**SPORTS MANAGEMENT TOOL**

**A PROJECT REPORT**

**Submitted By**

**ANIL KUMAR SINGH**

**2000290140018**

**Submitted in partial fulfilment of the**

**Requirements for the Degree of**

**MASTER OF COMPUTER APPLICATIONS**

**Under the Supervision of**

**Mr. ANKIT KUMAR**

### Assistant Professor

**( Department of Computer Applications)**



**Submitted to**

**Department Of Computer Applications**

**KIET Group of Institutions, Ghaziabad**

**Uttar Pradesh-201206**

**(13 Jan 2022)**

**CERTIFICATE**

Certified that **Anil Kumar Singh (2000290140018)** have carried out the project work having “**SPORTS MANAGEMENT TOOL**” for **Master of Computer Applications** from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)** (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

**Date- 12/01/2022**

**ANIL KUMAR SINGH**

**(2000290140018)**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

PROF. ANKIT VERMA

ASSISTANT PROFESSOR

**Department of Computer Applications**

**KIET Group of Institutions, Ghaziabad**

**Signature of Internal Examiner** **Signature of External Examiner**

**Dr. Ajay Shrivastava**

**Head, Department of Computer Applications**

**KIET Group of Institutions, Ghaziabad**

ABSTRACT

The focus of this project to provide the sports teams a handy tool to manage their team players data , find new ways to improve their performances and take their game at a level above the others.

The sports teams can register themselves on our website and create their profiles. Then the players can create their personal profiles. Our program takes inputs of the athletes and based on the inputs given , the guided information is provided to the teams in terms of plans for workout routines , resting periods and recovery exercises for injury management and workload management .

ACKNOWLEDGEMENTS

This Project was jointly undertaken by Anil Kumar Singh, under the guidance and supervision of Mr. Ankit Verma. Our primary thanks goes to him, who poured over every inch of our project with painstaking attention and helped us throughout the working of the project. It is our privilege to acknowledge our deepest gratitude to him for his inspiration which has helped us immensely. We are extremely grateful to him for his unstilted support and encouragement in the preparation of this project.

Words are not enough to express my gratitude to Dr. Ajay Kumar Shrivastava, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Anil Kumar Singh**

**List of Chapters**

Chapter 1 - Introduction

1.1 Project description

1.2 Project Scope

1.3 Hardware / Software used in Project

1.3.1 Hardware Specification

1.3.2 Software Specification

Chapter 2 Feasibility Study

2.1 Technical feasibility

2.2 Operational Feasibility

2.3 Behavioural Feasibility

Chapter 3 Software Requirement Specifications

3.1 Database Tables

3.2 Flow Chart

3.3 Use Case Diagram

3.4 Sequence Diagram

3.5 Activity Diagram

3.6 State Diagram

3.7 Component Diagram

Chapter 4 Form Design

4.1 Frontend

4.1.1 HTML

4.1.2 CSS

4.1.3 Bootstrap

4.2 Backend

4.2.1 PHP

4.2.2 Database

Chapter 5 DESIGN AND CODING

Module wise code

5.1 Module Description

5.2 Design Engineering

5.2.1 Architectural Design

5.2.2 Component Design

5.3 CODING

5.3.1 Code for Login Page

5.3.2 Code for Workout Routine Suggestor

Chapter 6 Testing

6.1 Unit Testing

6.2 Integration Testing

6.2.1 Top-Down and Bottom-up Approach

6.3 Software Verification and Validation

6.4 Classification methods

5.7 Test Cases

Chapter 7 References

Chapter-1 INTRODUCTION

The focus of this project to provide the sports teams a handy tool to manage their team players data , find new ways to improve their performances and take their game at a level above the others.

The sports teams can register themselves on our website and create their profiles. Then the players can create their personal profiles. Our program takes inputs of the athletes and based on the inputs given , the guided information is provided to the teams in terms of plans for workout routines , resting periods and recovery exercises for injury management and workload management .

**1.1 PROJECT DESCRIPTION**

The software has two main users ;

* one is the club or sports team that registers itself and the other user is ,
* the athlete which gives the inputs to the software and gets the required benefits.

The software will have different windows for the club and the athlete.

* For club : This window will include the registration portal , profile records , results etc.

For athlete : In this window the attributes like profile , input segments , report generator etc. will be present.

**1.3 HARDWARE/SOFTWARE SPECIFICATIONS**

1.3.1 Hardware Specification

* SSD 256 GB Storage
* 8 GB RAM
* 2.4 GHz AMD Ryzen 5 Processor

1.3.2 Software Specification

* Operating System: Windows 10
* IDE : Visual Studio Code
* XAMPP Server

**Chapter-2 Feasibility Study**

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it’s worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analyzed carefully. There are 3 parts in feasibility study.

