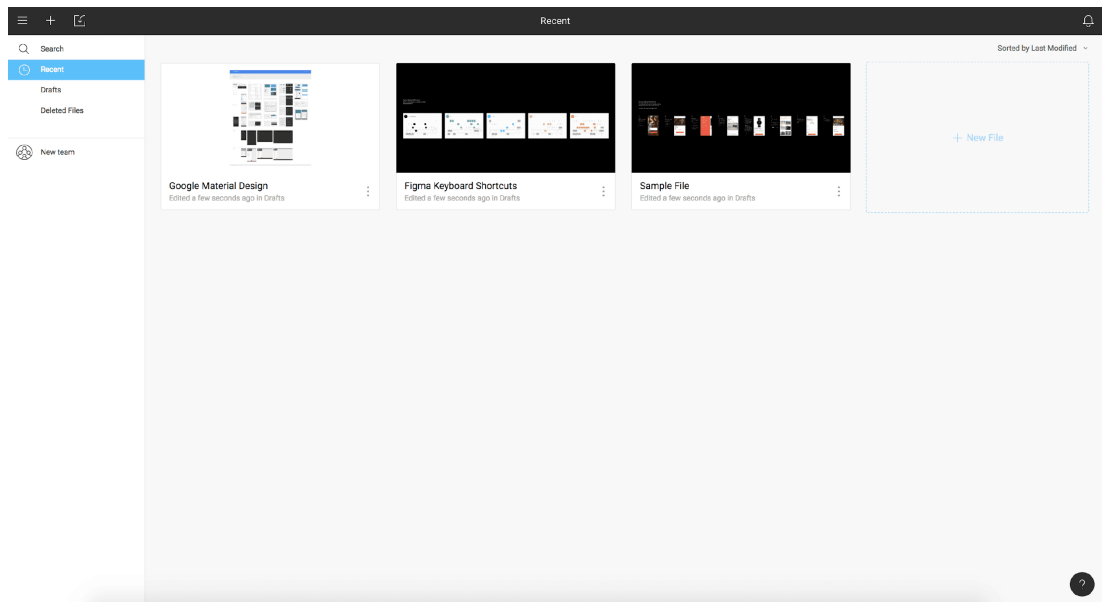
*: 'do not care' conditions (valid/invalid inputs)
n/a: option(s) that are not available for respective scenario

CHAPTER 4- Introduction to Figma- Tool for Designing Wireframe, Prototype

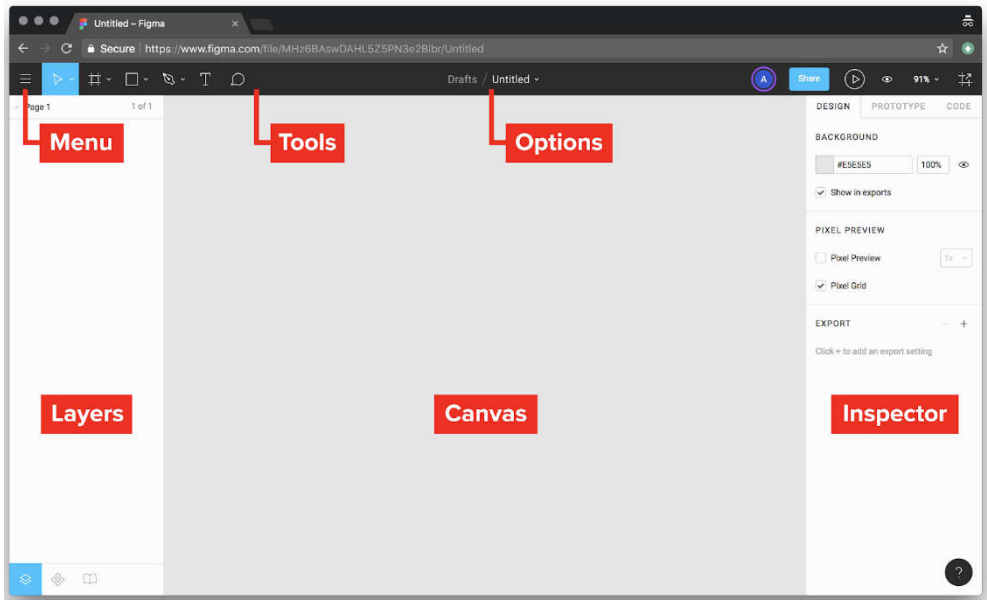
Figma is called **the collaborative interface design tool**. It gives users the ability to share a design file with multiple team members and get instant feedback from each other via comments. Figma also provides a lot of useful resources, plugins, and techniques that make your workflow smoother.

Begin by Setting up a Figma Account

To begin with Figma visit www.figma.com, website and "Sign up", and entering your details. Once you've done that, Figma will open up with a start screen like this. Click on "New File" and we'll get started!



2. Take a look around the Figma interface



The look and feel of the Figma interface is quite minimal, but it belies a set of powerful features. Here's an explanation of the interface's main areas (labelled above):

Menu:

Unlike regular desktop design apps, Figma's menus can be found by clicking the hamburger button in the top-left of the screen. Take a minute to browse around these menus and see what's there! You can also search for the specific command you need. Start typing in "rectangle" and you'll quickly find the Rectangle Tool, complete with a handy reminder of its keyboard shortcut (it's R, by the way).

Tools:

Here you can quickly access the tools you're likely to use most often: frames, shapes, text, etc. (We'll cover all these tools in the next couple of days!)

Options:

This area shows extra options for whichever tool you have selected. When no object is selected (as shown above), Figma displays the file name. When an object is selected, contextual options appear here.

Layers:

Where every element in the file is listed, organized into Frames and Groups.

Canvas:

This is where you create and review all your work.

Inspector:

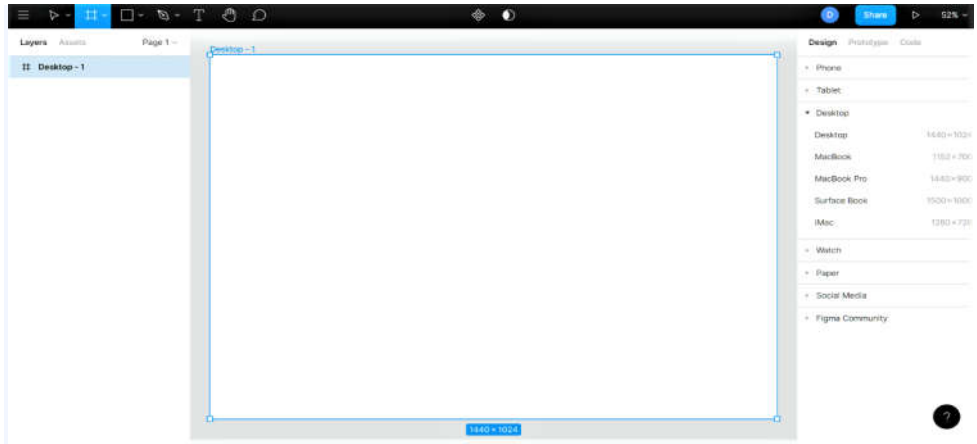
The Inspector shows contextual information and settings for whatever object is selected. In the image above, we're seeing options for the Canvas itself. Note that Figma gives us separate tabs in the Inspector (Design, Prototype, and Code)—we'll cover these later in the week.

Frame

The frame tool is what allows you to create frames on the canvas. You can nest frames inside each other as well. Let's start by adding a frame.

- To create a frame, either click on the Frame tool in the toolbar or press one of the following keys - F or A

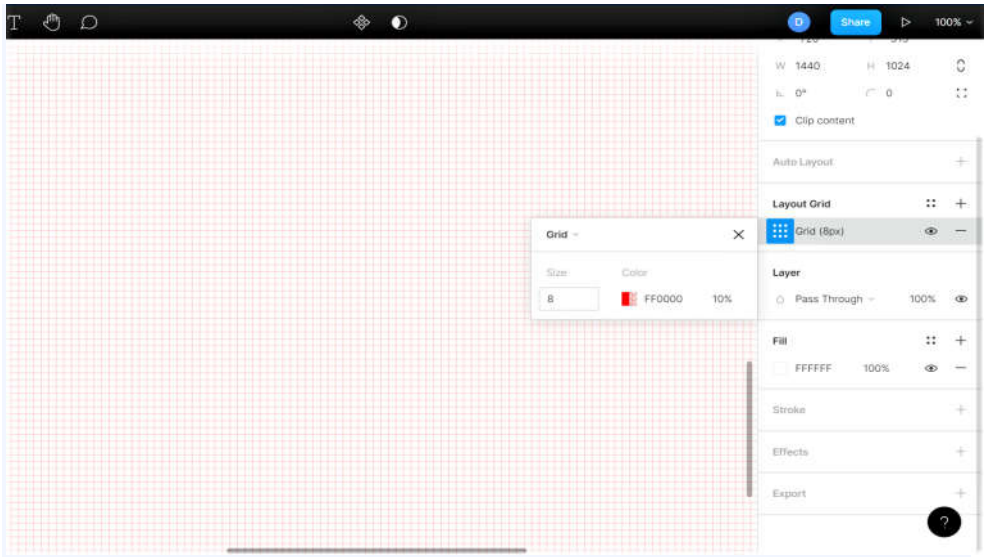
- Once you click on the frame tool, you can either choose one of the preset devices from the Properties panel or draw a custom one on the canvas



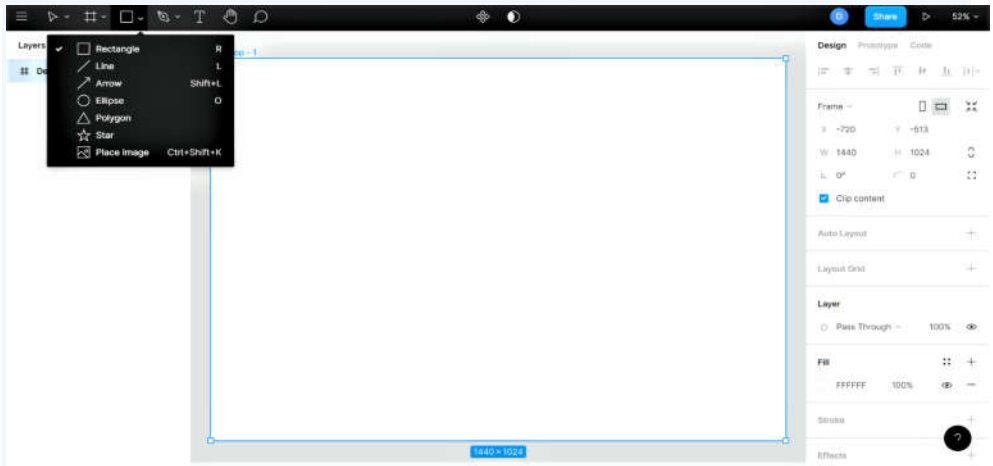
Practicing zooming in and out
The standard zoom commands are accessed with □+ and □-.
Change the font :-With the text layer selected, you can access settings in the Inspector to change the font, as well as font size, weight, and color. We've stuck with Roboto, but switched up to bold and uppercase text.

Create a Text layer :-Click T to access the text tool.

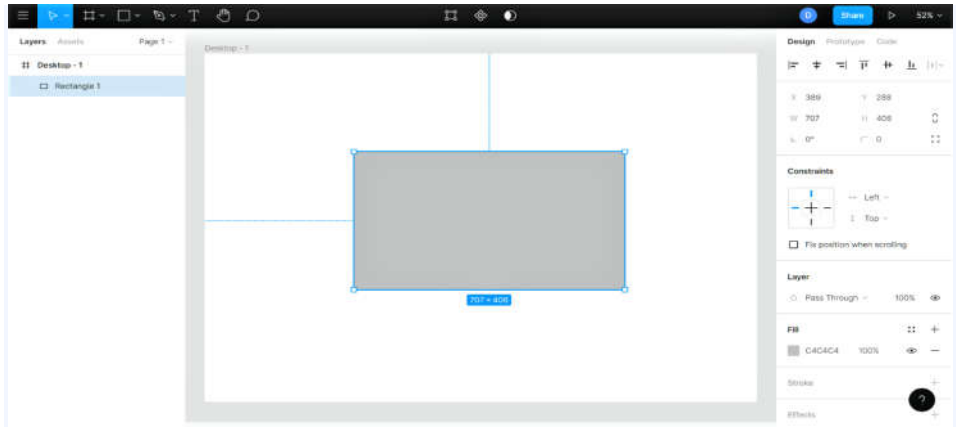
Grids :-Grids are essential to understanding the negative spacing when you're designing for iOS, Android and Web. For mobile, it's common to use an 8-point grid. For Web, it's a little less about spacing and more about division, like the 960 grids. It's good to have some grid in place, like the default 10-point grid. This ensures that all the elements fall on clean pixels, avoiding half pixels as much as possible.



Shape Tools :- You can access all the basic ones via the Shape tools in the toolbar. The shapes available include - Rectangle, Line, Arrow, Ellipse, Polygon and Star. To insert a shape, do the following:



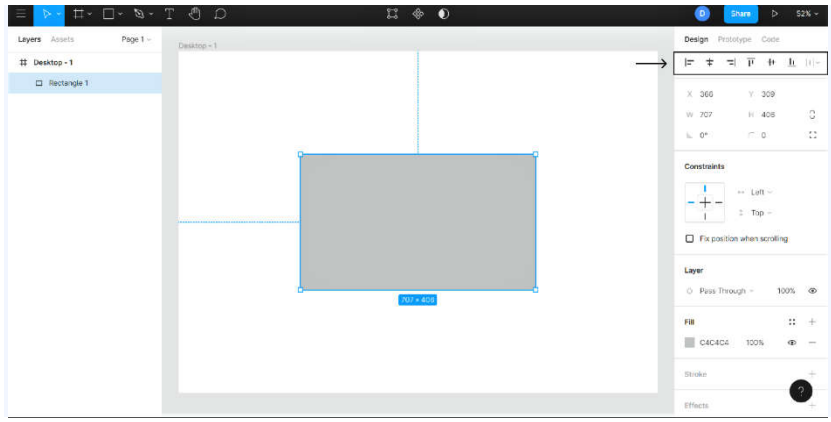
- Here, we'll be adding a rectangle, but you can add any shape you'd like



Alignment

The top of Properties panel contains various alignment options. Provided below is the list of all the options as well as their keyboard shortcuts:

- Align Left (Alt + A)
- Align Horizontal Centers (Alt + H)
- Align Right (Alt + D)
- Align Top (Alt + W)
- Align Vertical Centers (Alt + V)
- Align Bottom (Alt + S)



Resize

You can resize any layer by selecting it and then, dragging the corners. Hold Shift to keep the same aspect ratio and hold Option to resize from center in order to keep the same alignment. Alternatively, you can change the 'W' (width) and 'H' (height) values in the properties panel. Make sure to click the Constraint Proportions option so that it scales proportionately.

