

DECCAN COLLEGE OF ENGINEERING & TECHNOLOGY

Dar-us-Salam, Hyderabad- 500001.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING



Software Engineering Lab

Subject Code :(PC551CS)

B.E. V Semester

LAB RECORD

Name: -----

Roll no: -----

DECCAN COLLEGE OF ENGINEERING AND TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SOFTWARE ENGINEERING LAB

CERTIFICATE

This is to certify that _____, bearing Roll No. _____ a student of B.E V semester(CBCS), C.S.E branch has successfully completed Software Engineering Lab in Deccan College of Engineering and Technology, Dar-us-Salam, Hyderabad during the Academic year 2023-2024.

Internal Examiner

External Examiner

SYLLABUS

Course Code	Course Title					Core/Elective	
PC531CS	SOFTWARE ENGINEERING LAB					CORE	
Prerequisite	Contact Hours Per Week				CIE	SEE	Credits
	L	T	D	P			
-	-	-	-	2	25	50	2
Course Objectives To understand the software engineering methodologies for project development. To gain knowledge about open source tools for Computer Aided Software Engineering (CASE). To develop test plans and test cases to perform various testing.							
Course Outcomes Student will be able to: Analyze and design software requirements in an efficient manner. Use open source case tools to develop software Implement the design, debug and test the code							

I. FORWARDENGINEERING

Students have to form a team with a batch size of two or three and take up a **case study based project** to analyze, plan, design UML models and create a prototypical model (identifying deliverables) by coding the developed designs and finally documenting considering anyone example of the following domains:-

1. Academics (Eg: Course Registration System, Student marks analyzing system)
2. HealthCare(Eg:Expertsystemtoprescribemedicinesforgivensymptoms,RemoteDiagnostics,Patien
t/HospitalManagementSystem)
3. Finance(Eg:Banking:ATM/NetBanking,UPI:PayTM/PhonePay,Stocks:Zerodha)
4. E-Commerce(Eg:variousonlineshoppingportalslikeFlipKart/Amazon/Myntra)
5. Logistics(Eg.Postal/Courier:IndiaPost/DTDC/UPS/FedEx,Freight:Maersk)
6. Hospitality (Eg: Tourism Management: Telangana Tourism/Incredible India, Event
Management: MeraEvents/BookMyShow/Explara/EventBrite)
7. Social Networking(Eg:LinkedIn, FaceBook, Shaadi.com ,BharatMatrimony, Tinder)
8. CustomerSupport(Eg.BankingOmbudsman,IndianConsumerComplaintsForum)
9. Booking/Ticketing(Eg.forFood:Zomato/Swiggy/BigBasket/Grofers/JioMart,Hotel:OYO/Trivago
orTravel:
{ Cars:Uber/OLA/Zoom,Railways:IRCTC,Buses:OnlineTSRTC/RedBus/AbhiBus,Flights:MakeMyTrip/G
oibibo,Ships:Lakport})

II. REVERSE ENGINEERING: Students have to refer any project repository: GitLab/GitHub, execute the code in order to observe its functionalities/ features/ requirements and by the help of any to olderive the designs from the code for understanding the relationships among various subsystems/classes/components and if the tool partially generates models the identify by associating elements to judge/ mark the appropriate relationships.

III. TESTING: Prepare Test Plan and develop Test Case Hierarchy to monitor or uncover/reporter or using manual/automated testing tools

SoftwareRequired:StarUML/Umbrello,NetBeans/EclipseIDE,XAMPP/MEANstack,JUnit,JMeter,Seleniu
m, Bugzilla

LIST OF EXPERIMENTS

Do the following exercises for any case study given in the list of sample projects or any other projects:

S.No	Program Names	Dates	PageNo	Signature
*	Introduction of SDLC			
1	Development of problem statement. & Modules (as in Abstract)			
2	Preparation of Software Requirement Specification Document			
3	Preparation of Software Configuration Management			
4	Performing the UML Designing by using any CASE tools (Star UML).			
5	Perform Forward Engineering of Design Diagram in Star UML			
6	Re-Write the Code and execute the project (in any Standard IDE)			
7	Perform Reverse Engineering of code in Star UML			
8	Develop test cases for various testing techniques.			
9	Perform Automated Testing Using Jmeter			
10	Creating Test cases in Selenium			
*	Deployment			

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

There are various development approaches defined and designed which are used/employed during development process of software these approaches are also referred as “ Software Development Process Models” (eg: Waterfall model, Incremental model, etc.). Each process model follows a particular life cycle in order to ensure success in process of software development. Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverable required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements.

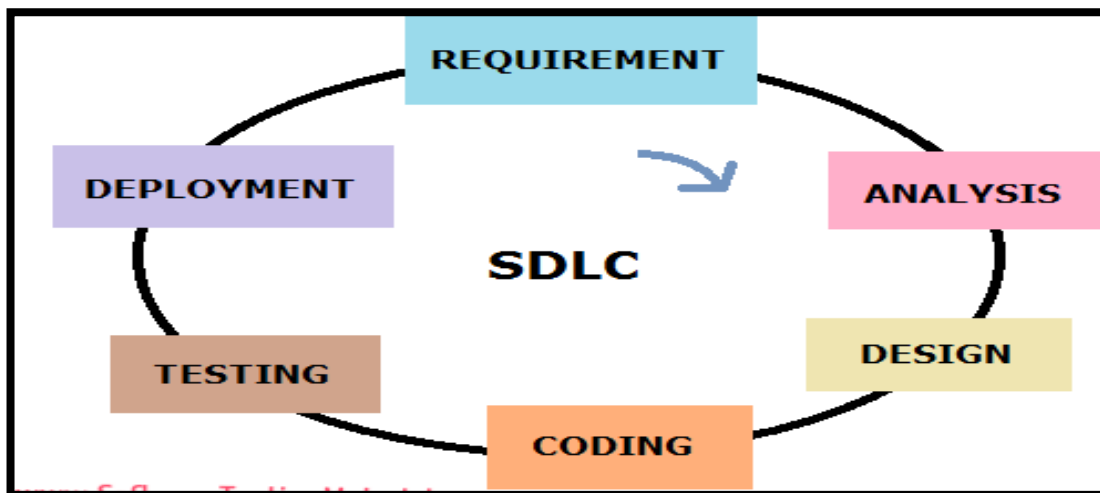
Life Cycle Phases

There are following six phases in every software development life cycle model:

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance

- Requirement gathering and analysis: Business requirements are gathered in this phase. This phase is the main focus of the project managers and stakeholders. Meetings with managers, stakeholders and users are held in order to determine the requirements like; who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase.
- After requirements gathering these requirements are analyzed for the their validity and the possibility of incorporating the requirements in the system to be developed is also studied. Finally, a requirements specification document is created which serves the purpose of guideline for the next phase of the model.

- **Design:** in this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.
- **Implementation/ Coding:** On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.
- **Testing:** After the code is developed it is tested again the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.
- **Deployment:** After successful testing the product is delivered/ deployed to the customer for their use.
- **Maintenance:** Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.



Software Development Life Cycle (SDLC)

Program 1

Development of problem statement & Modules:

-----Paragraph of Description of Application undertaken-----

This image shows a full page of white paper with horizontal blue dashed lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

Program2

Preparation of Software Requirement Specification Document

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Here is a template for an empty Software Requirements Specification (SRS) document. You can fill in the details based on your specific project requirements.

[Project Name]_____

Document Information

- Date: [Date]
- Authors: [List of Authors]
- Project Sponsor: [Name of Project Sponsor]
- Project Manager: [Name of Project Manager]

1. Introduction

1.1 Purpose

[Describe the purpose of the software, including its goals and objectives.]

1.2 Scope

[Define the scope of the project, outlining the features and functionalities that will be included.]

2. System Overview

2.1 System Description

[Provide a brief description of the overall system, its components, and how it fits into the existing environment.]

3. Functional Requirements

3.1 [Module Name]

3.1.1 Description

[Provide a brief description of the module.]