**2.1 Technical Feasibility**

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs and procedures. This can be qualified in terms of volume of data, trends, frequency of updating inorder to give an introduction to the technical system. The application is the fact that it has been developed on windows XP platform and a high configuration of 1GB RAM on Intel Pentium Dual core processor. This is technically feasible .The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

**2.2 Operational Feasibility**

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes. To ensure success, desired operational outcomes must be imparted during design and development.

These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters.

A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

**2.3 BEHAVIORAL FESIABILITY**

Establishing the cost-effectiveness of the proposed system i.e. if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need of online social networking facilities. Thus the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

**Chapter-3 Software Requirement Specifications**

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.

**3.1 Overall Description**

The software has two main users ;

* one is the club or sports team that registers itself and the other user is ,
* the athlete which gives the inputs to the software and gets the required benefits.

The software will have different windows for the club and the athlete.

* For club : This window will include the registration portal , profile records , results etc.
* For athlete : In this window the attributes like profile , input segments , report generator etc. will be present.

**3.1.1 PRODUCT FUNCTIONS**

* The club/sports team will be able to register online with a login name and password. This information is verified with the information in the database and the member is appended.
* They’ll be able to add, modify or delete the information provided by them when they register.
* The athlete can enter their profile details , physical measurements , workout routine records etc.
* The athlete will get the details of preferred workout sessions , rest routines etc.
* The teams can maintain the records about their players fitness and injury .

**3.1.2 USER CHARACTERISTICS**

Basic knowledge of using computers is adequate to use this application. Knowledge of how to use a mouse or keyboard and internet browser is necessary. The user interface will be friendly enough to guide the user.

**3.1.3 GENERAL CONSTRAINTS**

* Access to the web is required.
* Only those athletes belonging to the registered clubs can form up a profile and upload details.
* Precise information needs to be provided.

**3.1.4 ASSUMPTIONS AND DEPENDENCIES**

* It is assumed that athlete’s data will be made available for the project’s working. Until then , test data will be used for providing the demo for the presentations.
* It is assumed that the user is familiar with an internet browser and also familiar with handling the keyboard and mouse.
* Since the application is a web based application there is a need for the internet browser. It will be assumed that the users will possess decent internet connectivity.

**3.2 EXTERNAL INTERFACE REQUIREMENTS**

External interface requirements specify hardware, software, or database elements with which a system or component must interface. This section provides information to ensure that the system will communicate properly with external components.

**3.2.1 USER INTERFACE**

* The user must see the login portal first when he/she first opens the website .
* Both the sports team and athlete can enter the essential information for the registration.
* The teams should have a list & data of the players of their team who are enrolled in this program .

**3.3 FUNCTIONAL REQUIREMENTS**

A functional requirement document defines the functionality of a system or one of its subsystems. It also depends upon the type of software, expected users and the type of system where the software is used.

**FR 1 : Registration and login.**

The teams will first register themselves through the registration portal and create their login id.

**FR 2 : Personal profile.**

A personal profile of players will be created with the following information.

**3.3.1 Input personal details**

Personal details such as name, city & sport they play etc. will be entered under the personal details.

**3.3.2: Physical measurements**

Player’s age, height, weight etc.

**3.3.3: Game details**

Details of the game the athlete is playing like type of sport, pitch on which it is played etc.

**FR 3: Fitness routines.**

The exercises athlete should be doing/following in the regular training programme.

**FR 4: Injury Management**

Suggestion of resting periods, mobility exercises, recovery exercises and training frequency in the recovery period.

**3.4 PERFORMANCE REQUIREMENTS**

* The application should be portable and possible to users of Google Chrome as well as Internet Explorer.
* The response time for a particular analysis should be not be greater than 9-10 seconds for a respectable internet connection speed.
* The number of connections to the system should not slow down the application to a large degree.
* The data for the analysis will be obtained from the database of users, so the response time for a query from the client side to the database side should not be more than 5seconds.

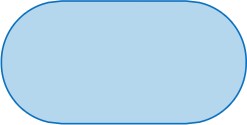
**3.5 FLOW CHART**

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.

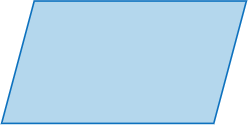
The process of drawing a flowchart for an algorithm is known as “flowcharting”.

## **Basic Symbols used in Flowchart Designs**

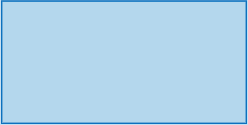
1. **Terminal:** The oval symbol indicates Start, Stop and Halt in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.