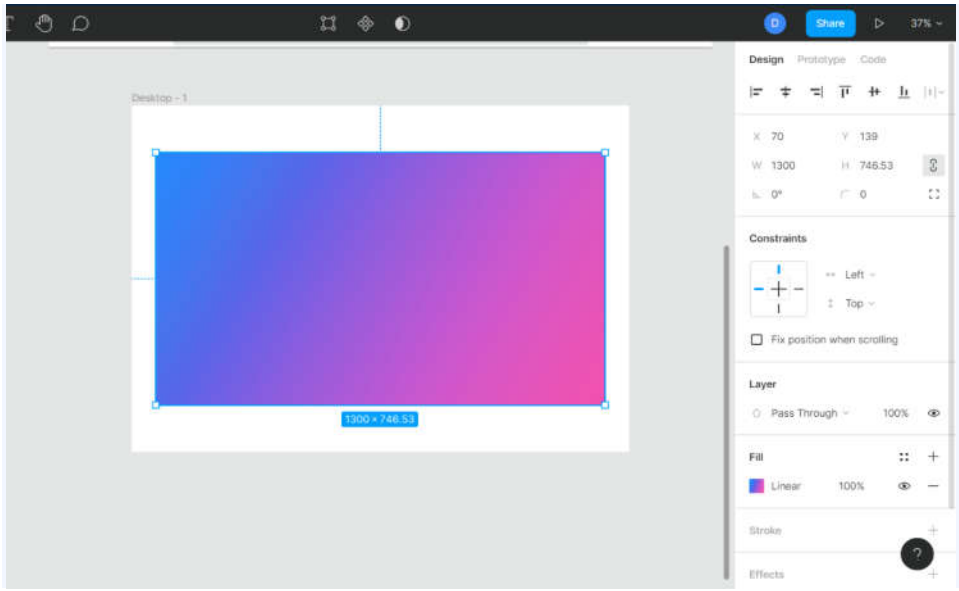
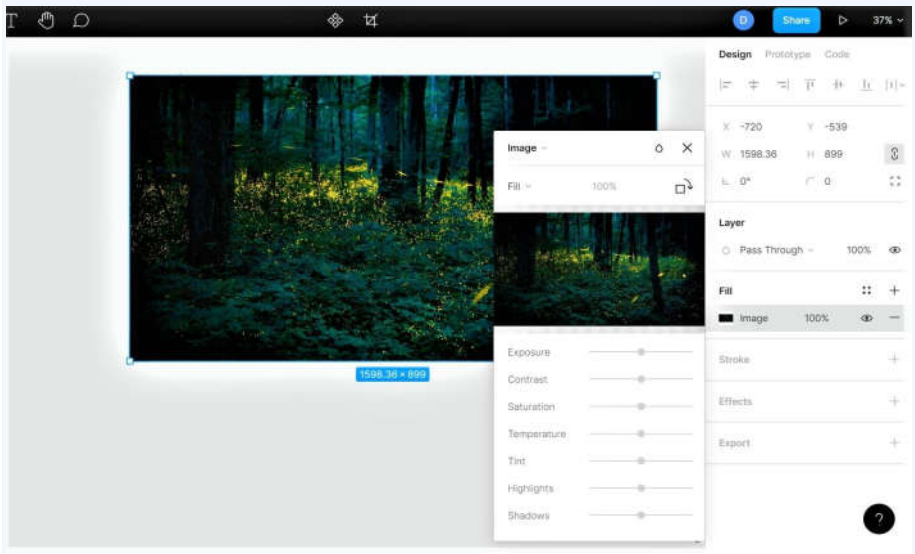


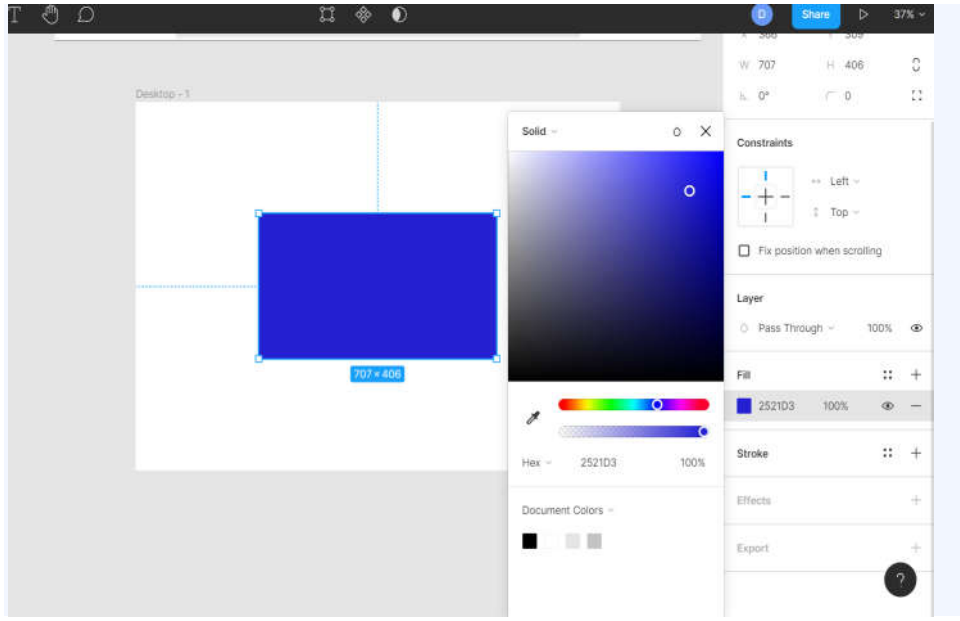
Image:-

In Figma, images can be edited on the fly after importing. You can control settings like Exposure, Contrast, Saturation and much more without having to leave the design tool.



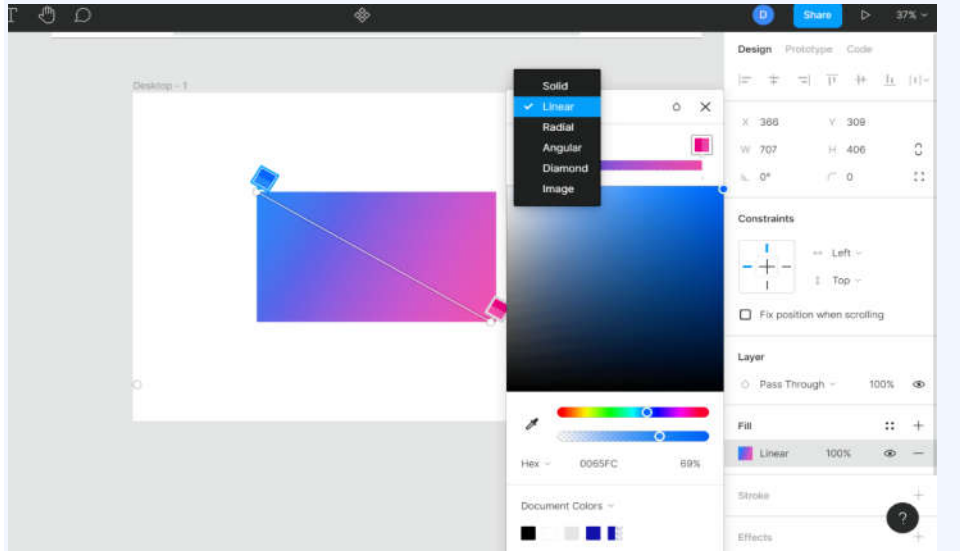
Color Picker

Click on the Fill to start customizing the background. As with most design tools, you can change the color by clicking the color on the wheel or via the eyedropper tool. Alternatively, you can input the HEX code or choose from one of the preset colors. The opacity value can also be changed from here.



Gradients

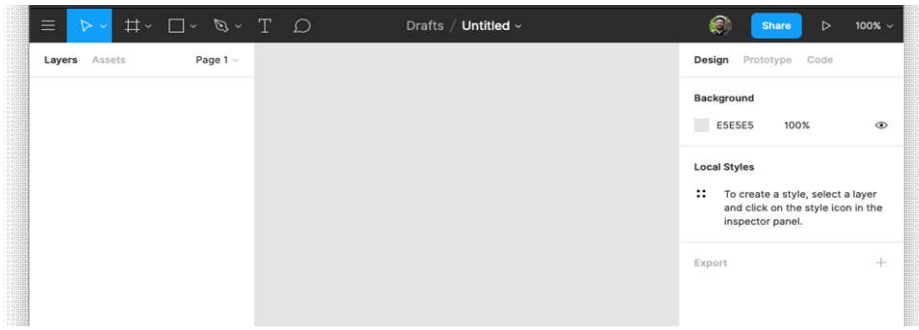
Figma also gives you the option of replacing the solid colors with gradients. There are multiple gradient options. You can find the gradient options in the color picker. You can play around with different combinations of colors, the opacity as well as the direction of the gradient!



Practical : To create a calculator using Figma Tool

- Go to https://www.figma.com and create an account
- Select the free tool option and then click on New Design.
- Create a new project using the + icon in the upper left corner and create a new project
- The new project screen will appear, like so:

The blank project screen looks like the image on the left. There are 4 parts



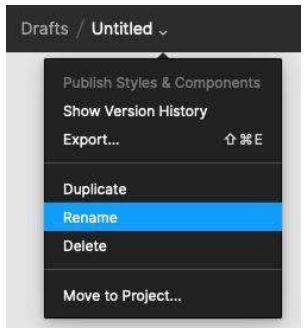
to the Figma UI:

The opening screen consist of the following components: -

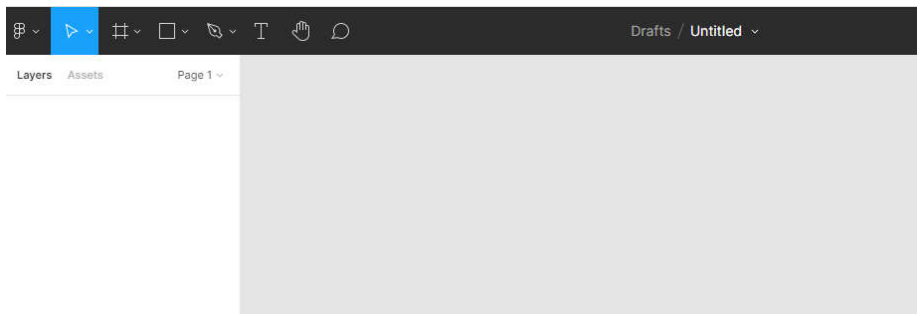
1. the toolbar (top), containing different design manipulation tools;
2. the layer list (left);
3. the inspector (right); and
4. the canvas (middle).

Notice that the inspector has 3 tabs: design, prototype

Click the arrow in the top-middle of your screen to rename your project to



"Calculator":



1. Frame

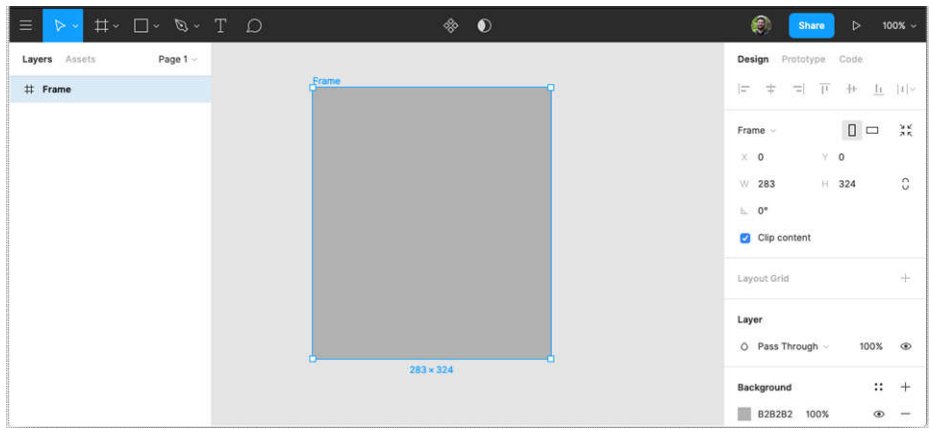
Select the "Frame" tool:

Create a frame for the calculator screen by dragging on the canvas to draw a rectangle. This frame is the main background for our calculator onto which all the elements will be placed. Don't worry about the exact size or position yet; we'll set that in the next step.

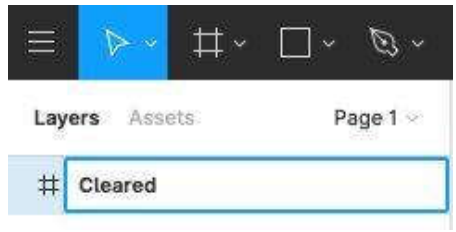
(General note: if you make a mistake, control-Z (Windows) / command-Z (macOS) works. Add a shift modifier to redo something you undid.)

2. Using the inspector

Verify that the default "Move" tool is selected. If the frame that you created isn't already selected, click on it to select it. In the inspector (right side), ensure that the "Design" tab is selected. Using the inspector, change the width of the frame to 283 and the height to 324. Also change the background color to #B2B2B2. Finally, set the X and Y each to 0, and scroll the canvas to find your frame again if necessary.



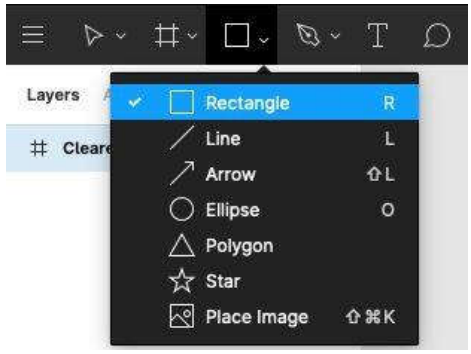
3. Layer names



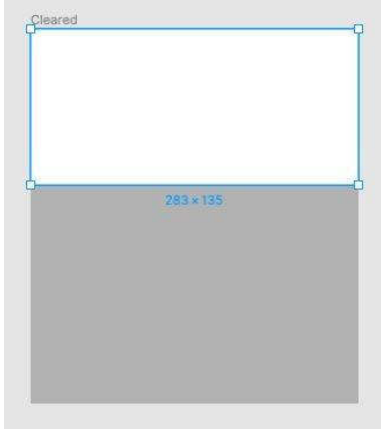
In the layer list (left side), double click on the frame that you created and rename it to "Cleared". This is the name we will use for this screen. By the end of the tutorial, we will create several screens (frames) for different states of the application and connect them together into a clickable prototype, so naming the frame helps keep things straight. "Cleared" means the initial, empty state of the calculator, before any buttons are pressed, or else after the "C" button is pressed.