3.1.2 Features

[List the features and functionalities of the module.]

3.1.3 Use Cases

[Describe the various use cases related to the module.]

3.2 [Next Module Name]

[Repeat the structure for each module in the system.]

4. Non-Functional Requirements

4.1 Performance

[Specify performance-related requirements, such as response times, scalability, etc.]

4.2 Security

[Outline security requirements, including user authentication, data encryption, etc.]

5. User Interfaces

5.1 [Interface Name]

5.1.1 Description

[Describe the purpose and functionality of the interface.]

5.1.2 Screenshots

[Include screenshots or mockups if available.]

5.2 [Next Interface Name]

[Repeat the structure for each user interface in the system.]

6. Constraints

[Identify any constraints or limitations that might affect the development or implementation of the software.]

7. Glossary

[List key terms and their definitions to ensure clarity and understanding.]

Program3

Preparation of Software Configuration Management

Software Requirements

Operating System : _____

Front End : _____

Server side Script : _____

Database : _____

Hardware Requirements

Processor : _____

RAM

Hard Disk

SDLC TOOLS USED

Designing:

Implementation:

Testing

Program 4

Performing the UML Designing by using any CASE tools (Star UML).

UNIFIED MODELING LANGUAGE (UML)

1. Model

A model is a simplification of reality.

A model provides the blueprints of a system.

A model may be structural, emphasizing the organization of the system, or it may be behavioral, emphasizing the dynamics of the system.

We build models so that we can better understand the system we are developing.

We build models of complex systems because we cannot comprehend such a system in its entirety.

Through modeling, we achieve four aims.

Models help us to visualize a system as it is or as we want it to be. Models permit us to specify the structure or behavior of a system.

1. Static Diagrams

- a) Use case diagrams
- b) Class diagrams
- c) Object diagrams
- d) Component diagrams
- e) Deployment diagrams

2. Dynamic diagrams

- a) Interaction diagrams
 - i) Sequence diagrams
 - ii) Collaboration diagrams
 - b) State machine diagrams
 - c) Activity diagrams
- Applications of UML:

Models give us a template that guides us in constructing a system.

Models document the decisions we have made

2. Principles of Modeling

The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped

Every model may be expressed at different levels of precision The best models are connected to reality

No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models

UML is a graphical notation used to visualize, specify, construct and document the artifact of software intensive. UML is appropriate for modeling systems ranging from Enterprise Information Systems to Distributed Web-based Application and even to Hard Real-time Embedded systems. UML effectively starts with forming a conceptual modeling of the language.

There are 2 types of diagrams. They are

UML is intended primarily for software intensive systems. It has been used effectively for such domains as

1. Enterprise Information Systems
2. Banking and Financial Services
3. Telecommunications
4. Transportation
5. Defense and Aerospace
6. Retail
7. Medical Electronics
8. Scientific
9. Distributed Web-based Services

Basic building blocks of UML:

The building blocks of UML can be categorized as

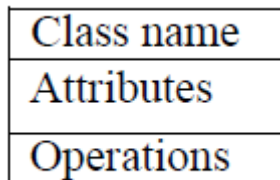
1. Things
2. Relationships and
3. Diagrams

Things:- Things are the most important building blocks of UML. Things can be

- a) Structural
 - b) Behavioral
 - c) Grouping
 - d) Annotational
- a) Structural Things: They define the static part of the model. They represent physical and conceptual elements.

Following are the structural things –

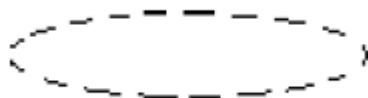
1. *Class*: - It describes a set of objects that share the same attributes, operations, relationships and semantics.



2. *Object*: - It is a collection of operations that specifies a service of a class or a component.



- 3 *Collaboration*: - It defines interaction between elements.

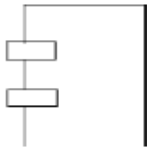


4. *Use case*: - They are used to identify different use case components of a particular software

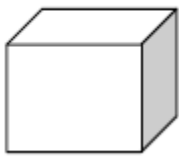
project. It is used to model the operation.



5. *Component*: - It is a physical and replaceable part that confirms to and provides realization of set of interfaces.



6. *Node*: - A physical resource that exists in runtime and represent a computational resource.



7. *Actor*: - The outside entity that communicates with a system. Typically a person playing a role on an external device.



b) Behavioral Things: They consist of dynamic parts of the UML model. The following are behavioral things –

1. *Interaction*: - It is defined as a behavior that consists of a group of message exchanged among

elements to accomplish a specific task. message

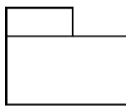


2. *State machine*: - It is useful when the states of an object in its life cycle. It defines the sequence of states and object goes through in response to events.



c) *Grouping Things*: They can be defined as a mechanism to group elements of UML model together. There is only one grouping thing available i.e., Package.

Package is used for gathering structural and behavioral things



d) *Annotational Things*: - They can be defined as a mechanism to capture remarks, description and comments of UML model elements. There is only one annotational thing available i.e., Note.

Note is used to render comments, constraints and so on of a UML element.

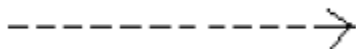


Relationships: -

The relationship is another most important building block of UML. They show how elements are associated with each other and their association describes the functionality of application.

There are 5 types of relationships. They are

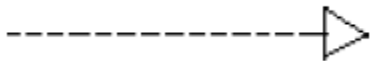
1. *Dependency*: It is a relationship between two things in which change in one element also affects another



2. Generalization: It can be defined as a relationship which connects a specialized element with a generalized element. It basically describes inheritance relationship in the object. It is a 'is_a' hierarchy.



3. Realization: It can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implement then. This relationship exists in case of interfaces.



4. Association: It is a set of links that connects elements of an UML model.



USE CASE DIAGRAM

A use case diagram describes a set of sequences in which each sequence indicates the relation with outside things. A use case involves the interaction of actor and system. There exist 3 types of relationships-

1. Association
2. Dependency
3. Generalization

Use case diagrams can contain

- □ Actors – “things” outside the system
- □ Use cases – system boundaries identifying what the system should do.

Use case diagram can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

Actor:- An actor represents system users. They help delimit the system requirements and give a clearer picture of what the system should do. An actor is someone or something that

- Interacts with or uses the system.
- Provides input to and receives information from the system.
- Is external to the system and has no control over the use cases.



Customer (actor)

Actors are discovered by examining,

- Who directly uses the system
- Who is responsible for maintaining the system?

- External hardware used by the system.
- Other systems that need to interact with the system.

Use case: - A use case can be described as a specific way of using the system from users (actors) perspective. Use case can be characterized as-

- A pattern of behavior the system exhibits.
- A sequence of related transactions performed by an actor and the system.
- Delivering something of value to the actor.



Transfer Funds

Note: Use cases often start with a “verb”.

Use cases provide a means to,

- Capture system requirements.
- Communicate with end users and domain experts.
- Test the system.

Every graphical representation has a textual description. The description of each use case is written in a use case specification. Use Case specification has :

Precondition – which states how and when the use case starts?

Main Flow – which lists the set of actions performed by the use case?

Alternate Flow – which lists the exceptions that are possible during executing the use case?

Post condition – which shows the result after the use case completes successfully.

Pseudo code

1. Right click on the model
2. Select Add Diagram – Use case diagram

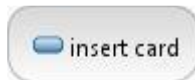
Use –Case Diagram for your Case Study:

ACTIVITYDIAGRAM

An activity diagram is a special case of state diagram. An activity diagram is like a flow Machine showing the flow a control from one activity to another. An activity diagram is used to model dynamic aspects of the system.

Activity diagram contains: 1.Activity states and action states 2.Transition

Action state:- These are atomic, executable computation which represents the execution of an action.



Activity state:- They can be decomposed. That is, their activity is represented by other activity diagrams.