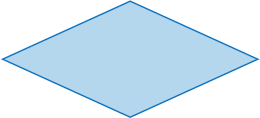
* **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.



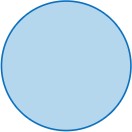
* **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.



* **Decision** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.



* **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.



* **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.

**3.6 DATA FLOW DIAGRAM**

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system software. The DFD also provides information about the outputs and inputs of each entity and the process itself.

**DFD LEVEL-0**

Diagram

Description automatically generated

**Figure 1.5.1 : DFD Level 0**

**DFD LEVEL – 1**

Diagram, schematic

Description automatically generated

**Figure 1.5.2 : DFD Level 1**

**3.6 USE CASE DIAGRAM**

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

* Scenarios in which your system or application interacts with people, organizations, or external systems
* Goals that your system or application helps those entities (known as actors) achieve

**Use case diagram components**

## To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

* **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
* **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
* **Goals:** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

## Use case diagram symbols and notation

The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams.

**Use cases:** Horizontally shaped ovals that represent the different uses that a user might have.

* **Actors:** Stick figures that represent the people actually employing the use cases.
* **Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
* **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
* **Packages:** A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

**3.7 SEQUENCE DIAGRAM**

To understand what a sequence diagram is, it's important to know the role of the [Unified](https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language) [Modeling Language,](https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language) better known as UML. UML is a modeling toolkit that guides the creation and notation of many types of diagrams, including behavior diagrams, interaction diagrams, and structure diagrams.

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Note that there are two types of sequence diagrams: UML diagrams and code-based diagrams. The latter is sourced from programming code and will not be covered in this guide.

## Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

* Represent the details of a UML use case.
* Model the logic of a sophisticated procedure, function, or operation.
* See how objects and components interact with each other to complete a process.
* Plan and understand the detailed functionality of an existing or future scenario.

# 3.8 **ACTIVITY DIAGRAM**

We use **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram. An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system.

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system.

## Activity Diagram Notations –

1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.

UML-State-Diagram

**Figure –** notation for initial state or start state

A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

For example – Here the initial state is the state of the system before the application is opened.



**Figure –** initial state symbol being used

1. **Action or Activity State –** An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

UML-Activity-Diagram

**Figure –** notation for an activity state

For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

**Figure –** activity state symbol being used

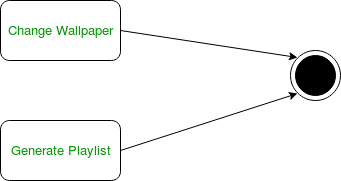


1. UML-Object-Diagram**Action Flow or Control flows –** Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

**Figure –** notation for control Flow

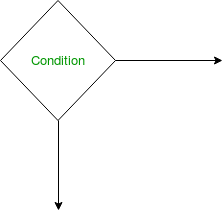
An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.



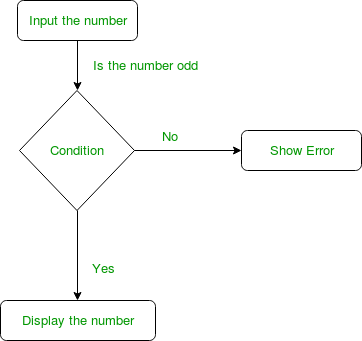
**Figure –** using action flows for transitions

1. **Decision node and Branching –** When we need to make a decision before deciding the flow of control, we use the decision node.

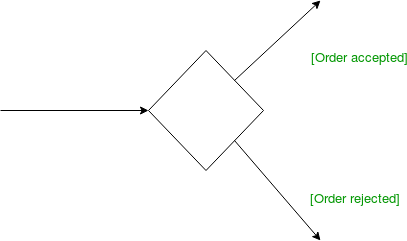


**Figure –** notation for decision node

The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.

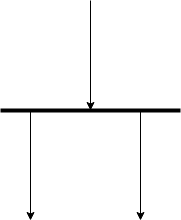


**Figure –** an activity diagram using decision node

1. **Guards –** A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

**Figure –** guards being used next to a decision node

The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

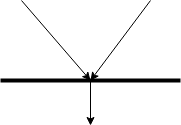
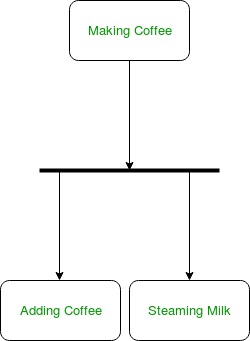
1. **Fork –** Fork nodes are used to support concurrent activities.

**Figure –** fork notation

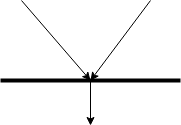
When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.

