



**Datta Meghe College of Engineering**  
**Airoli, Navi Mumbai**

**DEPARTMENT OF COMPUTER ENGINEERING**  
**ACADEMIC YEAR: 2023 – 24 (TERM – I)**

**List of Experiments**

**Course Name : SOFTWARE ENGINEERING LAB**

**Course Code : CSL501**

Sr. No	Name of experiment	CO's Covered	Page No.	Date of Performance	Date of Submission	Marks & Signature
01	Application of at least two traditional process models	CSL501.1				
02	Application of the Agile process models.	CSL501.1				
03	Preparation of software requirement specification (SRS) document in IEEE format.	CSL501.2				
04	Structured data flow analysis.	CSL501.2				
05	Use of metrics to estimate the cost.	CSL501.3				
06	Scheduling & tracking of the project.	CSL501.3				
07	Write test cases for black box testing.	CSL501.5				
08	Write test cases for white box testing.	CSL501.5				
09	Preparation of Risk Mitigation, Monitoring and Management Plan (RMMM).	CSL501.6				
10	Version controlling of the project.	CSL501.6				
11	Assignment-1	CSL501.1, CSL501.2, CSL501.3				
12	Assignment-2	CSL501.4, CSL501.5, CSL501.6				
13	Mini Project	CSL501.1				

This is to certify that Mr. / Miss \_\_\_\_\_ of \_\_\_\_\_ Roll No. \_\_\_\_\_ has performed the Experiments / Assignments / Tutorials / Case Study Work mentioned above in the premises of the institution.

Prof. Jayant Sawarkar  
**Practical Incharge**



**DATTA MEGHE COLLEGE OF ENGINEERING, AIROLI, NAVI  
MUMBAI  
DEPARTMENT OF COMPUTER ENGINEERING**

**Institute Vision** : To create value - based technocrats to fit in the world of work and research

**Institute Mission** :

- To adopt the best engineering practices
- To empower students to work in the world of technology and research
- To create competent human beings

**Department Vision** : To provide an intellectually stimulating environment for education, technological excellence in computer engineering field and professional training along with human values.

**Department Mission** :

**M1:** To promote an educational environment that combines academics with intellectual curiosity.

**M2:** To develop human resource with sound knowledge of theory and practical in the discipline of Computer Engineering and the ability to apply the knowledge to the benefit of society at large.

**M3:** To assimilate creative research and new technologies in order to facilitate students to be a lifelong learner who will contribute positively to the economic well-being of the nation.

**Program Educational Objectives (PEO)**

**PEO1:** To explicate optimal solutions through application of innovative computer science techniques that aid towards betterment of society.

**PEO2:** To adapt recent emerging technologies for enhancing their career opportunity prospects.

**PEO3:** To effectively communicate and collaborate as a member or leader in a team to manage multidisciplinary projects.

**PEO4:** To prepare graduates to involve in research, higher studies or to become entrepreneurs in long run.

**Program Specific Outcomes (PSO)**

**PSO1:** To apply basic and advanced computational and logical skills to provide solutions to computer engineering problems.

**PSO2:** Ability to apply standard practices and strategies in design and development of software and hardware-based systems and adapt to evolutionary changes in computing to meet the challenges of the future.

## **Program Outcomes as defined by NBA (PO)**

### **Engineering Graduates will be able to:**

- 1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- 2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- 3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- 4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- 5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- 6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- 7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- 8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- 9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- 11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- 12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**DATTA MEGHE COLLEGE OF ENGINEERING, AIROLI**

**DEPARTMENT OF COMPUTER ENGINEERING**

**COURSE NAME: SOFTWARE ENGINEERING LAB**

**COURSE CODE: CSL501**

**YEAR OF STUDY: T.E., SEMESTER: V**

**COURSE OUTCOMES**

<b>CSC502.1</b>	Identify requirements & assess the process models.
<b>CSC502.2</b>	Able to analyze and specify software requirements from various Stakeholders in a specific format.
<b>CSC502.3</b>	Plan, schedule and track the progress of the projects.
<b>CSC502.4</b>	Design the software projects.
<b>CSC502.5</b>	Do testing of software project.
<b>CSC502.6</b>	Identify risks; manage the change to assure quality in software projects.

