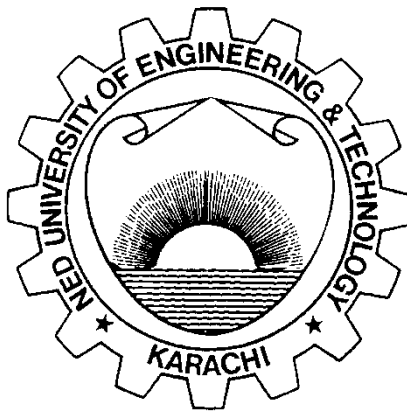


Practical Workbook

CS-326

Software Engineering



Name _____

Year _____

Batch _____

Roll No _____

Department: _____

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

Practical Workbook

CS-326

Software Engineering



Prepared by:

Ms. Fakhra Aftab

Revised in:

September 2019

**Department of Computer & Information Systems Engineering
NED University of Engineering & Technology**

INTRODUCTION

This workbook has been compiled to assist the conduct of practical classes for CS-326 Software Engineering. Practical work relevant to this course aims at providing students a chance to learn the complete Software Development Life Cycle (SDLC). SDLC has to be efficiently organized, and it is for this very reason that CASE (Computer Aided Software Engineering) tools are developed. With the help of CASE, the entire process can be automated and coordinated within the developed and adopted system life cycle. Therefore, variety of different example tools is covered in this workbook. In this way, students will be able to interact with modern CASE tools and can fully automate SDLC.

The Course Profile of CS-326 Software Engineering lays down the following Course Learning Outcome:

“Demonstrate the use of modern tools and techniques for software development and testing.”

All lab sessions of this workbook have been designed to assist the achievement of the above CLO. A rubric to evaluate student performance has been provided at the end of the workbook.

Lab session 1 explains the software documentation with the basic features of very powerful documentation tool ‘LaTeX’. Lab Session 2 covers Project Management tool ‘MS-Project’ for creating project plans. Lab sessions 3 - 6 are about learning the significance of Unified Modeling Language (UML) Diagrams in SDLC. These diagrams are developed using an open source tool named as ‘StarUML’. Lab session 7 discusses the Software Design Patterns. Lab session 8 deals with Program Testing Techniques in SDLC. Lab sessions 9 - 11 demonstrate Web Development & Testing using Agile Project Management (Scrum). Lab sessions 12 & 13 elaborate Version Controlling System via Git & GitHub. Lab session 14 explains Complex Engineering Activity. Appendix A covers more features of LaTeX. Rubric sheets for student’s evaluation are also attached.

CONTENTS

Lab Session No.	Title	Page No.	Teacher's Signature	Date
1	Explore the usage of any documentation tool in Software Development Life Cycle (SDLC)	1		
2	Practice any project management tool to prepare a project plan	11		
3	Carry out user view and structural view analysis for the suggested system: Use Case & Class Diagrams	19		
4	Practice function oriented diagram for the suggested system: Data Flow Diagram	25		
5	Practice behavioral view diagrams for the suggested system: State Transition, Sequence and Collaboration Diagrams	31		
6	Practice Collaboration and Deployment View Diagrams for the suggested system	39		
7	Use Design Patterns in SDLC	45		
8	Use the principles of program testing in SDLC	53		
9	Practice Web Development & Testing using Agile Project Management (Scrum)	59		
10	Demonstrate first sprint and plan second sprint of Web Development & Testing using Scrum	65		
11	Demonstrate second sprint and plan second sprint of Web Development & Testing using Scrum	71		
12	Explore Code repository tools for Version Controlling System (VCS)	77		
13	Practice conflict resolution for multiple contributors in VCS	83		
14	Complex Engineering Activity	87		
	Appendix A: LaTeX Formatting Features	90		
	Grading Rubric Sheet			

Lab Session 01

Explore the usage of any documentation tool in Software Development Life Cycle (SDLC)

Software documentation

All large software development projects, irrespective of application, generate a large amount of associated documentation. A high proportion of software process costs is incurred in producing this documentation. Furthermore, documentation errors and omissions can lead to errors by end-users and consequent system failures with their associated costs and disruption. Therefore, managers and software engineers should pay as much attention to documentation and its associated costs as to the development of the software itself. The documents associated with a software project and the system being developed, have a number of associated requirements:

1. They should act as a communication medium between members of the development team.
2. They should be a system information repository to be used by maintenance engineers.
3. They should provide information for management to help them plan, budget and schedule the software development process.
4. Some of the documents should tell users how to use and administer the system.

Types of Software Documentation

Generally, the software project documentation produced falls into two classes:

1. **Process documentation:** These documents record the process of development and maintenance. *Plans, schedules, process quality documents and organizational and project standards are process documentation.* The major characteristic of process documentation is that most of it becomes outdated. Plans may be drawn up on a weekly, fortnightly or monthly basis. Progress will normally be reported weekly. Although of interest to software historians, much of this process information is of little real use after it has gone out of date and there is not normally a need to preserve it after the system has been delivered.
2. **Product documentation:** This documentation describes the product that is being developed. *System documentation describes the product from the point of view of the engineers developing and maintaining the system; user documentation provides a product description that is oriented towards system users.* Unlike most process documentation, it has a relatively long life. It must evolve in step with the product, which it describes. Product documentation includes user documentation, which tells users how to use the software product and system documentation, which is principally intended for maintenance engineers. See Figure 2.1 for different types of User Documents.

System documentation consists of the following:

- Software Requirement Specification (SRS)

- Design and Architecture
- Source Code Documents
- Testing Documents
- Formal Technical Reviews

User documentation includes the following:

- End-users Documentation
- System-admin Documentation

Document structure

The document structure is the way in which the material in the document is organized into chapters and, within these chapters, into sections and subsections. Document structure has a major impact on readability and usability. As with software systems, you should design document structures so that the different parts are as independent as possible. The IEEE standard for user documentation proposes that the structure of a document should include the components shown in Figure 2.2.

Component	Description
Identification data	Data such as a title and identifier that uniquely identifies the document.
Table of contents	Chapter/section names and page numbers.
List of illustrations	Figure numbers and titles
Introduction	Defines the purpose of the document and a brief summary of the contents
Information for use of the documentation	Suggestions for different readers on how to use the documentation effectively.
Concept of operations	An explanation of the conceptual background to the use of the software.
Procedures	Directions on how to use the software to complete the tasks that it is designed to support.
Information on software commands	A description of each of the commands supported by the software.
Error messages and problem resolution	A description of the errors that can be reported and how to recover from these errors.
Glossary	Definitions of specialized terms used.
Related information sources	References or links to other documents that provide additional information
Navigational features	Features that allow readers to find their current location and move around the document.
Index	A list of key terms and the pages where these terms are referenced.
Search capability	In electronic documentation, a way of finding specific terms in the document.

Fig 1.1: Suggested components in a software user document

Document Preparation

Document preparation is the process of creating a document and formatting it for publication. Figure 2.3 shows the document preparation process as being split into 3 stages namely document creation, polishing and production. The three phases of preparation and associated support facilities are explained in the figure below:

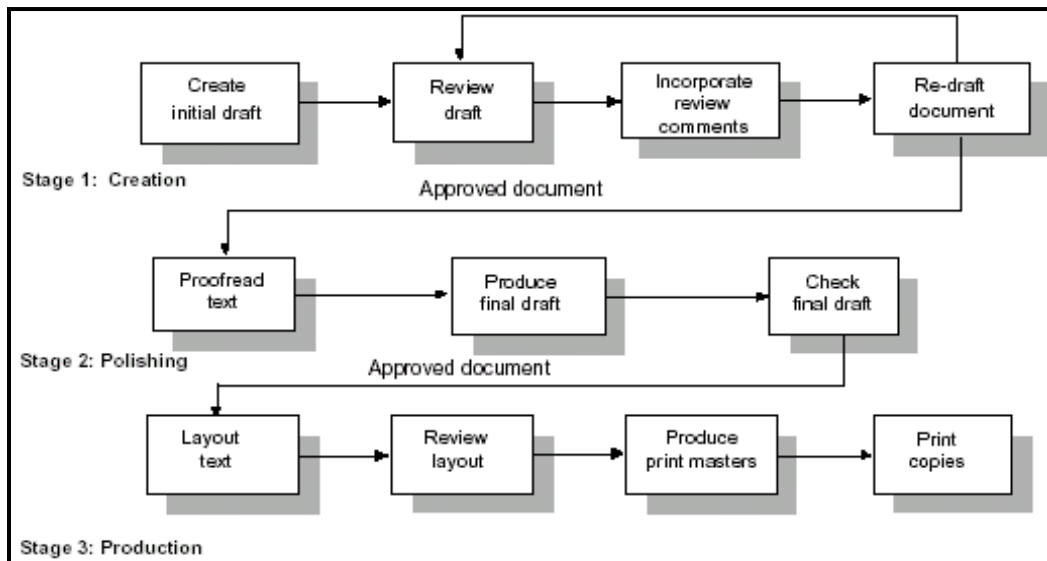


Fig 1.2: Stages of document preparation

LaTeX

LaTeX is a high-quality document preparation system; it includes features designed for the production of technical and scientific documentation. It allows users to very quickly tackle the more complicated parts of typesetting, such as inputting mathematics, creating tables of contents, referencing and creating bibliographies, and having a consistent layout across all sections. It is based on the WYSIWYM (what you see is what you mean) idea, meaning you only have focus on the contents of your document and the computer will take care of the formatting.

LaTeX is available as free software. To set up a basic TeX/LaTeX system, download and run the Basic MiKTeX Installer. TeXworks is used as editor for writing commands. We start off with a basic .tex file which contains LaTeX code. LaTeX uses control statements, which define how your content should be formatted. LaTeX compiler takes .tex file as input and convert it into a .pdf file.

First Piece of LaTeX

```
\documentclass{article}

\begin{document}
```

```
First document. This is a simple example, with no
extra parameters or packages included.
\end{document}
```

The first line of code declares the type of document, known as the *class*. The class controls the overall appearance of the document. In this case, the class is `article`, the simplest and most common LATEX class. Other types of documents you may be working on may require different classes such as **book** or **report**. Then the contents of our document are written, enclosed inside the `\begin{document}` and `\end{document}` tags. This is known as the *body* of the document. You can start writing here and make changes to the text if you wish. To see the result of these changes in the PDF you have to compile the document.

The Preamble of a document

In the previous example the text was entered after the `\begin{document}` command. Everything in your `.tex` file *before* this point is called the **preamble**. In the preamble you define the type of document you are writing, the language you are writing in, the *packages* you would like to use (more on this later) and several other elements. For instance, a normal document preamble would look like this:

```
\documentclass[12pt, letterpaper]{article}
```

Below a detailed description of `\documentclass`:

```
\documentclass[12pt, letterpaper]{article}
```

As said before, this defines the type of document. Some additional parameters included in the square brackets can be passed to the command. These parameters must be comma-separated. In the example, the extra parameters set the font size (**12pt**) and the paper size (**letterpaper**). Of course other font sizes (**9pt**, **11pt**, **12pt**) can be used, but if none is specified, the default size is **10pt**. As for the paper size other possible values are **a4paper** and **legalpaper**.

Adding a title, author and date

To add a title, author and date to our document, you must add three lines to the **preamble** (NOT the main body of the document). These lines are:

```
\title{First document}
\author{Hubert Farnsworth}
\thanks{funded by the Overleaf team}
```

This can be added after the name of the author, inside the braces of the **title** command. It will add a superscript and a footnote with the text inside the braces.

```
\date{September 2019}
```

Enter the date manually or use the command `\today` so the date will be updated automatically.

```
\documentclass[12pt, letterpaper, twoside]{article}

\title{First document}
\author{Hubert Farnsworth \thanks{funded by the Overleaf team}}
```



```
\date{September 2019}
```

Now a title, author and date are provided to the document, we can print this information on the document with the **\maketitle** command. This should be included in the **body** of the document at the place you want the title to be printed.

```
\begin{document}

\maketitle
We have now added a title, author and date to our first \LaTeX{}
document!
\end{document}
```

Adding Comments

To make a comment in LATEX, simply write a **%** symbol at the beginning of the line as shown below:

```
\begin{document}

\maketitle
\pagenumbering {gobble}
We have now added a title, author and date to our first \LaTeX{}
document!
% This line here is a comment. It will not be printed in the
document.
\end{document}
```

Basic Formatting



- **Abstracts**

In scientific documents it's a common practice to include a brief overview of the main subject of the paper. In LATEX there's the **abstract** environment for this. The **abstract** environment will put the text in a special format at the top of your document.

```
\begin{document}
\newpage
\begin{abstract}
This is a simple paragraph at the beginning of the
document. A brief introduction about the main subject.
\end{abstract}
\end{document}
```

- **Paragraphs and Newlines**

When writing the contents of your document, if you need to start a new paragraph you must hit the "Enter" key twice (to insert a double blank line). Notice that LATEX automatically indents paragraphs. To start a

new line without actually starting a new paragraph insert a break line point, this can be done by `\\` (a double backslash as in the example) or the `\newline` command.

```
\begin{document}
\begin{abstract}
This is a simple paragraph at the beginning of the document. A
brief introduction about the main subject.
\end{abstract}
Now that we have written our abstract, we can begin writing our
first paragraph.

This line will start a second Paragraph.
\end{document}
```

- **Chapters and Sections**

Commands to organize a document vary depending on the document type, the simplest form of organization is the sectioning, available in all formats.

```
\chapter{First Chapter}

\section{Introduction}

This is the first section.

Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Etiam lobortis facilisis sem. Nullam nec mi et
neque pharetra sollicitudin. Praesent imperdiet mi nec ante.
Donec ullamcorper, felis non sodales...

\section{Second Section}

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam lobortis facilisissem. Nullam nec mi et neque pharetra
sollicitudin. Praesent imperdiet mi nec ante...

\subsection{First Subsection}
Praesent imperdiet mi nec ante. Donec ullamcorper, felis non
sodales...

\section*{Unnumbered Section}
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam lobortis facilisissem
```

The command `\section{}` marks the beginning of a new section, inside the braces is set the title. Section numbering is automatic and can be disabled by including a `*` in the section command as `\section*{}`. We can also have `\subsection{}`s, and indeed `\subsubsection{}`s. The basic levels of depth are listed below:

-1	<code>\part{part}</code>
0	<code>\chapter{chapter}</code>
1	<code>\section{section}</code>
2	<code>\subsection{subsection}</code>
3	<code>\subsubsection{subsubsection}</code>
4	<code>\paragraph{paragraph}</code>
5	<code>\subparagraph{subparagraph}</code>

Note that `\part` and `\chapter` are only available in report and book document classes.

• Bold, Italics and Underlining

Following are some simple text formatting commands.

- **Bold:** Bold text in LaTeX is written with the `\textbf{...}` command.
- *Italics:* Italicised text in LaTeX is written with the `\textit{...}` command.
- Underline: Underlined text in LaTeX is written with the `\underline{...}` command.

Some of the `\textbf{greatest}`
discoveries in `\underline{science}`
were made by `\textbf{\textit{accident}}`.

• Unordered lists

Unordered lists are produced by the **itemize** environment. Each entry must be preceded by the control sequence `\item` as shown below. By default the individual entries are indicated with a black dot, so-called bullet. The text in the entries may be of any length.

```
\begin{itemize}
  \item The individual entries are indicated with a black dot, a
so-called bullet.
  \item The text in the entries may be of any length.
\end{itemize}
```

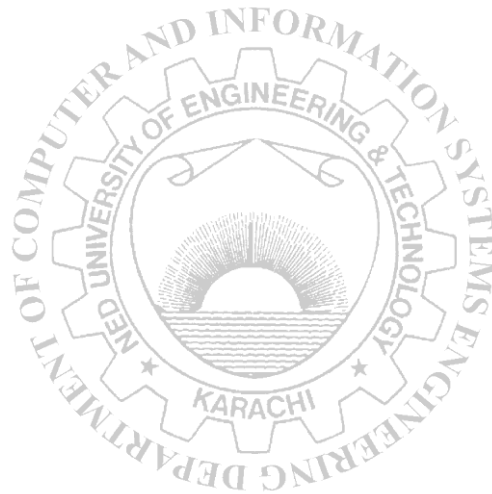
• Ordered lists

Ordered list have the same syntax inside a different environment. We make ordered lists using the **enumerate** environment:

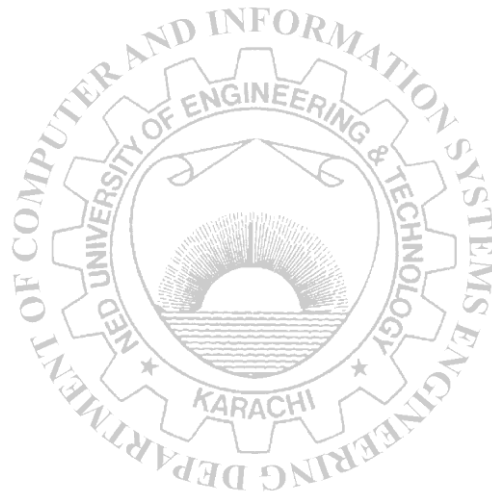
```
\begin{enumerate}
  \item This is the first entry in our list
  \item The list numbers increase with each entry we add
\end{enumerate}
```

EXERCISES

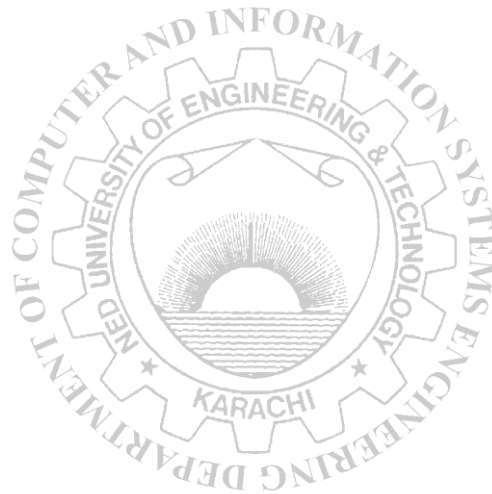
1. Construct a Software Requirement Specifications (SRS) document for a Software System of your choice using LaTeX. Attach the printout of the .tex file and .pdf file.
2. Develop a brief document describing the basic functionality of an IoT based Health Monitoring System. Also include any block diagram/ image for explaining the overall functionality of the system. Use LaTeX. Attach the printout of the .tex and .pdf files. (Feel free to you use the other available formatting options in LaTeX).
3. Suppose that you are developing a calculator application and gathering requirements for it. Your calculator is able to solve some basic linear equations as well. Prepare an initial draft of requirement specification document, highlighting various examples of equations your system is able to solve. Use LaTeX. Attach the printout of the .tex and .pdf files.



Attach print outs here:



Attach print outs here:



Lab Session 02

Practice any project management tool to prepare a project plan

Project management

Project management is the process of planning, organizing, and managing tasks and resources to accomplish a defined objective, usually within constraints on time, resources, or cost. Sound planning is one key to project success, but sticking to the plan and keeping the project on track is no less critical. Microsoft Project features can help in monitoring progress and keep small slips from turning into landslides.

First let us understand some basic terms and definitions that will be used quite often:

Tasks: They are a division of all the work that needs to be completed in order to accomplish the project goals.

Scope: of any project is a combination of all individual tasks and their goals.

Resources: can be people, equipment, materials or services that are needed to complete various tasks. The amount of resources affects the scope and time of any project.

STARTING A NEW PROJECT

You can create a Project from a Template File by choosing File > New from the menu. In the New File dialog box that opens, select the Project Templates tab and select the template that suits your project best and click OK. (You may choose a Blank Project Template and customize it)

Once a new Project page is opened, the Project Information dialog box opens.

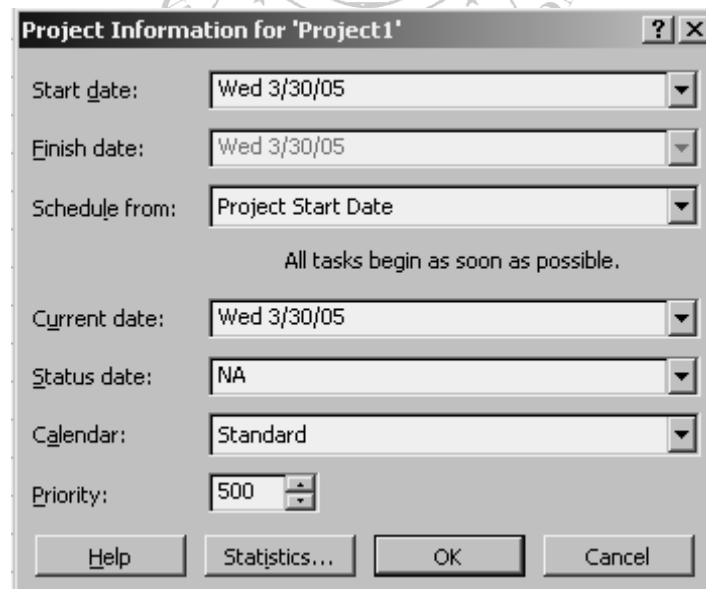


Fig 2.1: Project Information dialog box

Enter the start date or select an appropriate date by scrolling down the list. Click OK. Project automatically enters a default start time and stores it as part of the dates entered and the application window is displayed.