4. Adding the calculator display

Select the Rectangle tool from the toolbar:

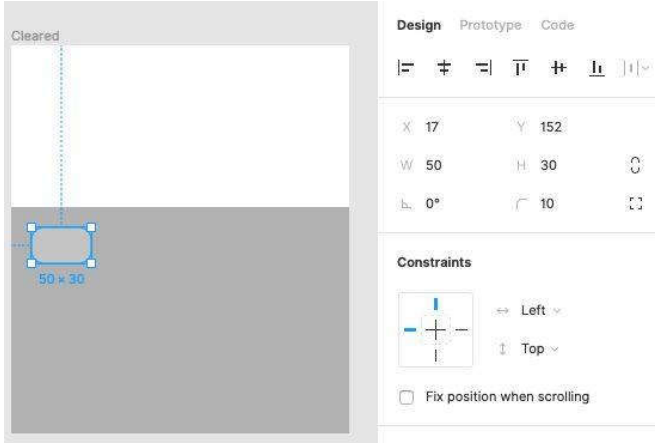


Drag an area on the canvas to add a rectangle. This will be the screen of the calculator to display numbers, operations, and results. Using the inspector, set the rectangle's X to 0, Y to 0, width to 283, height to 135, and color to #FFFFFF (i.e. white).



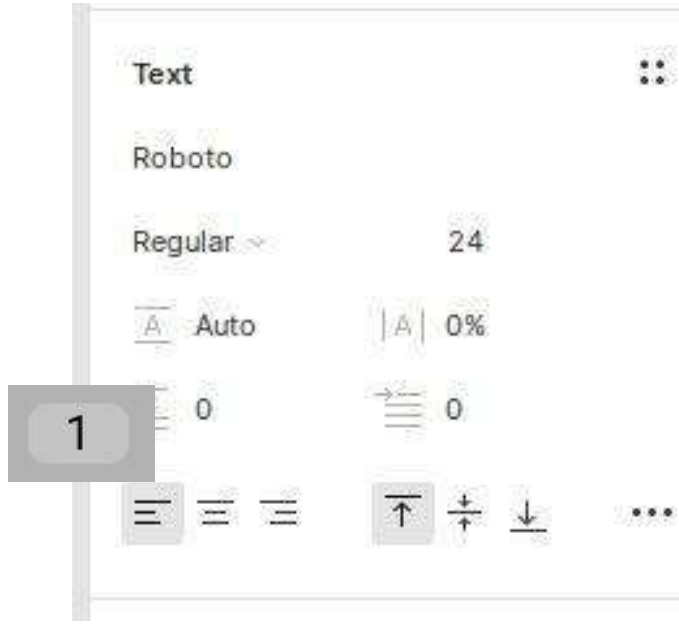
5. The first button

Add a smaller rectangle for the calculator buttons. Start by guessing a size and location similar to the one shown to the left, then use the inspector to fine-tune. Horizontally drag the width and height values in the inspector to set the width to 50 and height to 30. Use your arrow keys to set X to 17 and Y to 152. Set the color to #C4C4C4. Set the border radius to 10.



6. Select the text tool from the toolbar.

Click on the button you just created to add a text label, then type ‘1’ and hit escape. Either using the inspector or the canvas, make it the same size (width/height) and position (X/Y) as the button. Using the inspector, set the label’s horizontal alignment to center, vertical alignment to middle, font to Roboto, and size to 24, as shown to the right.

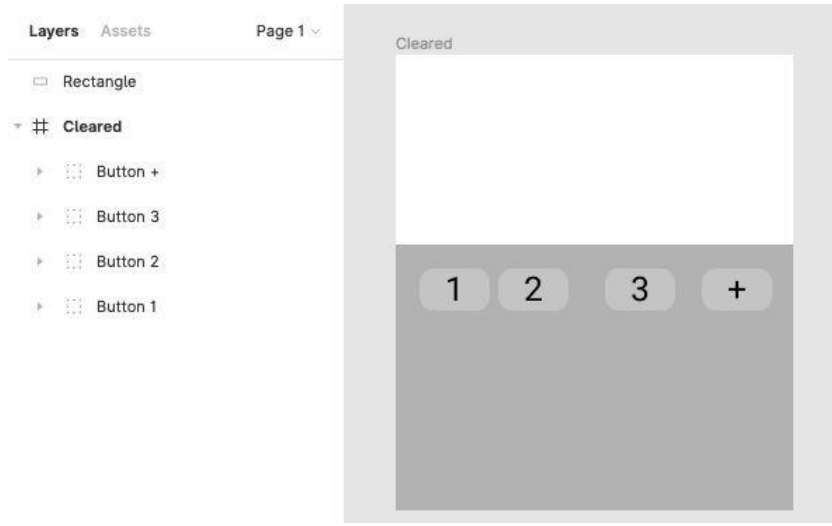


When done, your button and label should look like this:

7. Grouping elements

Find the button and the label in the layer list (left side). Select them both by holding control (Windows) or command (macOS) while clicking. Right click the selection and select “Group Selection”. Double click on the new group in the layer list to rename it to “Button 1”.

8. Duplicating elements



Select the group (not the button or label individually), then hit control-D (Windows) / command-D (macOS) to duplicate the group. Move the new group to the right by clicking and dragging the element in the canvas. Don't worry about getting the position just right yet, but put them roughly in a horizontal row. Rename the group "button 2", and change the text of the label to 2. Repeat to create a new button 3 and a new button +.Positioning elements

You may have noticed that dragging an element in the canvas pops up various alignment and spacing helpers. These are useful, but there are more powerful tools to use to get things just right.

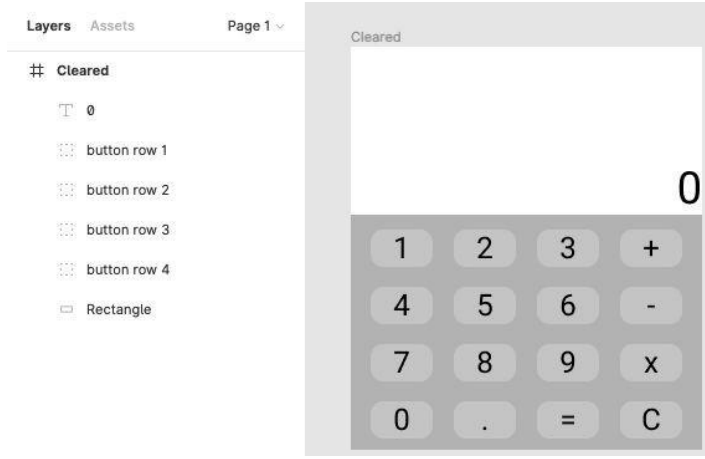
Select the 4 buttons. (Hint: use shift-click to add to the current selection.) If they're not vertically aligned, select "align vertical centers" under the "arrange" menu (see the "hamburger" icon in the top-left for the menu). Move them as a set to align the set to the center of the frame. Then use the menu to "distribute horizontal spacing" (under "arrange"). If the buttons feel too close or too far apart, move a button on either edge, re-align and re-distribute. Don't worry about matching the image here exactly; just do what feels right to you.

9. Duplicating rows

Apply your learning so far to fill out the remaining 3 rows of buttons. (Hint: group the buttons in a row in order to distribute rows vertically.)

10. Add the display label

In the initial state, our calculator displays 0. Add a text element 0 to the display screen, sized the same as the rectangle. Its alignment should be bottom-right, and its font should be Roboto, size 36. Group it with the display rectangle.

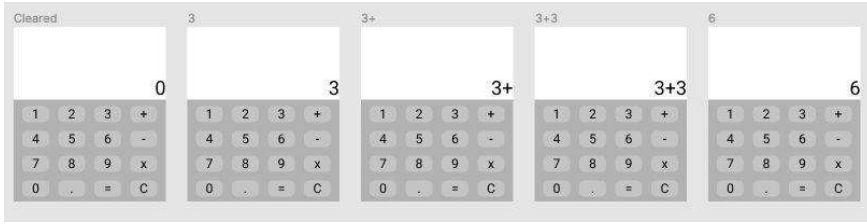


Note: if the text doesn't appear, you may have to move the display rectangle element down in the layer list to be after the text element.

11. Create other screens

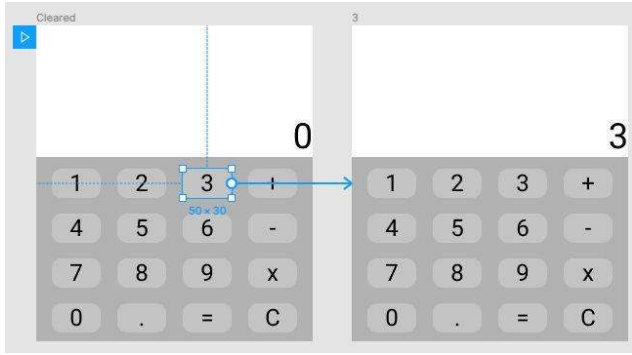
In the layer list, select the top-level frame, which you named "Cleared" in Step 3. Duplicate the frame, move it to the right of the first frame, change the display label to "3", and rename the frame to "3".

Create other screens whose labels and displays are "3+", "3+3", and "6", respectively, for a total of 5 screens. Hint: you can zoom out to see more canvas at a time—see the zoom control in the upper-right of Figma's UI.

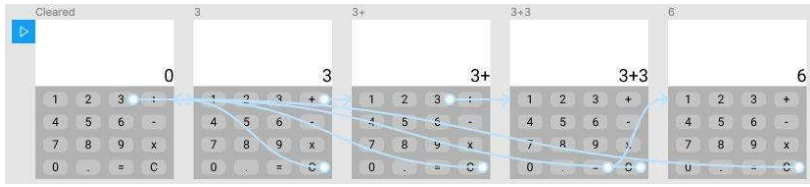


12. Create a clickable prototype

Select the first frame (named "Cleared"), and select the "prototype" tab of the inspector. Select Button 3 within the frame. Notice the blue circle on the right edge of the button. Drag that circle to the frame labeled "3".



Should look like this:



Likewise, connect the "+" button in the "3" Frame to the "3+" Frame, and likewise for the next "3" button and the following "=" button. Finally, connect the "C" button of all but the first frame to the "Cleared" frame. After deselecting all frames, your prototype connectionsPresent the clickable prototype

Finally, the payoff. Click the "play" triangle icon in the upper right of the Figma UI to see the clickable prototype in action.

PRACTICAL QUESTIONS

Create a low fidelity design for health app on android phone and list out the questions you will ask for testing the given prototype. Also create test cases for the same.

Create both low fidelity and high fidelity design for an ecommerce website and also generate the test cases to test the same.

❖❖❖❖

Module VI

6

INTRODUCTION TO IMPLEMENTATION IMPLEMENTATION

Learning Objectives :
To make the learners aware of the different factors to be considered while selecting and using a programming language.
To make them aware of the different code optimization techniques as well as the web optimization.
To make them aware of the importance of UI and UX during implementation of the project
To make them aware of the different software development tools

Learning Outcomes :-
The learner would be able to select the right programming language for the right project and also know how to optimise the code well to achieve better performance.

The learner would be aware of different development tools to automate the development process

What is software implementation

Software implementation refers to writing or reusing a code to execute a particular logic. **The implementation tools include:**

- CASE tools that enable analysts and designers to produce models of the system diagrams.
- Textual and Structured descriptions
- Code generator
- Compilers, Interpreters, Debuggers
- Visual Editors Integrated Development Environment (IDE)

Factors to be considered while selecting a programming language

The perfect selection of a programming language provides solutions that are concise, easy to debug, easy to extend, easy to document, and easy to fix.

The Targeted Platform

Target platform is very crucial for determining the programming languages, For example if a program is written in C and needs to be run on Windows and Linux platforms, it would require platform compilers and

two different executables. On the other hand, with Java, the program can un on any machine provided a Java Virtual Machine (JVM) is installed. Similarly in case of web pages they should all appear and work the same across all browsers. However, using CSS3 tags and HTML 5 without checking browser compatibility will cause the site to look and behave differently across different browsers. Also it has to be separately designed for mobile devices.

The Elasticity of a Language: This refers to the ability of the software to adapt to changes in the future. How easy it would be to accommodate changes to the software.

The Time to Production

This is the time taken to make the program go live, when the code is production-ready and works as intended. It is highly dependent on the size of code. In theory, the easier it is to learn a language, the smaller the code and thus less time taken to go live.

Performance: The ability of the software to respond to every request is its performance. Performance would be effective if the software code is optimal and occupies too little memory and also takes less time of the CPU.

Support and Community

. Languages with active forums are likely to be more popular than even greater languages without similar forums. Some of the offerings of community support include wikis, forums, tutorials and most importantly additional libraries, all of which help the language to grow.

Objective

It specifies the objective for which a program is being developed. In the event that one needs to develop commercial applications, then a business-oriented programming language like COBOL is the best candidate. For development of scientific applications, it is best to use a scientific oriented language like FORTRAN.

Similarly, if one is interested in developing programs related to Artificial Intelligence, then he or she should be comfortable using the LISP or ProLog languages. Object oriented languages are best suited for development of web-based applications. As for the development of system programs, a middle level language like C should be chosen.

Programmer Experience

If several programming languages or tools available to develop a software the programmer must select the once that he is not just familiar but have the expertise.

Ease of Development and Maintenance

Choose Programming language that is easy to maintain and reuse. Program sustainability should be for longer.

Efficiency

Efficiency is an important factor which need be considered before choosing a programming language for software development. One should consider the language in which the programs can be developed and executed rapidly.

Availability of an IDE

A powerful Integrated Development Environment goes a long way in increasing the productivity of a programmer. The language with an IDE of well supported development, debugging and compilation tools should be selected.