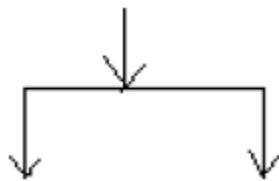
Branching:- In branching, we have one incoming transition and two or more outgoing transitions.

Decision

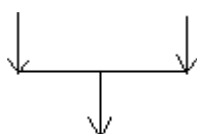


Decision

Forking:- It is a process of splitting a single flow of control into multiple flow of controls. Generally a fork has a single incoming flow of control but multi outgoing flow of control.



Joining:- It is exactly opposite of forking. It generally has multiple incoming flows of control but single outgoing flow of control.



Swim-lanes:- They represent the columns in the activity diagram to group the related activities. These are represented in the form of partitioned region. Swim-lanes are helpful when modeling a business work flow because they can represent organizational units or role with in a business model. Swim-lanes are very similar to an object because they provide a way to tell who is performing a certain role.



Pseudo code

1. Right click on the model
2. Select Add Diagram – Activity diagram

Activity Diagram of Your Case Study

CLASS DIAGRAM

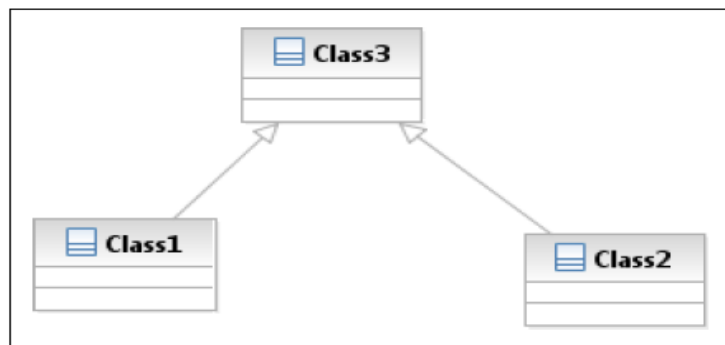
Class diagram contains icons representing classes, interfaces and their relationships.

Class:- A class is a set of objects that share a common structure and common behavior (the same attributes, operations and semantics). A class is a abstraction of real-world items. When these items exist in the real-world, they are instances of the class and are referred to as objects.

Class name
Attributes
Operations

Generalization relationship for classes: It shows that sub classes share the structure or behavior defined in one or more super classes. Use a generalize relationship to show “is_a” relationship.

Super class



Dependency Relationship: The dependency is a relationship between two model elements in which change in one element will affect the other model element. Typically in class diagrams, a dependency relationship indicates that the operations of the client invoke operation of the supplier.

Cardinality Adornment:- Cardinality specifies how many instances of one class may be associated with single instance of other class. When you apply a cardinality adornment to a class, you are indicating number of instances allowed for that class. A relationship, you are indicating number of links allowed between one instance of a class and the instances of another class.

<i>Valid Values:</i>	Value Description
0..0	Zero
0..1	zero or one
0..n	zero or more
1..1	One
1..n	one or more
N	unlimited number

Interface:- An interface specifies the externally visible operations of a class and/or component, and has no implementation of its own. An interface specifies only a limited part of behavior of class or a component.

Association relationship: An association provides a pathway of communications. The communication can be between use cases, actors, classes or interfaces. If two classes are usually considered independently, the relationship is an association.



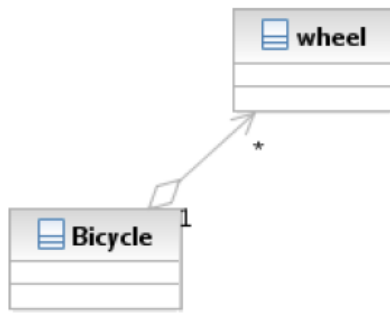
Unidirectional



Bi directional

An association is an orthogonal or straight solid line.

Aggregate relationship: Use the aggregate relationship to show a whole or part relationship between two classes.



The diamond end represents the class which is whole.

Pseudo code

1. Right click on the model
2. Select Add Diagram – Class diagram

Class Diagram of a of Your Case Study

OBJECT DIAGRAM

Object diagrams model the instances of things contained in the class diagrams. Object diagrams show a set of objects and their relationships at a point in time. They are used to model the static design view of a system.

Object diagrams contain:

- • Objects
- • Links

Pseudo code

1. Right click on the model
2. Select Add Diagram – Object diagram

Object Diagram of Your Case Study

INTERACTION DIAGRAM

An interaction is an important sequence of interactions between objects. There are two types of interaction diagrams,

1. Sequence Diagrams.
2. Collaboration diagrams.

1. **Sequence Diagram** : A sequence diagram is a graphical view of a scenario that shows object interaction in a time based sequence, what happens first, what happens next.

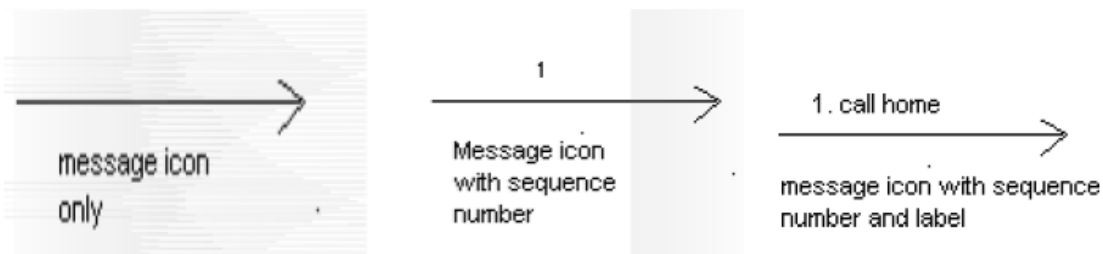
Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A sequence diagram has two dimensions: vertical placement represents time and horizontal placement represents different objects.

Link: Objects interact through their links to other objects. A link is an instance of an association, analogous to an object being instance of a class. A link should exist between two objects, including class utilities, only if there is a relationship between their corresponding classes.

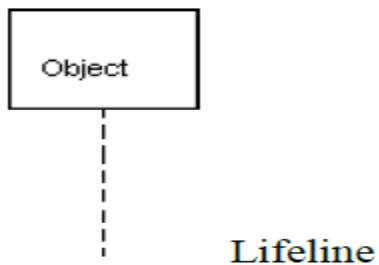


Message icons: A message icon represents the communication between objects indicating that an action will follow. The message icon is a horizontal, solid arrow connecting two lifelines together. A message icon in a sequence diagram represents exactly one message.



Lifeline: Each object appearing on the sequence diagram contains a dashed vertical line, called

lifeline, which represents the location of an object at a particular point in time. The lifeline also serves as a place for messages to start and stop and a place for the focus of control to reside.



Message or Event: a message is a communication carried between two objects that trigger an event. A message is represented in collaboration and sequence diagrams by a message icon which usually indicates its synchronization.

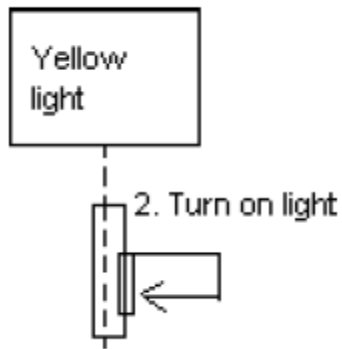
Synchronization types that are supported.

1. Synchronous Call
2. Asynchronous Call
3. Asynchronous Signal
4. Create
5. Delete
6. Reply

Message or Event: a message is a communication carried between two objects that trigger an event. A message is represented in collaboration and sequence diagrams by a message icon which usually indicates its synchronization.

Synchronization types that are supported.

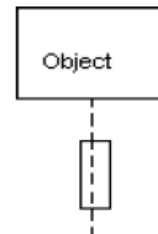
Message to self: It is a tool that sends a message from one object back to the same object. The sender of the message is same as the receiver.