**DATTA MEGHE COLLEGE OF ENGINEERING**  
**DEPARTMENT OF COMPUTER ENGINEERING**  
**ACADEMIC YEAR 2023-24 (TERM I)**  
**SUBJECT: SOFTWARE ENGINEERING LAB (CSL501)**  
**SEM: V      DIV : A & B**  
**RUBRICS FOR GRADING EXPERIMENTS**

<b>Rubric Number</b>	<b>Rubric Title</b>	<b>Criteria</b>	<b>Marks* (out of 15)</b>
<b>R1</b>	<b>Timeline</b>	On-time	<b>5</b>
		Delayed by not more than a Week	<b>3</b>
		Delayed more than a Week	<b>1</b>
<b>R2</b>	<b>Neatness</b>	Documented, Clean neat & properly maintain.	<b>5</b>
		Documented properly maintain	<b>3</b>
		Documented not properly maintain	<b>1</b>
<b>R3</b>	<b>Knowledge &amp; Implementation</b>	Correct implementation with Results and oral	<b>5</b>
		Implementation with some errors, oral	<b>3</b>
		Partial implementation, oral	<b>1</b>

**\*means obtained marks will be scaled to 15 for Experiments**

**DATTA MEGHE COLLEGE OF ENGINEERING**  
**DEPARTMENT OF COMPUTER ENGINEERING**  
**ACADEMIC YEAR 2023-24 (TERM I)**  
**SUBJECT: SOFTWARE ENGINEERING LAB (CSL501)**  
**SEM: V      DIV: A&B**  
**RUBRICS FOR GRADING ASSIGNMENTS**

<b>Rubric Number</b>	<b>Rubric Title</b>	<b>Criteria</b>	<b>Marks* (out of 5)</b>
<b>R1</b>	<b>Documentation</b>	<b>On-time</b>	<b>2</b>
		<b>Delayed by not more than a Week</b>	<b>1</b>
		<b>Delayed more than a Week</b>	<b>0</b>
<b>R2</b>	<b>Knowledge &amp; Concept</b>	<b>Clear understanding</b>	<b>2</b>
		<b>Partially understood</b>	<b>1</b>
		<b>Weak understanding</b>	<b>0</b>
<b>R3</b>	<b>Punctuality, Completion Time / Timeline</b>	<b>Correct Documentation</b>	<b>1</b>
		<b>Not documented properly</b>	<b>0</b>

**\*means obtained marks will be scaled to 5 for Assignments**

## **EXPERIMENT NO.:01**

**Date of Performance:**

**Date of Submission:**

**Aim:** Application of at least two traditional process models.

**Software Used:** Ms Word

**Theory:** Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

### **Types of Software Process Model**

Software processes, methodologies and frameworks range from specific prescriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a “sponsor” or “maintenance” organization distributes an official set of documents that describe the process.

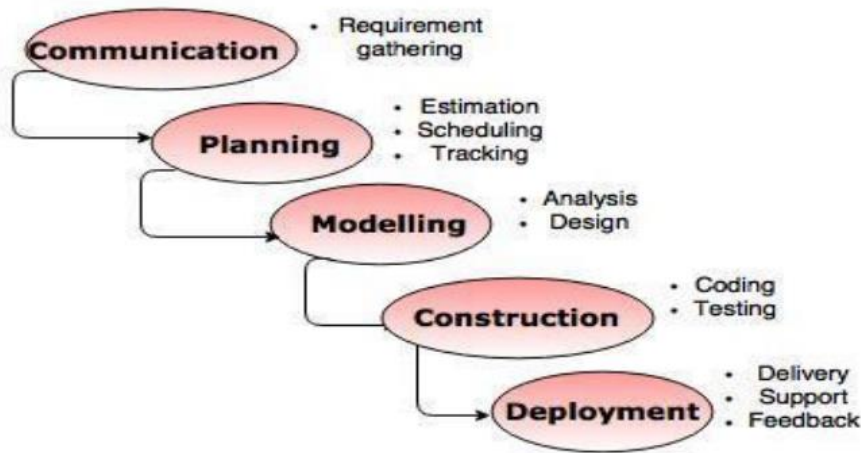
### **Software Process and Software Development Lifecycle Model**