Error Checking and Diagnosis

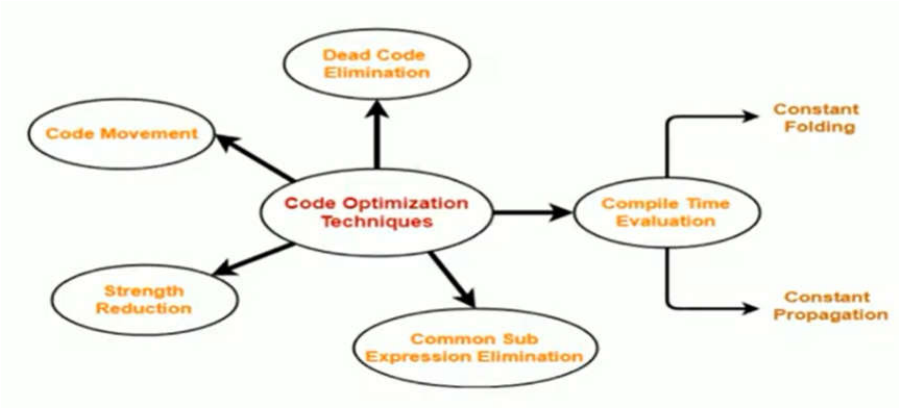
These factors involve finding the errors in a program and their causes. Programmers should choose programming languages which contain efficient error handling features. Error checking and diagnosis is very important and crucial in the development of quality and error-free programs. The task of code development and testing is easier when undertaken with a programming language with efficient and robust error detection and correction mechanisms.

A good example is Java. This language provides an efficient error handling mechanism of try/catch block. This feature in Java programs can be used to handle the unexpected errors that may occur during the execution of a program.

CODE OPTIMIZATION

It is a technique to improve the code by eliminating unnecessary code lines and arranging statements in such a sequence that speed up program execution and is to occupy limited memory. The objective of code optimization is to achieve faster execution, efficient memory usage and yield better performance.

CODE OPTIMIZATION TECHNIQUES



1) **Constant Folding**: It refers to a technique of evaluating the expressions whose operands are known to be constant at compile time itself e.g., length 22/7*d

2) **Constant Propagation:** In constant propagation if a variable is assigned a constant value, then subsequent use of that variable can be replaced as constant as long as no intervening assignment has changed the value of the variable.

Example-

pi = 3.14

radius = 10

Area of circle = pi x radius x radius

3) **Code Movement** : It is a technique of moving a block of code outside loop if it won't have any difference if it is executed outside or inside the loop

Example

Code before optimization	Code after optimization

for(int i=0;i<n;i++)
{
X=y+z
a[i]=6*i
}
	X=y+z
	for(int i=0;i<n;i++)
	{
	a[i]=6*i
	}

(4) **Dead code elimination:** Dead code elimination includes eliminating those code statements which are never going to get executed e.g.

```
I0
If(i==1)
{a=x+5;
}
```

Can be optimized directly to i=0;

(5) **Strength Reduction**: It is the replacement of expressions that we are expensive with cheaper and simple ones

B=A*2 before optimization

B=A+A After optimization

Strength reduction means replacing the high strength operator by the low strength.

```
i = 1;
while (i<10)
{
   y = i * 4;
}
 //After Reduction
i = 1
t = 4
{
  while( t<40)
  y = t;
  t = t + 4;
}
```

6. Common Sub-Expression Elimination-

The expression that has been already computed before and appears again in the code for computation

is called as **Common Sub-Expression**.

In this technique,

- As the name suggests, it involves eliminating the common sub expressions.
- The redundant expressions are eliminated to avoid their re-computation.
- The already computed result is used in the further program when required.

Example-

Code Before Optimization	**Code After Optimization**
S1 = 4 x i S2 = a[S1] S3 = 4 x j S4 = 4 x i // **Redundant Expression** S5 = n S6 = b[S4] + S5	S1 = 4 x i S2 = a[S1] S3 = 4 x j S5 = n S6 = b[S1] + S5

Website Speed Optimization -Light Weight Page Loading

The website speed makes the first impression about your business

High-performance websites result in high return visits, low bounce rates, higher conversions, engagement, higher ranks in organic search, and better user experience. Slow websites will cost you money and damaged reputation. By reducing the page load time, you will positively impact marketing and sales processes. Following are the ways to optimize the website

1**. Use a Content Delivery Network (CDN)**

A content delivery network is a network where there are several redundant distributed servers spread across several geographical locations that provide web content to end users about their location. This way request traffic will be distributed thereby reducing the load on single server.

2. **Move your website to a better host**

Website can be hosted on a shared host, virtual private servers and dedicated servers. Shared host are less expensive but with it you are sharing your resources like processor, RAM, hard disk etc. virtual private

servers and dedicated servers are much faster as there exist no resource sharing.

3**. Optimize the size of images on your website**

Images are necessary on a website to catch viewers attention. However, they consume more space. We can optimize the image size without compromising the quality by using compression tools like image optim, jpegmini or kraken.

4. **Reduce the number of plugins**

Include only those plugins which are necessary for your website to run smoothly remove the remaining which are unwanted. Plugins are common components of each website. They add specific features suggested by third parties. Unfortunately, the more plugins are installed, the more resources are needed to run them. As a result, the website works slower and also security issues can appear.

5. **Minimize the number of JavaScript and CSS files**

If your website contains a lot of JavaScript and CSS files, it leads to a large number of HTTP requests when your website visitors want to access particular files. These requests are treated individually by visitor's browser and slow down the website work. If you reduce the number of JavaScript and CSS files this will undoubtedly speed up your website

6. **Use website caching**

In case there are a lot of users accessing the page at one time servers work slowly and need more time to deliver the web page to each user. Caching is the process of storing the current version of your website on the hosting and presenting this version until your website is updated. This means that the web page doesn't render over and over again for each user. Cached web page doesn't need to send database requests each time.

7. Implement Compression of files

Compression of file is an effective way to reduce the size of files. It minimizes the HTTP requests and reduces the server response time. For e.g. Gzip compresses the files before sending them to the browser.

8. **Database optimization in CMS**

Database optimization is the an effective way to increase performance. If you use a content management system (CMS) packed with complex plugins, the database size increases and your website works slower. For instance, the WordPress CMS stores comments, blog posts, and other information that take up a lot of data storage. Each CMS requires its own optimization measures and also has a number of specific plugins. For WordPress, for example, you may consider **WP-Optimize**.

9. Reduce the use of web fonts

Web fonts have become very popular in website design. Unfortunately, the use of web fonts has a negative impact on the speed of page rendering. Web fonts add extra HTTP requests to external resources. The following measures will help you reduce the size of web font traffic:

- Use modern formats WOFF2 for modern browsers;
- Include only those character sets that are used on the site.
- Choose only the needed styles

10. Detect 404 errors

A 404-error means that a "Page isn't found". This message is provided by the hosting to browsers or search engines when the accessed content of a page no longer exists. To detect and correct a 404 error, you can use error detection tools to fix them.

11. Reduce redirects

Avoid website redirects as they create additional HTTP requests which negatively impact performance.

12. Use prefetching techniques

Prefetching entails reading and executing instructions before a user initiates them. The technique is rather common. It works well if you can anticipate user actions and, for instance, load some content or links in advance. Usually, modern browsers allow for prefetching by default as they assume user behaviour patterns. However, UX specialists and engineers are more likely to understand user behaviour and make "hints" for browsers to do prefetching work.

User Interface, User Experience and Usability are the most important thing today to be considered during the implementation process.

1. Usability: Making a website or app easy to use. It is perception of the end user on how that person can effectively, efficiently and satisfactory complete a task when using a product or application. The perception of usability is therefore a highly subjective way of looking at an application or product and is largely based on the knowledge and prior experiences a user already possesses when using the technology. Key elements in the perception of usability however are as the statement says: effectiveness, efficiency and perceived satisfaction when doing certain tasks, these elements can of course be quantified and measured to get a good view on the usability of a product or application.

2. User Interface (UI): Making a website or app attractive and effective according to users' preferences. It is a interaction between the Human and the system / product. The design of buttons, entry fields and layout of an application or website is where the user interacts with the system. You can therefore compare the user interface with the steering

wheel, pedals and dashboard of a car which are used to operate the vehicle in the same way an application has a UI in place to operate a system.

3. **User Experience(UX):** Making users feel positive about a website or app. User Experience is the overall experience of an end user with regards to a product or application or can even be related to the usage of a service in some cases. The experience however doesn't cover one task but covers the whole lifecycle from the orientation before buying to the disposal of the product / application. As stated in the terminology UX also covers the whole experience of the product or application and therefore covers aspects like UI and Usability within it as they are part of the experience.

Jakob Nielsen's has listed out 10 Heuristics Principles of Usability to be considered during implementation

1. Visibility of system status

The system should always keep users informed about current state and actions through appropriate visual cues and feedback within reasonable time.

2. Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

3. User control and freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

4. Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

5. Help users recognize, diagnose, and recover from errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

6. Consistency and standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

7. Recognition rather than recall

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

8. Flexibility and efficiency of use

Accelerators --unseen by the novice user --may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

9. Aesthetic and minimalist design

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

10. Help and Documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Today most of the development activity includes creating a web app or a cloud application. Also, E-Commerce is gaining its popularity compared to traditional commerce. So, it has become extremely important to focus on the User Interface/User Experience of a Website as the Website content is solely doing the sales. We don’t have any physical salesman pursuing any sales.

Characteristics of User-Friendly Website

1. Mobile Compatibility

Today all applications to be develop has to be compatible with mobile devices. As more and more people use their mobile phones to access the Internet, creating a mobile optimized website has become a necessity today

2. Accessible to All Users

Accessibility aspect is very important today. While designing an application one must consider the differently abled people and design website that would cater to their needs too. A user-friendly website should also be accessible to everyone including blind, disabled or the elderly. These users typically use screen-readers to access the Internet. There are many website accessibility guidelines highlights simple web design techniques that can be applied to make ensure your website can be accessed easily on-screen readers, making your website available to a larger audience.

3. Well Planned Information Architecture

The way information is organised and presented on your website is vital for good usability. However, it is often neglected. It has become even more important today as websites offer a wide range of information and resources to attract their target market. Plan your website sections and categories carefully and present information in a way that it is easy for users to find.

4. Well-Formatted Content That Is Easy to Scan

The average Internet user skims through the content on a web page instead of reading each and every word from top to down. Users tend to scan through key parts of the page quickly to determine if it is relevant to their needs.

It is important to format your content with this in mind. Correct use of headings, sub-headings, paragraphs, bullets or lists help to break up text, making it easy for readers to scan.

5. Fast Load Times

Website speed is of utmost important. Users are impatient while accessing the website. In fact, slow speed is one of the main reasons why visitors leave a website. Making sure your website loads within 4 to 6 seconds is important for good usability. It also affects your search engine ranking. There are many free tools that enable you to test the performance of your website and provides suggestion on improving the same. Website performance is affected through the use of third-party website plugins and widgets including website tracking, social media. Limiting their usage is advisable.

6. Browser Consistency

Web application should have compatibility with all browsers to gain popularity.

7. Effective Navigation

Good navigation is one of the most important aspects of website usability. Simple HTML or JavaScript menus tend to work best and appear consistent on all browsers and platforms.

It is equally important for the navigation to be clutter-free. Try to limit the number of menu items as far as possible. A drop-down menu or sub-navigation may work better on large site with many sections and pages. DHTML, Javascript libraries such as Motols and Ajax offer innovate navigation system.

8. Good Error Handling

Good usability demands good error handling and description on-screen messages However, it is often overlooked. Correct handling of errors at a

code level ensures the website is robust and free from bugs. Displaying the right error message improves the user experience and overall usability.

9. Valid Mark-Up & Clean Code

A website that adheres to the relevant web design best practices and standards is often more robust and dependable. It also ensures the website will load faster and appear consistent across browsers and devices. It also makes it easier to locate problems and troubleshoot if the need arises.

More information and mark-up validation tools can be found on W3C's website.

10. Contrasting Colour Scheme

The right contrast between the background of the website and content is one of the most basic yet most important web design principles that should never be overlooked. Good contrast between background and text e.g. black text on a white background makes your content legible and easy to read. Lack of contrast, on the other hand, makes it very difficult for visitors to read your content.

11. Usable Forms : Forms are meant for capturing the input. A well defined form would allow users to interact smoothly with the website.

CHARACTERISTICS/ TRAITS OF USABILITY

UX designers, by their very nature solve problems and seek solutions that creatively align user needs and business goals. A sound solution to a design problem identifies the nature and context of use, whilst taking into account the limitations and constraints in which the resulting product/application will be used.

Useful: Your content should be original and fulfill a need

Usable: Site must be easy to use

Desirable: Image, identity, brand, and other design elements are used to evoke emotion and appreciation

Findable: Content needs to be navigable and locatable onsite and offsite

Accessible: Content needs to be accessible to people with disabilities

Credible: Users must trust and believe what you tell them

Other Traits of UX

Equitable

If a product is **equitable**, it means a design is helpful to people with diverse abilities and backgrounds. In other words, the product's design addresses the needs of a diverse audience and ensures a high-quality experience is delivered to all users regardless of background, gender, race, or ability. Equity means providing people with the tools they need to accomplish their goals and support improved quality of life.

Enjoyable

If a product is **enjoyable**, it means the design delights the user. The design reflects what the user may be thinking or feeling and creates a positive connection with them.

Useful

If a product is **useful**, that means it solves user problems. In other words, the design intentionally solves a user problem that the designer has identified. It's important to note that, while similar, useful and usable have different meanings. A product that is useful isn't always usable. The same is true for the opposite. The distinction between the two is that usability refers to the product working well and being easy to use, while usefulness refers directly to the ability to solve user problems.

Popular Software Development and Programming Tools

This includes different tools used during the software development life cycle like the requirement gathering tools, designing tools, Development Tools, Testing Tools both for functional and non-functional testing and Configuration Management Tools

Requirement gathering tools

Microsoft Visio, Mind Genius, Jama Software, Code Beamer ALM, ACCompa, Caliber, Perforce Heliz RM, Pearls, ReqSuite,Visure, Orcanos are some of the requirement gathering tools that provides the following features :-

- Figuring out the goals and objectives of the stakeholders
- Documenting everything; a step that is often undervalued
- Getting approval and confirmation explicitly

- Remaining agile and understanding that changes will come
- Handling approvals
- Documenting changes
- Managing relationships between requirements
- Removing ambiguity, assumptions, and wishful thinking
- Making certain all requirements are clear, realistic, and agreed upon
- Reducing Redundancy
- Traceability of Requirements
- Risk Management
- Enterprise Architect is another is a requirement management tool.

It integrates seamlessly with other development tools by creating requirements in the model.