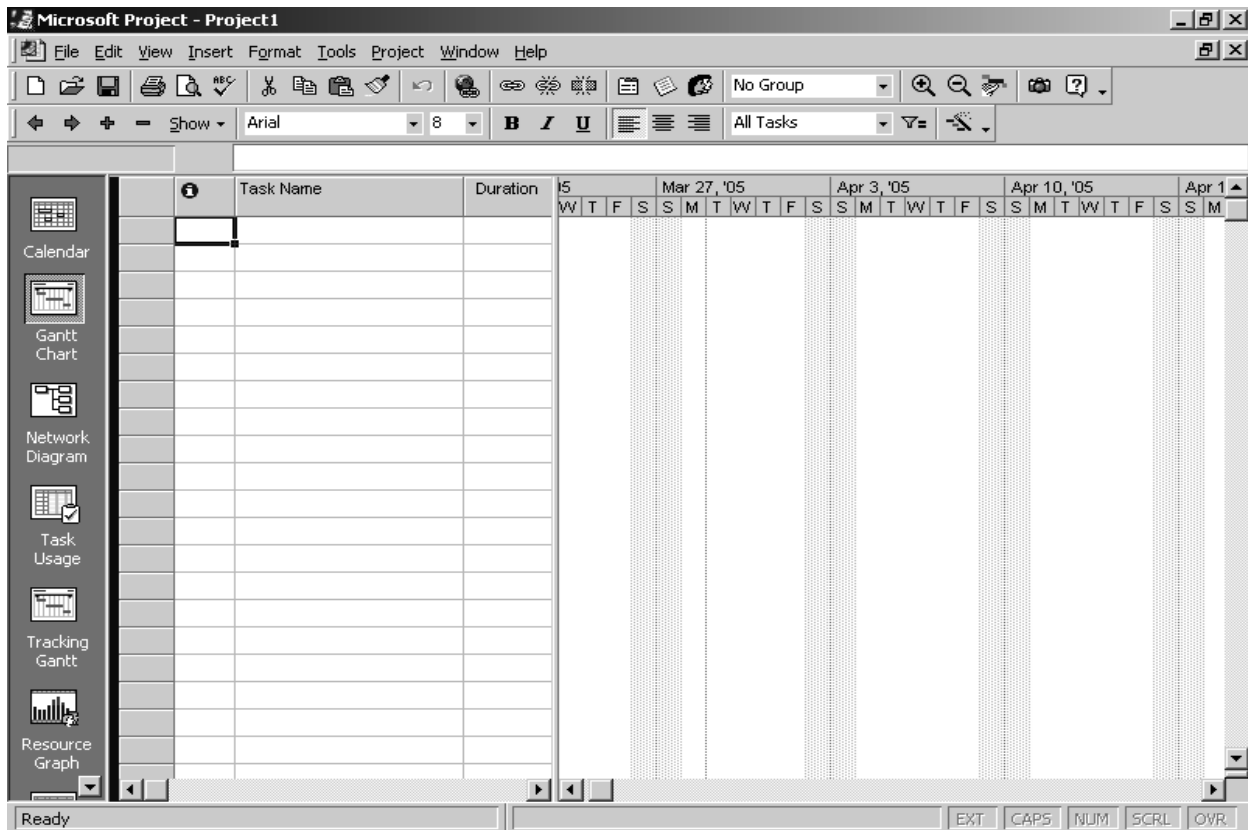


Fig 2.2: Main Window

Views

Views allow you to examine your project from different angles based on what information you want displayed at any given time. You can use a combination of views in the same window at the same time.

Project Views are categorized into two types:

- Task Views (5 types)
- Resource Views (3 types)

The Project worksheet is located in the main part of the application window and displays different information depending on the view you choose. The default view is the Gantt Chart View.

Tasks

Goals of any project need to be defined in terms of tasks. There are four major types of tasks:

1. *Summary tasks* - contain subtasks and their related properties
2. *Subtasks* - are smaller tasks that are a part of a summary task

3. *Recurring tasks* - are tasks that occur at regular intervals
4. *Milestones* - are tasks that are set to zero duration and are like interim goals in the project

- ***Entering Tasks and assigning task duration***

Click in the first cell and type the task name. Press enter to move to the next row. By default, estimated duration of each task is a day. But, there are very few tasks that can be completed in a day's time. You can enter the task duration as and when you decide upon a suitable estimate.

Double-clicking a task or clicking on the Task Information button on the standard toolbar opens the *Task Information box*. You can fill in more detailed descriptions of tasks and attach documents related to the task in the options available in this box.

To enter a *milestone*, enter the name and set its duration to zero. Project represents it as a diamond shape instead of a bar in the Gantt chart.

To copy tasks and their contents, click on the task ID number at the left of the task and copy and paste as usual.

You can enter *Recurring tasks* by clicking on Insert > Recurring task and filling in the duration and recurrence pattern for the task.

Any action you perform on a summary task - deleting it, moving or copying it apply to its subtasks too.

- ***Outlining tasks***

Once the summary tasks have been entered in a task table, you will need to insert subtasks in the blank rows and indent them under the summary task. This is accomplished with the help of the outlining tool.

Outlining is already active when you launch a project and its tools are found at the left end of the Formatting bar. To enter a subtask, enter the task in a blank cell in the Task Name column and click the Indent button on the Outlining tool bar. The Show feature in this toolbar is drop down tool that gives you an option of different Outline levels.

A summary task is outdented to the left cell border, is bold and has a Collapse (-) (Hide subtasks) button in front of it and its respective subtasks are indented with respect to it.

Advantages of Outlining:

- It creates multiple levels of subtasks that roll up into a summary task
- Collapse and expand summary tasks when necessary
- Apply a Work Breakdown structure
- Move, copy or delete entire groups of tasks

LINKS

Tasks are usually scheduled to start as soon as possible i.e. the first working day after the project start date.

Dependencies

Dependencies specify the manner in which two tasks are linked. Because Microsoft Project must maintain the manner in which tasks are linked, dependencies can affect the way a task is scheduled. In a scenario where there are two tasks, the following dependencies exist:

Finish to Start – Task 2 cannot start until task 1 finishes.

Start to Finish – Task 2 cannot finish until task 1 starts.

Start to Start – Task 2 cannot start until task 1 starts.

Finish to Finish – Task 2 cannot finish until task 1 finishes.

Tasks can be linked by following these steps:

1. Double-click a task and open the Task Information dialog box.
2. Click the predecessor tab.
3. Select the required predecessor from the drop down menu.
4. Select the type of dependency from drop down menu in the Type column.
5. Click OK.

The Split task button splits tasks that may be completed in parts at different times with breaks in their duration times.

CONSTRAINTS

Certain tasks need to be completed within a certain date. Intermediate deadlines may need to be specified. By assigning constraints to a task you can account for scheduling problems. There are about 8 types of constraints and they come under the flexible or inflexible category. They are:

- **As Late As Possible** – Sets the start date of your task as late in the Project as possible, without pushing out the Project finish date.
- **As Soon As Possible** – Sets the start date of your task as soon as possible without preceding the project start date.
- **Must Finish On** – Sets the finish date of your task to the specified date.
- **Must Start On** – Sets the start date of your task to the specified date.
- **Start No Earlier Than** – Sets the start date of your task to the specified date or later.
- **Start No Later Than** – Sets the start date of your task to the specified date or earlier.
- **Finish No Earlier Than** – Sets the finish date of your task to the specified date or later.
- **Finish No Later Than** – Sets the finish date of your task to the specified date or earlier.

Flexible constraints (demarcated by a red dot in Microsoft Project 2000) restrict scheduling to a great extent whereas flexible constraints (blue dot) allow Project to calculate the schedule and make appropriate adjustments based on the constraint applied.

Inflexible constraints can cause conflicts between successive and preceding tasks at times and you may need to remove such a constraint.

To apply a constraint:

1. Open the Task Information dialog box.
2. Click the Advanced tab and open the Constraint type list by clicking on the drop-down arrow and select it.
3. Select a date for the Constraint and click OK.

UPDATE COMPLETED TASKS

If you have tasks in your project that have been completed as they were scheduled, you can quickly update them to 100% complete all at once, up to a date you specify.

1. On the View menu, click Gantt Chart.
2. On the Tools menu, point to Tracking, and then click Update Project
3. Click Update work as complete through and then type or select the date through which you want progress updated.
4. If you don't specify a date, Microsoft Project uses the current or status date.
5. Click Set 0% or 100% complete only.
6. Click Entire Project

RESOURCES

Once you determine that you need to include resources into your project you will need to answer the following questions:

- What kind of resources do you need?
- How many of each resource do you need?
- Where will you get these resources?
- How do you determine what your project is going to cost?

Resources are of two types - *work* resources and *material* resources.

Work resources complete tasks by expending time on them. They are usually people and equipment that have been assigned to work on the project.

Material resources are supplies and stocks that are needed to complete a project.

A new feature in Microsoft Project 2000 is that it allows you to track material resources and assign them to tasks.

Entering Resource Information in Project

The Assign Resources dialog box is used to create list of names in the resource pool.

To enter resource lists:

1. Click the Assign Resources button on the Standard Tool bar or Tools > Resources > Assign resources.

2. Select a cell in the name column and type a response name. Press Enter
3. Repeat step 2 until you enter all the resource names.
4. Click OK.

Resource names cannot contain slash (/), brackets [] and commas (.). Another way of defining your resource list is through the Resource Sheet View.

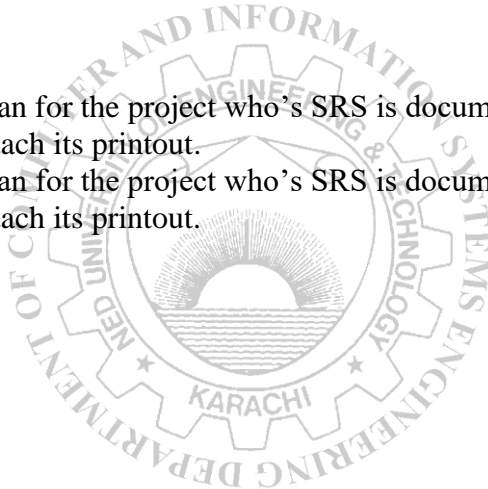
1. Display Resource Sheet View by choosing View > Resource Sheet or click the Resource Sheet icon on the View bar.
2. Enter your information. (To enter material resource use the Material Label field)

To assign a resource to a task:

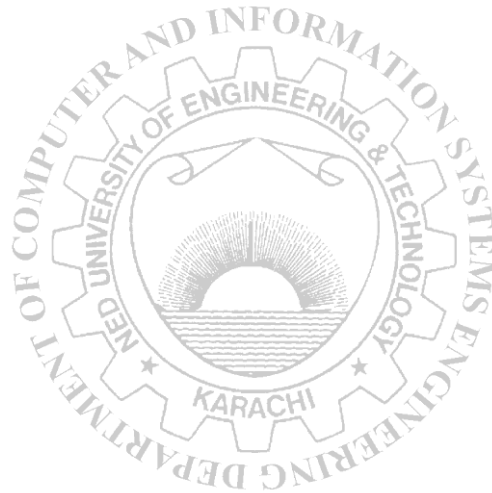
1. Open the Gantt Chart View and the Assign Resources Dialog box.
2. In the Entry Table select the tasks for which you want to assign resources.
3. In the Assign Resources Dialog box, select the resource (resources) you want to assign.
4. Click the Assign button.

EXERCISE

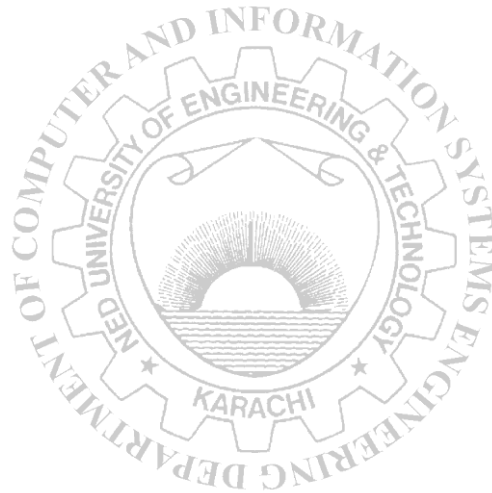
1. Construct a project plan for the project who's SRS is documented in Lab Session 01 (Exercise#1). Also attach its printout.
2. Construct a project plan for the project who's SRS is documented in Lab Session 01 (Exercise#2). Also attach its printout.



Attach print outs here:



Attach print outs here:



Lab Session 03

Carry out user view and structural view analysis for the suggested system: Use Case and Class Diagrams

Use case diagram

Use case diagrams are used to describe a set of actions (**use cases**) that some system or systems (**subject**) should or can perform in collaboration with one or more **external users** of the system (**actors**). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

(System) Use case diagrams are used to specify:

- (external) requirements, required usages of a system under design or analysis (subject) - to capture what the system is supposed to do;
- the functionality offered by a subject – what the system can do;
- Requirements the specified subject poses on its environment - by defining how environment should interact with the subject so that it will be able to perform its services.

Create Use Case Diagram

To create a Use Case Diagram in StarUML:

1. Select first an element where a new Use Case Diagram to be contained as a child.
2. Select **Model | Add Diagram | Use Case Diagram** in Menu Bar or select **Add Diagram | Use Case Diagram** in Context Menu.
3. To create use case subjects, actors, use cases, include and extend relationship etc., select the desired option from toolbox and drag on the main diagram. Upon double clicking, all the respective options will be made available.

Major elements of the UML use case diagram are shown on the picture below.

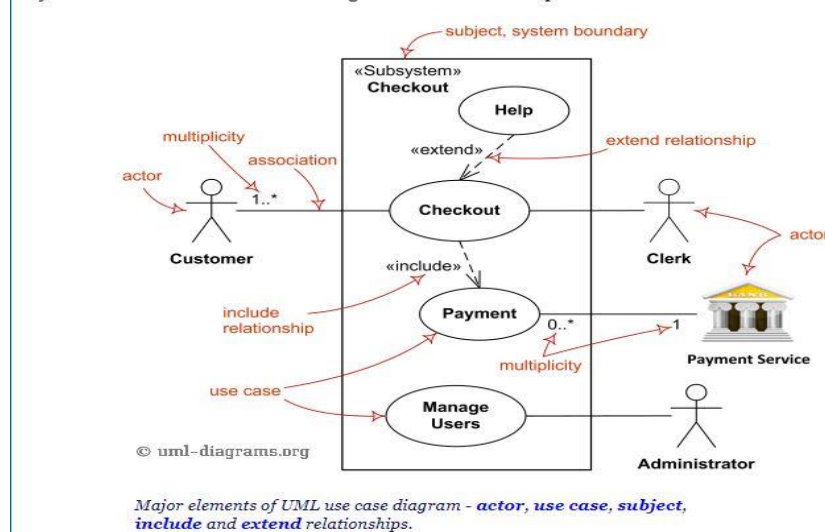


Fig 3.1 Major Elements of UML Use Case Diagram

Class diagram

Class diagram is UML **structure diagram** which shows structure of the designed system at the level of **classes and interfaces**, shows their **features, constraints** and **relationships** - **associations, generalizations, dependencies**, etc.

Create Class Diagram

To create a Class Diagram in StarUML:

1. First select an element where a new Class Diagram to be contained as a child.
2. Select **Model | Add Diagram | Class Diagram** in the Menu Bar or select **Add Diagram | Class Diagram** in Context Menu.
3. To create classes, enumerations, association, composition, generalization and composition relationships, select the respective option from the toolbox.

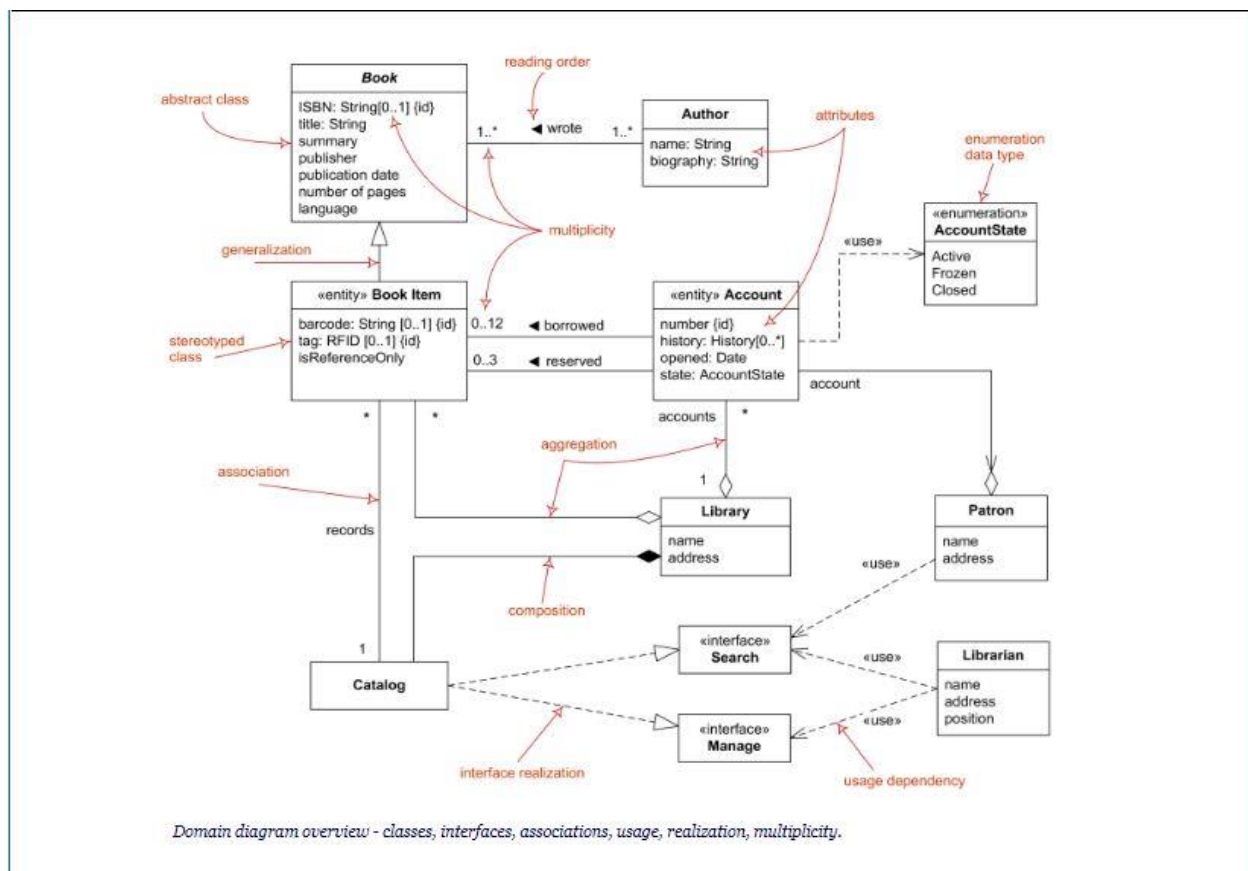


Fig 3.2 Major Elements of UML Class Diagram

Online Shopping System

Web Customer actor uses some web site to make purchases online. Top level **use cases** are **View Items**, **Make Purchase** and **Client Register**. View Items use case could be used by customer as top level use case if 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.

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 the customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.