We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities. For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation

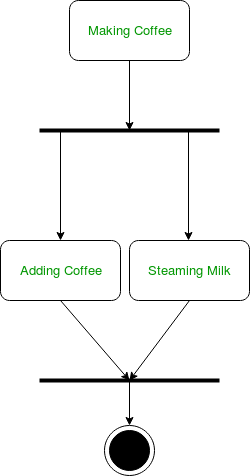
**Figure –** a diagram using fork

1. **Join –** Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

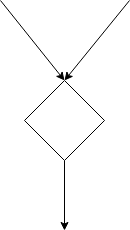
**Figure –** join notation



For example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.



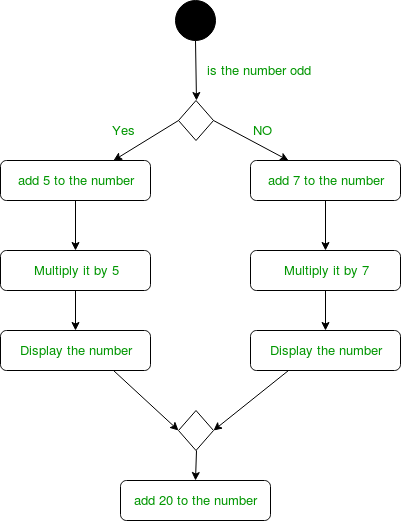
**Figure –** a diagram using join notation

1. **Merge or Merge Event –** Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

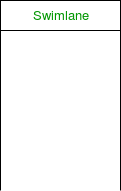
**Figure –** merge notation

For example – In the diagram below: we can’t have both sides executing concurrently, but they finally merge into one. A number can’t be both odd and even at the same time.

**Figure –** an activity diagram using merge notation



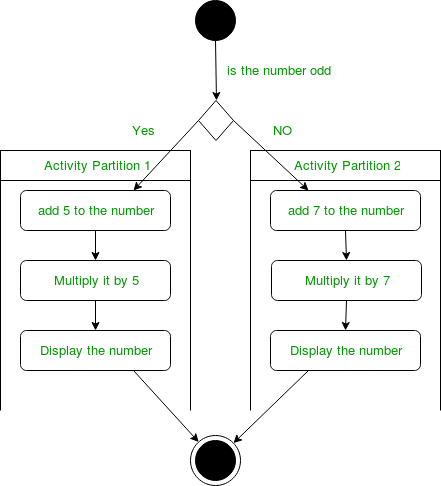
**9. Swimlanes –** We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It’s similar to creating a function in a program. It’s not mandatory to do so, but, it is a recommended practice.



**Figure –** swimlanes notation

We use a rectangular column to represent a swimlane as shown in the figure above.

For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane



**Figure –** an activity diagram making use of swimlanes

## **10. Time Event –**



**Figure –** time event notation

We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event.

For example – Let us assume that the processing of an image takes takes a lot of time. Then it can be represented as shown below



**11. Final State or End State –** The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

UML-State-Diagram

**Figure –** notation for final state

Diagram, schematic

Description automatically generated

**Figure 5.1.1 : Design level DFD**

**CHAPTER 4**

**4.1 Frontend**

The front-end is what a user sees and interacts with (user interface). The back-end is part of the application that is hidden from the user (what some would call, under the hood). This part is responsible for data processing, storing the data, and mathematical operations

**4.1.1 HTML**

The Hyper Text Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. ... HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items.

HTML can embed programs written in a [scripting language](https://en.wikipedia.org/wiki/Scripting_language) such as [JavaScript](https://en.wikipedia.org/wiki/JavaScript), which affects the behaviour and content of web pages. Inclusion of CSS defines the look and layout of content. The [World Wide Web Consortium](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.[[2]](https://en.wikipedia.org/wiki/HTML#cite_note-deprecated-2) A form of HTML, known as [HTML5](https://en.wikipedia.org/wiki/HTML5), is used to display video and audio, primarily using the <**canvas**> element, in collaboration with JavaScript.

**4.1.2 CSS**

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

SS (Cascading Style Sheets) is used to style and layout web pages — for example, to alter the font, colour, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features.

There are three types of CSS which are given below:

* Inline CSS.
* Internal or Embedded CSS.
* External CSS.

**4.1.3 Bootstrap**

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Bootstrap is a potent front-end framework used to create modern websites and web apps. It's open-source and free to use, yet features numerous HTML and CSS templates for UI interface elements such as buttons and forms. Bootstrap also supports JavaScript extensions

**4.2 BACKEND**

The back end refers to parts of a computer application or a program's code that allow it to operate and that cannot be accessed by a user. ... A back-end application or program supports front-end user services, and interfaces with any required resources.