Designing Tools

Designing Tools help in creating a high and low architectural designs, low and highfidelity designs, wireframes, mockups and prototypes. These are some of the designing tools widely used

1. Star UML – A popular UML modeling tool
2. OpenText Provision – An extensive business process architecture tool
3. Proofhub
4. Adobe Photoshop
5. Adobe Illustrator
6. Visual Paradigm – A design and management tool for business IT development
7. FileStage
8. PicsArt
9. Desygner
10. Design Bold
11. Fotor
12. Mockplus
13. Marvel-prototyping tool
14. Figma
15. Pixelmator Pro

Development Tools

RAD Studio

RAD Studio is a Powerful IDE For Building Native Apps on Windows, Android, iOS, macOS and Linux. It enables you to design Beautiful Desktop and Mobile App UIs with less coding effort. Write once, compile everywhere.

FinanceLayer

FinanceLayer is a real-time API for finance news that uses JSON payload to get the most dynamic financial data. You can get live finance news and article feeds on any website or application.

Studio 3T

Studio 3T for MongoDB helps you to build queries fast, generate instant code, import/export in multiple formats, and much more.

Linx

Linx is a low code IDE and server. IT pros use Linx to quickly create custom automated business processes, integrate applications, expose web services and to efficiently handle high workloads.

DbSchema

DbSchema is a visual database designer & manager for any SQL, NoSQL, or Cloud database. The tool enables you to visually design & interact with the database schema, design the schema in a team and deploy it on multiple databases, generate HTML5 diagram documentation, visually explore the data and build queries, and so much more.

NetBeans

NetBeans is a popular, Free, open-source IDE. It is one of the best application development tools that allows developing desktop, mobile and web applications.

Cloud9 IDE

Cloud9 IDE is an online integrated software development environment. It is one of the best software design tools that supports many programming languages like C, C++, PHP, Ruby, Perl, Python, JavaScript and Node.js.

Zend Studio

Zend Studio allows software developers to code faster, debug more easily. It is next-generation PHP IDE designed to create apps for boosting developers' productivity.

Bootstrap

Bootstrap is a responsive framework for developing with HTML, CSS, and JS. It is one of the best software programming tools that has many in-builds components, which you can easily drag and drop to assemble responsive web pages.

HTML5 Builder

HTML5 Builder is a software solution for building the web and mobile apps. It can develop an app using a single HTML5, CSS3, JavaScript and PHP codebase. It helps to target multiple mobile operating systems, devices and Web browsers.

Azure

Microsoft Azure is widely used by developers to build, deploy and manage web applications.

Github

GitHub allows developers to review code, manage projects, and build software. It offers right tool for different development jobs.

Axure

Axure provides the capability to produce wireframes, prototypes, and create documentation. This tool is used by business analysts, product managers, and IT consultants around the world.

SendBird

Sendbird is used as a messaging and Chat API for Mobile Apps and Websites. It offers scalability for a massive audience. It also prevents spam flooding of chat rooms.

Review and Testing Tools

Collaborator

The below are some of the code and document review tool for development teams that take quality seriously.

- Colloborator-Peer review
- SmartBear Collaborator
- Embold
- CodeScene
- Codebrag
- Gerrit
- Codestriker
- Rhodecode

- Phabricator
- Crucible
- Veracode
- Review Board

Testing Tools- Functional and Non Functional– Selenium, TestRail,Xray, Practitest, TestPad, Zephyr Scale,SpiraTest, Testiny, Test Monitor, Avo Assure, Kobiton, Parasoft. Lambda Test, QTP, Watir, Testim,

CrossBrowserTesting, SauceLabs, -Cross Browser Testing Tool

Webload, Ghostlab, Loadrunner, Wapt, Jmeter, LoadImpact- Load Testing Tool

Jira, Mantis, Bugzilla, Redmine- Defect Tracking tool

Appium, Espresso, Perfecto, Digital.ai,Robotium- Mobile Testing Tool

Tools for Configuration Management

The goals of Software Configuration Management are generally Configuration, Identification, Configuration idioms and baselines, configuration control, implementing a control change process.

This is usually achieved by setting up a change control board whose primary function is to approve or reject all change request that is sent against any baseline. Configuration status accounting, reporting and recording all the necessary information on the status of the development process.

- SolarWinds Server Configuration Monitor
- Auvik
- CFEngine Configuration Tool
- Puppet Configuration Tool
- CHEF Configuration Tool
- Ansible Configuration Tool
- SALTSTACK Configuration Tool
- JUJU Configuration Tool
- RUDDER
- Bamboo Configuration Management
- TeamCity Configuration Tool
- Octopus Deploy

PRACTICAL QUESTIONS

a) Create an E-Commerce Website using HTML, Microsoft Publisher, Asp.NET or any free web making tools like web builder, word press

b) Optimize the below given codes

(i)

```
do
{
  item = 10;
  value = value + item;
} while(value<100)
```

(ii)

```
c = a * b
x = a
till
d = x * b + 4
```

(iii)

```
a = 200;
      while(a>0)
 {
    b = x + y;
    if (a % b == 0}
    printf("%d", a);
  }
```

(iv)

```
1.          some_list = [1, 2, 3, 4]
that_list_plus_1 = []
that_list_plus_2 = []
 for i in some_list:
that_list_plus_1.append(i + 1)
for i in some_list:
that_list_plus_2.append(i + 2)
```

c) Consider a website https://machpowertools.com/ or any other website and list out the tips for improving the User Interface, User Experience and Usability of the same.

d) On the basis of your own experience of using a typical bank cash machine (ATM), and after learning the Jakob Nielsen's 10 Heuristics Principles, describe what designer should and should not do.

Jakob Nielsen's 10 Heuristics Principles

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

Example: 1st Principle

1. **Visibility of system status:** ATM should give immediate feedback once users entered valid or invalid password. (If password is valid, ATM provide welcome message, if password is invalid display error message with solution).

❖❖❖❖

Module VII

7

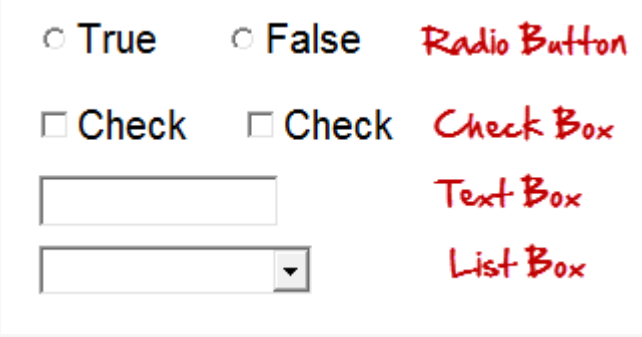
INTRODUCTION TO USER INTERFACE TESTING

Learning Objectives:
To make the learners aware of the different aspects of User Interface and Usability Testing, different testing approaches, Test case designing, test scripts, testing tools etc.

Learning Outcomes: The learner will be able to design an user interface for his project and would also have the ability to identify the user Interface flaw and usability problem of an existing project or website.

Introduction: -

Interface refers to the environment with which you interact. User interface (UI) are of two types of Command User Interface (CUI) and Graphical User Interface (GUI). Command User Interface is the environment in which the user interacts with the computer system using commands. Whereas in GUI interface it is less of keying and more of clicks. The below is the example of GUI interface



UI testing, also known as GUI testing, is a technique for testing the features of any software that a user will interact with. This usually involves testing the visual components to ensure that they are meeting the outlined requirements - both in terms of functionality and performance.

It includes testing all visual indicators and graphical elements like the menus, radio buttons, text boxes, checkboxes, toolbars, colours, fonts, and more.

The main aspects checked in **UI testing include:**

- ***Visual Design-*** *Physical Appearance*
- ***Functionality-*** *Task performed. Output received*

- ***Performance-*** *Response Time with every click*
- ***Compliance-*** *Legal and Business requirement Compliance*
- ***Usability-*** **Usability includes the following: -**

Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?

Efficiency: Once users have learned the design, how quickly can they perform tasks?

Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?

Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?

Satisfaction: How pleasant is it to use the design? (Nielsen, Usability 101: Introduction to Usability)

Guess ability: How efficiently a user can use the system the first time without orientation.

Performance: The response time of the application with every click.

Reusability/Sustainability: The ability to reuse the application or the modules in the application ahead. It is about knowing how long the application can sustain in the market. If the application is agile and can accept changes it will sustain longer in the market.

Purpose of UI Testing

- To check how the application handles user actions carried out using the keyboard, mouse, and other input devices.
- To check if all visual elements are displayed and working correctly.
- Check all the GUI elements for size, position, width, length, and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
- Check you can execute the intended functionality of the application using the GUI
- Check Error Messages are displayed correctly
- Check for Clear demarcation of different sections on screen
- Check Font used in an application is readable
- Check the alignment of the text is proper
- Check the Colour of the font and warning messages is aesthetically pleasing

- Check that the images have good clarity
- Check that the images are properly aligned
- Check the positioning of GUI elements for different screen resolution

UI Testing essential Checklist for creating Test Cases

- **Data type errors** – Ensure only valid data can be entered for specific data types such as currency and dates.
- **Field widths** – If a certain text box permits a specified number of characters, then make it clear on the user interface that the data entered shouldn't exceed the character limit. (For instance, a field that allows 50 characters in the application's database should not allow users to enter more than 50 characters on the interface).
- **Navigational elements** – Verify all navigational buttons on the page are working correctly, and that they redirect users to the right page or screen.
- **Progress bars** – When displaying screens that take time to render results, a progress bar should be used to show the user that a process is still running.
- **Type-ahead** – If your UI uses drop-down lists, ensure you include type ahead. In a drop-down menu with hundreds of items, typing the first letter should skip the list to items beginning with that letter such that users will not have to check through a long list.
- **Table scrolling** – If data in your tables extends to another page, then the scroll function should allow users to scroll the data but keep all headers intact.
- **Error logging** – When the system experiences a fatal error, ensure the application writes the error details to an event viewer or log file for later review.
- **Menu items** – Ensure the application only displays valid menu items that are available at a particular state.
- **Working shortcuts** – For applications that support shortcuts, verify whether they work correctly, no matter the browser, platform, or device being used.
- **Confirm action buttons** – Ensure the UI has working confirm button every time the user wants to save or delete an item
- Testing the size, position, width, height of the elements.
- Testing of the error messages that are getting displayed.
- Testing the different sections of the screen.
- Testing of the font whether it is readable or not.

- Testing of the screen in different resolutions with the help of zooming in and zooming out like 640 x 480, 600×800, etc.
- Testing the alignment of the texts and other elements like icons, buttons, etc. are in proper place or not.
- Testing the colors of the fonts.
- Testing the colors of the error messages, warning messages.
- Testing whether the image has good clarity or not.
- Testing the alignment of the images.
- Testing of the spelling.
- The user must not get frustrated while using the system interface.
- Testing whether the interface is attractive or not.
- Testing of the scrollbars according to the size of the page if any.
- Testing of the disabled fields if any.
- Testing of the size of the images.
- Testing of the headings whether it is properly aligned or not.
- Testing of the color of the hyperlink.
- Payment transactions or any other critical transactions to be evaluated end to end**.**

Steps in UI Testing

1. **Define your goal.** Make sure you have a solid understanding of the problems you are trying to solve and the goals you are trying to achieve.
2. **Know your audience.** This will help you ask the right questions which leads to a design that is more focused on your users' specific needs.
3. **Test early and often.** Testing is a necessity in UX design. When creating your web design, it's important to run tests often so you can validate components of your design. Prototype your design, test it with your users, and use the feedback to improve your next design.

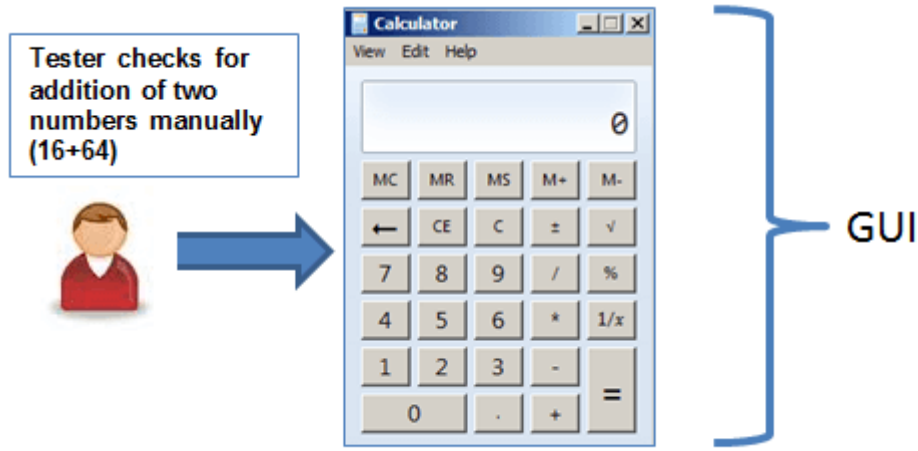
Approaches to UI Testing

There are three main GUI testing approaches, namely:

1. Manual Testing

In manual testing, a human tester performs a set of operations to check whether the application is functioning correctly and that the graphical

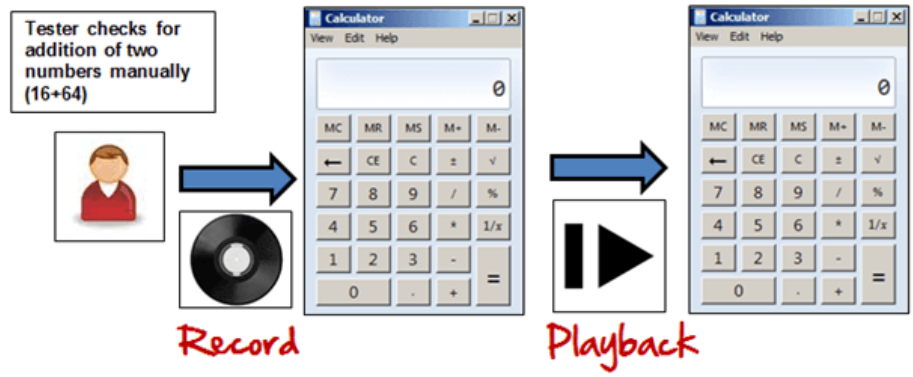
elements conform to the documented requirements. Manual-based testing can be time-consuming, and the test coverage (Total no of test cases to be executed) is extremely low. Additionally, the quality of testing in this approach depends on the knowledge and capabilities of the testing team. Under this approach, graphical screens are checked manually by testers in conformance with the requirements stated in the Software requirements Specification document.



They will be manually checking 16+64, -16+64,16.66+64,-16.66+64 and other permutations and combinations.