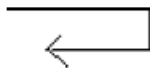
Tools:

1. Object

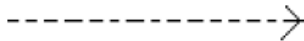
2. Object message



3. Message to self



4. Return message



5. Destruction marker



Pseudo code

1. Right click on the model
2. Select Add Diagram – Sequence diagram

Sequence Diagram of Your Case Study

Collaboration diagram : A collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object-like entities in the current model. Collaboration diagrams contain icons representing objects.

1. Right click on the model

2. Select Add Diagram – Communication diagram

Sequence and collaboration diagrams are semantically equivalent as both show the interaction among objects. From one diagram we can generate another diagram. To generate a collaboration diagram from sequence diagram, right click on sequence diagram, select - Add Diagram - communication diagram . Similarly, a sequence diagram can be generated from collaboration diagram.

Pseudo code

1. Right click on the model

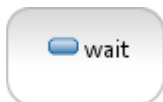
2. Select Add Diagram – Communication diagram

Collaboration Diagram of Your Case Study

STATE MACHINE DIAGRAM

State Machine diagrams model the dynamic behavior of individual classes or any other kind of object. They show the sequence of states that an object goes through the events that cause a transition from one state to another and the actions that result from a state change. A state Machine diagram is typically used to model the discrete stages of an objects lifetime. A state Machine diagram typically contains one start state and multiple end states.

State :- A state represents a condition or situation during the life of an object during which it satisfies some condition or waits for an event. Each state represents a cumulative history of its behavior. States can be shared between state machines. Transitions cannot be shared.



Naming: The name of the state must be unique to its enclosing class, within the state

Actions: Actions on states can occur at one of four times On entry

On exit Do On event

Start state:- A start state (also called an “initial state”) explicitly shows the beginning of the execution of the state machine on the state Machine diagram or beginning of the workflow on an activity diagram. Normally, one outgoing transition can be placed from the start state.

However, multiple transitions may be placed on start state, if at least one of them is labeled with a condition. No incoming transitions are allowed.

The start state icon is a small, filled circle that may contain the name (Begin process).

● Begin process

End state:- An end state represents a final or terminal state on an activity or state Machine diagram..

Transitions can only occur into an end state.

The end state icon is a filled circle inside a slightly larger unfilled circle that may contain the name (End process).

● End process

State transition:- A state transition indicates that an action in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when

certain conditions are satisfied. A state transition is a relationship between two states, two activities or between an activity or a state. The icon for a state transition is a line with an arrow head pointing toward the destination state or activity.

We can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

Naming: We should label each state transition with the name of at least one event that causes the state transition. We do not have to use unique labels for the state transitions because the same event can cause a transition to many different states or activities.

Tools:

1. State



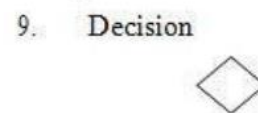
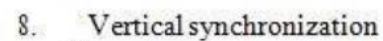
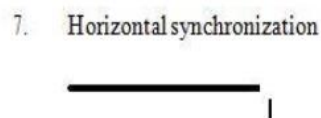
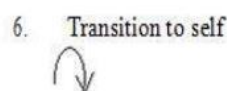
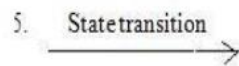
2 Activity



3. Start state



4. End state



Pseudo code

1. Right click on the model

1. Select Add Diagram – State Machine diagram

State-Machine Diagram of Your Case Study

COMPONENT DIAGRAM

Component diagrams provide a physical view of the current model. A component diagram shows the organizations and dependencies among software components, including source code component, binary code component, and executable component. These diagrams also show the externally visible behavior of the component by displaying the interfaces of the components.

Component diagrams contain:

- Component package
- Components
- Interfaces
- Dependency relationship

Pseudo code

1. Right click on the model
2. Select Add Diagram–Component diagram

Component Diagram of Your Case Study

DEPLOYMENT DIAGRAM

A deployment diagram shows processors, devices and connections. Each model contains a single deployment diagram which shows the connections between nodes.

Node:- A node is a hardware component capable of executing programs. Each node must have a name, there are no constraints on the node name because nodes denote hardware rather than software entities.

Pseudo code:

- Right click on the model
- Select Add Diagram– Deployment diagram

Deployment Diagram of Your Case Study

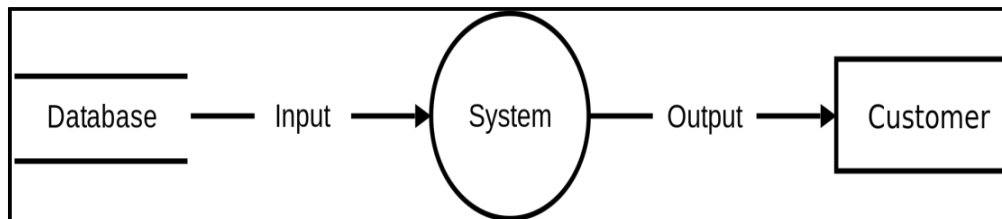
BUILDING AN E-R DIAGRAM:

- Level 1—Model All Data Objects (Entities) And Their “Connections” To One Another
- Level 2—Model All Entities And Relationships
- Level 3—Model All Entities, Relationships, And The Attributes That Provide Further Depth

E-R Diagram:

DATA FLOW DIAGRAMS

A **data flow diagram** (DFD) is a way of representing a flow of a data of a processor a system (usually an information_system) The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.



For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which sub divides this process into sub-processes.

The data-flow diagram is part of the structured-analysis modeling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan.

DFD consists of processes, flows, warehouses, and terminators. There are several ways to view these DFD components.^[4]

Process

The process (function, transformation) is part of a system that transforms inputs to outputs. The symbol of a process is a circle, an oval, a rectangle or a rectangle with rounded corners (according to the type of notation). The process is named in one word, a short sentence, or a phrase that is clearly to express its essence.^[2]

Data Flow

Data flow (flow, dataflow) shows the transfer of information (sometimes also material) from one part of the system to another. The symbol of the flow is the arrow. The flow should have a name that determines what information (or what material)is being moved. Exceptions are flows where it is clear what information is transferred through the entities that are linked to these flows. Material shifts are modeled in systems that are not merely informative. Flow should only

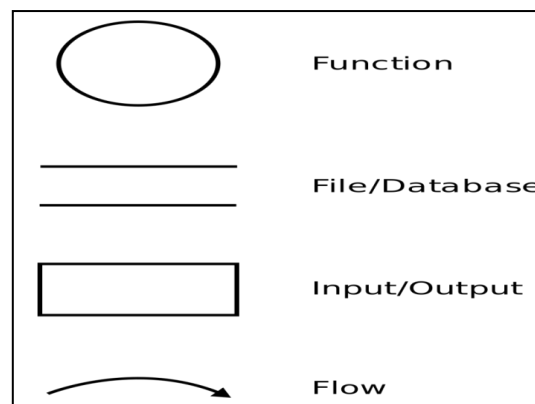
transmit one type of information (material). The arrow shows the flow direction (it can also be bi-directional if the information to/from the entity is logically dependent - e.g. question and answer). Flows link processes, warehouses and terminators.^[2]

Warehouse

The warehouse (data store, data store, file, database) is used to store data for later use. The symbol of the store is two horizontal lines, the other way of view is shown in the DFD Notation. The name of the warehouse is a plural noun (e.g. orders) - it derives from the input and output streams of the warehouse. The warehouse does not have to be just a data file, for example, a folder with documents, a filing cabinet, and optical discs. Therefore, viewing the warehouse in DFD is independent of implementation. The flow from the warehouse usually represents the reading of the data stored in the warehouse, and the flow to the warehouse usually expresses data entry or updating (sometimes also deleting data). Warehouse is represented by two parallel lines between which the memory name is located (It can be modeled as a UML buffer node).^[2]

Terminator

The Terminator is an external entity that communicates with the system and stands outside of the system. It can be, for example, various organizations (eg a bank), groups of people (e.g. customers), authorities (e.g. a tax office) or a department (e.g. a human-resources department) of the same organization, which does not belong to the model system. The terminator may be an other system with which the modeled system communicates



Sample Data Flow Diagram of Your Case Study

Program5

Perform Forward Engineering of Design Diagram in Star UML

Steps to use StarUml to construct forward engineering:

Step 1: Create a new Logial View through Rose.