The back-end is part of the application that is hidden from the user (what some would call, under the hood). This part is responsible for data processing, storing the data, and mathematical operations

**4.2.1 PHP**

PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites. It is integrated with several popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP is an open-source scripting language designed for creating dynamic web pages that effectively work with databases. It is also used as a general-purpose programming language.

Most important advantage of PHP is that it's open source and free from cost. It can be downloaded at anywhere and readily available to use for event of web applications. It is platform independent. PHP based applications can run on any OS like UNIX, Linux and Windows, etc.

**CHAPTER-5 DESIGN AND CODING**

**5.1 MODULE DESCRIPTION**

A module is a separate unit of software or hardware. Typical characteristics of modular components include portability, which allows them to be used in a variety of systems, and interoperability, which allows them to function with the components of other systems.

* **Athlete profile manager**

INPUT – Player’s physical details and game details

DESC – Takes the input and sends it to the next module

OUTPUT – Details of athlete

* **Club profile manager**

INPUT – Athlete’s record & club registration

DESC – Maintains athlete’s records

OUTPUT – Records of athletes

* **Workout routine suggestor**

INPUT – Info about current workout routines

DESC – Analysis of workout routines

OUTPUT – New workout routines

* **Injury surveillance**

INPUT – Injury details

DESC – Provides information about resting exercises

OUTPUT – Returns recovery exercises

**DATA DICTIONARY**

|  |  |
| --- | --- |
| **DATA** | **DESCRIPTION** |
| Sports Club Info (1) | Club Name + Club Code + City |
| Club Name | String |
| Club Code | Digits(4) |
| City | String |
| Sports Club Info (2) | Club Code + Password |
| Password | Alphanumeric |
|  |  |
| Athlete Info | Name + D.O.B + Age + Phone no. + Club Name |
| Name | F name + L name |
| F name | String |
| L name | String |
| D.O.B | Date + Month + Year |
| Date | Digits (2) |
| Month | Digits (2) |
| Year | Digits (4) |
| Age | Digits (2) |
| Gender | Character |
| Phone no. | Digits (10) |
| Email | [character + character +… +special character] |
| Name of sport | String |
|  |  |
| Endurance hours | Digits(2) |
| Flexibility hours | Digits(2) |
| Strength hours | Digits(2) |
| Endurance days | Digits(2) |
| Flexibility days | Digits(2) |
| Strength days | Digits(2) |
|  |  |
| String | [character + character + character + …] |
| Character | [A-Z | a-z] |
| Digit | [0-9] |
| Alphanumeric | [character + character + special character + …] |
| Special character | [ !@#$ ] |

**5.2 DESIGN ENGINEERING**

Engineering Design is the process of devising a system, component or process to meet the desired needs. It is a decision making process (often iterative), in which the basic science and mathematics and engineering sciences are applied to convert resources optimally to meet a stated objective.

**5.2.1 ARCHITECTURAL DESIGN**

IEEE defines architectural design as 'the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

Diagram, schematic

Description automatically generated

**5.2.2 COMPONENT LEVEL DESIGN**

In software engineering, after the planning stage of an application or system, called requirements modelling , the architectural design of the software follows. At this point it is designed on a higher level. After that, the process of taking the components identified in the architectural design and getting down to a 'nuts and bolts' level of designing the proposed software is called component-level design. This level of design defines the interface, algorithms, data structures and communication methods of each component.

**LEVEL – 1 FACTORING**

The first-level factoring is straight forward , after the most abstract input and output data items are identified in the data flow diagram. The main module is the overall control module , which will form the main program or procedure in the implementation of the design.

A picture containing diagram

Description automatically generated

**LEVEL – 2 FACTORING**

##### **Factoring of input, output, and transform branches**

First level Modules must be factored into subordinate modules that will distribute the work of a module. Each of the input, output, and transformation modules must be considered for factoring.

Diagram

Description automatically generated

**5.3 CODING**

**5.3.1 CODE FOR LOGIN PAGE**

<html>

<head>

    <title>

        LOG IN | COLLEGE SPORTS MANAGEMENT

    </title>

    <style>

    p.serif {

  font-family: "Comic Sans MS", Times, serif;

  font-size: 2.5em; /\* 40px/16=2.5em \*/

  color: #FF0000;

  font-weight: bold;

}

    </style>

</head>

<body style="background-image:url('ground.jpg');">

<table align="center" border="1" width="1290">

<tr>

<td height="300" width="200" bgcolor="#FF0000" COLSPAN="4" style="background-image:url(clipart-sporty-9.jpg);">