2. Automation Tool Testing-Record-and-Playback Testing

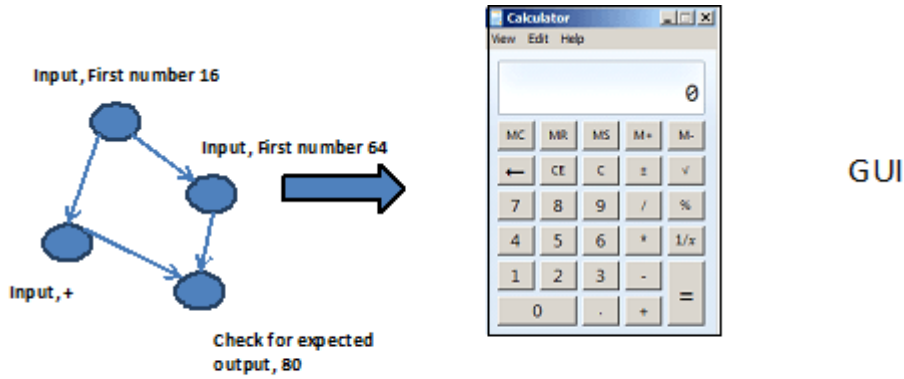
This is executed using automation tools. The automated UI testing tool records all tasks, actions, and interactions with the application. The recorded steps are then reproduced, executed, and compared with the expected behaviour. For further testing, the replay phase can be repeated with various data sets. GUI testing can be done using automation tools. This is done in 2 parts. During Record, test steps are captured by the automation tool. During playback, the recorded test steps are executed on the Application Under Test. Example of such tools – QTP, Selenium. Mainly used for website testing.



You are initially testing it manually and while you are testing it manually you are recording your actions. Later if there is any change and you want

to rerun the test case to see whether the functionality works fine you will playback what you have recorded.

3. Model Based Testing



Model Based Testing

A model is a graphical description of a system's behaviour. It helps us to understand and predict the system behaviour. This provides a deeper understanding of the system, which allows the tester to generate highly efficient test cases. In the models, we determine the inputs and outputs of the system, which are in turn, used to run the testsThe following needs to be considered for this model-based testing:

- Build the model
- Determine System Inputs for the model
- Calculate and verify the expected output for the model
- Execute the tests
- Compare the actual output with the expected output
- A decision on further action on the model

Some of the modelling techniques from which test cases can be derived:

- Charts – Depicts the state of a system and checks the state after some input.
- Decision Tables – Tables used to determine results for each input applied

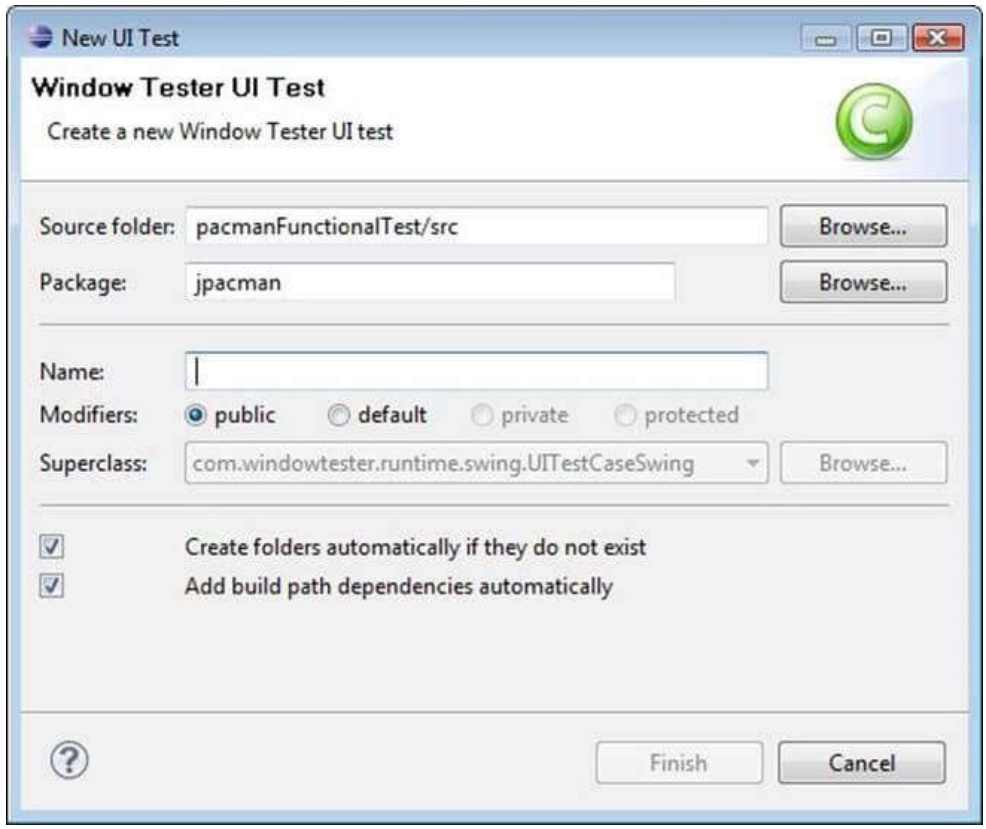
Model based testing is an evolving technique for generating test cases from the requirements**. Its main advantage, compared to above two methods, is that it can determine undesirable states that your GUI can attain.** The model-based approach is great because it allows a higher level of automation. It also covers a higher number of states in the system, thereby improving the test coverage.

OPEN SOURCE AUTOMATED TESTING TOOLS

Product	Licensed Under
AutoHotkey	GPL
Selenium	Apache
Sikuli	MIT
Robot Framework	Apache
Water	BSD
Dojo Toolkit	BSD

EXAMPLE ON GUI TESTING TEST CASES

Here we will use some sample test cases for the following screen.



Following below is the example of the Test cases, which consists of UI and Usability test scenarios.

TC 01- Verify that the text box with the label "**Source Folder**" is aligned properly.

TC 02 – Verify that the text box with the label "**Package**" is aligned properly.

TC 03 – Verify that label with the name "**Browse**" is a button which is located at the end of TextBox with the name "**Source Folder**."

TC 04 – Verify that label with the name "**Browse**" is a button which is located at the end of TextBox with the name "**Package**."

TC 05 – Verify that the text box with the label "**Name**" is aligned properly.

TC 06 – Verify that the label "**Modifiers**" consists of 4 radio buttons with the name public, default, private, protected.

TC 07 – Verify that the label "**Modifiers**" consists of 4 radio buttons which are aligned properly in a row.

TC 08 – Verify that the label "**Superclass**" under the label "**Modifiers**" consists of a dropdown which must be properly aligned.

TC 09 – Verify that the label "**Superclass**" consists of a button with the label "**Browse**" on it which must be properly aligned.

TC 10 – Verify that clicking on any radio button the default mouse pointer must be changed to the hand mouse pointer.

TC 11 – Verify that user must not be able to type in the dropdown of "**Superclass**."

TC 12 – Verify that there must be a proper error generated if something has been mistakenly chosen.

TC 13 – Verify that the error must be generated in the RED color wherever it is necessary.

TC 14 – Verify that proper labels must be used in the error messages.

TC 15 – Verify that the single radio buttons must be selected by default every time.

TC 16 – Verify that the TAB button must be work properly while jumping on another field next to previous.

TC 17 – Verify that all the pages must contain the proper title.

TC 18 – Verify that the page text must be properly aligned.

TC 19 – Verify that after updating any field a proper confirmation message must be displayed.

TC 20 – Verify that only 1 radio button must be selected, and more than single checkboxes may be selected.

GUI Testing Techniques

Scripted testing

In scripted testing, software testers design and then execute pre-planned scripts to uncover defects and verify that an application does what it is supposed to do. For example, a script might direct a tester through the process of placing a specific order on an online shopping site. The script defines the entries that the tester makes on each screen and the expected outcome of each entry. The tester analyses the results and reports any defects that are found to the development team. Scripted testing may be performed manually or supported by test automation.

Scripted testing benefits

Because scripted testing is pre-planned and has tangible outputs – the test scripts and testing reports – scripted testing gives product managers and customers confidence that an application has been rigorously tested. By creating test scripts early in the development process, teams can uncover missing requirements or design defects before they make it into the code. While test scripts must be created by a more experienced tester with knowledge of the system, less experienced/knowledgeable testers can perform the actual testing. Finally, test scripts can be reused in the future for regression testing and can also be automated for greater efficiency.

Scripted testing challenges

Scripted testing requires a lot of up-front planning, which can cause time pressure especially in agile development environments. Test scripts must be updated as the AUT changes. More importantly, studies have suggested that the rigid structure of test scripts may cause testers to miss defects that would be uncovered by exploratory testing, or that the time required to develop test cases and execute them manually does not deliver payback in terms of the number and severity of defects found relative to exploratory testing. Read more about defect detection rates in scripted vs. exploratory testing.

Exploratory testing

Rather than following pre-written test scripts, exploratory testers draw on their knowledge and experience to learn about the AUT, design tests and then immediately execute the tests. After analysing the results, testers may identify additional tests to be performed and/or provide feedback to developers.

Although exploratory testing does not use detailed test scripts, there is still pre-planning. For example, in session-based exploratory testing, testers create a document called a test charter to set goals for the planned tests and set a time frame for a period of focused exploratory testing. Sessions of exploratory testing are documented by a session report and reviewed in a follow-up debriefing meeting.

Likewise, during scripted testing, there may be some decisions available to testers including the ability to create new tests on the fly. For that reason, it is helpful to think of scripted testing and exploratory testing as being two ends of a continuum rather than being opposites.

Both scripted and exploratory testing can be completely manual, or they can be assisted by automation. For example, an exploratory tester might decide to use test automation to conduct a series of tests over a range of data values.

Exploratory testing benefits

As time planning and writing test cases is reduced, testers have more time to focus on the actual testing of the AUT. Testers who are challenged to

use their knowledge, skills, and creativity to identify defects and ensure conformance to requirements may be more engaged and may find more defects than testers who are restricted to scripts written by others.

Exploratory testing challenges

Exploratory testing requires testers to have a deep understanding of the performance requirements of the AUT as well as skill in software testing. Due to the realities of time constraints and resource availability, it may be impractical to try to cover an entire AUT with exploratory testing. Exploratory tests are not as repeatable as scripted tests, which is a major drawback for regression testing. Further, relying on exploratory testing alone can create concern in product managers or customers that code will not be covered and defects will be missed.

User experience testing

In user experience testing, actual end-users or user representatives evaluate an application for its ease of use, visual appeal, and ability to meet their needs. The results of testing may be gathered by real-time observations of users as they explore the application on-site. Increasingly, this type of testing is done virtually using a cloud-based platform. As an alternative, project teams can do beta testing, where a complete or nearly complete application is made available for ad hoc testing by end users at their location, with responses gathered by feedback forms. By its nature, user experience testing is manual and exploratory.

Don't confuse user experience testing (UX) with user acceptance testing (UAT). As discussed earlier, UAT is a testing level which verifies that a given application meets requirements. For example, imagine that shortly after an update is released to a shopping website, the site receives many complaints that customers are unable to save items to a wish list. However, UAT verified that pressing the "wish list" button correctly added items to the wish list. So, what's wrong? UX testing might have revealed that the wish list button was improperly placed on the screen, making it difficult for shoppers to find.

You can conduct UX testing at any point in the development phase where user feedback is needed. It is not necessary to have a completed application prior to involving users. For example, focus groups can respond to screen-mock-ups or virtual walk-throughs of an application early in development.

User experience testing benefits

UX testing provides the essential perspective of the end-user, and therefore it can identify defects that may be invisible to developers and testers due to their familiarity with a product. For example, a few years ago, a leading web-based e-mail provider developed a social sharing tool. This tool was beta tested by thousands of the provider's own employees but not by end-users prior to its initial release. Only after the product was released into production did the provider begin receiving end-user

feedback, which was overwhelmingly negative due to privacy concerns. Early UX testing would likely have revealed these concerns and saved the provider millions of dollars in development costs.

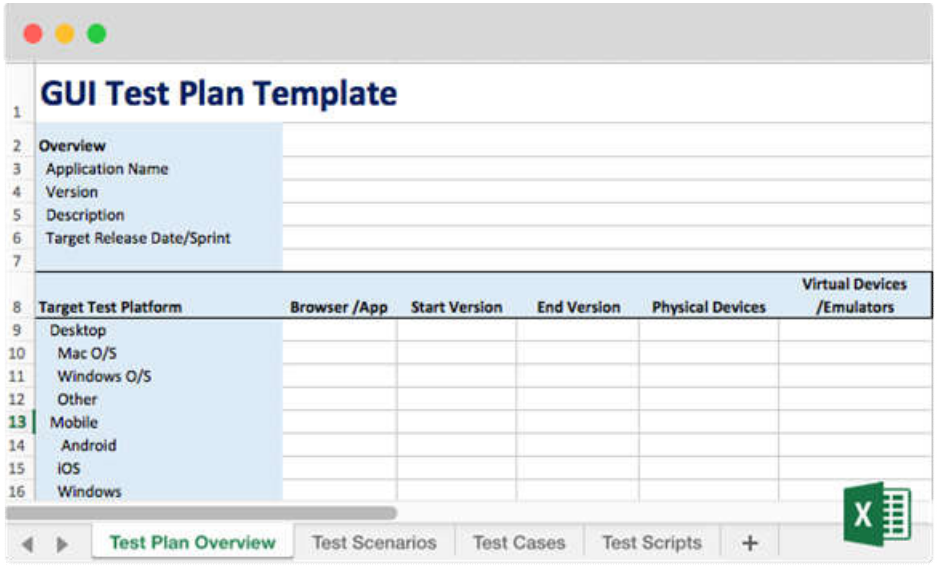
User experience testing challenges

UX testing requires identification and recruitment of user testers that accurately represent the target user base, such as novices, experienced users, young people, older adults, etc. While it is not necessary to have a very large group of users, it is important to cover the expected user personas. Recruiting user testers may be a time-consuming and potentially costly process. Although it is possible for user experience testing to be done by user proxies such as product owners, it can be difficult for them to set aside their knowledge of the AUT and fully step into the role of an end user. Finally, if UX testing is limited to beta testing, this occurs very late in the development cycle when it is expensive to find and fix defects.