Step 2: Build a class relationship diagram, add: package, class relationship, structure, comments, etc.

Step 3: Click the Tools drop-down menu in the upper menu bar—Extension Manager

Step 4: Find Java, click install, the same for other languages

Step 5: Click Tools—Java—Generate Code

Step 6: Select the model you want to perform forward and click OK, and then the Java file is generated Click OK to the model of the direction, and then a Java file is generate

Output:

Program6

Re-Write the Code and execute the project (in any Standard IDE)

Sample Code of your designed Application & output Screens:

Program 7

Perform Reverse Engineering of code in Star UML

Steps to build reverse engineering using StarUml:

step one: Create a new Logical View through StarUml.

Step two: Click on the Tools drop-down menu Java—Reverse Code in the upper menu bar and select the folder where the code is located.

Step three: The class in the code appears in the tree structure on the left

Step Four: Drag and drop the generated UML primitives to the workspace, as shown in the figure below

Step Five: Save the mdj file

Output:

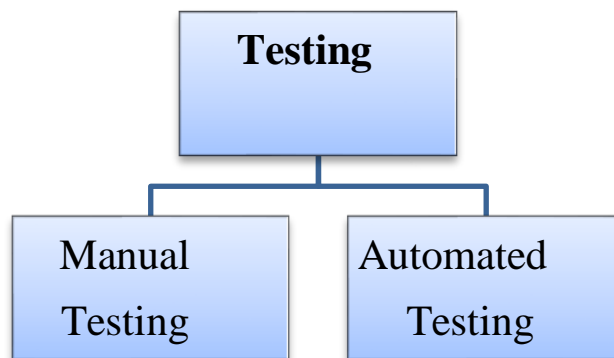
SOFTWARE TESTING

Testing Description

The purpose of testing is to assess product quality. It helps to strengthen and stabilize the architecture early in the development cycle. We can verify through testing, the various interactions, integration of components and the requirements which were implemented. It provides timely feedback to resolve the quality issues, in a timely and cost effective manner.

The test workflow involves the following:

- Verifying the interactions of components.
- Verifying the proper integration of components.
- Verifying that all requirements have been implemented correctly.
- Identifying and ensuring that all discovered defects are addressed before the software is deployed.



Manual Testing is a process of finding out the defects or bugs in a software program. In this method, the tester plays an important role of end-user and verifies that all the features of the application are working correctly. The tester manually executes test cases without using any automation tools. The tester prepares a test plan document which describes the detailed and systematic approach to testing of software applications. Test cases are planned to cover almost 100% of the software application. As manual testing involves complete test cases it is a time-consuming test.

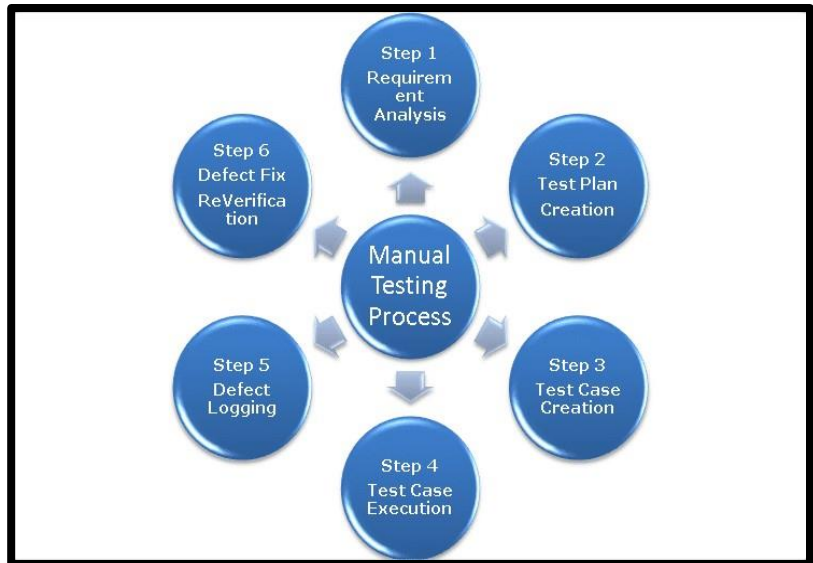
The differences between actual and desired results are treated as defects. The defects are then fixed by the developer of software application. The tester retests the defects to ensure that defects are fixed. The goal of Manual testing is to ensure that application is defect & error-free and is

working fine to provide good quality work to customers

Procedure of Manual

Testing

1. Requirement Analysis
2. Test Plan Creation
3. Test case Creation
4. Test case Execution
5. Defect Logging
6. Defect Fix & Re-Verification



Check Login Functionality there many possible test cases are:

- Test Case 1: Check results on entering valid User Id & Password
- Test Case 2: Check results on entering Invalid User ID & Password
- Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

This is nothing but a Test Case.

Program8

Develop test cases for various testing techniques

Writing Test Cases in Manual Testing

Program9

Perform Automated Testing Using JMeter:

TESTING A SAMPLE WEB PAGE OF MAILING APPLICATION USING JMETER:

JMeter is an open-source, Java-based functional and load testing tool which was initially developed for web but has since extended to support all major protocols, like FTP, SMTP, JMS, and JDBC.

The basics of JMeter and you can configure and run load tests using JMeter.

Objective

When running a prolonged load test, you might want to receive the test reports automatically via email upon test completion, or you might be running scheduled tests and want to get test reports by email. Whatever the case, our goal is to configure JMeter to send the generated test reports automatically by email.

There are two ways you can achieve this using JMeter. The first method doesn't require any external plugins but depends on JMeter's configuration to overcome the inherent problem (described below) while sending test reports via SMTP sampler.

The second method requires you to install JMeter Standard Plugins (via Plugins Manager or via external jars) but provides more flexibility in the reporting section.

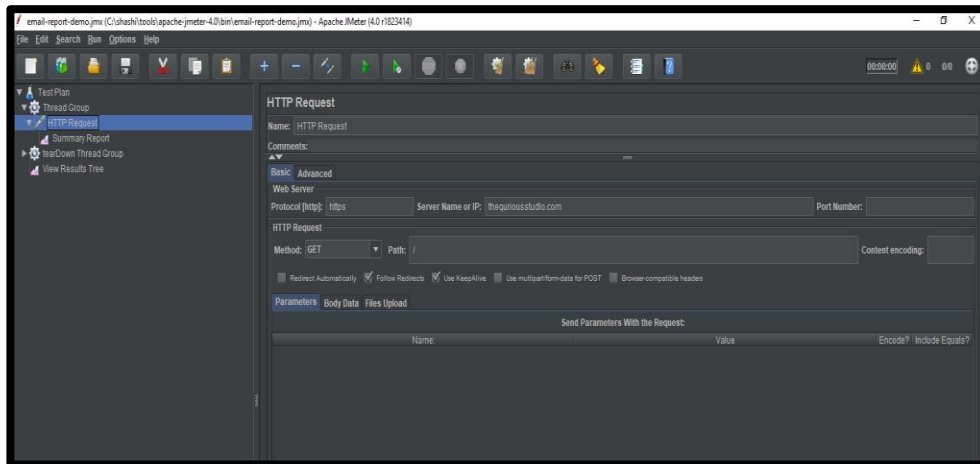
Prerequisites

The test has been performed with Apache JMeter v4.0, which requires JDK 8. The test script consists of a basic HTTP sampler with SMTP Sampler and "Summary Report" listener for the first method and "jp@gc Flexible File Writer" for the second method.