One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Spiral model
- Prototype model

1. Topic Name: Multi-language translation tool

Model Used: Waterfall Model:



**Fig. - The Waterfall model**

Waterfall model is a sequential software development process that follows a linear and rigid progression through several distinct phases.

### **Application of the Waterfall model on the multi-language translation tool**

#### **1. Requirements Analysis**

- **Gather Requirements:** Identify and document the requirements for the translation tool. This includes understanding the languages to be supported, the types of text or documents to be translated, and any specific features needed (e.g., context-aware translation, user interface requirements).
- **User Needs:** Consult with potential users to gather their needs and expectations. This helps in defining functional requirements (e.g., accuracy, language support) and non-functional requirements (e.g., performance, security).

#### **2. System Design**

- **Architectural Design:** Create a high-level design of the system architecture. This might include decisions on whether to use rule-based, statistical, or neural network-based translation methods, and how these methods will be integrated.
- **Detailed Design:** Develop detailed designs for each component of the translation tool. This includes user interfaces, translation algorithms, databases for language pairs, and integration points.



### 3. Implementation

- **Development:** Write code based on the detailed design specifications. This involves programming the translation algorithms, creating user interfaces, and setting up the necessary databases and backend services.
- **Integration:** Ensure that all components of the tool work together as expected. This includes integrating translation engines with the user interface and backend systems.

### 4. Verification

- **Testing:** Conduct thorough testing to ensure that the translation tool meets the specified requirements. This includes:
  - **Unit Testing:** Test individual components for functionality.
  - **Integration Testing:** Verify that integrated components work together correctly.
  - **System Testing:** Ensure the entire system functions as intended, including usability and performance testing.
  - **Acceptance Testing:** Validate that the tool meets user needs and requirements.

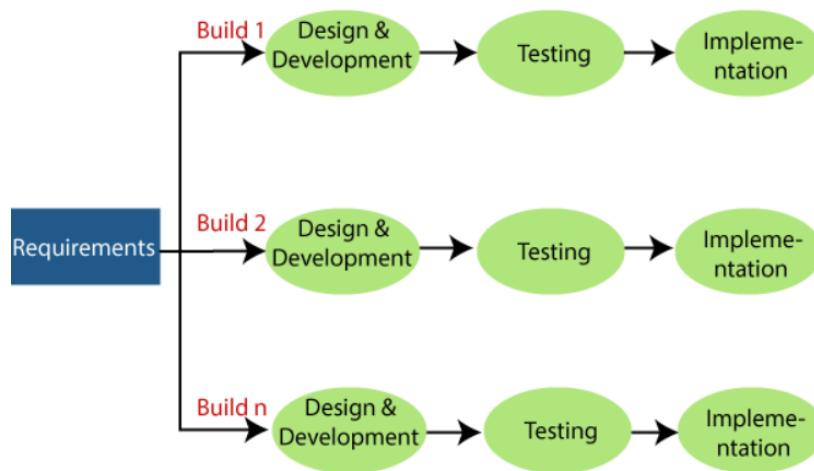
### 5. Deployment

- **Release:** Deploy the translation tool to the production environment. This involves setting up the necessary infrastructure and ensuring that the tool is accessible to users.
- **Documentation:** Provide user manuals, technical documentation, and support materials to assist users and administrators.