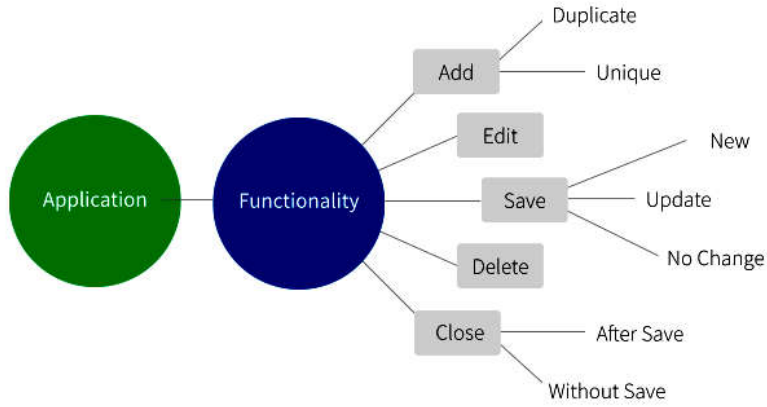
Artifacts for GUI Testing

- Test Plan has to be created
- Test Scenarios has to be created

GUI Test Plan Template



The sample concept map below shows the result of a brainstorming session for a generic application and includes GUI events such as add, edit, delete, save, and close. To create a concept map, testers apply heuristics: their knowledge of the AUT(Application Under Test) combined with general testing principles.



Partial concept map for GUI testing

Partial concept map for GUI testing

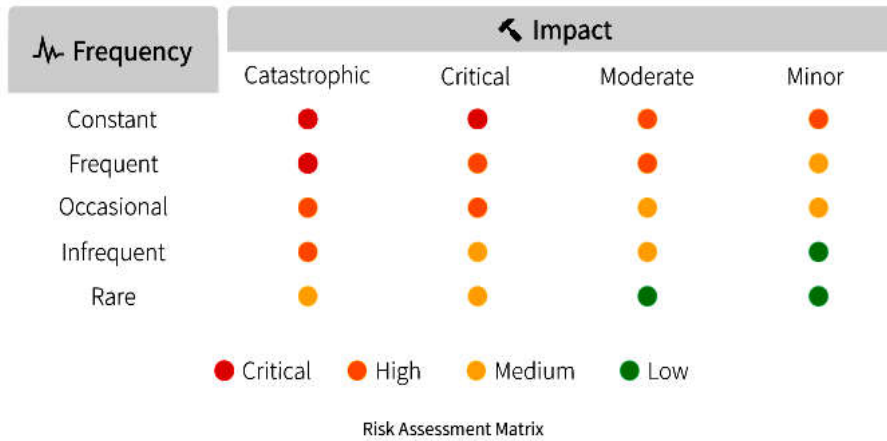
For example, to verify the navigation of a cloud-based application, plan tests such as the following:

Sample areas to test for web navigation

- Compatibility with all common browsers
- Proper functioning of the page when the user clicks the back button or the refresh button
- Page behavior after a user returns to the page using a bookmark or their browser history
- Page behavior when the user has multiple browser windows open on the AUT at the same time.

Prioritize the test cases as the resources for testing are limited and there is mostly a time constraint to deliver functionality. Risk-based testing is about grouping the different test cases based on their impact factor and the probability of occurrences. All test cases with high impact and high probability of occurrences have to executed first.

Ris



Risk Assessment Matrix

For example, let's assume that the password reset process does not work at all. Once a user encounters it, the frequency is "constant" and the effect is "catastrophic" as the user is locked out of the application. Therefore, testing the password reset process is a critical priority. If the testing team determines that there is only enough time to inspect critical and high priority events, then medium events could be addressed through exploratory testing, while low priority events may not be tested at all.

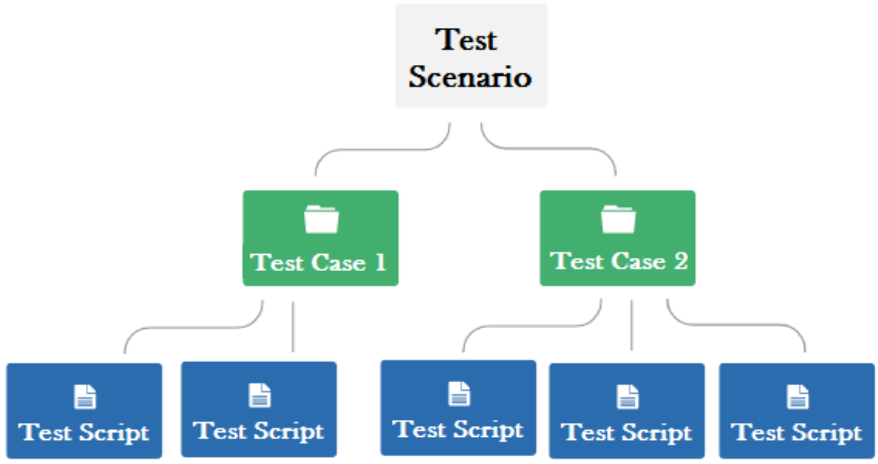
UI Testing Scenarios

When performing a UI test, the QA team needs to prepare a test plan that identifies the areas of an application that should be tested as well as the testing resources available. With this information, testers can now define the test scenarios, create test cases, and write the test scripts.

A test scenario is a document that highlights how the application under test will be used in real life. A simple test scenario in most applications would be, "users will successfully sign in with a valid username or ID and password" In this scenario, we can have test cases for multiple GUI events. This includes when a user:

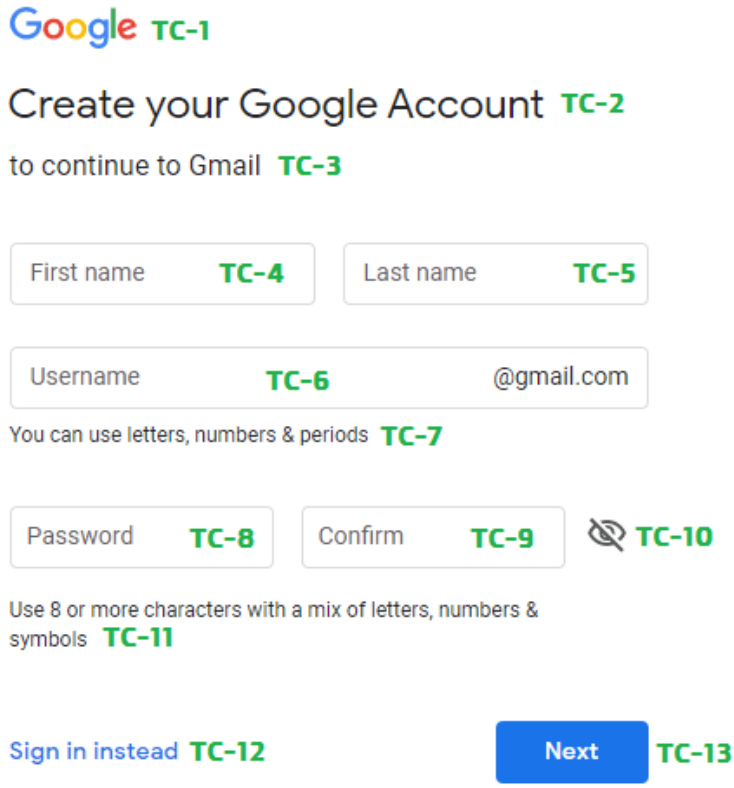
- Provides a valid username and password combination
- Enters an invalid username
- Enters a valid username but an invalid password
- Forgets and tries to reset the password
- Tries to copy a password from the password field
- Tries to copy a password to the password field
- Hits the help button

Although scenarios are not required when creating UI test cases, they guide their development. Basically, they serve as the base from which test cases and test scripts are developed as shown below:



A Practical UI Test

There are many things taken into consideration when testing a user interface. Let's consider this Google sign-up form as an example.



Using the above form, we identify 13 test cases, labelled TC-1 to TC-13. At the very least, we should perform the following UI checks:

TC-1

- Check the page label, position, and font.

TC-2

- Validate whether the page heading is correct.
- Check the font used.

TC-3

- Check the cursor focus on the default field.
- Test the mandatory fields by clicking next while the form is blank.
- Check the position and alignment of the text box.
- Check the acceptance of both valid and invalid characters in the field labels.

TC-4

- Check the position and alignment of the text box.
- Check field labels, validate the acceptance of both valid and invalid characters.

TC-5

- Check the position and alignment of the text box.
- Check field labels, validate the acceptance of both valid and invalid characters.

TC-6

- Test the error message by entering both permitted and prohibited characters.
- Verify error message correctness.

TC-7

- Test pop-ups and hyperlinks.

TC-8

- Check field labels, validate the acceptance of both valid and invalid characters.
- Check the position and alignment of the text box.

TC-9

- Save an unmatched password.
- Check field labels, validate the acceptance of both valid and invalid characters.
- Check the position and alignment of the text box.

TC-10

- Verify icon position.
- Test the icon shows or hides the user password.
- Check the image quality.

TC-11

- Test the error message by entering both permitted and prohibited characters.
- Verify error message correctness.

TC-12

- Test pop-ups and hyperlinks.

TC-13

- Test form submission.
- Check button position and clarity.

Challenges in UI Testing

Software testers face a number of issues when performing UI tests. Some of the most notable challenges include:

- **Constantly changing UI** – It is common to upgrade applications constantly to accommodate new features and functionalities. When upgrades are made frequently, performing comprehensive UI tests becomes a challenge.
- **Increasing testing complexity** – Modern applications have significantly complex features including embedded frames, complex flowcharts, maps, diagrams, and other web elements. This makes UI tests to become more challenging.
- **UI tests can be time-consuming** – creating effective UI test scripts and executing the tests can take time especially when a tester is not using the right tool.
- **Maintaining UI test scripts** – As developers make changes to the user interface, it becomes challenging to maintain the test scripts.
- **Handling multiple errors** – When performing complex UI tests under tight timelines, testers spend a lot of time creating scripts. In such scenarios, fixing errors during the testing process becomes a challenge.
- **Computing the ROI for UI test automation** – Since the UI keeps changing, so do the tests change. This increases the amount of time spent on UI testing, thereby delaying the delivery process. In the end, it becomes difficult to calculate the ROI for continuously performing UI tests.

Tips to overcome the UI challenges.

1. Select the right UI test automation tool

The first step when resolving software testing challenges is choosing the right automation tool. There are various testing tools in the market that you can use for your project. However, focus on choosing one that integrates seamlessly with your workflow. A great UI automation tool has record/playback capabilities, supports reusable tests, and requires minimal maintenance. It also supports reporting and has defect tracking capabilities.

2. Utilize an object repository

One approach for reducing test maintenance and the associated costs is using a shared repository. It is also a great idea to reduce the number of UI test cases during the initial testing stages, then increase the coverage as you move forward. This ensures a higher success rate in your test cases.

3. Consider codeless automation testing tools

To eliminate the trouble of making repetitive changes in the test code, developers and QA teams should leverage the power of codeless automation. If you're a Selenium fan, for instance, Perfecto Scriptless will automate your entire test creation and execution process, thereby saving you a great deal of time and cost.

4. Organizational code review standards

The coding culture of an enterprise has a significant impact on how well their teams address testing challenges in the application development cycle. For this reason, organizations should focus on training their teams on the best test automation practices, so there are specific criteria for code review or modifications across the enterprise. A good approach would be engaging test automation experts in some intense brainstorming sessions.

CHAPTER 2- INTRODUCTION TO TEST SCRIPTS

Test script is a line-by-line instruction that has to be executed to test the application under test.

It is different from test cases. A test script provides a clearly defined procedure for a tester to follow. The test script may include information such as the following:

Test Script ID	a unique identifier for the test script.
Title	the title of the test script, such as "User enters a valid username and password, maximum length."
Test Case ID	a link to the unique ID for the test case.
Test Setup	the requirements for the test environment; could be stored separately in a test data spreadsheet.
Data	either the literal values for the tester to enter; or a link to an external spreadsheet or database containing combinations of usernames and passwords to test.

Procedure	the step-by-step instructions for the tester, such as the following example: 1. Start the AUT. The log on screen appears. 2. Click on the username field. 3. Enter the first username from the spreadsheet. 4. Enter the password from the spreadsheet. 5. Click the Log On button.
Actual Result	completed by tester after testing.
Status	pass/fail/blocked status of the test script.
Defect Cross-Reference*	if a defect is found, enter the code from the defect tracking system here, to connect the test case with the defect.

Here are the main difference between Test Cast and Test Script:

Test Case	Test Script
Test case is a step by step procedure that is used to test an application.	The test script is a set of instructions to test an application automatically.
Test Cases are used for manual testing environment.	Test Script is used in the automation testing environment.
It is done manually.	It is done according to the scripting format.
The test case template includes Test ID, test data, test procedure, actual and expected results, etc.	In the Test Script, we can use different commands to develop a script.

TEST SCRIPTS CREATION TECHNIQUES

Record/playback:

In this method, the tester needs to write any code instead of just to record the user's actions. However, the tester will need to do coding to fix things that go wrong or fine-tune the automation behavior.

This method is easier than writing a complete test script from scratch because you already have the complete code. It is mostly used in a simplified programming language such as VBScript.

Keyword/data-driven scripting:

In this method, there is a clear separation between testers and developers. In data-driven scripting, the tester defines the test using keywords without knowledge of the underlying code.

Here, the developers' job is to implement the test script code for the keywords and update this code when needed. So in this method, the tester need not worry about the system. However, they will highly rely upon development resources for any new functionality you want to test automatically.

Writing Code Using the Programming Language:

If you like to create test script using this method, you will typically still have the ability to record or playback and generate a simple script.

Although, as a tester, you finally need to go beyond record/playback and learn how to code simple scripts. It is important to understand that you can choose your programming language even if your application is written in Java.

However, it does not mean that you need to write your test scripts in Java, which can be difficult to learn. Instead, you can write your test scripts in an easier language like JavaScript or Ruby (or any easier language you wish to use).

Example of a Test script

For example, to check the login function on a website, your test script might do the following:

- Specify how the automation tool can locate the "Username" and "Password" fields in the login screen. Let us say, by their CSS element IDs.
- Load the website homepage, then click on the "login" link. Verify that the Login screen that appears and the "Username" and "Password" fields are visible.
- Next, type the username "Charles" and password "123456" identify the "Confirm" button and click it.
- They need to specify how a user can locate the title of the Welcome screen that appears after login- say, by its CSS element ID.
- Verify that the title of the Welcome screen is visible.

- Read the title of the welcome screen.
- Insert that the title text is “Welcome Charles”.
- If the title text is as per the expectation, a record that the test passed. Otherwise, an album that the test failed.