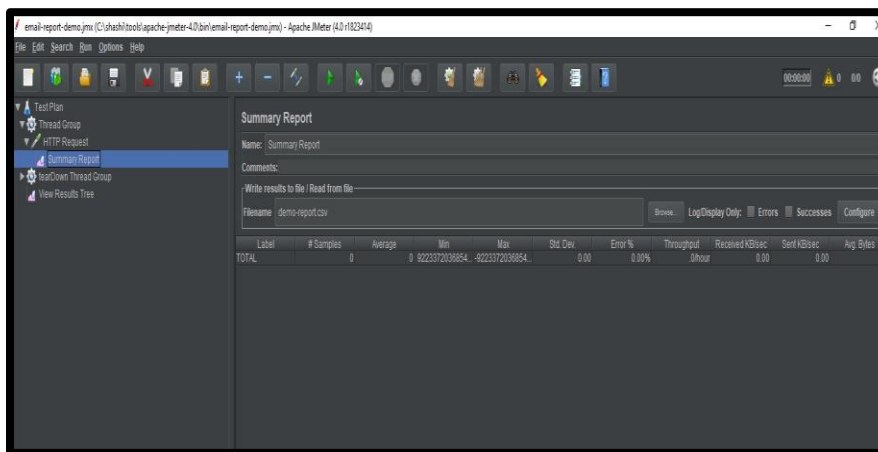
Using SMTP Sampler and Default "Summary Report" Listener Add a Thread Group to the Test Plan.

Add an HTTP Sampler and set up the required parameters, like Server Name, Method,

Path, etc.



Add the "Summary Report" listener to the Thread Group. Configure the File Name parameter to define the file where reports will be saved (the default file path is JMeter's bin directory). The File



Name parameter is important as it will be referenced by the SMTP sampler.

Add a tearDown Thread group to the Test Plan. The tearDown thread group will run only after the test has been execute, so it will run after the first Thread Group is completed.

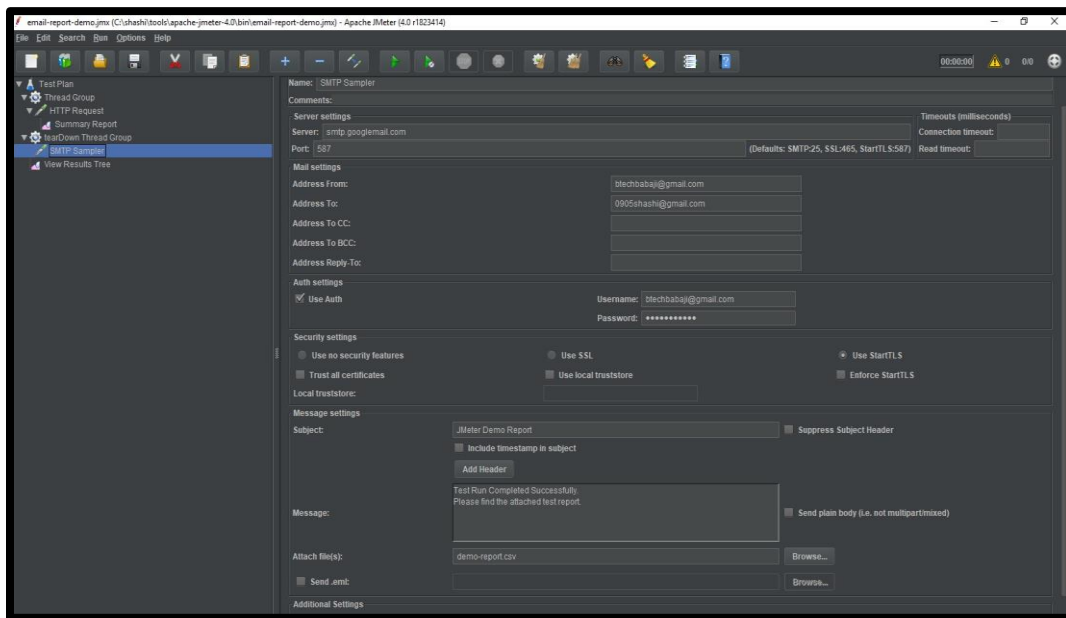
Add the SMTP Sampler to the Tear-Down Thread group and configure the mail server parameters. A demo configuration using Gmail. The SMTP server would use the following parameter values:

- **Server-** smtp.googlemail.com
- **Port-** 587

o**Address From** -
sender@gmail.com
o**Address To** -
recipient@gmail.com o**Auth**
Settings

✦ **username** - sender@gmail.com

✦ **password** - sender's Gmail password o **Security Settings** -



Use StartTLSo **Message Settings** -

- ✦ **Subject** - Email Subject/Title to be used
- ✦ **Message** - Email body goes here
- ✦ **Attach File(s)** - enter the filename used in the Summary Report Listener in Step 3

When using Gmail as an SMTP server, you might need to configure Gmail's settings to allow "LessSecured" apps to sign in; otherwise, Google may block JMeter from

sending any emails.

Now your test plan is almost complete, but if you run the test now, you will get an empty file in the mail without any data. This is due to the fact that the report file is written only after the test script has finished execution (which includes our `tearDown Thread Group`). When the SMTP Sampler is executed, it gets only an empty file without any data. The reason JMeter does this is to provide better performance — instead of executing write operations on a per-line basis, it is done all at once after test execution.

However, this property of JMeter can easily be controlled by modifying the "user.properties" (or "jmeter.properties") configuration file located in JMeter's bin directory.

Find and modify (or add, if not present) the following section in the file, setting the `autoflush` parameter to *true*:

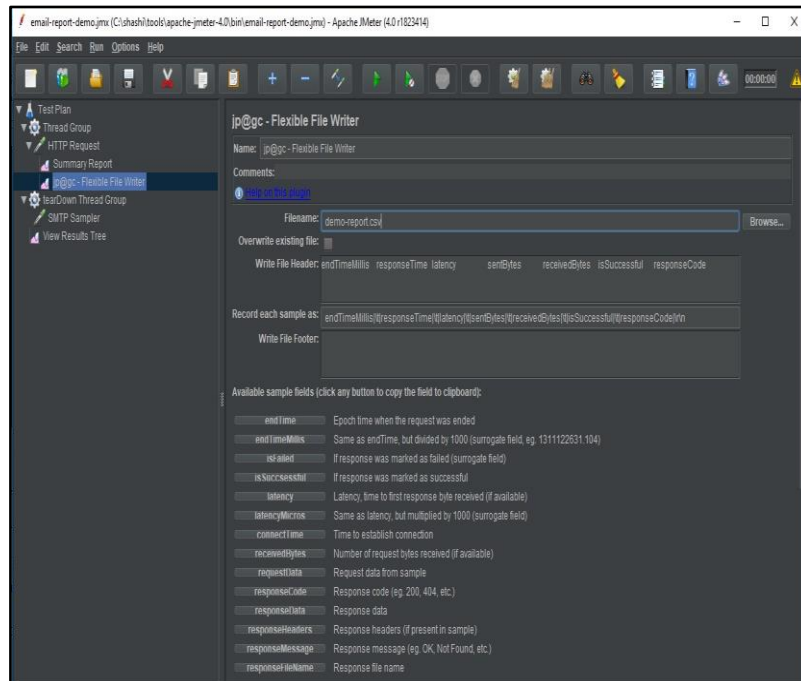
```
# AutoFlush on each line written in XML or CSV output  
  
# Setting this to true will result in less test results data loss in case of Crash  
# but with impact on performances, particularly for intensive tests (low or no pauses)  
# Since JMeter 2.10, this is false by default jmeter.save.saveservice.autoflush=true
```

Now, restart the JMeter and run the test again. You should be able to get the test report on mail with complete data.