UML Use Case Diagram Example

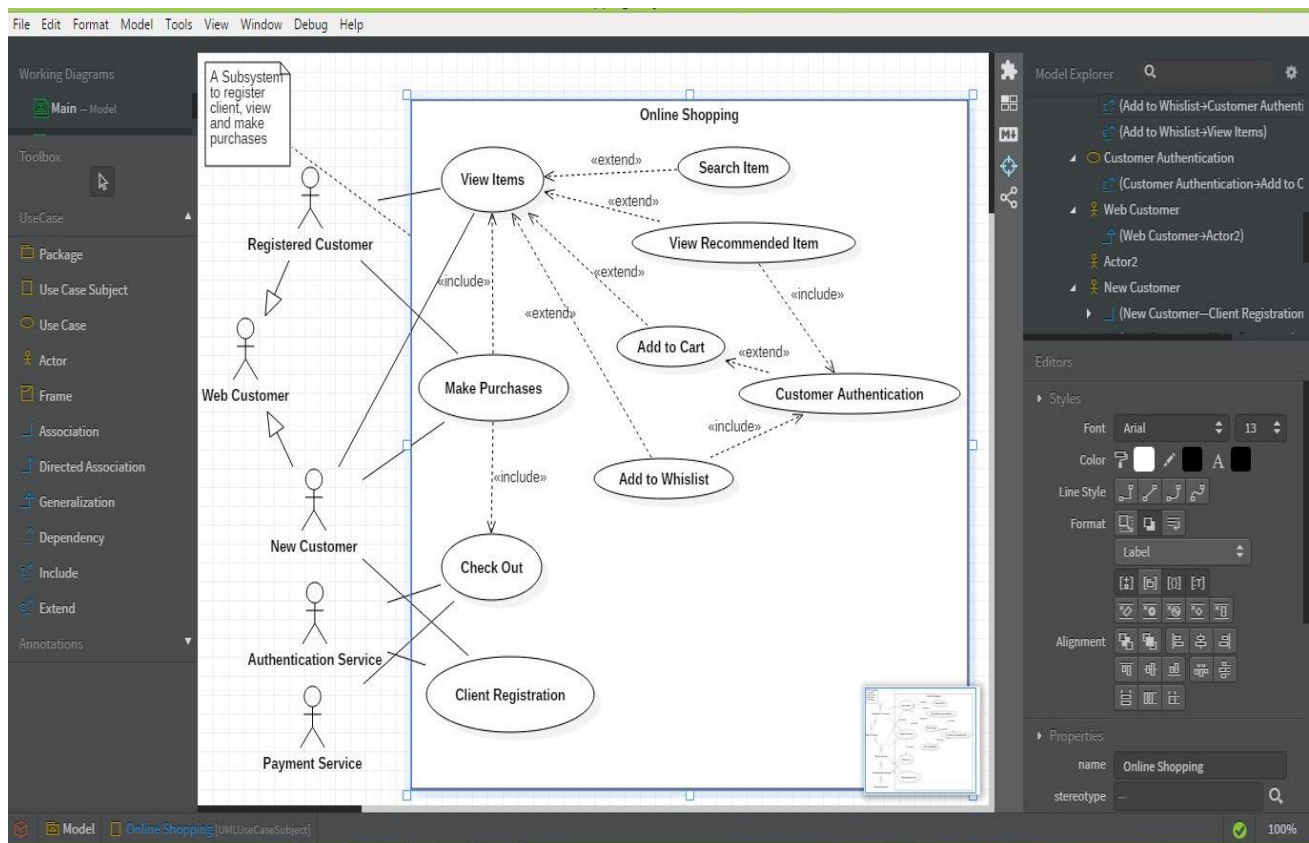


Fig 3.3 UML Use Case Diagram for Online Shopping System

UML Class Diagram Example

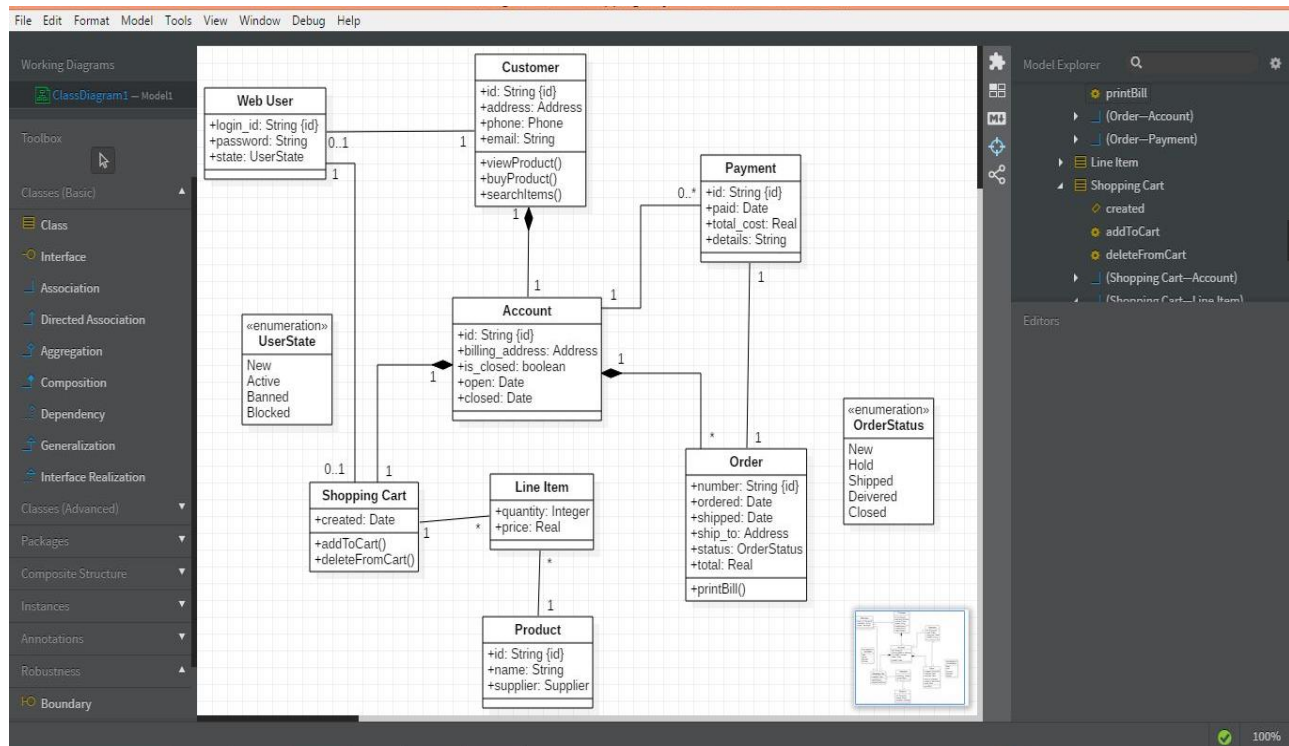
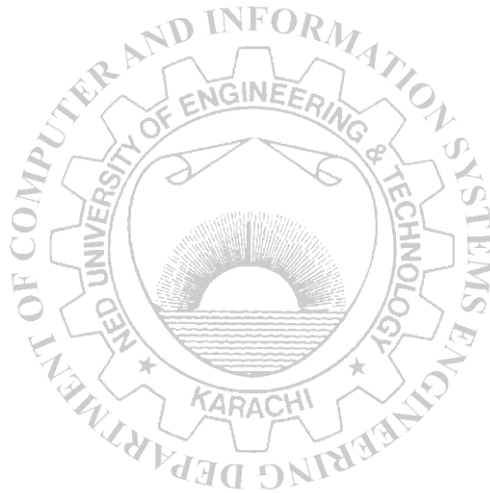


Fig 3.4 UML Class Diagram for Online Shopping System

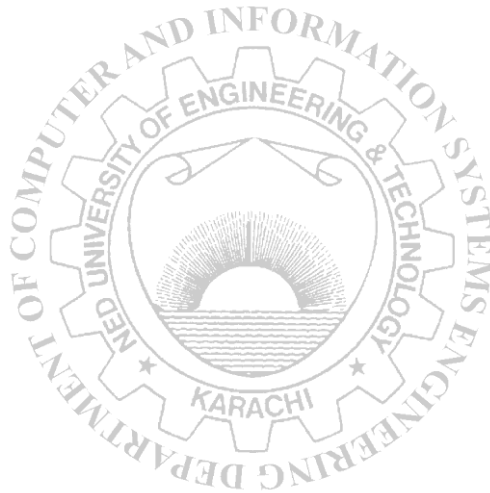
EXERCISES:

1. Construct use case and class diagram for an automated teller machine (ATM) or the automatic banking machine (ABM). Customer uses a bank ATM to check balances of his/her bank accounts, deposit funds, withdraw cash and/or transfer funds (use cases). ATM Technician provides maintenance and repairs to the ATM. Also attach printout.
2. In “Online Shopping System”, Check Out use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie (“Remember me”). Web site authentication service is used in all these use cases. Checkout use case also includes Payment use case which could be done by using external credit payment service. Extend the given use case diagram incorporating the given information. Also attach the printout.
3. For a hypothetical “Food Ordering App”, identify possible classes, attributes and their operations. Also show the relationships among various classes using StarUML. Attach its printout.

Attach print outs here:



Attach print outs here:



Lab Session 04

Practice function oriented UML diagram for the suggested system: Data Flow Diagrams

Data flow diagram (DFD)

Data Flow Diagrams (DFD) are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

DFD Symbols

There are four basic symbols that are used to represent a data-flow diagram.

1. Process:

A process receives input data and produces output with a different content or form. Every process has a name that identifies the function it performs.

Notation

- A rounded rectangle represents a process
- Processes are given IDs for easy referencing

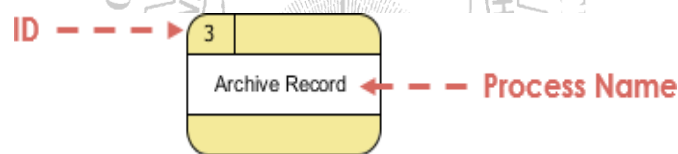


Fig 4.1 Process Notation

Process Example

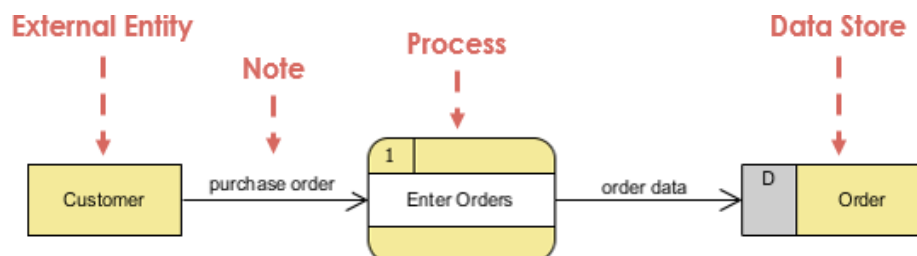


Fig 4.2 Process Example

2. Data Store:

A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data because one or more processes need to use the stored data in a later time.

Notation

- Data can be written into the data store, which is depicted by an outgoing arrow

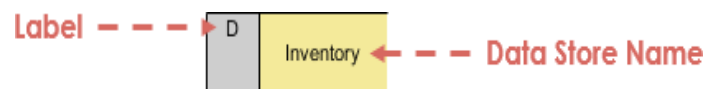


Fig 4.3 Data Store Notation

- Data can be read from a data store, which is depicted by an incoming arrow.

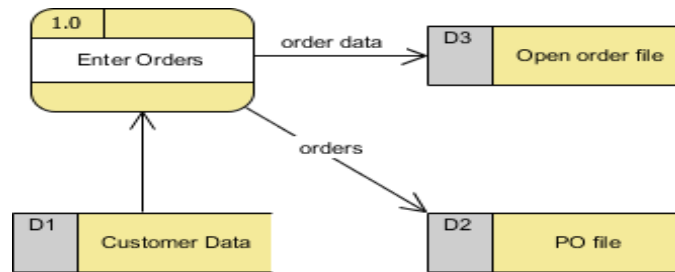
Data Store Example

Fig 4.4 Data Store Example

3. Data Flow

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or it can represent a set of data element (or a data structure).

Notation

- Straight lines with incoming arrows are input data flow
- Straight lines with outgoing arrows are output data flows

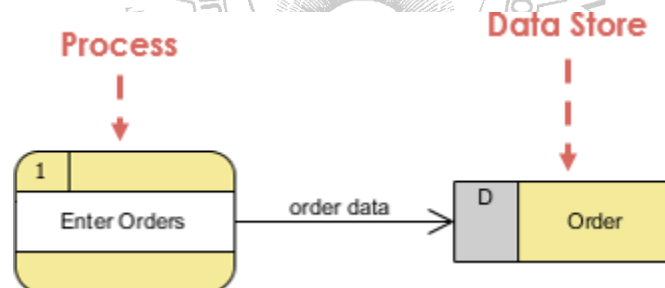
Data flow Example

Fig 4.4 Data Flow Example

4. External Entity

An external entity is a person, department, outside organization, or other information system that provides data to the system or receives outputs from the system. External entities are components outside of the boundaries of the information systems. They represent how the information system interacts with the outside world.

Notation

- A customer submitting an order and then receive a bill from the system
- A vendor issue an invoice



Fig 4.5 External Entity Notation

External Entity Example

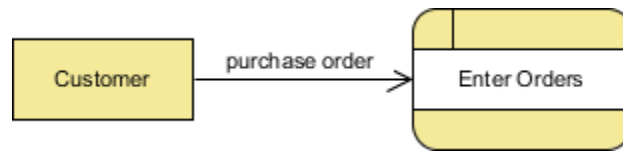


Fig 4.6 External Entity Example

Creating DFD in StarUML:

- Select **Model | Add Diagram | Data Flow Diagram** in Menu Bar or select **Add Diagram | Data Flow Diagram** in Context Menu.
- To create external entities, processes, data flow and data store, select the desired option from toolbox and drag on the main diagram. Upon double clicking, all the respective options will be made available.

The Food Ordering System

- **Context/Level 0 DFD**

A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0. Some of the benefits of a Context Diagram are:

- Shows the overview of the boundaries of a system
- No technical knowledge is required to understand with the simple notation
- Simple to draw, amend and elaborate as its limited notation

The figure below shows a context Data Flow Diagram that is drawn for a Food Ordering System. It contains a process that represents the system to model the "*Food Ordering System*". It also shows the participants who will interact with the system, called the external entities. In this example, *Supplier*, *Kitchen*, *Manager* and *Customer* are the entities who will interact with the system. In between the process and the external entities, there are data flow (connectors) that indicate the existence of information exchange between the entities and the system.

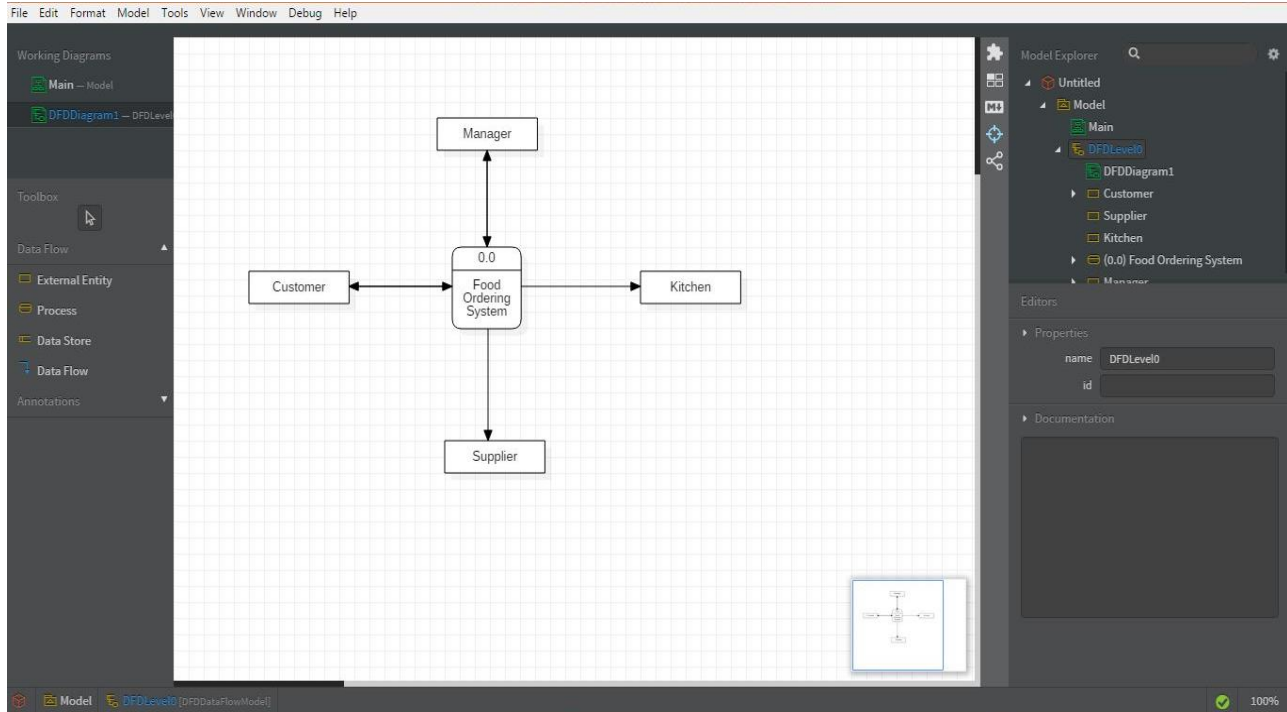


Fig 4.7 DFD Level 0 for Food Ordering System

• Level 1 DFD

The figure below shows the level 1 DFD, which is the decomposition (i.e. break down) of the Food Ordering System process shown in the context DFD. Read through the diagram and then we will introduce some of the key concepts based on this diagram.

The Food Order System Data Flow Diagram example contains three processes, four external entities and two data stores.

Based on the diagram, we know that a *Customer* can place an *Order*. The *Order Food* process receives the *Order*, forwards it to the *Kitchen*, store it in the *Order* data store, and store the updated *Inventory details* in the *Inventory* data store. The process also deliver a *Bill* to the *Customer*.

Manager can receive *Reports* through the *Generate Reports* process, which takes *Inventory details* and *Orders* as input from the *Inventory* and *Order* data store respectively.

Manager can also initiate the *Order Inventory* process by providing *Inventory order*. The process forwards the *Inventory order* to the *Supplier* and stores the updated *Inventory details* in the *Inventory* data store.

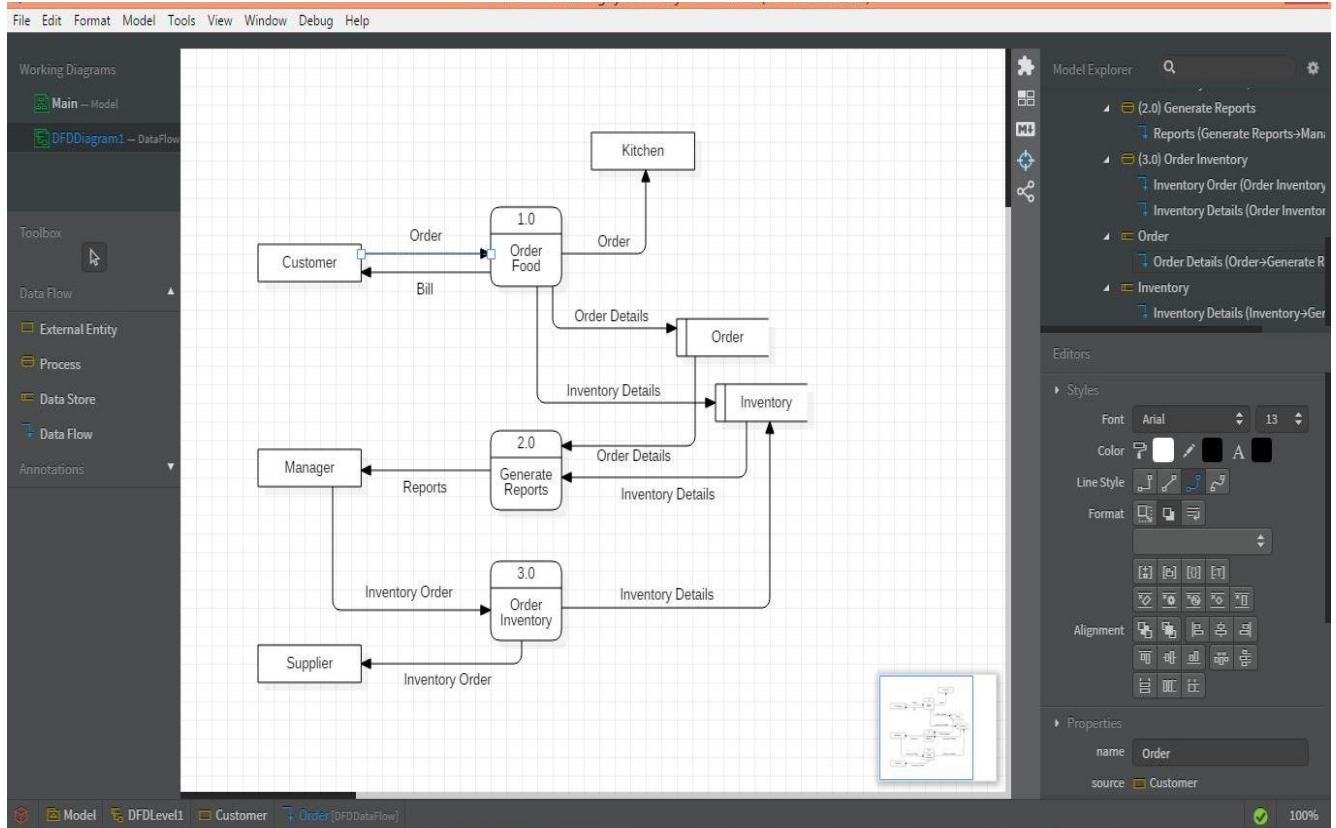
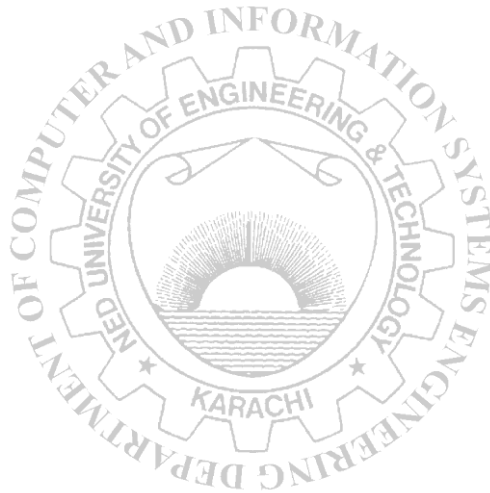


Fig 4.8 DFD Level 1 for Food Ordering System

EXERCISES:

1. Construct Level 0 and Level 1 DFD for a “Super Market App” using StarUML. Identify the processes, data stores and external entities clearly. Attach printout.
2. In the “Super Market App”, customers can view, search and shop all the required stuff easily. Construct a Level 2 DFD for “Search Item” process of Super Market App using StarUML. Attach printout.
3. Construct a Level 2 DFD for “2.0 Generate Reports” process of “Food Ordering System”. Also attach printout.