<marquee scrolldelay="100">

<body>

</body>

<p class="serif" >COLLEGE SPORTS MANAGEMENT TOOL</p>

</tr>

<tr>

<td width="100" align="center"><a href="index.html">HOME</A></td>

<td width="100" align="center"><a href="index.html">ABOUT US</A></td>

<td width="100" align="center"><a href="index.html">FEATURES</A></td>

<td width="100" align="center"><a href="index.html">HELP</A></td>

</tr>

<tr>

<td width="100" height="200" style="background-image:url('events.jpg');>

<MARQUEE DIRECTION="up" bgcolor="orange" SCROLLDELAY="200">

NEW FEEDS<BR>

NEW FEEDS<BR>

NEW FEEDS<BR>

NEW FEEDS<BR>

NEW FEEDS<BR>NEW FEEDS<BR>NEW FEEDS<BR>NEW FEEDS<BR>NEW FEEDS<BR>NEW FEEDS<BR>NEW FEEDS<BR>

</marquee></td>

<td width="100" bgcolor="orange" COLSPAN="3"><marquee direction="up" scrolldelay="200"><font size="5" color="brown" >The focus of this website to provide the sports teams a handy tool to manage their team players data , find new ways to improve their performances and take their game one level above the others.<br>

The sports teams can register themselves on our website and create profile for their players , our programme takes inputs of the athletes and based on the inputs given , the help is provided to the teams in terms of plans of workout sessions, injury management and workload management.</font></marquee></td>

</tr>

</table>

<form action="post.php" method="post" align=center>

            Enter Details:

            <br>

            Name :

            <br>

            <input type = "text" placeholder = "Name" name = "name" required>

            <br>Dob

            <br>

            <input type = "date" placeholder = "DoB" name = "dob" required>

            <br>

            Gender

            <br>

            <input type = "text" placeholder = "Gender" name = "Gender" required>

            <br>

            Phone no.

            <br>

            <input type = "text" placeholder = "Phone no." name = "Phone" required>

            <br>

            Email

            <br>

            <input type = "text" placeholder = "Email" name = "Email" required>

            <br>

            Name of sport

            <br>

            <input type = "text" placeholder = "Name of sport" name = "sport" required>

            <br>

            Name of club

            <br>

            <input type = "text" placeholder = "Name of club" name = "club" required>

            <br>

            <br>

            Enter Username :

                <br>

                    <input type = "text" placeholder = "Username" name = "usr" required>

            <br>

            <br>

            Enter Password :

                <br>

                    <input type = "password" placeholder = "Password" name = "pwd" required>

            <br>

            <br>

                        Click Below To Sign In :

                    <br>

                    <input type = "SUBMIT" name= "signin" value ="Sign In" style="font-weight:bold">

            <br>

            <br>

            Reset The Fields :

            <br>

            <input type = "RESET" value = "Reset" style="font-weight:bold">

            <br>

            <br>

            </form>

</body>

</html>

**5.3.2CODE FOR WORKOUT ROUTINE SUGGESTOR**

<html>

<title></title>

<head> <h1>WORKOUT ROUTINE SUGGESTOR</h1></head>

<body>

ENTER THE FOLLOWING DETAILS

<form method='POST'>

Enter the no of Endurance days

<input type="number" name="ed">

Enter the no of Endurance hours

<input type="number" name="eh">

<br>

Enter the no of Flexibility days

<input type="number" name="fd">

Enter the no of Flexibilty hours

<input type="number" name="fh">

<br>

Enter the no of Strength days

<input type="number" name="sd">

Enter the no of Strength hours

<input type="number" name="sh">

<br>

Age

<input type="nunmber" name="age">

<br>

<input type ="submit" value="SELECT" name="b1">

</form>

</body>

</html>

<?php

if(isset($\_POST['b1']))

{

    $ed=$\_POST['ed'];

    $fd=$\_POST['fd'];

    $sd=$\_POST['sd'];

    $eh=$\_POST['eh'];

    $fh=$\_POST['fh'];

    $sh=$\_POST['sh'];

    $age=$\_POST['age'];

    $favgWL=9.33;

    $savgWL=15.5;

    $tavgWL=19;

    $eWL=$ed\*$eh\*4;

    $fWL=$fd\*$fh\*2;

    $sWL=$sd\*$sh\*3;

    $avgWL=($eWL+$fWL+$sWL)/3;

//  $age\_c=4;

//  if($age<15)

    //      $age\_c=1;

//  }else

//  if ($age>=15 && $age<=24)

//  {

//      $age\_c=2;

 //    }else $age\_c=3;

switch ($age)

{

    case "14": if($avgWL>(1.1\*$favgWL) || $avgWL<(0.8\*$favgWL) )

    {

        echo " Endurance hours=1.5 Endurance days=3 <br> Flexibility hours=1 Flexibilty days=2 <br> Strength hours=1 Strength days=2";