The waterfall model's sequential nature makes it ideal for projects with well-defined requirements and a clear path to development. However, be aware that this model is less flexible when it comes to accommodating changes during development. If your translation tool project requires frequent updates or iterative improvements, consider whether a more iterative model like Agile might be better suited.

## 2. Topic Name: Multi-language translation tool

Model Used: Waterfall Model:



An incremental model, in the context of software development or project management, refers to an iterative approach where development is divided into smaller, manageable parts or increments.

### Key Phases of the Incremental Model

#### 1. Initial Planning and Requirements Analysis

- **High-Level Requirements:** Start with gathering and defining high-level requirements for the entire system. These requirements are not detailed but provide an overview of what the system should achieve.
- **Prioritization:** Determine which features or components are the most critical and can be implemented first.

#### 2. Incremental Development

- **Design and Implementation of Each Increment:**
  - **Design:** For each increment, create a detailed design that includes specifications and architecture. Each increment focuses on a specific subset of the system's overall functionality.
  - **Development:** Implement the design of the increment. This could involve coding, integrating with existing increments, and ensuring that each increment functions correctly.
- **Testing:** Each increment is tested independently to ensure it meets the requirements and integrates well with the previously developed increments.

### 3. Integration and Testing

- **System Integration:** After the implementation of each increment, integrate it with the previously developed increments. This ensures that the new functionality works with the existing system.
- **Testing:** Conduct integration testing to verify that all increments work together as expected. Additionally, perform regression testing to ensure that new increments do not adversely affect existing functionality.

### 4. User Feedback and Refinement

- **User Review:** Involve users early and often by providing them with the new functionality as it becomes available. Collect feedback on each increment.
- **Refinement:** Based on user feedback, make necessary adjustments and improvements to the increments.

### 5. Finalization and Deployment

- **Complete System:** Continue developing and integrating increments until the full system is complete.
- **Deployment:** Deploy the final version of the system after thorough testing of the complete functionality.
- **Documentation and Training:** Provide documentation and training materials to users, if necessary.

### Advantages of the Incremental Model

- **Early Delivery:** Provides the opportunity to deliver and use portions of the system early in the development process.
- **Flexibility:** Easier to accommodate changes and incorporate user feedback throughout development.
- **Risk Management:** Reduces risk by breaking down the project into smaller, manageable pieces. Problems can be identified and resolved in smaller increments rather than all at once.
- **Improved User Engagement:** Continuous user feedback allows for improvements and ensures that the final product better meets user needs.

### Disadvantages of the Incremental Model

- **Integration Challenges:** Continuous integration of increments can sometimes lead to integration issues, especially if not managed properly.

- **Scope Creep:** Frequent changes and additions based on user feedback can lead to scope creep if not controlled.
- **Planning and Coordination:** Requires careful planning and coordination to ensure that each increment aligns with the overall project goals.

### Example of Using the Incremental Model in a Multi-Language Translation Tool

1. **Initial Increment:** Develop and release a basic translation tool with support for one or two languages. Focus on core translation functionality.
2. **Second Increment:** Add support for additional languages and enhance translation accuracy. Test these new features and integrate them with the existing system.
3. **Third Increment:** Introduce advanced features such as context-aware translation or user customization options. Continue testing and integration.
4. **Subsequent Increments:** Continue adding features, refining performance, and expanding language support based on user feedback and requirements.

By using the Incremental Model, you can build a multi-language translation tool progressively, allowing users to start benefiting from the tool early on and providing opportunities to refine and enhance the system based on real-world use and feedback.

**Conclusion:** Applying traditional process models like incremental and waterfall demonstrates their effectiveness in providing structured development and thorough validation each suited to different project needs.