Using SMTP Sampler With jp@gc Flexible File Writer

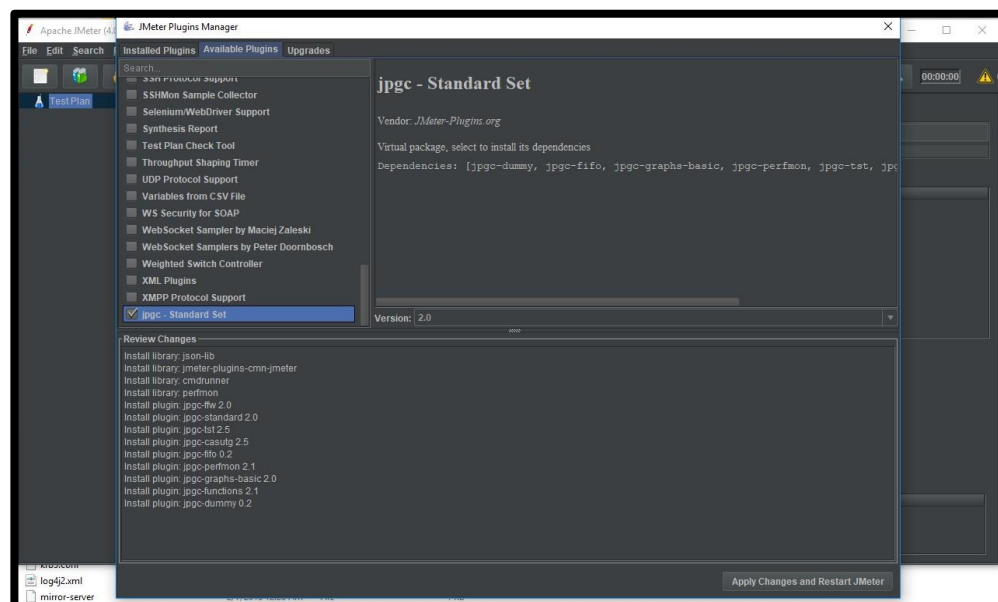
The `jp@gc` standard plugin set for JMeter can be used to extend its capabilities, including features like writing test results in flexible formats, enabling server monitoring, and recording performance metrics.

The standard plugin set can be installed via Plugins Manager or by directly downloading the jar and putting it into the `/lib/ext` directory. However, Plugins Manager provides easy-to-use plugin management capabilities via UI. To install Plugins Manager, download its



jar.

Once installed, you can access the Plugins Manager via File Menu in JMeter (under the "options" tab). To install the jp@gc Standard Plugin Set (which includes Flexible File Writer), go to the "Available Plugins" tab and mark the checkbox for "jp@gc Standard Set." Click the button in the bottom-right corner to "Apply Changes and Restart JMeter."



After restarting JMeter, you can find Flexible File Writer in the list of Listeners. Now, replace the

"Summary Report" listener in our test script with the "Flexible File Writer."

When you use Flexible File Writer, you don't have to modify the configuration file to enable Autoflush (done in Step 6) as it keeps writing the report file during test execution.

Notice the flexibility with the Flexible File Writer Listener. You can select the parameters to be included in the report by just selecting them from the list. You can also specify the format in which the data is written in the file.

You can read more about the usage and features of Flexible File Writer.

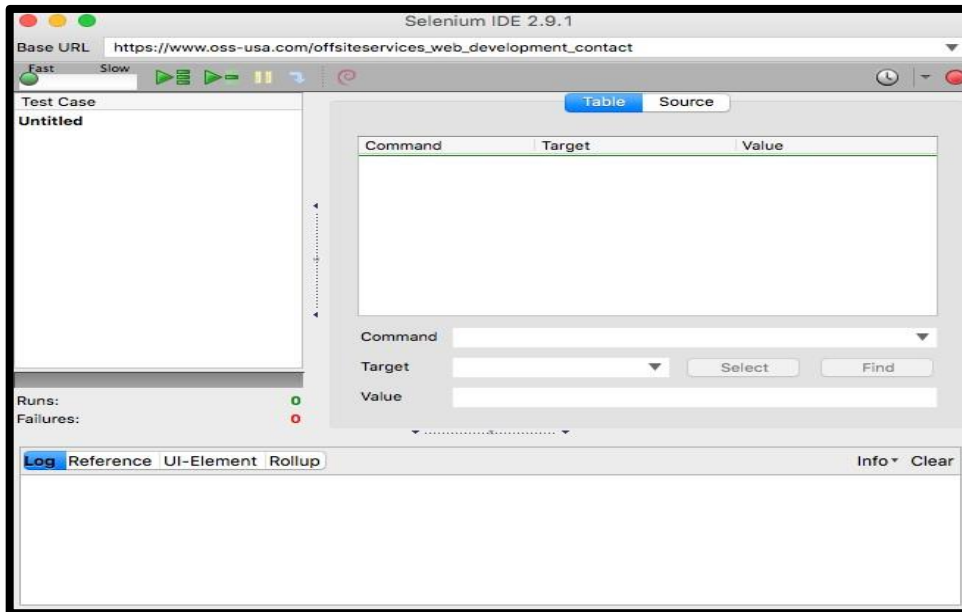
The choice between the two methods above depends solely on your test requirements. If you only need a basic summary report, you can go with the inbuilt Summary Report Listener.

In case you need more flexibilities or control over the reports and want to explore more new features, then try out the jp@gc set of plugins.

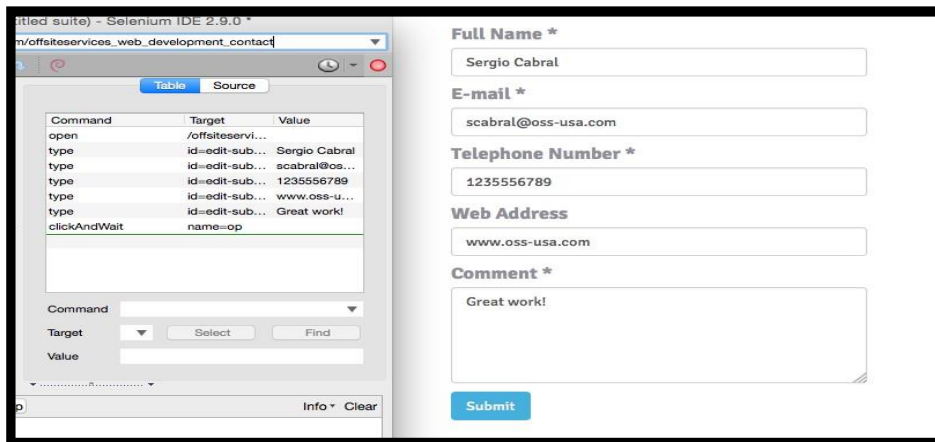
Program 10:

Creating A Test Case In Selenium

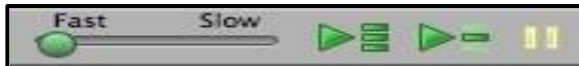
1. Start Selenium IDE from Firefox's Tool Bar.
2. Add "Base URL" to specify web page on which the test is going to be performed.
This URL can be changed to perform the same test case on a different URL.



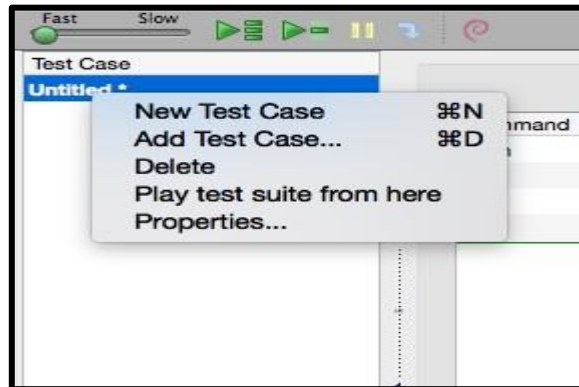
3. Once you've added the Base URL you can press the "Record" button (red round button top right) and start performing a manual test to the site (filling inputs, clicking links, etc.).
4. As you perform your manual test, Selenium will populate its commands table with all your interactions on the page.