    }else

    {

        echo "<br> Endurance hours=$eh Endurance days=$ed <br> Flexibility hours=$fh Flexibilty days=$fd <br> Strength hours=$sh Strength days=$sd";

    }

    case "17":  if($avgWL>(1.1\*$savgWL) || $avgWL<(0.8\*$savgWL) )

    {

        echo "<br> Endurance hours=2 Endurance days=3 <br> Flexibility hours=1.5 Flexibilty days=3 <br> Strength hours=1.5 Strength days=3";

    }else

    {

        echo " <br>Endurance hours=$eh Endurance days=$ed <br> Flexibility hours=$fh Flexibilty days=$fd <br> Strength hours=$sh Strength days=$sd";

    }

    case "23" :  if($avgWL>(1.1\*$tavgWL) || $avgWL<(0.8\*$tavgWL) )

    {

        echo "<br> Endurance hours=2.5 Endurance days=3 <br> Flexibility hours=1.5 Flexibilty days=3 <br> Strength hours=2 Strength days=3";

    }else

    {

        echo " <br>Endurance hours=$eh Endurance days=$ed <br> Flexibility hours=$fh Flexibilty days=$fd <br> Strength hours=$sh Strength days=$sd";

    }

    default :  echo "<br> Endurance hours=$eh Endurance days=$ed <br> Flexibility hours=$fh Flexibilty days=$fd <br> Strength hours=$sh Strength days=$sd";

}

}

echo "<br>Inside wr suggestor";

?>

**CHAPTER 6 TESTING**

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Any engineered product can be tested using any of the following two approaches:

**6.1 Unit Testing**

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control

data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing. Ideally, each test case is independent from the others. Substitutes such

as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

## **6.2 INTEGRATION TESTING**

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase

in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its

input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing

## **6.2.1 Top-down And Bottom-up**

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage. Top-down testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module. Sandwich testing is an approach to combine top down testing with bottom up testing.

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control

data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for component testing. Ideally, each test case is independent from the others. Substitutes such

as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

## **6.3 SOFTWARE VERIFICATION AND VALIDATION**

**Introduction**

In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle.

Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements.This is done through dynamic testing and other forms of review.Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

* Validation : Are we building the right product?
* Verification : Are we building the product right?

According to the Capability Maturity Model (CMMI-SW v1.1)

Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that "you built it right". Software validation ensures that "you built the right thing".

Software validation confirms that the product, as provided, will fulfill its intended use.

From Testing Perspective

* Fault – wrong or missing function in the code.
* Failure – the manifestation of a fault during execution.
* Malfunction – according to its specification the system does not meet its specified functionality

Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required. Within the modeling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

* M&S Verification is the process of determining that a ⦁ computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.
* M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).

## **6.4 Classification of Methods**

In mission-critical software systems, where flawless performance is absolutely necessary, formal methods may be used to ensure the correct operation of a system. However, often for non- mission-critical software systems, formal methods prove to be very costly and an alternative method of software V&V must be sought out. In such cases, syntactic methods are often used.

## **6.4.1 Test Cases**

A test case is a tool used in the process. Test cases may be prepared for software verification and software validation to determine if the product was built according to the requirements of the user. Other methods, such as reviews, may be used early in the life cycle to provide for software validation.

## **Black-Box Testing**

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

**Test Procedure**

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposed to do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

## **Test Cases**

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are

primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output, often with the help of

an oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

## **White-Box Testing**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT). White-box testing can be applied at the unit, integration and system levels of the software testing process.

Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test.

Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

## **Levels**

1. **) Unit testing :** White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.
2. **) Integration testing :** White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.

**3) Regression testing :** White-box testing during regression testing is the use of recycled white- box test cases at the unit and integration testing levels.

**FLOW GRAPH**

Diagram

Description automatically generated

**INDEPENDENT PATHS**

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined. The value of V(G) provides the upper bound on the number of linearly independent paths through the program control structure.

**Basis Path Set:**

1. 1,2,3,4,5,6,7,8,9,13,14,15,1
2. 1,2,3,4,5,6,7,8,9,13,14,16,1
3. 1,2,3,4,5,6,7,8,9,13,17,18,1
4. 1,2,3,4,5,6,7,8,9,13,17,19,1
5. 1,2,3,4,5,6,7,8,9,13,20,21,1
6. 1,2,3,4,5,6,7,8,9,13,20,22,1
7. 1,2,3,4,5,6,7,8,10,11,13,14,15,1
8. 1,2,3,4,5,6,7,8,10,12,13,14,15,1
9. 1,2,3,4,5,6,7,8,10,11,13,17,18,1