### Sign and Remark:

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## EXPERIMENT NO.:02

**Date of Performance:**

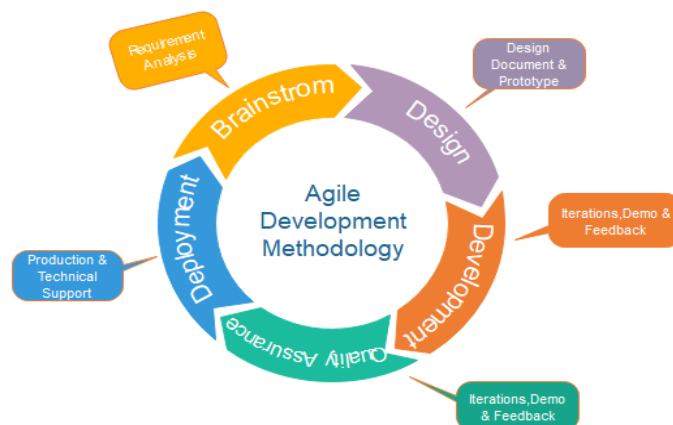
**Date of Submission:**

**Aim:** Application of the Agile process models.

**Software Used: JIRA**

Theory: The meaning of Agile is swift or versatile. “**Agile process model**” refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

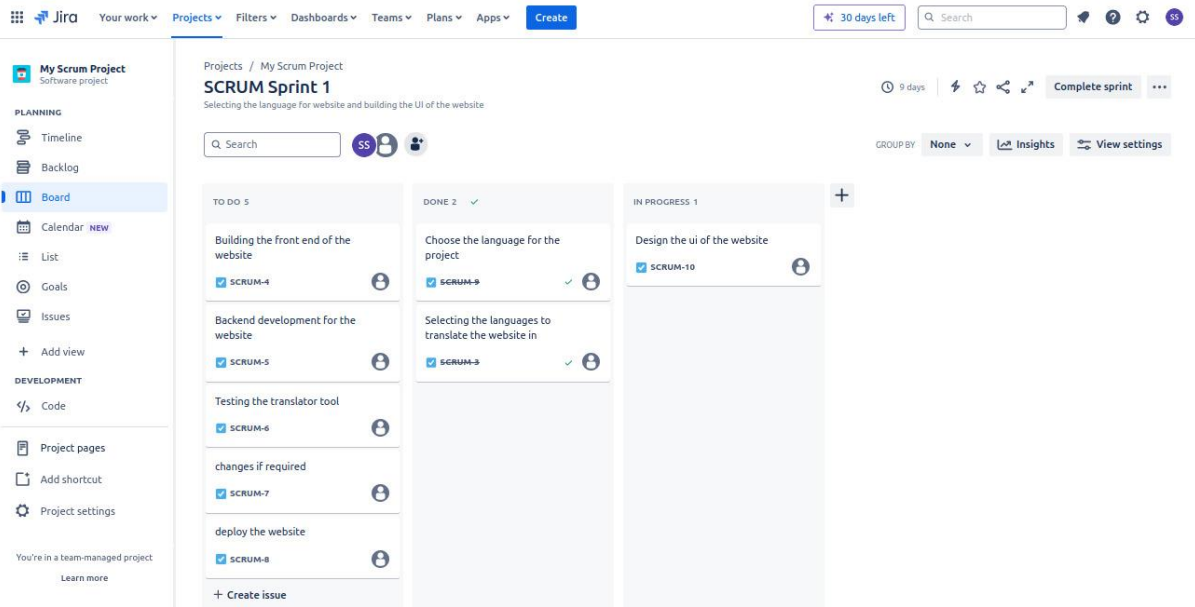


Every iteration involves cross functional teams working simultaneously on various areas like

- Planning
- Requirements Analysis
- Design
- Coding
- Unit Testing and
- Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.

Kanban methodology is based on the idea of continuous releases. Work is tracked using a kanban board that displays the statuses of work in columns and lanes. There are four important pillars to kanban to help teams ship products: continuous releases, WIP (work in progress) limits, the list of work, and columns or lanes. Here are some tools that come out-of-the-box in Jira Software's kanban template to help you run kanban with your team.