Attach print outs here:



Lab Session 05

Practice behavioral view UML diagrams for the suggested system: State Chart, Sequence and Collaboration Diagrams

State chart diagram

State machine diagram is a **behavior diagram** which shows discrete behavior of a part of designed system through finite state transitions. State machine diagrams can also be used to express the usage protocol of part of a system.

The UML defines the Simple State, Composite State and Submachine state.

Simple State

A **simple state** is a state that does not have sub-states - it has no **regions** and it has no **submachine states**. Simple state is shown as a rectangle with rounded corners and the state name inside the rectangle. Simple state may have compartments. The compartments of the state are:

- name compartment
- internal activities compartment
- internal transitions compartment

Name compartment holds the (optional) name of the state, as a string. States without names are called **anonymous states** and are all considered distinct (different) states. Name compartments should not be used if a name tab is used and vice versa.

Internal activities compartment holds a list of internal actions or state (do) activities (behaviors) that are performed while the element is in the state. The activity label identifies the circumstances under which the behavior specified by the activity expression will be invoked. The behavior expression may use any attributes and association ends that are in the scope of the owning entity. For list items where the expression is empty, the slash separator is optional.

Several labels are reserved for special purposes and cannot be used as event names. The following are the reserved activity labels:

- entry (behavior performed upon entry to the state)
- do (ongoing behavior, performed as long as the element is in the state)
- exit (behavior performed upon exit from the state)

Internal transition compartment contains a list of internal transitions, where each item has the form as described for trigger. Each event name may appear more than once per state if the guard conditions are different. The event parameters and the guard conditions are optional. If the event has parameters, they can be used in the expression through the current event variable.



Fig 5.1 Simple state with name and internal activities compartments

Creating State Chart Diagram in StarUML

To create a State-chart Diagram:

1. Select first an element where a new Statechart Diagram to be contained as a child.
2. **Model | Add Diagram | Statechart Diagram** in Menu Bar or select **Add Diagram | Statechart Diagram** in Context Menu.
3. Toolbox contains all the required options namely simple states, internal activities (entry, do and exit), transitions, initial and final states etc. to create the required state chart diagram.

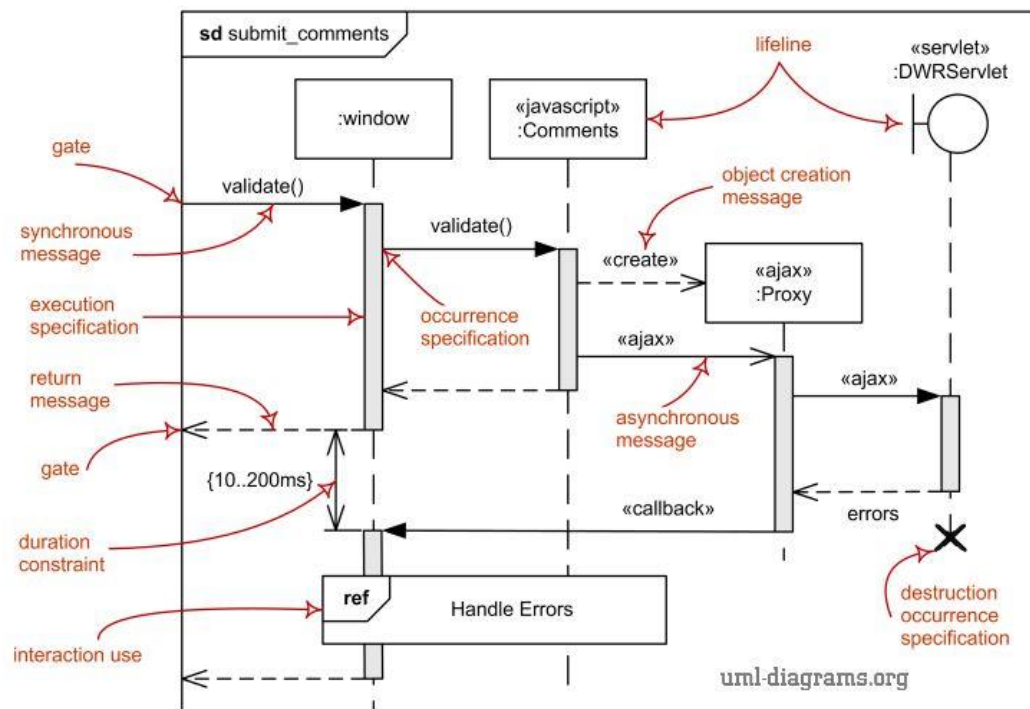
Sequence diagram

Sequence diagram is the most common kind of **interaction diagram**, which focuses on the **message** interchange between a numbers of **lifelines**.

Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

The following nodes and edges are typically drawn in a **UML sequence diagram**: **lifeline**, **execution specification**, **message**, **combined fragment**, **interaction use**, **state invariant**, **continuation**, **destruction occurrence**.

Major elements of the sequence diagram are shown on the picture below.



Major elements of UML sequence diagram.

Fig 5.2 Major Elements of UML Sequence Diagram

Create Sequence Diagram

To create a Sequence Diagram:

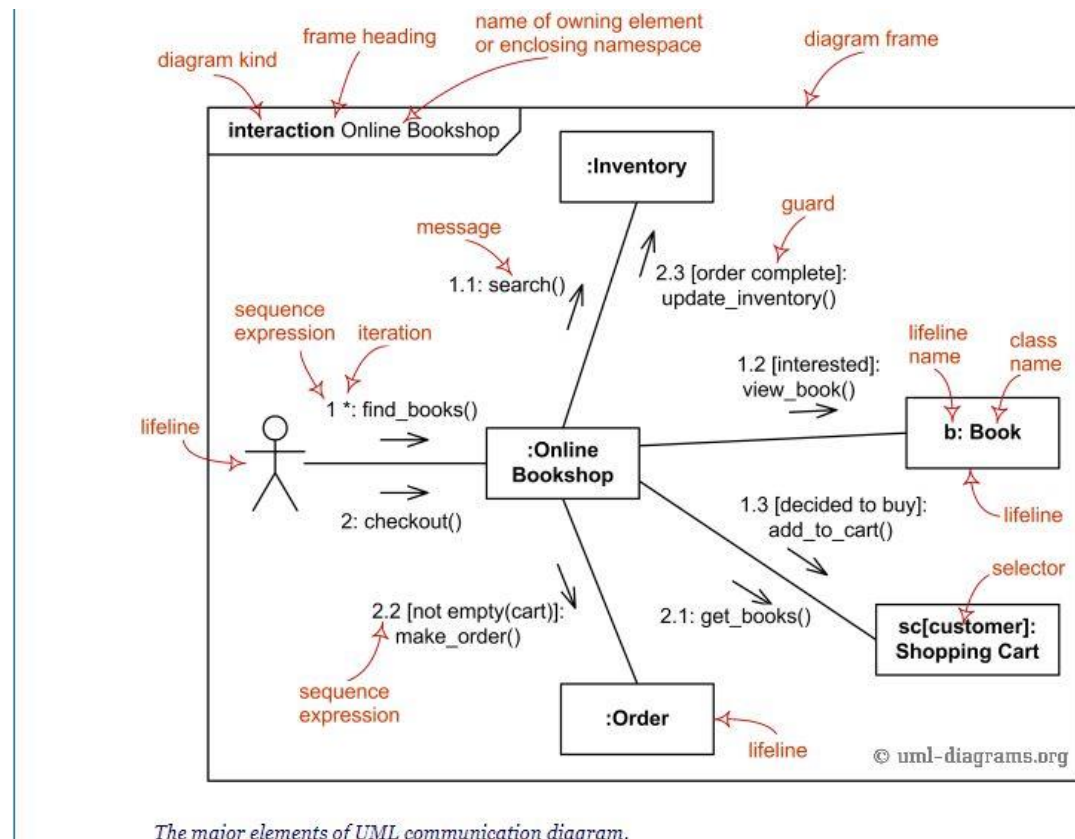
1. Select first an element where a new Sequence Diagram to be contained as a child.
2. Select **Model | Add Diagram | Sequence Diagram** in Menu Bar or select **Add Diagram | Sequence Diagram** in Context Menu.
3. All the major elements of UML sequence diagram (shown above) are available in the toolbox.

Collaboration or communication diagram

Communication diagram (called **collaboration diagram** in UML 1.x) is a kind of UML **interaction diagram** which shows interactions between objects and/or **parts** (represented as **lifelines**) using sequenced messages in a free-form arrangement.

Communication diagram corresponds (i.e. could be converted to/from or replaced by) to a simple **sequence diagram** without structuring mechanisms such as interaction uses and combined fragments. It is also assumed that **message overtaking** (i.e., the order of the receptions are different from the order of sending of a given set of messages) will not take place or is irrelevant.

The following nodes and edges are drawn in a UML communication diagrams: **frame**, **lifeline**, and **message**. These major elements of the communication diagram are shown on the picture below:



The major elements of UML communication diagram.

Fig 5.3 Major Elements of UML Communication Diagram

Create Communication/Collaboration Diagram

To create a Communication Diagram:

1. Select first an element where a new Communication Diagram to be contained as a child.

2. Select **Model | Add Diagram | Communication Diagram** in Menu Bar or select **Add Diagram | Communication Diagram** in Context Menu.
3. All the major elements of UML sequence diagram (shown above) are available in the toolbox.

Bank Automated Teller Machine (ATM) System

State Transition / State Chart Diagram

Purpose: An example of UML behavioral state machine diagram describing Bank Automated Teller Machine (ATM) top level state machine.

Summary: ATM is initially turned off. After the power is turned on, ATM performs startup action and enters **Self Test** state. If the test fails, ATM goes into **Out of Service** state, otherwise there is **triggerless transition** to the **Idle** state. In this state ATM waits for customer interaction.

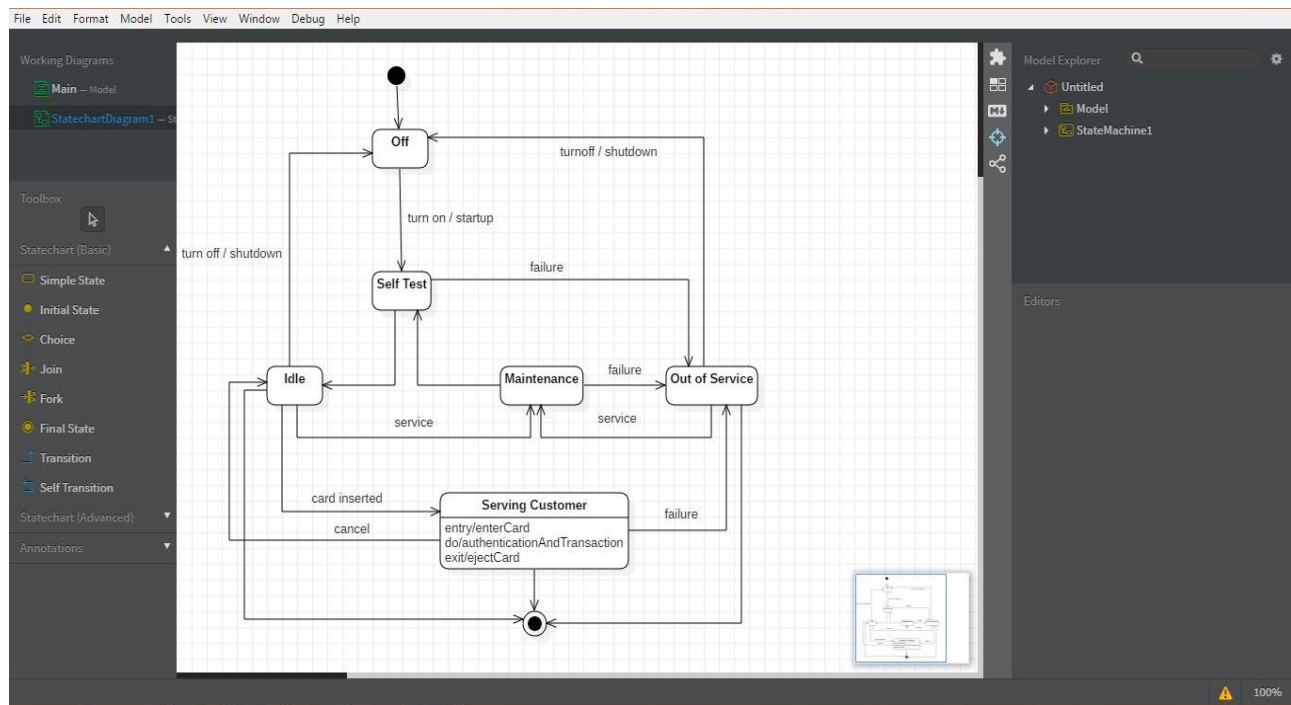


Fig 5.4 State Chart Diagram for ATM System

Sequence and Communication Diagram

Here, customer inserts his card in the ATM machine. The card is verified through bank and after authentication is done, customer is allowed to do transactions. He chooses to withdraw some cash. His request is processed and amount is debited from his account. The customer receives cash, card and receipt from the respective slots.

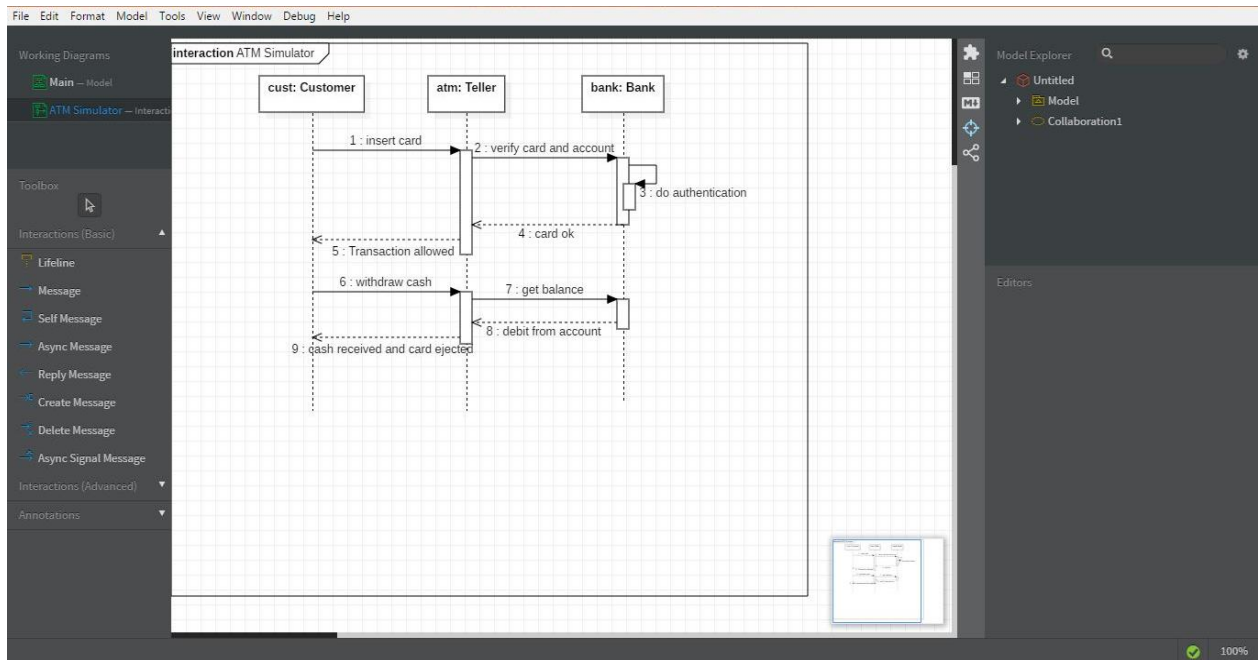


Fig 5.4 UML Sequence Diagram for ATM System

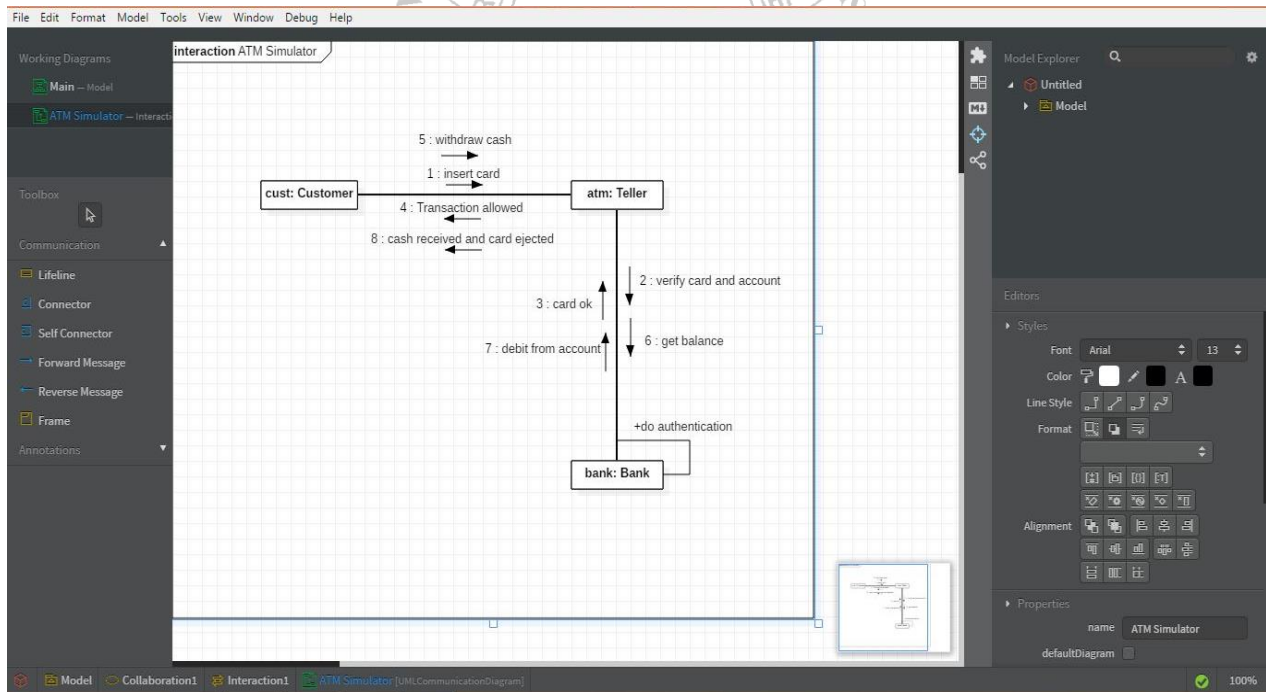
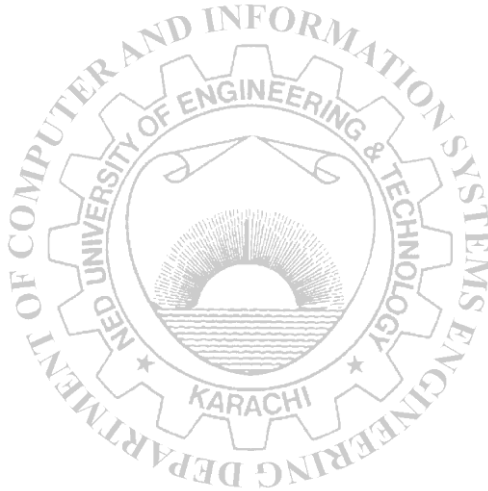


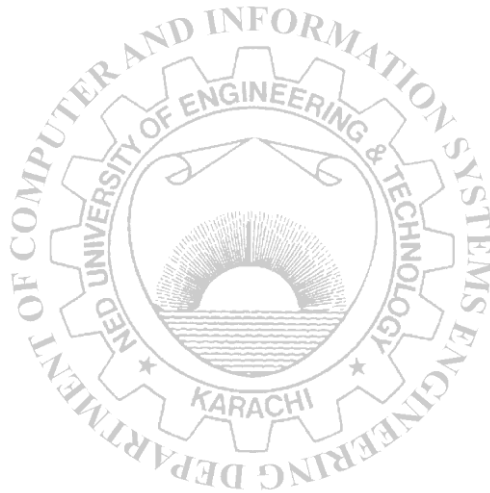
Fig 5.5 UML Collaboration Diagram for ATM System

EXERCISES

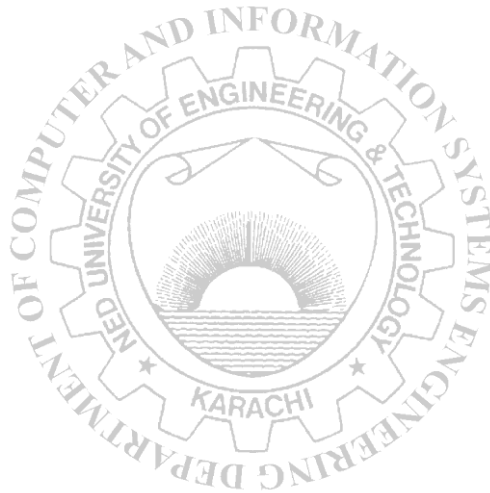
1. Create a State Chart Diagram for Microwave Oven Operation using StarUML. Also attach print out.
2. Create Sequence and Collaboration diagrams for Online Bookshop where the Web customer actor can search inventory, view and buy books online. Also attach the printout.
3. UML diagrams can also be created by means of variety of different online tools. Visual Paradigm is one of them. Create any three UML diagrams using this online tool for the ‘Online Bookshop System’. Attach print out.