Test Script Template Example in Excel

Functional Area	Test Name	Pre-requisites	Test Steps	Expected Results	Pass/Fail	Actual Result (if fail)
(Example) Sign-up	Sign-in success	Valid credentials required	1. Enter valid username & password 2. Click Login	Home page displayed after successful login		

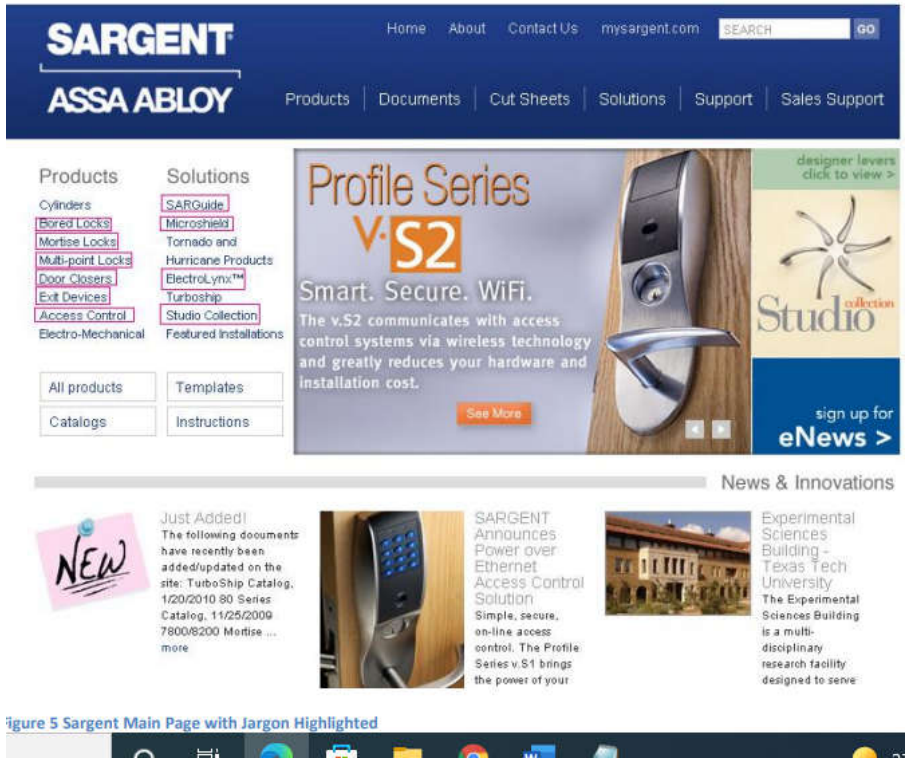
Selenium Automation Tool Code for automating login page in chrome using Selenium web driver

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
public class LoginAutomation {
@Test
public void login() {
System.setProperty("webdriver.chrome.driver", "path of driver");
WebDriver driver=new ChromeDriver();
driver.manage().window().maximize();
driver.get("https://www.browserstack.com/users/sign_in");
WebElement username=driver.findElement(By.id("user_email_Login"));
WebElement password=driver.findElement(By.id("user_password"));
WebElement login=driver.findElement(By.name("commit"));
username.sendKeys("abc@gmail.com");
password.sendKeys("your_password");
```

```
login.click();

String actualUrl="https://live.browserstack.com/dashboard";

String expectedUrl= driver.getCurrentUrl();

Assert.assertEquals(expectedUrl,actualUrl);

}

}
```

On executing the code, Selenium will navigate to the Chrome browser and open the Browserstack login page. Then, it will log in using the relevant credentials. It will also check the test case status using Assert and try to match the URL

Case Study Example: Evaluating a website for its Usability and User Interface



igure 5 Sargent Main Page with Jargon Highlighted

Website Name: Sargent Manufacturing's Homepage
www.sargentlock.com

Usability and User Interface Problems: -

a) Lot of technical jargons which a layman audience will not understand. Industry terms are not explained (e.g., cylindrical, mortise, exit, "access control, etc.), leaving an end user surfing the site to guess through context or research each term as they come across them.

b) These links are the primary way of navigating the site. Without knowing these terms, the customer must follow a guess-and-check method

for finding information on their product; none of these terms have a hover over definition.

c) Not clear image to describe what the product is

d) Appearance is cluttered not providing clarity

e) Positioning of links and menus could be made better

f) Ineffective search

g) Product feature not fully listed

h) No video to demonstrate working of the product

i) No chat bots for solving the queries of the viewers

j) No page for Frequently asked Questions (FAQ's)

k) Non appealing visual, color combinations, fonts, font size etc.

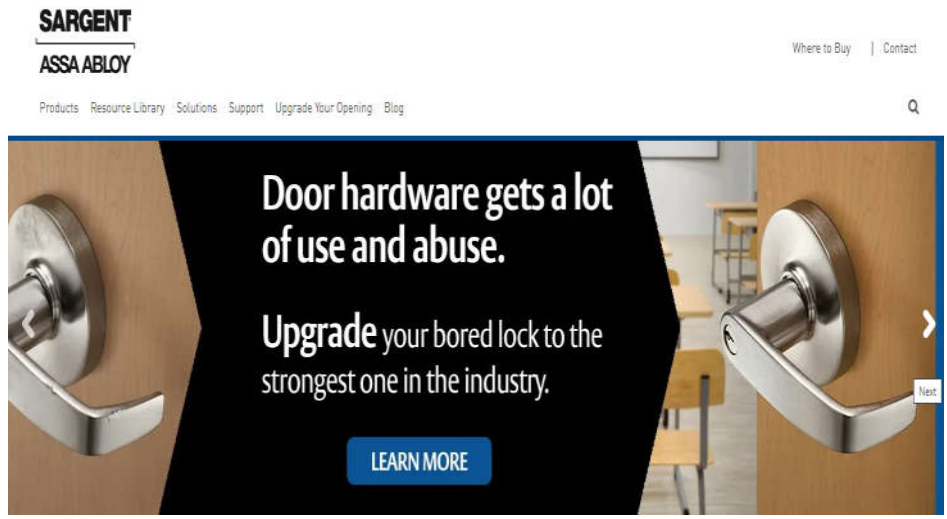
l) They are into selling locks. There are different types and subtypes of locks. All locks details displayed together no grouping. Search is difficult

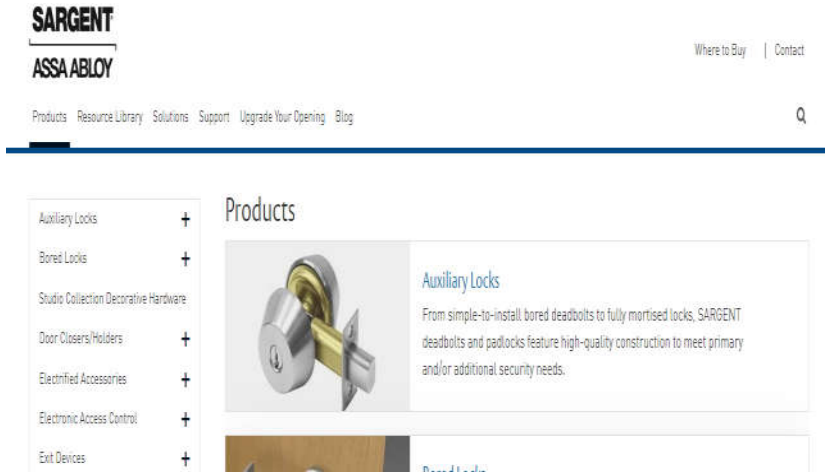
m) Images huge in size so page loading takes lot of time.

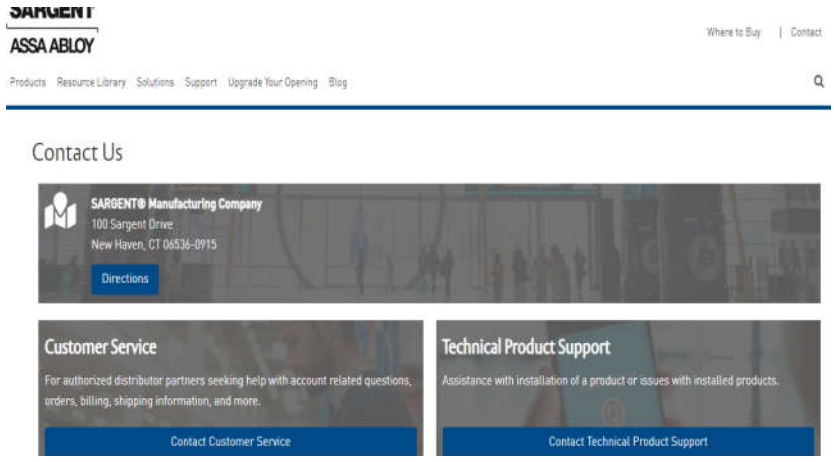
n) Contact detail does not have the complete address and email address. Has only the contact number.

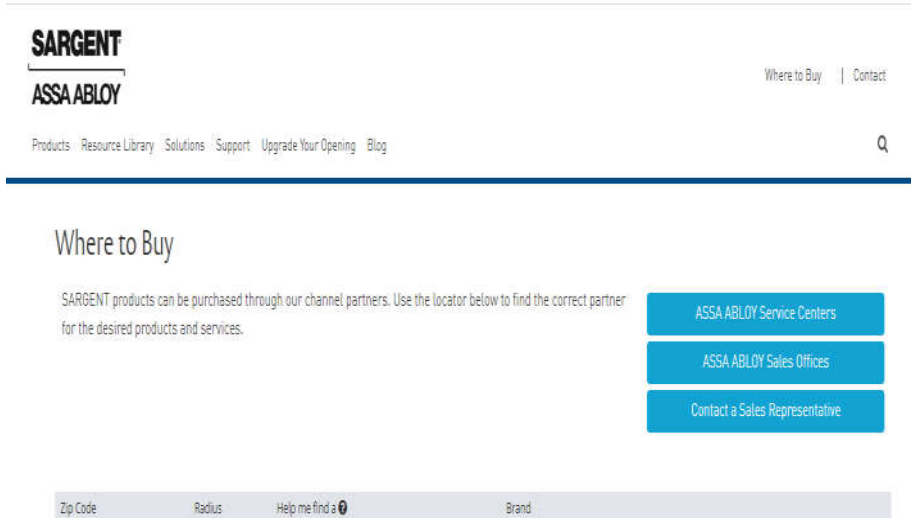
o) No clear details on where to buy their authorized dealers etc.

All the above usability and user interface issues resolved in the new website designed by them. Following is the visuals of new website of sargent:-









All the Usability and User Interface issues discuss above are being overcome in the new website of sargent manufacturing company.

HUMAN COMPUTER EVALUATION EXAMPLES

Choose an appropriate evaluation method for each of the following situations. In each case identify

(i) The participants.
(ii) The technique used.
(iii) Representative tasks to be examined.
(iv) Measurements that would be appropriate.
(v) An outline plan for carrying out the evaluation.

(a) You are at an early stage in the design of a spreadsheet package and you wish to test what type of icons will be easiest to learn.

(b) You have a prototype for a theatre booking system to be used by potential theatre-goers to reduce queues at the box office.

(c) You have designed and implemented a new game system and want to evaluate it before release.

(d) You have developed a group decision support system for a solicitor's office.

(e) You have been asked to develop a system to store and manage student exam results and would like to test two different designs prior to implementation or prototyping.

SOLUTIONS:-

Note that these answers are illustrative; there are many possible evaluation techniques that could be appropriate to the scenarios described.

Spreadsheet package

(i) Subjects	Typical users: secretaries, academics, students, accountants, home users, schoolchildren
(ii) Technique	Heuristic evaluation
(iii) Representative tasks	Sorting data, printing spreadsheet, formatting cells, adding functions, producing graphs
(iv) Measurements	Speed of recognition, accuracy of recognition, user-perceived clarity
(v) Outline plan	Test the subjects with examples of each icon in various styles, noting responses.

Theatre booking system

(i) Subjects	Theatre-goers, the general public
(ii) Technique	Think aloud
(iii) Representative tasks	Finding next available tickets for a show, selecting seats, changing seats, changing date of booking
(iv) Measurements	Qualitative measures of users' comfort with system, measures of cognitive complexity, quantitative measures of time taken to perform task, errors made
(v) Outline plan	Present users with prototype system and tasks, record their observations whilst carrying out the tasks and refine results into categories identified in (iv).

New game system

(i) Subjects	The game's target audience: age, sex, typical profile should be determined for the game in advance and the test users should be selected from this population, plus a few from outside to see if it has wider appeal
(ii) Technique	Think aloud
(iii) Representative tasks	Whatever gameplay tasks there are - character movement, problem solving, etc.
(iv) Measurements	Speed of response, scores achieved, extent of game mastered.
(v) Outline plan	Allow subjects to play game and talk as they do so. Collect qualitative and quantitative evidence, follow up with questionnaire to assess satisfaction with gaming experience, etc.

Group decision support system

(i) Subjects	Solicitors, legal assistants, possibly clients
(ii) Technique	Cognitive walkthrough
(iii) Representative tasks	Anything requiring shared decision making compensation claims, plea bargaining, complex issues with a diverse range of expertise needed.

(iv) Measurements	Accuracy of information presented and accessible, veracity of audit trail of discussion, screen clutter and confusion, confusion owing to turn-taking protocols
(v) Outline plan	Evaluate by having experts walk through the system performing tasks, commenting as necessary.

Exam result management

(i) Subjects	Exams officer, secretaries, academics
(ii) Technique	Think aloud, questionnaires
(iii) Representative tasks	Storing marks, altering marks, deleting marks, collating information, security protection
(iv) Measurements	Ease of use, levels of security and error correction provided, accuracy of user
(v) Outline plan	Users perform tasks set, with running verbal commentary on immediate thoughts and considered views gained by questionnaire at end.

PRACTICAL QUESTIONS

Q1. Choose an appropriate **Evaluation Method** for each of the following situations. In each case suggest the answers in the given spaces below:

You have a prototype for a Admission fees payment process to be used by potential students to reduce physical queues at college office.

(i) Participants _____

(ii) The technique used_____

(iii) Representative tasks to be examined_____

(iv) Measurements that would be appropriate_____

(v) An outline plan for carrying out the evaluation_____

Q2. Consider a Website www.mu.ac.in or any other website of your choice and identify the different User Interface and Usability problems with the website and provide suggestions to improvise the same. Also write Test scenarios and Test Cases for testing the same.

Q3. Design an User Interface for

a) Welcome screen

b) Multiplication and Addition of any two numbers

Q4. Design an user interface for assigning a grade to students based on the subjects marks

Q5. Design an User Interface Screen for Calculator

Q6. Design an User Interface for registration of a student for admissions.

Q7. Design an User Interfaces for ATM Machine

Q8. Design an User Interfaces for for Smart Phone

Q9. Design an User Interface Screen for Online Examination

Q10. Design an User Interfaces for Booking of Movie Tickets

Q11. Design an User Interface Screen for Paintbrush.