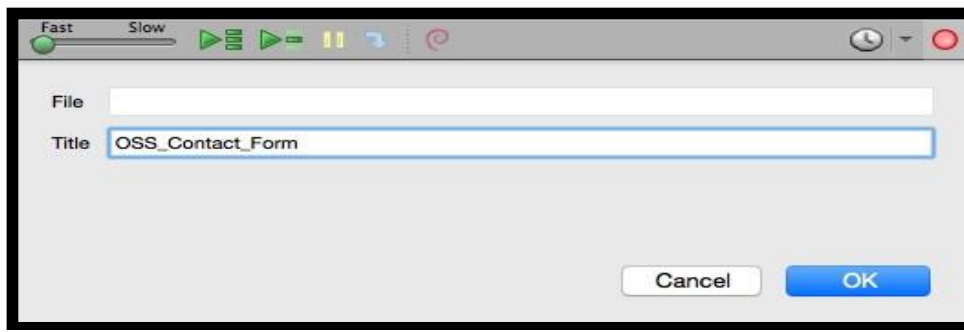
5. After you've finished recording your test, press the "Record" button again to stop the recording process.
6. Once your test is done recording, you can use the "Play" buttons to play either and entire Test Suite or current Test Case.



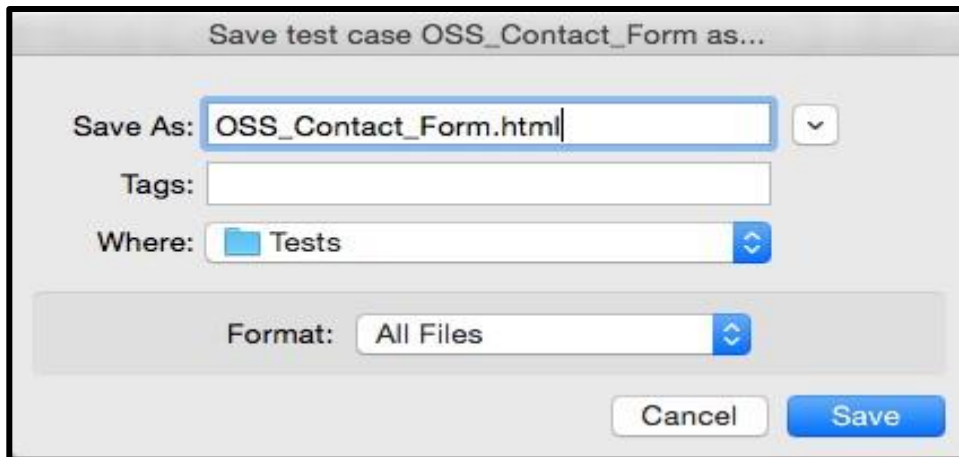
7. Right click on the test case's title, at the moment "Untitled *" and click on "Properties".



8. Here you can change the Test Case's Title to something more descriptive as you will probably have many test cases in a single Test Suite.



9. Once your test case has a title, you can proceed to save it. Click Firefox's menu File> Save Test Case As ... and add a name to the file. Test Cases could be saved as HTML files.



10. If you save a Test Case as an HTML file you can later double click on it and see the table on any browser.

OSS_Contact_Form		
open	/offsiteservices_web_development_contact	
type	id=edit-submitted-full-name	Sergio Cabral
type	id=edit-submitted-e-mail	scabral@oss-usa.com
type	id=edit-submitted-telephone-number	1235556789
type	id=edit-submitted-web-address	www.oss-usa.com
type	id=edit-submitted-comment	Great work!
clickAndWait	name=op	

Editing an existing Test Case it's simple:

1. Start Selenium IDE from Firefox's Tool Bar.
2. Click Firefox's menu File> Open ... and browse to the Test Case file that you want to work with.

Once loaded there are several things you can do to a Test Case:

- You can "Edit" a command
- You can "Add" new commands
- You can "Delete" commands **Editing a Command**

1. Click on a command from the Commands Table to select it.

2. Right below the commands table you'll see the selected command's information

Command	Target	Value
open	/offsiteservices_web_d...	
type	id=edit-submitted-full-...	Sergio Cabral
type	id=edit-submitted-e-mail	scabral@oss-usa.c...
type	id=edit-submitted-tele...	1235556789
type	id=edit-submitted-web...	www.oss-usa.com
type	id=edit-submitted-co...	Great work!
clickAndWait	name=op	

Command	type	
Target	id=edit-submit	Select Find
Value	1235556789	

3. Here you can change the command executed selecting from the drop down menu.
You CAN Find an Entire list of Selenium's commands in the [Selenium Reference](#) ("broken link").

type	id=edit-submitted-co...	Great work!
clickAndWait	name=op	

Command	type
Target	
Value	

type

addLocationStrategy
addLocationStrategyAndWait
addScript
addScriptAndWait
addSelection
addSelectionAndWait
allowNativeXpath
allowNativeXpathAndWait
altKeyDown
altKeyDownAndWait
altKeyUp
altKeyUpAndWait
answerOnNextPrompt
assertAlert

4. Once a command is selected you can see a short reference at the bottom of the app, where you'll find the command's arguments and description.

Log Reference UI-Element Rollup

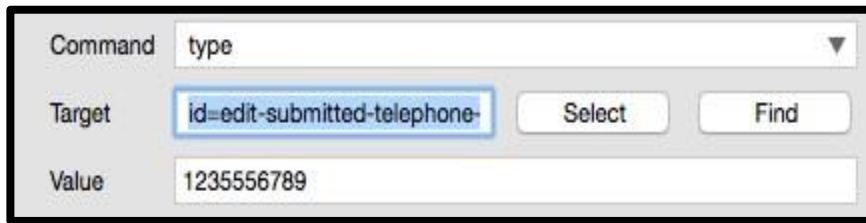
type(locator, value)
Arguments:

- locator - an element locator
- value - the value to type

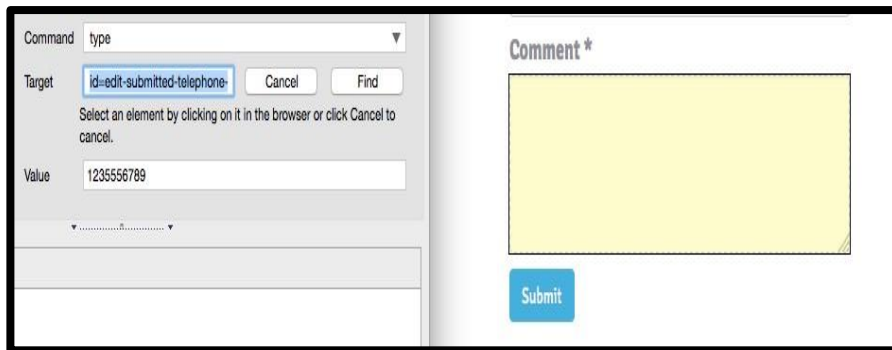
Sets the value of an input field, as though you typed it in.

Can also be used to set the value of combo boxes, check boxes, etc. In these cases, value should be the value of the option selected, not the visible text.

5. You can also edit the target of the command by changing the target's id.



6. If you do not know the target's id you can click on "Select" and then click on the website's element you want to apply the command to, this will get you its id.



7. And finally you can change the value used with the selected command by changing the "Value" field.
8. After you've finished editing commands you can proceed to save your test case as previously explained.

Adding a new command

1. Right click anywhere on the command's table and select the "Insert New Command" option.
2. An empty new command will be added to the table.
3. Edit it as previously explained.
4. After edited, drag and drop the command to its required position.
5. After you've finished adding commands you can proceed to save your test case as previously explained.

Deleting command

1. Go to your commands table and right click on the command you want to eliminate.
2. Select "Delete" from the menu.
3. After you've finished deleting commands you can proceed to save your test case as previously explained.

Conclusion of SDLC:

Software is developed following SDLC phases with Forward & Reverse Engineering, it can be implemented to fulfill all the client requirements. The system interface is very user friendly, and the overall system has been successfully tested. It has a broad future scope as new features can be incorporated in the present proposed system.

This system is designed using Unified Modeling Language concepts. The interfaces designed for the system is very user friendly and attractive. It has successfully implemented as per the client requirement.

The system has successfully passed the both testing at the development site and is under the testing phase in the presence of the client.