Conclusion:

<https://www.atlassian.com/software/jira>

[https://www.youtube.com/watch?v=uM\\_m6EzMg3k](https://www.youtube.com/watch?v=uM_m6EzMg3k)

Sign and Remark:

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## **EXPERIMENT NO.:03**

**Date of Performance:**

**Date of Submission:**

**Aim: Preparation of Software Requirement Specification (SRS) document in IEEE format**

**Software Used: Ms-Word**

### **Theory:**

The purpose of this document is to give a detailed description of the requirements for the software. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications. This document is primarily intended to be proposed to a customer for its approval and a reference for developing the first version of the system for the development team.

### Table of Contents

#### Introduction

##### 1.1 Purpose

##### 1.2 Scope

##### 1.3 Definitions, acronyms, and abbreviations

##### 1.4 References

#### 2. Overall description

##### 2.1 Product perspective

##### 2.2 Product functions

##### 2.3 User characteristics

##### 2.4 Constraints

##### 2.5 Assumptions and dependencies

##### 2.6 Apportioning of requirements

#### 3. Specific requirements

##### 3.1.1 User interfaces

##### 3.1.2 Hardware interfaces

##### 3.1.3 Software interfaces

##### 3.1.4 Communications interfaces

## 3.2 Functional requirements

### 3.2.1 User Class 1

### 3.2.2 User Class 2

### 3.2.3 User Class 3

## 3.3 Performance requirements

## 3.4 Design constraints

## 3.5 Software system attributes

## 4. Prioritization and Release Plan

### 4.1 Choice of prioritization method

## Appendix

## 1. Introduction

### 1.1 Purpose

The purpose of the Multi-Language Translation Tool (MLTT) project is to develop a comprehensive application that enables users to translate content seamlessly across multiple languages. The tool aims to provide an intuitive and user-friendly interface for both individuals and organizations, facilitating accurate and efficient translation. The goal is to enhance global communication, content accessibility, and interaction by offering a versatile and dynamic translation solution.

### 1.2 Scope

The scope of the MLTT project includes the design and development of an application with features that support text translation, speech translation, document translation, and contextual understanding. Key functionalities include user registration and profiles, translation memory, language selection, translation history, and integration with third-party services. The project also involves ensuring compatibility across various devices and platforms through responsive design.

### 1.3 Definitions, Acronyms, and Abbreviations

- MLTT: Multi-Language Translation Tool
- NLP: Natural Language Processing, the technology used for understanding and generating human language.



- OCR: Optical Character Recognition, technology to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data.
- API: Application Programming Interface, a set of protocols for building and interacting with software applications.
- UI: User Interface, the visual elements through which users interact with the application.
- UX: User Experience, the overall experience and satisfaction a user derives from interacting with the application.
- ML: Machine Learning, a type of AI that allows systems to learn and improve from experience without being explicitly programmed.
- Translation Memory: A database that stores previously translated segments to improve consistency and efficiency in future translations.
- Responsive Design: A design approach that ensures the application is usable on various devices and screen sizes.

#### 1.4 References

- Books: Refer to books on translation technologies, natural language processing, and user experience design. Examples include "Speech and Language Processing" by Daniel Jurafsky and James H. Martin and "Designing Interfaces" by Jenifer Tidwell.
- Online Platforms: Websites such as TechCrunch, Towards Data Science, and various tech blogs provide articles, tutorials, and best practices for translation technology development, web design, and user experience.

## 2. Overall Description

A Multi-Language Translation Tool is designed to facilitate seamless translation of text, speech, and documents across various languages. The tool incorporates advanced NLP and machine learning algorithms to provide high-quality translations. Key functionalities include customizable translation settings, user profiles, and integration with third-party services for enhanced capabilities.