Attach print outs here:



Attach print outs here:



Lab Session 06

Practice Component and Deployment view UML diagrams for the suggested systems

Component diagram in UML

The main purpose of a component diagram is to show the structural relationships between the components of a system. In UML, Components are made up of software objects that have been classified to serve a similar purpose. Components are considered autonomous, encapsulated units within a system or subsystem that provide one or more interfaces.

Component Diagram Notations

1. Component

A component is drawn as a rectangle with optional compartments stacked vertically. A component can be represented as just a rectangle with the component's name and the component stereotype text and/or icon. The component stereotype's text is "<<component>>" and the component stereotype icon is a rectangle with two smaller rectangles.

1.1 Component Interfaces

- **Provide Interface**

Provided interfaces define "a set of public attributes and operations that must be provided by the classes that implement a given interface".

- **Required Interface**

Required interfaces define "a set of public attributes and operations that are required by the classes that depend upon a given interface".

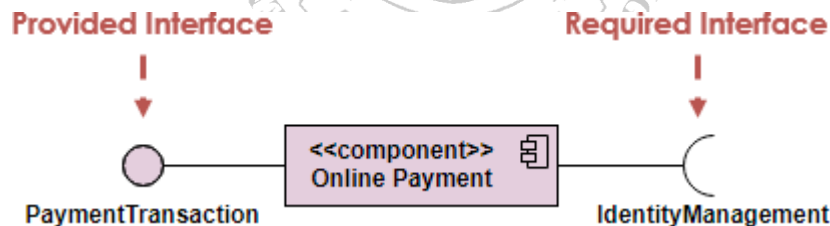


Fig 6.1 Component & its Interfaces

2. Port

A port (definition) indicates that the component itself does not provide the required interfaces (e.g., required or provided). Instead, the component delegates the interface(s) to an internal class.

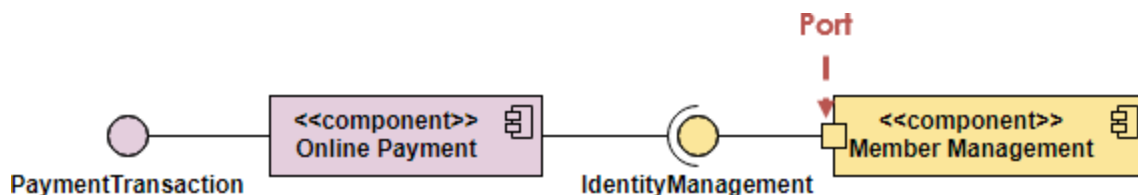


Fig 6.2 Ports

Deployment diagram in UML

In the UML, deployment diagrams is used to visualize the static aspect of these physical nodes and their relationships and to specify their details for construction. Deployment diagrams are one of the two kinds of diagrams used in modeling the physical aspects of an object-oriented system. A deployment diagram shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams address the static deployment view of an architecture. They are related to component diagrams in that a node typically encloses one or more components.

Deployment Diagram Notations

1. Node

A node is a run-time physical object that represents a computational resource, generally having memory and processing capability. You can model node types and node instances. You may model the component instances that run or live on a node by drawing them within the node. You may model which nodes communicate with one another using the Connection relationship line.

2. Dependency

A dependency indicates that one model element (source) depends on another model element (target), such that a change to the target element may require a change to the source element in the dependency. In a deployment diagram, you can use the dependency relationship to show the capability of a node type to support a component type. You may also use the relationship to show the dependency between component types.

3. Connection

A connection depicts the communication path used by the hardware to communicate usually indicates the method i.e. TCP/IP.

4. Artifact

Artifacts represent concrete elements in the physical world that are the result of a development process. Examples of artifacts are executable files, libraries, archives, database schemas, configuration files, etc.

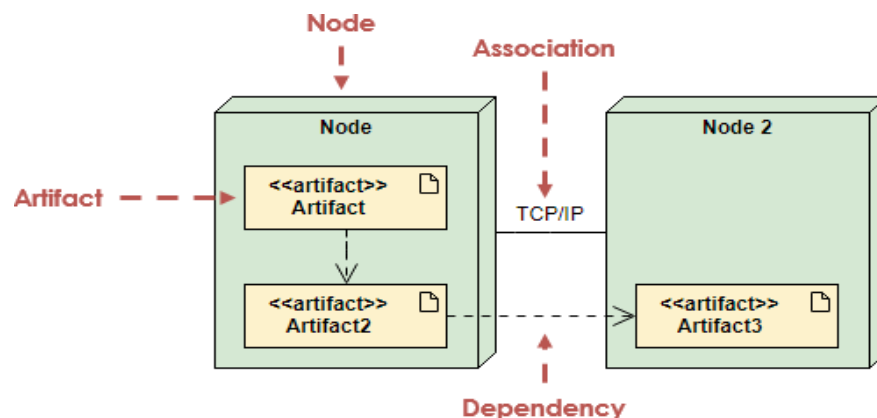


Fig 6.2 UML Deployment Diagram Example

Component Diagram Example - Components in Deployment Diagram

Models the physical deployment of software components with UML deployment diagram. In deployment diagram, hardware components (e.g. web server, mail server, application server) are presented as nodes, with the software components that run inside the hardware components presented as artifacts.

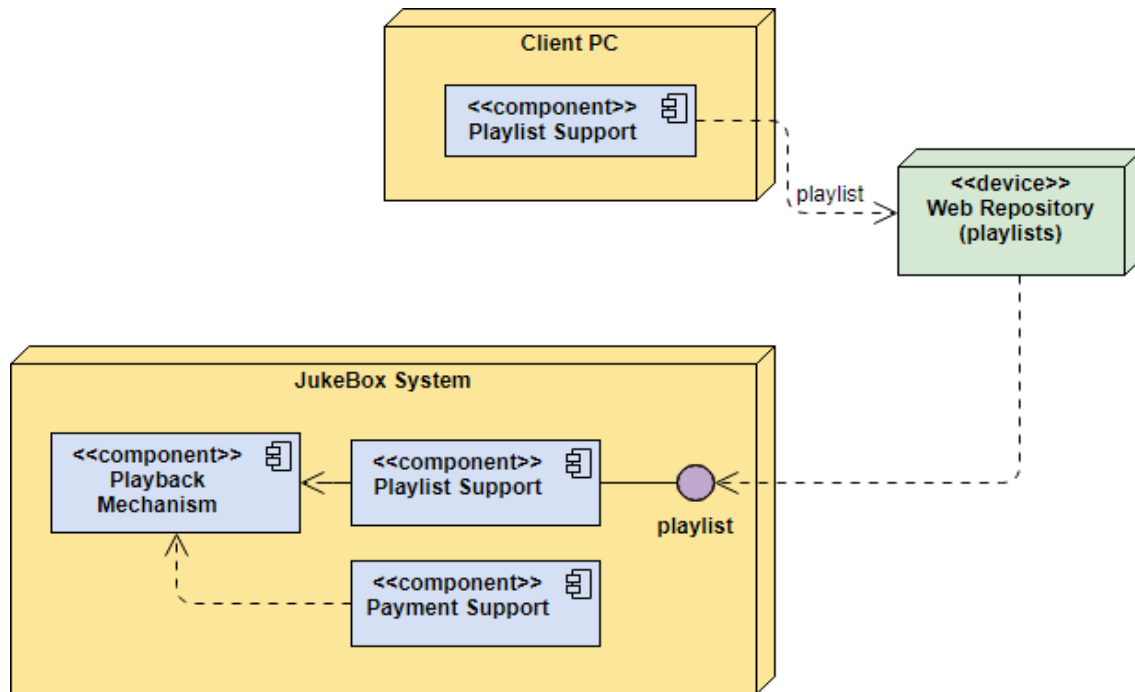


Fig 6.3 UML Component Deployment Diagram Example

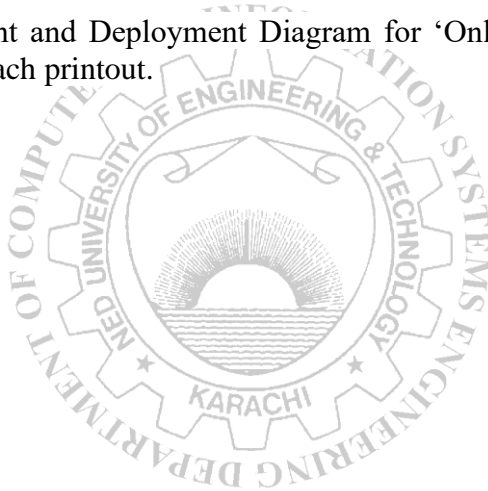
EXERCISES:

- Develop the Component Diagram following the specifications given below: (Attach printout)
 - Create a package called Registration
 - Create the main component diagram for the Registration package
 - Create three components on the Main component diagram for the Registration package – Registrar.exe, Courses.dll and People.dll //Create dummy files for this purpose
 - Create a dependency relationship between
 - Registrar.exe and Courses.dll
 - Registrar.exe and People.dll
- Develop the Deployment Diagram following the specifications given below: (Attach printout)

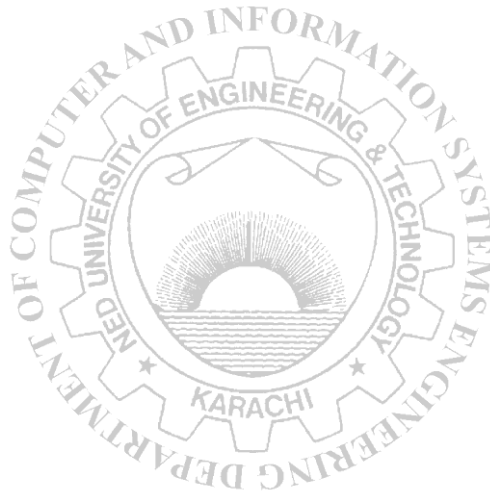
- Add the following nodes (processors): Registration, Database, Main Building, Dorm, Library
- Add the connections as shown in the table below:

Node	Connected to Node
Registration	Database
Registration	Main Building
Registration	Dorm
Registration	Library

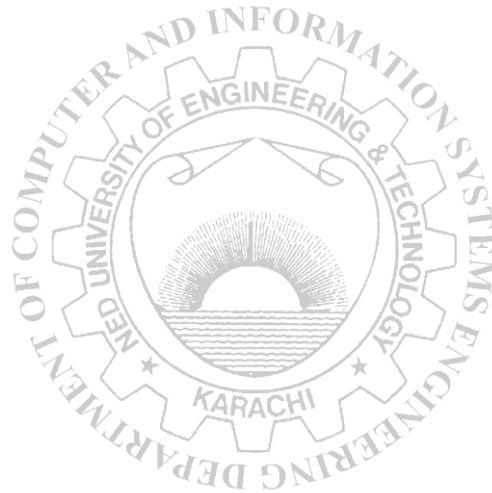
3. Construct a deployment diagram for an online shopping system using StarUML. Identify all the nodes, components, artifacts, interfaces, connections and dependencies. Also attach the printout.
4. Create the Component and Deployment Diagram for 'Online Bookshop System' using Visual Paradigm. Attach printout.



Attach print outs here:



Attach print outs here:



Lab Session 07

Use Design Principles in SDLC

Design patterns

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

Builder pattern

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. A builder class builds the final object step by step. This builder is independent of other objects.

Implementation

We have considered a business case of fast-food restaurant where a typical meal could be a burger and a cold drink. Burger could be either a Veg Burger or Chicken Burger and will be packed by a wrapper. Cold drink could be either a coke or pepsi and will be packed in a bottle.

We are going to create an *Item* interface representing food items such as burgers and cold drinks and concrete classes implementing the *Item* interface and a *Packing* interface representing packaging of food items and concrete classes implementing the *Packing* interface as burger would be packed in wrapper and cold drink would be packed as bottle.

We then create a *Meal* class having *ArrayList* of *Item* and a *MealBuilder* to build different types of *Meal* objects by combining *Item*. *BuilderPatternDemo*, our demo class will use *MealBuilder* to build a *Meal*.

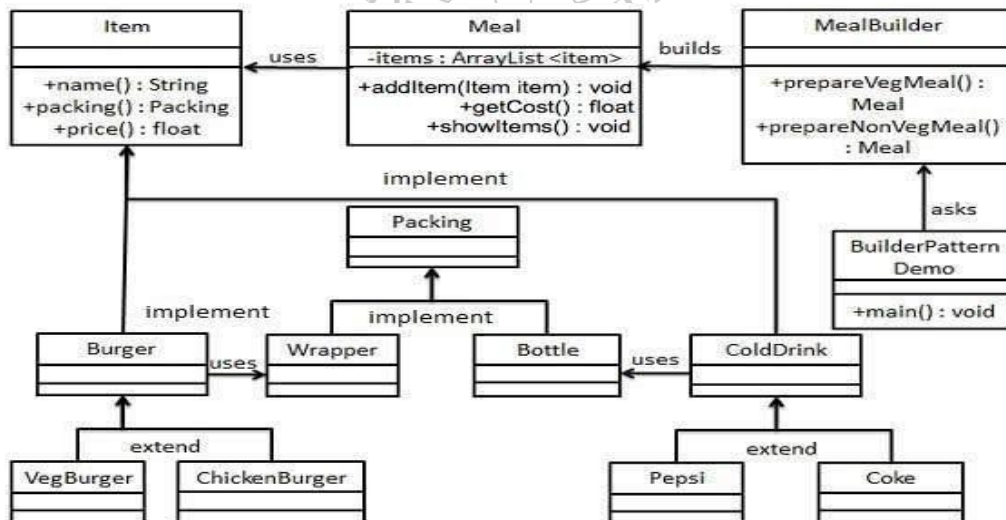


Fig 7.1 Builder Pattern for Fast Food Restaurant System

Step 1

Create an interface Item representing food item and packing.

Item.java

```
public interface Item {  
    public String name();  
    public Packing packing();  
    public float price();  
}
```

Packing.java

```
public interface Packing {  
    public String pack();  
}
```

Step 2

Create concrete classes implementing the Packing interface.

Wrapper.java

```
public class Wrapper implements Packing {  
  
    @Override  
    public String pack() {  
        return "Wrapper";  
    }  
}
```

Bottle.java

```
public class Bottle implements Packing {  
  
    @Override  
    public String pack() {  
        return "Bottle";  
    }  
}
```

Step 3

Create abstract classes implementing the item interface providing default functionalities.

Burger.java

```
public abstract class Burger implements Item {  
  
    @Override  
    public Packing packing() {  
        return new Wrapper();  
    }  
  
    @Override  
    public abstract float price();  
}
```

ColdDrink.java

```
public abstract class ColdDrink implements Item {  
    @Override  
    public Packing packing() {  
        return new Bottle();  
    }  
}
```

```
        @Override
        public abstract float price();
    }
```

Step 4

Create concrete classes extending Burger and ColdDrink classes

VegBurger.java

```
public class VegBurger extends Burger {
```

```
    @Override
    public float price() {
        return 25.0f;
    }
```

```
    @Override
    public String name() {
        return "Veg Burger";
    }
}
```

ChickenBurger.java

```
public class ChickenBurger extends Burger {
```

```
    @Override
    public float price() {
        return 50.5f;
    }

    @Override
    public String name() {
        return "Chicken Burger";
    }
}
```

Coke.java

```
public class Coke extends ColdDrink {
```

```
    @Override
    public float price() {
        return 30.0f;
    }
```

```
    @Override
    public String name() {
        return "Coke";
    }
}
```

Pepsi.java

```
public class Pepsi extends ColdDrink {
```

```
    @Override
```

```
public float price() {  
    return 35.0f;  
}  
  
@Override  
public String name() {  
    return "Pepsi";  
}  
}
```

Step 5

Create a Meal class having Item objects defined above.

Meal.java

```
import java.util.ArrayList;  
import java.util.List;  
  
public class Meal {  
    private List<Item> items = new ArrayList<Item>();  
  
    public void addItem(Item item){  
        items.add(item);  
    }  
  
    public float getCost(){  
        float cost = 0.0f;  
  
        for (Item item : items) {  
            cost += item.price();  
        }  
        return cost;  
    }  
  
    public void showItems(){  
  
        for (Item item : items) {  
            System.out.print("Item : " + item.name());  
            System.out.print(", Packing : " + item.packing().pack());  
            System.out.println(", Price : " + item.price());  
        }  
    }  
}
```

Step 6

Create a MealBuilder class, the actual builder class responsible to create Meal objects.

MealBuilder.java

```
public class MealBuilder {  
  
    public Meal prepareVegMeal () {  
        Meal meal = new Meal();  
        meal.addItem(new VegBurger());  
        meal.addItem(new Coke());  
    }  
}
```

```
        return meal;
    }

    public Meal prepareNonVegMeal () {
        Meal meal = new Meal();
        meal.addItem(new ChickenBurger());
        meal.addItem(new Pepsi());
        return meal;
    }
}
```

Step 7

BuilderPatternDemo uses MealBuilder to demonstrate builder pattern.

BuilderPatternDemo.java

```
public class BuilderPatternDemo {
    public static void main(String[] args) {

        MealBuilder mealBuilder = new MealBuilder();

        Meal vegMeal = mealBuilder.prepareVegMeal();
        System.out.println("Veg Meal");
        vegMeal.showItems();
        System.out.println("Total Cost: " + vegMeal.getCost());

        Meal nonVegMeal = mealBuilder.prepareNonVegMeal();
        System.out.println("\n\nNon-Veg Meal");
        nonVegMeal.showItems();
        System.out.println("Total Cost: " + nonVegMeal.getCost());
    }
}
```

Step 8

Verify the output.

Veg Meal

Item : Veg Burger, Packing : Wrapper, Price : 25.0

Item : Coke, Packing : Bottle, Price : 30.0

Total Cost: 55.0

Non-Veg Meal

Item : Chicken Burger, Packing : Wrapper, Price : 50.5

Item : Pepsi, Packing : Bottle, Price : 35.0

Total Cost: 85.5

Singleton pattern

Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

Implementation

We're going to create a *SingleObject* class. *SingleObject* class have its constructor as private and have a static instance of itself.

SingleObject class provides a static method to get its static instance to outside world. *SingletonPatternDemo*, our demo class will use *SingleObject* class to get a *SingleObject* object.

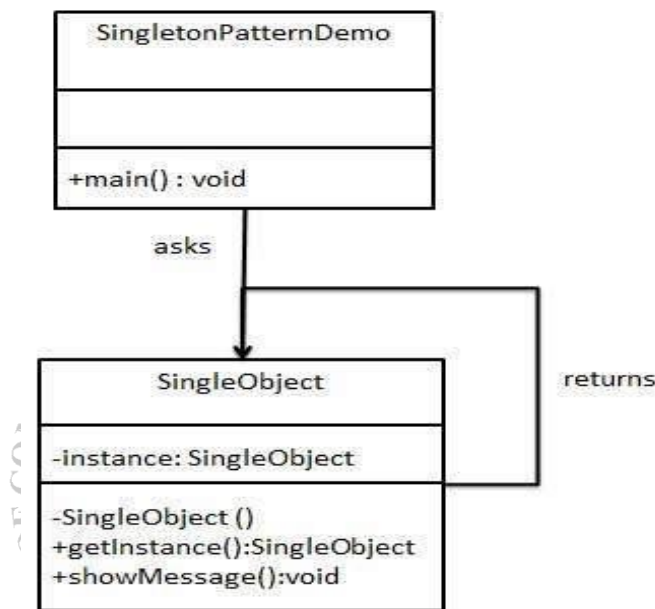


Fig 7.2 Singleton Pattern

Step 1

Create a Singleton Class.

SingleObject.java

```

public class SingleObject {

    //create an object of SingleObject
    private static SingleObject instance = new SingleObject();

    //make the constructor private so that this class cannot be
    //instantiated
    private SingleObject(){}

    //Get the only object available
    public static SingleObject getInstance(){
        return instance;
    }

    public void showMessage(){
  
```

```
        System.out.println("Hello World!");  
    }  
}
```

Step 2

Get the only object from the singleton class.