**CYCLOMATIC COPMLEXITY**

1. Closed region = 9
2. Euler’s formula = edges – nodes + 2= 23 -16 + 2=9
3. No. of predicate nodes + 1= 8+1=9

**6.5 TEST CASES**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TEST**  **NO.** | **TEST CASE DESCRIPTION** | **INPUT DESCRIPTION** | **INPUT** | **EXPECTED OUTPUT** | **ACTUAL**  **OUTPUT** | **RESULT**  **(PASS/FAIL)** |
| 1. | ‘First Name’ should not take numbers as valid input | Entering numbers in First Name | Abcd21 | Does not take the numbers and displays nothing if user presses a number key | Does not take the numbers and displays nothing if user presses a number key | PASS |
| 2. | ‘First Name’ should be required i.e. user cannot register without entering this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 3. | ‘Club Name’ should not take numbers as valid input | Entering numbers in Club Name | Efg33 | Does not take the numbers and displays nothing if user presses a number key | Does not take the numbers and displays nothing if user presses a number key | PASS |
| 4. | ‘Last Name’ should not take numbers as valid input | Entering numbers in Last Name | Xyz22 | Does not take the numbers and displays nothing if user presses a number key | Does not take the numbers and displays nothing if user presses a number key | PASS |
| 5. | ‘Last Name’ should be required i.e. user cannot register without entering this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | FAIL |
| 6. | Club code should be required i.e. user cannot register without entering this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 7. | ‘Gender’ should be required i.e. user cannot register without selecting this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 8. | ‘Date of Birth’ should not take invalid date and month values | Enter invalid day and month values | 33-14-2016 | Change the values to the maximum valid value in each case i.e. 31 for day and 12 for month | Change the values to the maximum valid value in each case i.e. 31 for day and 12 for month | PASS |
| 9. | ‘Date of Birth’ should not take invalid day values for respective months | Enter invalid day value for respective months | 30-02-2018 | Shows a pop up: ‘Please enter a valid value. This field is incomplete or has an invalid value’ | Shows a pop up: ‘Please enter a valid value. This field is incomplete or has an invalid value’ | FAIL |
| 10. | ‘Date of Birth’ should be required i.e. user cannot register without selecting this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 11. | ‘Name of club’ should be required i.e. user cannot register without selecting this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 12. | ‘Password’ should be required i.e. user cannot register without selecting this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 13. | ‘Confirm Password’ must match the ‘Password’ field | Entering different data than the one entered in the ‘Password’ field | (Assuming password to be ‘Hello213’)  ‘Hellu213’ | Shows a pop-up: ‘Password entered does not match!!!’ | Shows a pop-up: ‘Password entered does not match!!!’ | PASS |
| 14. | ‘Confirm Password’ should be required i.e. user cannot register without selecting this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 15. | ‘Email’ must contain an ‘@’ | Entering an email without an @ | abcgmail.com | Shows a pop-up: ‘Please include an ‘@’ in the email address. ‘abcgmail.com’ is missing an ‘@’ | Shows a pop-up: ‘Please include an ‘@’ in the email address. ‘abcgmail.com’ is missing an ‘@’ | FAIL |
| 16. | ‘Email’ must contain a part following the ‘@’ | Entering an email without a part following the ‘@’ | abc@ | Shows a pop-up: ‘Please enter a part following the ‘@’. ‘abc@’ is incomplete | Shows a pop-up: ‘Please enter a part following the ‘@’. ‘abc@’ is incomplete | PASS |
| 17. | ‘Email’ should be required i.e. user cannot register without selecting this field | Leaving field empty and attempt to register | NULL | Shows a pop-up: ‘Please fill out this field’ | Shows a pop-up: ‘Please fill out this field’ | PASS |
| 18. | ‘Phone no.’ field should not take alphabets/special characters as valid inputs | Entering alphabets in ‘Phone no.’ | 98871aa120 | Does not take the input and displays nothing if alphabets/special characters are entered | Accepts character values | FAIL |
| 19. | ‘Phone no.’ field should not take more than 10 digits | Entering more than 10 digits in the field | 87891345790 | Stops taking input after 10 digits | Stops taking input after 10 digits | PASS |

**7 . REFRENCES**

#### Software Engineering- A Practitioner’s Approach by Roger S. Pressman: 6th Edition McGraw Hill, 2005

1. An Integrated Approach to Software Engineering by Pankaj Jalote: 3rd Edition Springer, 2005

* htt[ps://www.tutorialspoint.com/index.htm](http://www.tutorialspoint.com/index.htm)
* htt[ps://www.javatpoint.com](http://www.javatpoint.com/)
* htt[ps://www.w3schools.com](http://www.w3schools.com/)
* https://html.com