## 2.1 Product Perspective

The product perspective for MLTT involves understanding the features, functionalities, and design considerations essential for a successful translation tool. Key aspects include:

- **User Experience:** Providing an intuitive and engaging interface for users to interact with translation features.
- **Translation Accuracy:** Ensuring high-quality translations through advanced NLP and ML algorithms.
- **Integration Capabilities:** Offering API access for integration with other applications and services.
- **Contextual Understanding:** Enhancing translation accuracy by understanding the context of the content.

## 2.2 Product Functions

Key functions of the MLTT include:

- **Text Translation:** Translating written text between supported languages.
- **Speech Translation:** Converting spoken language into text and translating it.
- **Document Translation:** Translating entire documents while preserving formatting.
- **Translation Memory:** Storing and reusing previously translated segments.
- **User Profiles:** Allowing users to manage their translation history and preferences.
- **Integration:** Connecting with external APIs and services for additional functionality.

## 2.3 User Characteristics

Understanding user characteristics is crucial for designing a translation tool that meets diverse needs. Considerations include:

- **Technical Proficiency:** Catering to both novice and advanced users.
- **Language Proficiency:** Supporting users with varying levels of language proficiency.
- **Accessibility Needs:** Ensuring the tool is usable by people with disabilities.
- **Use Cases:** Accommodating different translation needs such as personal use, business, and professional translation.

## 2.4 Constraints

Constraints to consider include:

- **Language Limitations:** Not all languages may be supported, and some translations may have limited functionality.
- **Data Privacy:** Ensuring compliance with data protection regulations and safeguarding user data.
- **Performance:** Ensuring the tool can handle high volumes of translation requests efficiently.
- **Compatibility:** Ensuring the tool works across different devices and browsers.

## 2.5 Assumptions and Dependencies

Assumptions and dependencies include:

- **Internet Connectivity:** Assuming users will have access to the internet for full functionality.
- **Web Browsers:** Compatibility with major web browsers and operating systems.
- **NLP and ML Models:** Availability and accuracy of language models for translation.
- **User Engagement:** Assumption that users will actively use the translation features.

## 2.6 Apportioning of Requirements

Requirements for the MLTT can be apportioned as follows:

- **Core Functionality:** Essential features such as text and speech translation.
- **User Interface and Experience:** Design and usability enhancements.
- **Integration:** API and third-party service integrations.
- **Advanced Features:** Additional functionalities like contextual understanding and translation memory.

## 3. Specific Requirements

### 3.1 User Interfaces

- **Graphical Interface:** The tool shall provide a user-friendly graphical interface for initiating and managing translations.

- **Responsive Design:** The interface shall be responsive and accessible across various devices.
- **Customizable Settings:** Users shall be able to customize translation preferences and settings.

### 3.2 Performance Requirements

- **Response Time:** The tool shall deliver translations within 2 seconds for text input and 5 seconds for document uploads.
- **Load Handling:** The tool shall efficiently handle high traffic and multiple concurrent users.

### 3.3 Software System Attributes

- **Usability:** The tool shall be easy to use and navigate.
- **Scalability:** The tool shall be scalable to accommodate growing user numbers and content volume.
- **Security:** The tool shall ensure the security of user data and translation content.
- **Reliability:** The tool shall be reliable with minimal downtime.

## 4. Prioritization and Release Plan

### Prioritization

**MVP (Minimum Viable Product):**

Basic text and speech translation features.

User registration and profile management.

Core language support.

**Release 1: Enhanced Features:**

Advanced text and document translation tools.

Improved language support and customization.

Basic translation memory integration.

#### Release 2: User Interaction:

- Commenting and feedback features.
- Integration with additional third-party services.
- Enhanced translation history and management.

#### Release 3: Personalization and Localization:

- User customization options.
- Multi-language support and localization.
- Advanced contextual understanding and recommendations.

#### Release 4: Performance and Expansion:

- Performance optimization and scalability improvements.
- Support for multimedia content and extended file formats.
- Expanded analytics and reporting capabilities.