SingletonPatternDemo.java

```
public class SingletonPatternDemo {  
    public static void main(String[] args) {  
  
        //illegal construct  
        //Compile Time Error: The constructor SingleObject() is not  
        visible  
        //SingleObject object = new SingleObject();  
  
        //Get the only object available  
        SingleObject object = SingleObject.getInstance();  
  
        //show the message  
        object.showMessage();  
    }  
}
```

Step 3

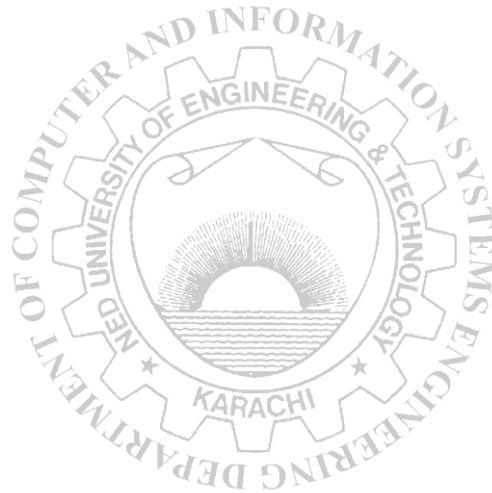
Verify the output.

Hello World!

EXERCISES

1. Implement each of the type of design pattern covering some real world entity. Attach printout of the code along with output.
2. Explore Factory Patterns in Java and elaborate by using an example. Attach printout of code along with the output.

Attach print outs here:



Lab Session 08

Use principles of Program Testing in SDLC

Program testing

Following are the steps of program testing in SDLC:

1. Choosing Test Data

The choice of data to be used to test programs and functions, depends on the project under development. Hence, only general guidelines can be established for choosing test data.

First we should note that *the quality of test data is more important than its quantity*. Many sample runs that do the same calculations in the same cases provide no more effective a test than one run. Program testing can be used to show the presence of faults, but never their absence.

It is possible that other cases remain that have never been tested even after many sample runs. For any program of substantial complexity, it is impossible to perform exhaustive tests, yet the careful choice of test data can provide substantial confidence in the program.

Everyone, for example, has great confidence that the typical computer can add two floating point numbers correctly, but this confidence is certainly not based on testing the computer having it add all possible floating point numbers and checking the results. If a double precision floating point number takes 64 bits, then there are 2^{128} distinct pairs of numbers that could be added. This number is astronomically large: all computers manufactured to date have performed altogether but a tiny fraction of this number of additions. Our confidence that computers add correctly is based on tests of each component separately, that is, by checking that each of the 64 bits is added correctly, and that carrying from one place to another is done correctly.

2. Testing Methods

a) The Black-Box Method

Most users of a large program are not interested in the details of its functioning; they only wish to obtain answers. That is, they wish to treat the program as a black box; hence the name of this method.

Data Selection for Black Box Testing

Test data should be chosen according to the specifications of the problem, without regard to the internal details of the program, to check that the program operates correctly. At a minimum the test data should be selected in the following ways:

- **Easy Values**

The program should be debugged with data that are easy to check. Generally, programmers who try a program only for complicated data, and think it works properly, are embarrassed when an independent tester tries trivial data.

- **Typical Realistic Values**

Always try a program on data chosen to represent how the program will be used. These data should be sufficiently simple so that the results can be checked by hand.

- **Extreme Values**

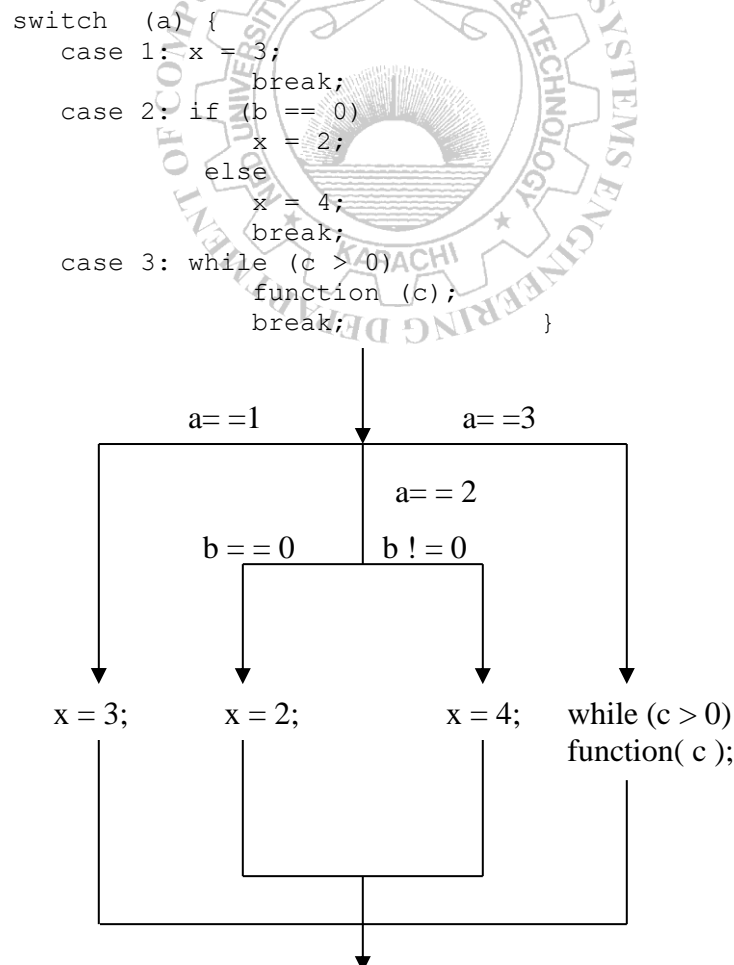
Many programs err at the limits of their range of applications. It is very easy for counters or array bounds to be off by one.

- **Illegal Values**

“Garbage In, Garbage Out” is an old saying in computer circles that should not be respected. When a good program has garbage coming in, then its output should at least be a sensible error message. Indeed, the program should provide some indication of the likely errors in input and perform any calculations that remain possible after discarding the erroneous input.

b) The Glass-Box Method

The second approach to choosing test data begins with the observation that a program can hardly be regarded as thoroughly tested if there are some parts of its code that, in fact, have never been executed. In the *glass-box method* of testing, the logical structure of the program is examined, and for each alternative that may occur, test data are devised that will lead to that alternative. Thus care is taken to choose data to check each possibility in every case statement, each clause of every if statement, and the termination condition of each loop. If the program has several selection or iteration statements then it will require different combinations of test data to check all the paths that are possible. Consider a short program segment with its possible execution paths.



For a large program *glass-box* approach is clearly not practicable, but for a single small module, it is an excellent testing method. In a well-designed program, each module will involve few loops and alternatives. Hence only a few well-chosen test cases will suffice to test each module on its own.

In *glass-box* testing, the advantages of modular program design become evident. Let us consider a typical example of project involving 50 functions, each of which can involve 5 different cases or alternatives. If we were to test the whole program as one, we would need 5^{50} test cases to be sure that each alternative was tested. Each module separately requires only 5 (easier) test cases, for a total of $5 \times 50 = 250$. Hence a problem of impossible size has been reduced to one that, for a large program, is of quite modest size.

Comparison

In practice, *black-box* testing is usually more effective in uncovering errors. One reason is that the most subtle programming errors often occur not within a function but in the interface between functions, in misunderstanding of the exact conditions and standards of information interchange between functions.

A reasonable testing philosophy for a large project would be to apply *glass-box* methods to each small module as it is written and use *black-box* test data to test larger sections of the program when they are complete.

TEST CASE TEMPLATE

Several standard fields of a sample Test Case template are listed below:

- **Test case ID:** Unique ID is required for each test case. Follow some convention to indicate the types of the test. **E.g.** 'TC_UI_1' indicating 'user interface test case #1'.
- **Test priority (Low/Medium/High):** This is very useful while test execution. Test priority for business rules and functional test cases can be medium or higher whereas minor user interface cases can be of a low priority. Test priority should always be set by the reviewer.
- **Module Name:** Mention the name of the main module or the sub-module.
- **Test Designed By:** Name of the Tester.
- **Test Designed Date:** Date when it was written.
- **Test Executed By:** Name of the Tester who executed this test. To be filled only after test execution.
- **Test Execution Date:** Date when the test was executed.
- **Test Title/Name:** Test case title. **E.g.** verify login page with a valid username and password.
- **Test Summary/Description:** Describe the test objective in brief.
- **Pre-conditions:** Any prerequisite that must be fulfilled before the execution of this test case. List all the pre-conditions in order to execute this test case successfully.
- **Dependencies:** Mention any dependencies on the other test cases or test requirement.
- **Test Steps:** List all the test execution steps in detail. Write test steps in the order in which they should be executed. Make sure to provide as many details as you can. **Tip** – In order to manage a

test case efficiently with a lesser number of fields use this field to describe the test conditions, test data and user roles for running the test.

- **Test Data:** Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.
- **Expected Result:** What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on the screen.
- **Post-condition:** What should be the state of the system after executing this test case?
- **Actual result:** Actual test result should be filled after test execution. Describe the system behavior after test execution.
- **Status (Pass/Fail):** If actual result is not as per the expected result, then mark this test as **failed**. Otherwise, update it as **passed**.

Test Case Template						
Project Name:						
Test Case ID: <i>Fun_10</i>			Test Designed by: <i><Name></i>			
Test Priority (Low/Medium/High): <i>Med</i>			Test Designed date: <i><Date></i>			
Module Name: <i>Google login screen</i>			Test Executed by: <i><Name></i>			
Test Title: <i>Verify login with valid username and password</i>			Test Execution date: <i><Date></i>			
Description: <i>Test the Google login page</i>						
Pre-conditions: <i>User has valid username and password</i>						
Dependencies:						
Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User= <i>sample@gmail.com</i>	User should be able to login	User is navigated to	Fail	
2	Provide valid username	Password: <i>1234</i>		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					
Post-conditions: <i>User is validated with database and successfully login to account. The account session details are logged in database.</i>						

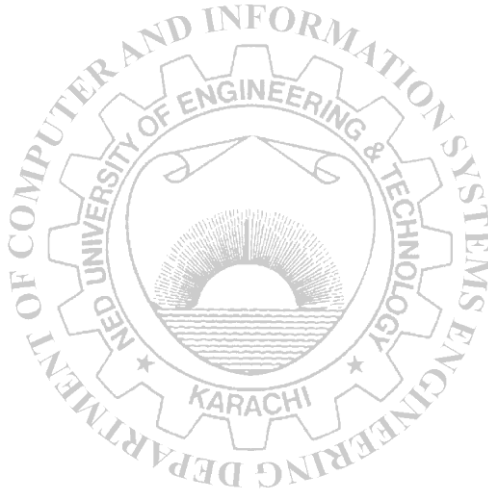
Fig 8.1 Test Case Template

EXERCISES

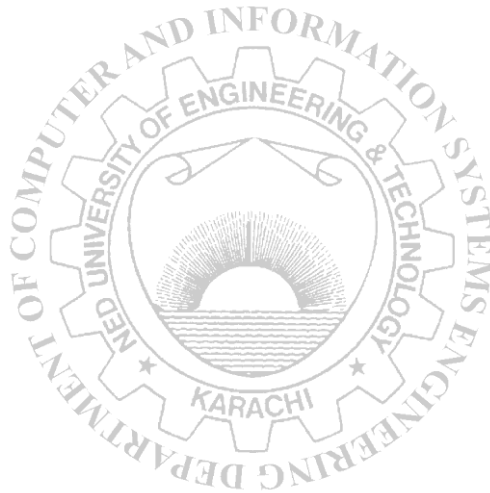
- Find suitable *black-box* test data for each of the following. Also prepare the complete Test Case Document (Attach print outs)
 - A function that returns the largest of its three parameters, which are real numbers.
 - A function that returns the least common multiple of its two parameters, which must be positive integers.
 - A function that returns the square root of a real number.
 - A function that sorts three integers, given as its parameters, into ascending order.

- v) A function that sorts an array A of integers indexed from 0 to a variable n-1 into ascending order, where A and n are both parameters.
- b) Find suitable *glass-box* test data for the following statement. Attach the print out of the example code and test data.

```
if (a < b) if (c > d) x = 1; else if (c == d) x = 2;  
else x = 3; else if (a == b) x = 4; else if (c == d) x = 5;  
else x = 6;
```



Attach print outs here:



Lab Session 09

Practice Web Development & Testing using Agile Project Management (Scrum)

Agile project management (Scrum)

Scrum project management is a methodology for managing software delivery that comes under the broader umbrella of agile project management. It provides a lightweight process framework that embraces iterative and incremental practices, helping organizations deliver working software more frequently. Projects progress via a series of iterations called sprints; at the end of each sprint the team produces a potentially deliverable product increment.

The Scrum approach to project management enables software development organizations to prioritize the work that matters most and break it down into manageable chunks. Scrum is about collaborating and communicating both with the people who are doing the work and the people who need the work done. It's about delivering often and responding to feedback, increasing business value by ensuring that customers get what they actually want.

Activities in Scrum Project Management

The main activity in Scrum project management is the Sprint, a time boxed iteration that usually lasts between 1-4 weeks, with the most common sprint length being 2 weeks.

Sprint Planning Meeting: At the start of each sprint a planning meeting is held to discuss the work that is to be done. The product owner and the team meet to discuss the highest-priority items on the product backlog. Team members figure out how many items they can commit to and then create a sprint backlog, which is a list of the tasks to complete during the sprint.

Daily scrum or daily standup: Each day during the sprint team members share what they worked on the prior day, will work on today, and identify any impediments. Daily scrums serve to synchronize the work of team members as they discuss the work of the sprint. These meetings are time boxed to no more than 15 minutes.

Sprint Review: At the end of a sprint the team demonstrates the functionality added during the sprint. The goal of this meeting is to get feedback from the product owner and any users or other stakeholders who have been invited to the review.

Sprint Retrospective: At the end of each sprint the team participates in a retrospective meeting to reflect on the sprint that is ending and identify opportunities to improve in the new sprint.

SPRINT PLANNING

1. Requirement Specifications:

We intend to design and develop a basic website with a proper UI and DML operations. The web pages should be presentable, user friendly and responsive. A website is designed to serve a purpose, usually to solve a problem. For example, a job board has a purpose where employers can post jobs and job seekers can find and apply for jobs. Once applied, there should be a way for candidates and employers to communicate and keep up to date with a job application. If you build a job board and it only lets you post jobs, that's not enough. All the contents should be relevant. You do not want to put car details on a job

board. If your website is a job board, it should have content and tips related to being a good candidate such as how to create a professional resume and how to behave in an interview. You do not want to post about cars or sports on a job board. It should be functional, scalable and secure.

2. Mock Design – Wireframing:

A wireframe (sometimes called mockups or low-fidelity prototypes) is a simple presentation of a website which only shows a website at the structural level. However, simple doesn't mean a wireframe is easy to create. Wire-framing is a necessary step during the whole interactive design process, and it often takes place early in the project lifecycle. A wireframe is a graphical skeleton of a website, similar to an architectural blueprint.

- **Start with a Sitemap**

You'll want to create a sitemap of the existing website or design a new information architecture, discuss content strategy, and build personas. A hierarchy diagram best represents a sitemap.

- **Create a Wireframe**

Wireframes will help visualize and provide a structure for these ideas. Once you begin creating your wireframe don't use any distracting colors or images. You want the wireframe to focus on layout and behavior. Even the font you use should be generic. Images should be represented with a box with an X through them. Copy on the page can be placeholder copy (Lorem Ipsum) or canned copy like generic calls to action such as "sign up" or "buy." As you iterate your way to a final wireframe, you may find that generic copy doesn't do your vision justice. You can replace your initial placeholders with more detailed copy, especially if that copy takes up more space. That will affect the layout and other elements on the page. An example wire-frame is provided in Fig 9.1

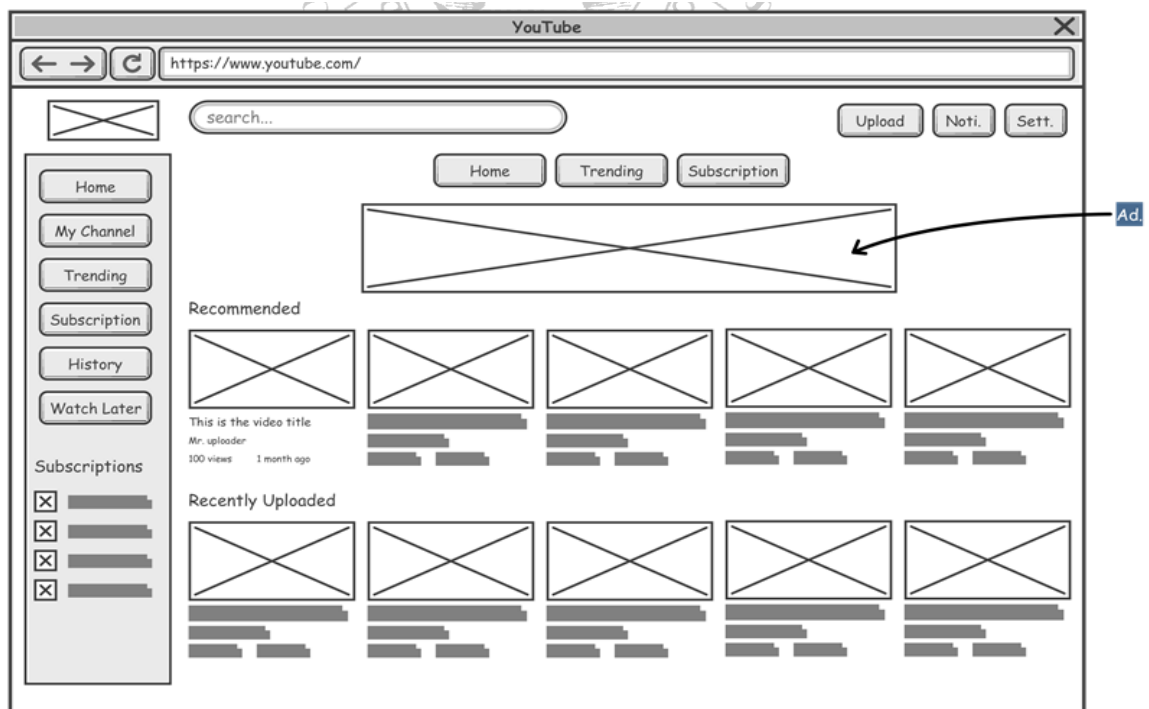


Fig 9.1 Wire-frame Example for You Tube Website

Following are some of the best open source and online wire-framing tools:

- Axure
- Mockplus
- Mockplus iDoc
- Adobe XD
- Wireframe CC
- Balsamiq Mockups
- Fireworks
- Photoshop

Web development using ASP.NET

ASP.NET is a server side scripting technology that enables scripts (embedded in web pages) to be executed by an Internet server. It allows developers to create dynamic web pages. Dynamic web pages are pages whose content is dynamically regenerated each time the web page is requested. It is a program that runs inside the windows server component IIS (Internet Information Services). An ASP.NET file can contain HTML, XML, and scripts. Scripts in an ASP.NET file are executed on the server. An ASP.NET file has the file extension ".aspx".

Creating a New Website:

We will create website using Microsoft Visual Studio. The website can be built either using Visual C# or Visual Basic. Select the default ASP.Net Website from the template and type in the name for the project after selecting the language (here we select Visual C#).

The default.aspx is included in the project. To rename it, right click on the Default.aspx select Rename from the popup menu and then name it to First.aspx.

- **Source View**

Source view displays the HTML markup for the Web page, which you can edit. By default, all HTML elements and scripts are displayed when the Source View is initially selected. Elements can be dragged from the Toolbox, just as they are dragged when editing a Web page in Design view, and then see their markup inserted into the document.

To select Source view, click the Source tab at the bottom of the HTML Designer window.

- **Design View**

Design view displays ASP.NET Web pages, master pages, content pages, HTML pages, and user controls. It allows you to add text and elements and then position and size them and set their properties using special menus or the Properties window. To select Design view, click the Design tab at the bottom of the HTML Designer window.

- **Properties Window**

It's a window where you can edit properties or events for the controls. To access Properties Window, Select Properties Window on the View menu.

- **Solution Explorer**

Solution Explorer provides you with an organized view of your projects and their files as well as ready access to the commands that pertain to them. A toolbar associated with this window offers commonly used commands for the item you highlight in the list. To access Solution Explorer, select Solution Explorer on the View menu.

Working with Master Pages in ASP.Net

One of the most prominent features of ASP.NET is a technique called Master Pages. It provides the means to define the overall layout of an ASP.NET application at a single point (the .master file) and reuse it in all content pages that derive from the master. Master Pages are templates that can be created to control the look and feel of the content pages. If multiple pages share the same header, footer, navigation bar, and content elements, for example, Master Pages will make this work much easier. This model allows developers to define and centralize a site-wide page layout, thereby making it easier to create a consistent look and feel across all pages that can easily be updated.

Creating a Master Page

To create a master page, right-click on the project name in the Solution Explorer and choose Add New Item. Then select the Master Page type from the list of templates and rename it as FirstMasterPage.master. The master page can be edited using the design view as well as source view. If a style sheet is to be added here, select the document in the properties window and provide the link of the .css file in the Style Sheet property.

By default a Content Place Holder is visible in the Master page. In the source view, the following code is visible for this Content Place holder:

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
```

Any text that is written in between these tags will be visible in the master page only. This means that the contents of this place holder will vary from webpage to webpage while the contents outside it will remain static. Multiple content holders can be added to give space for more contents. A Content Place Holder can be added in a webpage by dragging it from the Standard toolbox.

Sometimes it is difficult to move a Content Place Holder or to align it to a desired place. To do so, add a table to the webpage and drag the Content Place Holder to a desired cell. This will help in achieving a uniform and more professional look.

Adding Master Page to a Webpage

A Master Page can be added to a new webpage as well as the existing pages. To add the Master page to a new webpage, first click the Add New Item from the root directory and then add a new Web Form. On the lower right corner of this Add New Item window, there is a check box to Select Master page. Check this option and click OK. Another window will pop up asking to select which Master Page you want to add. Select the desired master page and click OK.

To add Master page to an existing page add the following code, with necessary changes as per your requirements, should be added in the Source View of the webpage.

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
AutoEventWireup="true" CodeFile="Home.aspx.cs" Inherits="Home"
Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
    </asp:Content>
```

Here the MasterPageFile should contain the file name of the Master Page that is present in the project. The CodeFile contains the name of the webpage in which this code is added. The ContentPlaceHolderID should also be changed if it is changed on the Master Page.

Creating a Site Map

One of the challenges of managing a website composed of more than a handful of pages is providing a straightforward way for visitors to navigate through the site. To begin with, the site's navigational structure must be defined. Next, this structure must be translated into navigable user interface elements, such as menus or breadcrumbs. Finally, this whole process needs to be maintained and updated as new pages are added to the site and existing ones removed.

The ASP.NET 2.0 site navigation system provides a means for a developer to define a site map and to then access this information through a programmatic API. ASP.NET ships with a site map provider that expects site map data to be stored in an XML file formatted in a particular way.

To add a site map, click on the root directory to add a new item and select the Site Map from the Add New Item Window. Change the Site Map file to look something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
<siteMapNode url="Home.aspx" title="Home" description="This is the home page"
>
<siteMapNode url="Login.aspx" title="Login" description="This is the Login
page" />
<siteMapNode url="SignIn.aspx" title="Sign In" description="This is the Sign
Up page" />
</siteMapNode>
</siteMap>
```

The site map file is an XML file. The site map file must have the <siteMap> node as its root node, which must contain precisely one <siteMapNode> child element. That first <siteMapNode> element can then contain an arbitrary number of descendent <siteMapNode> elements.

There are three options to include a site map to a webpage:

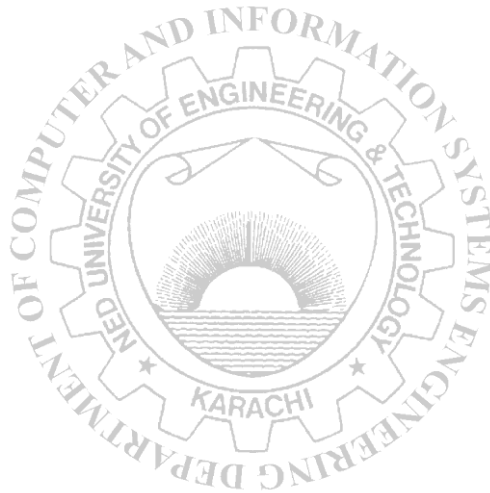
1. SiteMapPath
2. Menu
3. TreeView

Drag any of the desired view from the Navigation Toolbox into the Master Page or any other webpage. By using the Smart Tag, select the new data source option from the Choose Data Source menu. Select the SiteMap from the pop up window and click OK. The menus can be formatted by using the Auto Format option.

EXERCISES

1. Suppose that you have to develop a website in a 3 weeks period. We will use Scrum to rapidly deliver the product in a limited time frame. Provide the Requirement Specification Document for a Web-based Project. Highlight all the functional and non-functional user requirements. Point out all the deliverables in 1st, 2nd and final sprint. Attach print out.
2. Design wireframes for the website you are going to develop using any online wireframing/mock design tool (E.g. Balsamiq). Also attach the print outs.
3. Create a master page for your website using ASP.Net. Attach the code snippet and output.
4. Test your master page by comparing it with the wireframes that you have created while sprint planning.

Attach print outs here:



Lab Session 10

Demonstrate first sprint & plan second sprint of Web Development & Testing using Scrum

Sprint review

At the end of first sprint, before moving further every group member is required present all the tasks completed up till now. If any task still needs to be completed or requires improvement, sprint backlog document should be prepared and must be carried out in the second sprint.

Sprint planning

In the second sprint, we plan to add basic controls in the website. By the end of this sprint, LOGIN and SIGNUP pages will be added in the website.

Adding basic controls in a website using ASP.NET

- **Adding Button Control on the Screen**

Double click on the Button icon present in the Standard toolbox to add it to the website. Drag to the desired location. The default ID for this Button is Button1. To change this ID go to properties window, type the ID as **OKButton**. Also change the Text property of this control to **OK**. Drag a label on the website and remove the text label from its properties. Double click the **OKButton** to open its code and add the following code. Use Ctrl+Shift+S to save the project.

```
protected void OKButton_Click(object sender, EventArgs e)
{
    Label1.Text = "Hello User";
}
```

- **Move the Control on the Screen**

To drag it to the desired position on the page, go to Format → Set Position → Absolute. This option can also be applied to the controls using Cascading Style Sheets. The Cascading Style Sheets or CSS allow you to control the layout and look of your page easily. CSS tags or properties are easy to use and affect the look and feel or style of your pages by setting its Property: CSS Class to the already defined Style Sheet.

- **Adding a Hyperlink**

From the toolbox drag a Hyperlink control to the webpage. Change its Text property to **CIS Department** and its NavigateUrl property to <http://www.neduet.edu.pk/CISE>. To open this link in a new browser, change its Target property to **_blank**.

- **Adding a Table**

To insert a table on a webpage, click Table → Insert Table. An Insert Table window pops up enabling a user to select from the available templates or creating a customized table with required number of cells and cell formatting.

- **Add a Dropdown List**

Drag the Drop down list from the standard toolbox. Change its ID to DeptDropDown. Click on the arrow on upper right hand corner of the control to see the various options available. Select the Edit Items to add a number of options from which the user can select. The ListItem Collection Editor opens in a pop up window. Click Add to add a new entry in the drop down list and type in the name of the entry in the Text property. Click Add to enter another option or OK to exit.

- **Adding Radio Buttons**

To add radio buttons on a webpage, drag the number of radio buttons required (here consider two radio buttons are being dragged) from the standard toolbox. Rename them as studentRadioButton and facultyRadioButton. Also change their text property to Student and faculty respectively. In the GroupName property, write User for both of the radio buttons which makes them to belong to same group.

- **Field Validation**

Validation controls are available in the toolbox. Some of these are discussed below:

i) Required Field Validator

Required field validator is used for required fields. Fields that cannot be left empty is validated using the required field validator. A good example will be a TextBox which is used for taking the username as the input. Since the user must enter the username in the TextBox in order to access the website. The required field validator can be used with the TextBox control to validate the input. If user does not enter any data in the TextBox than an error message is displayed and further processing of the request will be stopped.

In order to set the required validator on the TextBox control just drag and drop the validator on the webform and set its ControlToValidate property to the TextBox id that is to be validated.

ii) Regular Expression Validator

Regular Expression Validator is used to check the user input against some built in regular expressions. The control provides the RegularExpression collection property that can be used to select the desired Regular Expression. Some of the built in expressions are Email, postal code, telephone number and many more. If the user input does not match the regular expression expected an error will occur and the request for the resources will not be authorized unless the user corrects the input data.

iii) Range Validator

The Range Validator control is used to check whether the input control contains value in the specified range. You can check the range of values against different data types such as String, Date, and Integer and so on. The two most important properties of the range validator control is the Maximum value and the minimum value. The range can be used to restrict the user data. Suppose if the user is entering a roll number so the range can be set from 1 to 500.

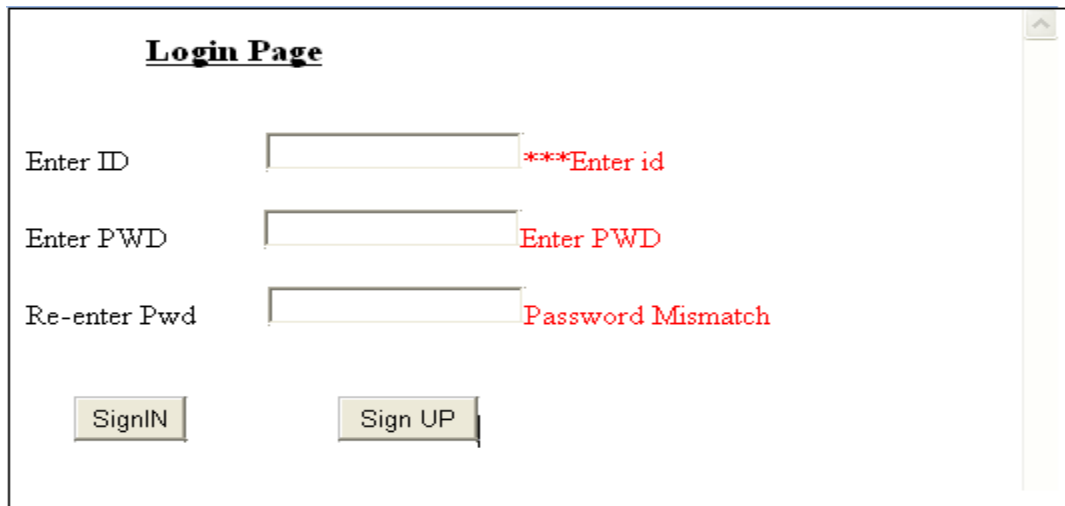
iv) Compare Validator

The compare validator control is used to compare the input server control's value. The compare validator can be used to compare against a value or another control. If both the ControlToCompare and ValueToCompare properties are set for a CompareValidator control, the ControlToCompare property takes precedence.

It can be used to compare whether the user has selected a value from the dropdown list or not. For this purpose, select the dropdown list from ControlToValidate property and in ValueToCompare write down the string e.g. Select a Department (which must be the first entity in the dropdown list) and set the Operator property to NotEqual.

Another good use of the compare validator is to check whether the passwords entered by the user in the two TextBoxes while registering for a website are same or not. This validator also performs validation on the client side as well as the server side. So, no postback will be required when doing the comparison and hence resources will be saved and performance will increase.

Creating a login page by using different controls



The screenshot shows a web form titled "Login Page". It contains three text boxes for input: "Enter ID", "Enter PWD", and "Re-enter Pwd". To the right of the "Enter ID" box is a red error message "***Enter id". To the right of the "Enter PWD" box is a red error message "Enter PWD". To the right of the "Re-enter Pwd" box is a red error message "Password Mismatch". At the bottom of the form are two buttons: "SignIN" and "Sign UP".

Fig 10.1 LOGIN Page using Validation Controls

ASP.Net Website Administration Tool

In ASP.Net, a login page can be inserted very easily by a series of mouse clicks i.e. no time is spent in writing extensive code for the authentication purposes. Open the ASP.Net Configuration from the Website menu to view the website administration tools. This option provides, amongst other facilities, the ability to update the security settings for a website.

Various options are available here, through which new user can be created or an existing user can be edited. Roles of the users can be created and the access rules can also be created or edited.

To create a new role, click on the Enable Roles option and then click on the Create or manage roles link. Enter the name of the role (e.g. Administrator or Premium User) and click OK. Now once a role is created, its users can be defined and the access rules for the user of this role can be set. To add a new user, first set the authentication type to From the Internet. Select the Create User link and fill up the form for a new user. Make sure to enter a password with having minimum 7 characters including a non-alphanumeric character. Close this explorer window and return to the Visual Studio environment.

- **Creating a Login Page**

Add a Web form to the project and name it Login.aspx. In its design view, drag the Login control from the Login Toolbox. From the Smart Tag, this control can be formatted or the text boxes or the labels present in it can be edited too. In the DestinationPageUrl property, write down the name of the webpage which the

user can visit after logging in. On the Home Page, a login hyperlink can be created which will link to this login page or set the login page as the start page.

The User login name and status can also be displayed in the webpage. Drag the Login Status and Login Name control on the webpage to do so. The Change Password control can be used to change the existing user's password. A new can sign up if the Create user Wizard is added to the webpage.

Post Back Action

A Post back is an action taken by an interactive webpage, when the entire page and its contents are sent to the server for processing some information and then, the server post the same page back to the browser. This is done to verify user names and passwords, for example, when processing an on-line order form data, or other similar tasks that require server interaction.

Each Asp.Net page, when loaded, goes through a regular creation and destruction cycle like Initialization, Page load etc., in the beginning and unload while closing it. This Postback is a read only property with each Asp.Net Page (System.Web.UI.Page) class. This is false when the first time the page is loaded and is true when the page is submitted and processed. This enables users to write the code depending on if thePostBack is true or false (with the use of the function Page.IsPostBack).

Working on a Post Back Event

For some web pages it may be required to fill up the entities in a drop down list depending upon selection made by user. For instance, the user is filling up a form that asks for the country and city the user belongs. Instead of typing in the country, the user can simple select from the drop down list. As soon as the country is selected, the drop down list of the city becomes populated with the cities of that country. This action is taken in response of the user's action and is made possible by thePostBack event.

Add a textbox, a button and label to the webpage and write the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack == false)
    {
        TextBox1.Text = "";
    }
}
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = TextBox1.Text;
}
```

This code helps the user to enter some text in the textbox and when the button is clicked it copies that text to the label. If you want to make sure that the textbox is empty at the time the page is first time loaded then you write `TextBox1.Text = ""`; But this will empty the textbox as well as the label whenever the button is clicked. To empty the contents of the textbox for only the first time the condition `if (Page.IsPostBack == false)` is inserted. The `IsPostBack` is a property that is false only when the page is loaded for the first time.

• Adding a Calendar Control

Whenever it is required to take date as input from the user, a useful control called Calendar can be added from the standard toolbox. To illustrate its working, add a textbox and set its `ReadOnly` property to `True`. Also change its ID to `dateTextbox`. Double click on the control to open its coding and add the following code:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
```



```
{  
    dateTextBox.Text = Calendar1.SelectedDate.ToString("MM/dd/yyyy");  
}
```

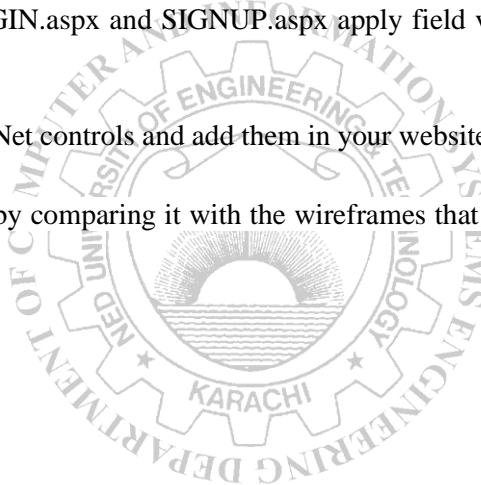
The date clicked by the user is Postback and is displayed in the textbox which is un-edit.

Debugging the Website

Press F5 or the play button in the standard toolbar to start debugging. A window displays a message that **Debugging Not Enabled**. Select the option *Add web.config file with debugging enabled* and click OK. The Visual Studio Web Developer 2008 launches a mini testing web server that will work for the purpose of testing the website just created locally. A popup on the taskbar shows ASP.Net Development Server <http://localhost:50701/First/>. The website will now open on the website explorer.

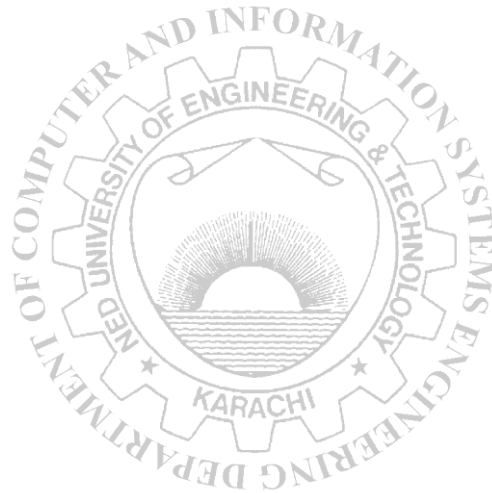
EXERCISES:

1. Prepare a brief Sprint Backlog and Planning document. Attach print out.
2. Design web pages LOGIN.aspx and SIGNUP.aspx apply field validation on it. Also attach print outs.
3. Explore different ASP.Net controls and add them in your website according to the SRS.
4. Test your master page by comparing it with the wireframes that you have created while 1st sprint planning.



Attach print outs here:

.



Lab Session 11

Demonstrate second sprint & plan third sprint of Web Development & Testing using Scrum

Sprint review

At the end of first sprint, before moving further every group member is required present all the tasks completed up till now. If any task still needs to be completed or requires improvement, sprint backlog document should be prepared and must be carried out in the second sprint.

Sprint planning

In the third sprint, we plan to complete the website by integrating required database tables. The look and feel of the website can also be improved. By the end of this sprint, along with the website, complete Test Cases document will also be presented by every group.

Working with Databases in ASP.Net

Microsoft provides a tool window in Visual Studio IDE called the “Server Explorer” which is a management console for any server (or system) and is used to open databases, manage and control the data of different databases, manage system services, and so on. Whenever a project is opened in ASP.Net the server explorer is also visible often tabbed with the Solution explorer.

ActiveX Data Objects

ADO.NET is a set of classes that expose data access services to the .NET programmer. ADO.NET provides functionality to developers writing managed code, consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. Its primary objects are:

DataSet objects are in-memory representations of data. They contain multiple DataTable objects, which contain columns and rows, just like normal database tables. You can even define relations between tables to create parent-child relationships. The DataSet is specifically designed to help manage data in memory and to support disconnected operations on data.

Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle.

Command objects are used to execute commands to a database across a data connection. They provide three methods that are used to execute SQL commands on the database:

- **ExecuteNonQuery:** Executes commands that have no return values such as INSERT, UPDATE or DELETE.
- **ExecuteScalar:** Returns a single value from a database query.
- **ExecuteReader:** Returns a result set by way of a DataReader object.

DataReader object provides a forward-only, read-only, connected stream record set from a database. It allows you to obtain the results of a SELECT statement from a command object.

DataAdapter object contains a reference to the connection object and opens and closes the connection automatically when reading from or writing to the database. Additionally, the data adapter contains command object references for SELECT, INSERT, UPDATE, and DELETE operations on the data.

Working with SQL database

- **Creating a Database**

Right click on the root directory on the Solution Explorer window and select Add New Item. From the pop-up window select the SQL Database and assign a name to the database (e.g. StudentDB.mdf). To add a new table click on Server Explorer then right click the Tables Option shown beneath the database filename and click Add New Table. Insert all the column names of the database along with their data types in the database. Also rename the table with an appropriate name.

- **Connecting to Database**

Drag a SQL Data source from the Data Toolbox into the webpage. Click on the arrow present on the upper right corner of the Data Source (known as *Smart tag*). Select Configure Data Source to make a new connection. A popup window will appear asking the user to choose the database from a drop down list. Select the desired database and click Next. After passing through a series of steps in which the connection string will be saved and also the required columns in the table are selected, the database connection is established.

- **Displaying the Data on the Webpage**

To view the data present in a table on the webpage, drag the table to be shown on to the design window or drag the GridView or DetailsView in the data Toolbox and click on the arrow on its upper right corner and in choose the data source option, select the desired SQLDataSource.

Working with SQL Database using ADO.NET

- **Creating a Database**

Right click on the root directory on the Solution Explorer window and select Add New Item. From the pop-up window select the SQL Database and assign a name to the database (e.g. ItemDB.mdf). To add a new table click on Server Explorer then right click the Tables Option shown beneath the database filename and click Add New Table. Insert all the column names of the database along with their data types in the database. Also rename the table with an appropriate name.

Consider an example to create the columns with name “ID” and “CategoryName” and select appropriate data types. Select the column ID as primary key. Also rename the table as “Categories”.

- **To display the database records**

The **SqlConnection** object holds the connection string that connects the database engine to the ItemDB.mdf database. To provide appropriate connection string to this connection object, drag a SQL Data source from the Data Toolbox into the webpage. Select Configure Data Source to make a new connection. A popup window will appear asking the user to choose the database from a drop down list. Select the desired database. Open the **ConnectionString** tab below to get the required string. This connection string is to be copied in the code.

The **SqlCommand.ExecuteReader()** method creates an **SqlDataReader** object to read these records. The **DataGrid** is connected to the data reader through its **DataGrid.DataSource** property.

When the **DataGrid.DataBind()** method executes, database records are moved from the database to the **DataGrid**, which displays them one record per row.

From the toolbox, drag a **DataGrid** to the Web form. Switch to code view by double-clicking the form. Add this line to the using statements at the beginning of **Default.aspx.cs**.

```
using System.Data.SqlClient;
Insert this code into the Page Load method:
private void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
        ReadData();
}
Add the ReadData method just after the Page Load method
public void ReadData()
{
    SqlDataReader rdr = null;
    SqlConnection conn = null;
    try
    {
        conn = new SqlConnection("Data Source=(local);Initial
        Catalog=Northwind; Integrated Security=SSPI");

        // Open the connection
        conn.Open();

        // 1. Instantiate a new command with a query and connection

        SqlCommand cmd = new SqlCommand("select CategoryName from Categories",
        conn);

        // 2. Call Execute reader to get query results

        rdr = cmd.ExecuteReader();
        Datagrid1.DataSource = rdr;
        Datagrid1.DataBind();

        // print the CategoryName of each record

        while (rdr.Read())
        {
```

```
        Console.WriteLine(rdr[0]);
    }
}
finally
{
    // close the reader
    if (rdr != null)
    {
        rdr.Close();
    }

    // Close the connection
    if (conn != null)
    {
        conn.Close();
    }
}
```

Press F5 to launch the Web application under the debugger. The contents of the database should appear in the browser.

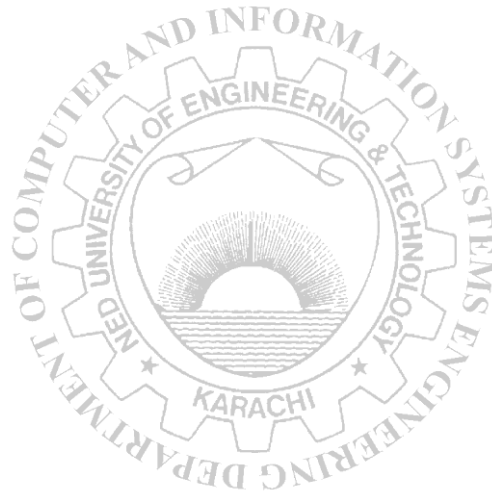
Exploring style sheets in ASP.NET

Professionally, whenever a webpage is designed is more often contain more than one webpage. The basic design of the webpage like the fonts used or the background color etc. should be consistent to give it a uniform professional look. For this purpose style sheets are used. CSS is a Style Sheet language that is used to describe the presentation of the *markup tags* present in HTML. The style sheet reduces the complexity in the code by providing all the similar styles at one place, thus reducing repetition and length of code.

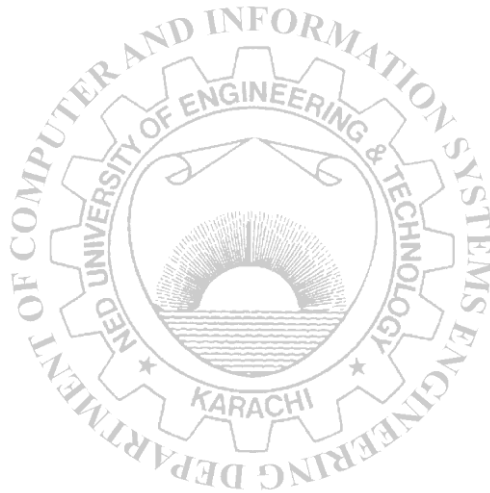
EXERCISE

1. Construct a complete sprint review and backlog document. Attach print out.
2. Create all the required database tables in the website and link them to the appropriate controls in the website.
3. Add the basic DML operations (Select, Update, Delete) to the webpages.
4. Compare all the functionalities of the website with the latest specification document.
5. Test all the functions of the website by providing variety of different inputs. Prepare all test cases. Attach the printout of Test Case Document.

Attach print outs here:



Attach print outs here:



Lab Session 12

Explore code repository tool for Version Controlling System (VCS)

Version control basics

A version control system is a system that tracks incremental versions (or revisions) of files and, in some cases, directories over time. Of course, merely tracking the various versions of a user's (or group of users') files and directories isn't very interesting in itself.

If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

Examples of VCS tools are: CVS, Subversion, Perforce, Git, Mercurial, Bazaar, Darcs and etc.

Centralized VS Distributed Version Controlling

Centralized version controlling systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control. This setup offers many advantages for example, everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what, and it's far easier to administer a CVCS than it is to deal with local databases on every client. However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.

Distributed Version Controlling – Git

The Repository At the core of the version control system is a repository, which is the central store of that system's data. The repository usually stores information in the form of a filesystem tree—a hierarchy of files and directories.

Git Workflow

There are four fundamental elements in the Git Workflow.

- Working Directory
- Staging Area
- Local Repository
- Remote Repository.

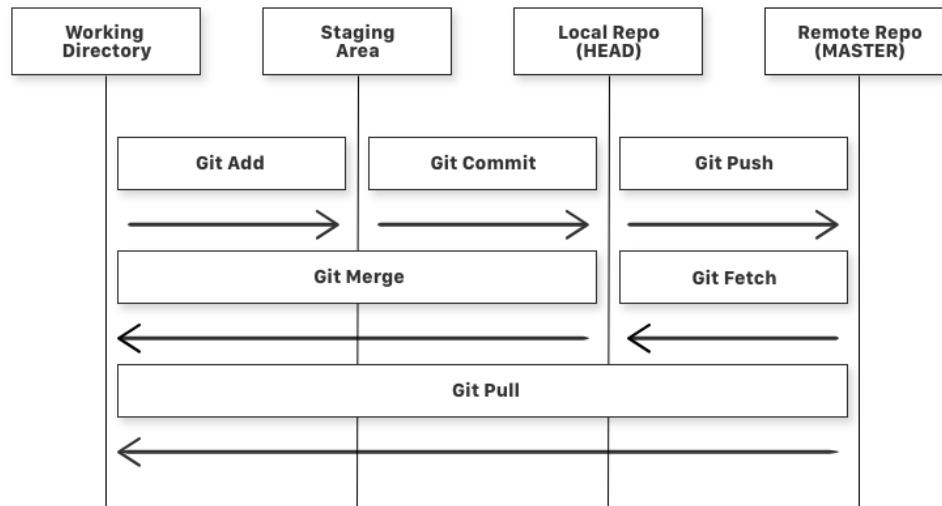


Fig 12.1 Git Workflow

If you consider a file in your Working Directory, it can be in three possible states.

1. **It can be staged** means the files with the updated changes are marked to be committed to the local repository but not yet committed.
2. **It can be modified** means the files with the updated changes are not yet stored in the local repository.
3. **It can be committed** means that the changes you made to your file are safely stored in the local repository.

Git Commands

- **git add** is a command used to add a file that is in the working directory to the staging area.
- **git commit** is a command used to add all files that are staged to the local repository.
- **git push** is a command used to add all committed files in the local repository to the remote repository. So in the remote repository, all files and changes will be visible to anyone with access to the remote repository.
- **git fetch** is a command used to get files from the remote repository to the local repository but not into the working directory.
- **git merge** is a command used to get the files from the local repository into the working directory.
- **git pull** is command used to get files from the remote repository directly into the working directory. It is equivalent to a **git fetch** and a **git merge**.

- **git status** shows the working tree status. It displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git.
- **git version** shows the current version of git.

Steps to place files under Git

1. Make a GitHub Account.
2. Install Git on your machine. Use either command prompt or Git Bash utility.
3. Once the setup is installed, mention your user name and email address in the configuration file since every Git commit will use this information to identify you as the author.

```
$ git config --global user.name "YOUR_USERNAME"
$ git config --global user.email "im_abc@example.com"
$ git config --global --list # To check the info you just provided
```

4. Create a folder at your machine's Desktop or any suitable place. Create a simple .txt file for example README.txt and initialize git via command prompt.

```
$ git init
```

5. Add files to the Staging Area for commit.

```
$ git add README.txt # To add a specific file
```

6. Before we commit let's see what files are staged:

```
$ git status # Lists all new or modified files to be committed
```

7. Commit Changes you made to your Git Repo. Now to commit files you added to your git repository.

```
$ git commit -m "First commit"
```

8. Add a remote origin and Push. Now each time you make changes in your files and save it, it won't be automatically updated on GitHub. All the changes we made in the file are updated in the local repository. Now to update the changes to the master:

```
$ git remote add origin remote_repository_URL # Sets the new remote
```

9. Now the **git push** command pushes the changes in your local repository up to the remote repository you specified as the origin.

```
$ git push -u origin master # Pushes changes to origin
```

Creating Branches in Git

While using GitHub, its better to keep the master branch clean, which mean without any changes. We can create at any time a branch from master.

Each time that you want to commit a bug or a feature, you need to create a branch for it, which will be a copy of your master branch.

Before creating a new branch, pull the changes from upstream. Your master needs to be up to date.

```
$ git pull
```

Create the branch from master on your local machine and switch in this branch:

```
$ git checkout -b [name_of_your_new_branch]
```

If you want to create a branch (say bug_fixes) from a remote branch say development, then the command would be:

```
$ git checkout -b bug_fixes origin/development
```

Push the branch on github:

```
$ git push origin [name_of_your_new_branch]
```

When you want to commit something in your branch, be sure to be in your branch. Add -u parameter to set upstream. You can see all branches created by using:

```
$ git branch -a
```

Which will show all the branches in your current repository. Let suppose name of your new branch is master_clean then the following message will be appeared in the :

```
* approval_messages
master
master_clean
```

Add a new remote for your branch, git provide [name_of_your_remote] as origin:

```
$ git remote add [name_of_your_remote] [name_of_your_new_branch]
```

Push changes from your commit into your branch:

```
$ git push [name_of_your_new_remote] [url]
```

Update your branch when the original branch from official repository has been updated:

```
$ git fetch [name_of_your_remote]
```

Then you need to apply to merge changes, if your branch is derived from develop you need to do:

```
$ git merge [name_of_your_remote]/develop
```

Delete a branch on your local filesystem:

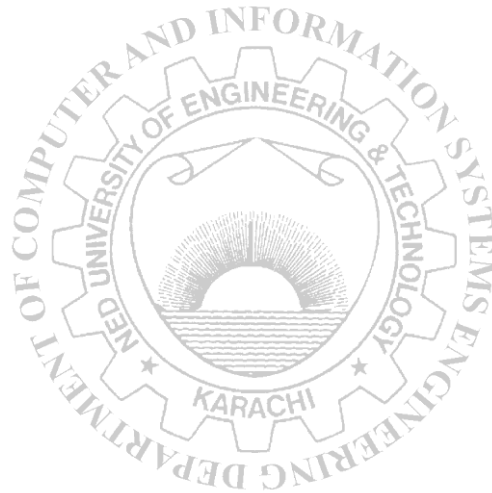
```
$ git branch -d [name_of_your_new_branch]
```

EXERCISES

1. Follow all the given steps to create a repository at GitHub and place your code files there. Use GitBash. Also attach the screenshots of each step.
2. Practice the following Git commands and explain their purposes. Also provide the screenshots of your results.

Commands	Functions
git diff	
git rm [file]	
git log	
git show [commit]	
git tag [commitID]	
git branch --merged	
git revert <name of bad commit>	
git checkout [branch name]	
git checkout -b [branch name]	

Attach print outs here:



Lab Session 13

Practice Conflict Resolution with multiple contributors in a VCS repository

Merge Conflicts

Merge conflicts occur when competing changes are made to the same line of a file, or when one person edits a file and another person deletes the same file.

1. Competing Line Change Merge Conflicts

To resolve a merge conflict caused by competing line changes, you must choose which changes to incorporate from the different branches in a new commit.

For example, if you and another person both edited the file *styleguide.md* on the same lines in different branches of the same Git repository, you'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

1. Open Git Bash.
2. Navigate into the local Git repository that has the merge conflict.
`cd REPOSITORY-NAME`
3. Generate a list of the files affected by the merge conflict. In this example, the file *styleguide.md* has a merge conflict.

```
$ git status
> # On branch branch-b
> # You have unmerged paths.
> #   (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> #   (use "git add ..." to mark resolution)
> #
> # both modified:      styleguide.md
> #
> no changes added to commit (use "git add" and/or "git commit -a")
```

4. Open your favorite text editor, such as Notepad++, and navigate to the file that has merge conflicts.
5. To see the beginning of the merge conflict in your file, search the file for the conflict marker <<<<<<<. When you open the file in your text editor, you'll see the changes from the HEAD or base branch after the line <<<<<<< HEAD. Next, you'll see =====, which divides your changes from the changes in the other branch, followed by >>>>>>> BRANCH-NAME. In this example, one person wrote "open an issue" in the base or HEAD branch and another person wrote "ask your question in IRC" in the compare branch or branch-a.

```
If you have questions, please
<<<<<<< HEAD
open an issue
=====
```

```
ask your question in IRC.
>>>>>> branch-a
```

- Decide if you want to keep only your branch's changes, keep only the other branch's changes, or make a brand new change, which may incorporate changes from both branches. Delete the conflict markers <<<<<<<, =====, >>>>>>> and make the changes you want in the final merge. In this example, both changes are incorporated into the final merge:

If you have questions, please open an issue or ask in our IRC channel if it's more urgent.

- Add or stage your changes.
\$ git add .
- Commit your changes with a comment.
\$ git commit -m "Resolved merge conflict by incorporating both suggestions."

We can now merge the branches on the command line or push changes to the remote repository on GitHub and merge our changes in a pull request.

2. Removed File Merge Conflicts

To resolve a merge conflict caused by competing changes to a file, where a person deletes a file in one branch and another person edits the same file, you must choose whether to delete or keep the removed file in a new commit.

For example, if you edited a file, such as *README.md*, and another person removed the same file in another branch in the same Git repository, you'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

- Open Git Bash.
- Navigate into the local Git repository that has the merge conflict.
cd *REPOSITORY-NAME*
- Generate a list of the files affected by the merge conflict. In this example, the file *README.md* has a merge conflict.
\$ git status
> # On branch master
> # Your branch and 'origin/master' have diverged,
> # and have 1 and 2 different commits each, respectively.
> # (use "git pull" to merge the remote branch into yours)
> # You have unmerged paths.
> # (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> # (use "git add/rm ..." as appropriate to mark resolution)
> #
> # deleted by us: README.md
> #
> # no changes added to commit (use "git add" and/or "git commit -a")

4. Open your favorite text editor, such as Notepad++, and navigate to the file that has merge conflicts.
5. Decide if you want keep the removed file. You may want to view the latest changes made to the removed file in your text editor. To add the removed file back to your repository:

```
$ git add README.md
```

To remove this file from your repository:

```
$ git rm README.md  
> README.md: needs merge  
> rm 'README.md'
```

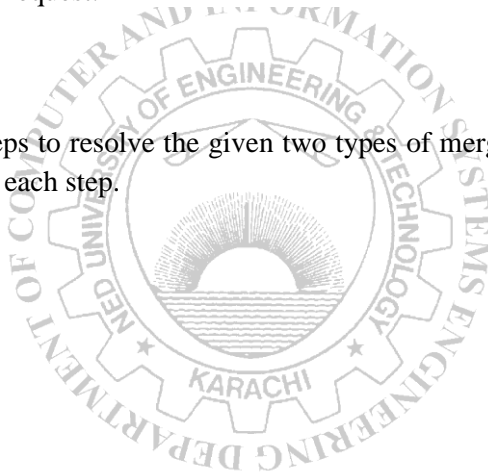
6. Commit your changes with a comment.

```
$ git commit -m "Resolved merge conflict by keeping README.md file."  
> [branch-d 6f89e49] Merge branch 'branch-c' into branch-d
```

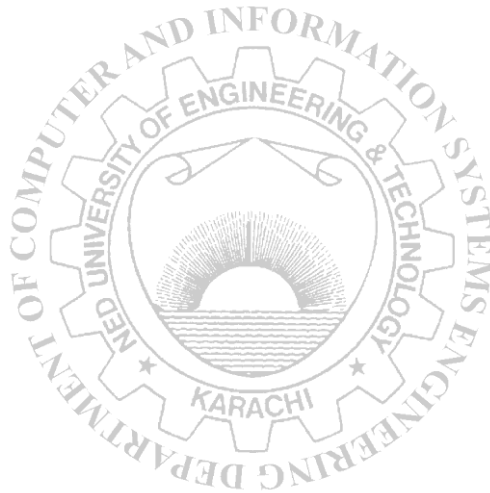
We can now merge the branches on the command line or push changes to the remote repository on GitHub and merge our changes in a pull request.

EXERCISE

1. Follow all the given steps to resolve the given two types of merge conflicts using GitBash. Also attach the screenshot of each step.



Attach print outs here:



Lab Session 14

Complex Engineering Activity

PROBLEM STATEMENT

--

COMPLEX PROBLEM SOLVING ATTRIBUTES COVERED

--

TASK DESCRIPTION

--

DELIVERABLES

GRADING RUBRIC

SOLUTION

Appendix A

LaTeX Formatting Features

Out of variety of very powerful and significant Latex, two are discussed over here:

1. Adding images

We will now look at how to add images to a LATEX document. Below is an example on how to include a picture.

```
\documentclass{article}
\usepackage{graphicx}
\graphicspath{ {images/} }

\begin{document}
The universe is immense and it seems to be homogeneous,
in a large scale, everywhere we look at.

\includegraphics{universe}

There's a picture of a galaxy above
\end{document}
```

LATEX cannot manage images by itself, so you will need to use a *package*. Packages can be used to change the default look of your LATEX document, or to allow more functionalities. In this case, you need to include an image in our document, so you should use the **graphicx** package. This package gives new commands, `\includegraphics{...}` and `\graphicspath{...}`. To use the **graphicx** package, include the following line in your preamble: `\usepackage{graphicx}`

The command `\graphicspath{ {images/} }` tells LATEX that the images are kept in a folder named *images* under the current directory.

The `\includegraphics{universe}` command is the one that actually included the image in the document. Here *universe* is the name of the file containing the image without the extension, then *universe.PNG* becomes *universe*. The file name of the image should not contain white spaces nor multiple dots.

- **Captions, Labels and References**

Images can be captioned, labelled and referenced by means of the **figure** environment as shown below:

```
\begin{figure}[h]
\centering
\includegraphics[width=0.25\textwidth]{mesh}
\caption{a nice plot}
\label{fig:mesh1}
\end{figure}

As you can see in the figure \ref{fig:mesh1}, the
function grows near 0. Also, in the page \pageref{fig:mesh1}
is the same example.
```

There are three important commands in the example:

- `\caption{a nice plot}`: As you may expect this command sets the caption for the figure. If you create a list of figures this caption will be used there. You can place it above or below the figure.
- `\label{fig:mesh1}`: If you need to refer the image within your document, set a label with this command. The label will number the image, and combined with the next command will allow you to reference it.
- `\ref{fig:mesh1}`: This code will be substituted by the number corresponding to the referenced figure.

When placing images in a LATEX document, we should always put them inside a **figure** environment or similar so that LATEX will position the image in a way that fits in with the rest of your text.

2. Adding Math to LATEX

One of the main advantages of LATEX is the ease at which mathematical expressions can be written. LATEX allows two writing modes for mathematical expressions: the **inline** mode and the **display** mode. The first one is used to write formulas that are part of a text. The second one is used to write expressions that are not part of a text or paragraph, and are therefore put on separate lines. Let's see an example of the **inline** mode:

In physics, the mass-energy equivalence is stated by the equation $E=mc^2$, discovered in 1905 by Albert Einstein.

To put your equations in *inline* mode use one of these delimiters: `\(... \)`, `$... $` or `\begin{math} ... \end{math}`. They all work and the choice is a matter of taste. The *displayed* mode has two versions: numbered and unnumbered.

The mass-energy equivalence is described by the famous equation

```
\[ E=mc^2 \]
```

discovered in 1905 by Albert Einstein.

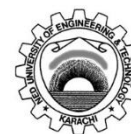
In natural units ($c = 1$), the formula expresses the identity

```
\begin{equation}
```

```
E=m
```

```
\end{equation}
```

NED University of Engineering & Technology
Department of Computer and Information Systems Engineering



Course Code and Title: CS-326 Software Engineering

Laboratory Session No. _____

Date: _____

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution exhibits redundancy and partially covers the problem.	The solution exhibits redundancy or partially covers the problem.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
Weighted CLO Score				
Remarks				
Instructor's Signature with Date				

NED University of Engineering & Technology
Department of Computer and Information Systems Engineering



Course Code and Title: CS-326 Software Engineering

Laboratory Session No. _____

Date: _____

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution exhibits redundancy and partially covers the problem.	The solution exhibits redundancy or partially covers the problem.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
Weighted CLO Score				
Remarks				
Instructor's Signature with Date				

NED University of Engineering & Technology
Department of Computer and Information Systems Engineering



Course Code and Title: CS-326 Software Engineering

Laboratory Session No. _____

Date: _____

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what level has the student understood the problem?	The student has not understood the problem at all.	The student understands the problem inadequately.	The student understands the problem adequately.	The student understands the problem comprehensively.
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed solution?	The solution does not address the problem adequately.	The solution exhibits redundancy and partially covers the problem.	The solution exhibits redundancy or partially covers the problem.	The solution is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the task?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
Weighted CLO Score				
Remarks				
Instructor's Signature with Date				