#### Release Plan:

##### Month 1-2: MVP Development:

- Develop core translation features and user management.
- Implement basic language support.

##### Month 3-4: Release 1: Enhanced Features:

- Introduce advanced translation tools and improved language support.
- Integrate basic translation memory features.

##### Month 5-6: Release 2: User Interaction:

- Add commenting and feedback features.
- Enhance translation history and third-party integrations

##### Month 7-8: Release 3: Personalization and Localization:

- Develop user customization options and multi-language support.
- Implement advanced contextual understanding.

Month 9-10: Release 4: Performance and Expansion:

Optimize performance and scalability.

Support multimedia content and expand analytics features.

**Conclusion:**

In conclusion, the development of the Multi-Language Translation Tool offers significant opportunities to enhance global communication and content accessibility. By implementing a well-structured development plan and prioritizing key features, the tool can deliver a robust and user-friendly experience, enabling seamless translation and interaction across multiple languages.

**Sign and Remark:**

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## **EXPERIMENT NO.:04**

**Date of Performance:**

**Date of Submission:**

**Aim:** Structured Data Flow Analysis

### **Software Used: Star UML**

Theory: Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes –

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

### **Structured Analysis Tools**

During Structured Analysis, various tools and techniques are used for system development. They are –

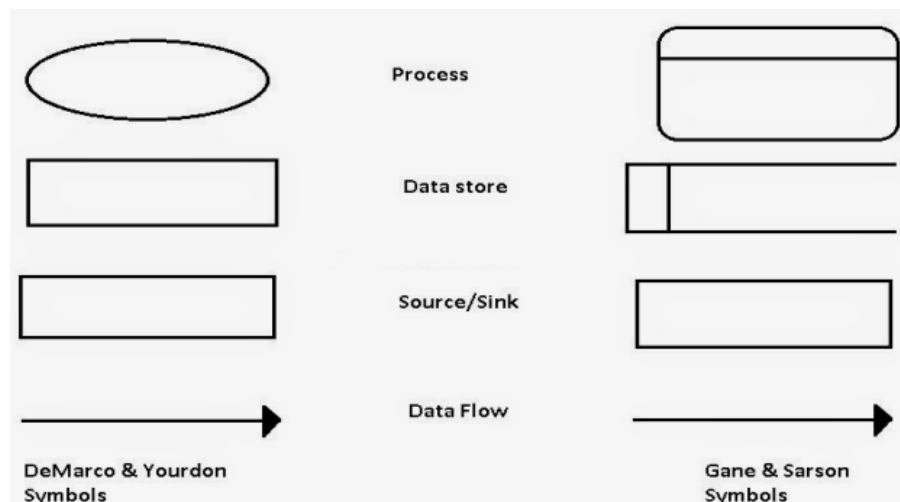
- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudo code

### **Data Flow Diagrams (DFD)**

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

## Symbols used in DFD



## Used Case Diagram

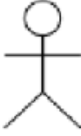
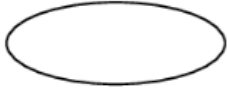
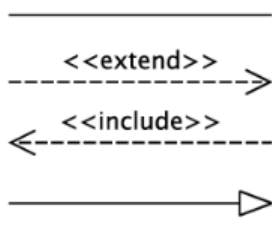
In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and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

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

### UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case



Symbol	Reference Name
	Actor
	Use case
	Relationship

**Conclusion:**

Link to download software: <https://staruml.io/download>

**Sign and Remark:**

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## **EXPERIMENT NO.:05**

**Date of Performance:**

**Date of Submission:**

**Aim:** Use of Metrics to estimate the cost

**Software Used: Ms-Word**

### **Software Cost Estimation**

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually.  
To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

### **Uses of Cost Estimation**

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.
3. Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.
4. Estimation determines how much money, effort, resources, and time it will take to build a specific system or product.

### **Loc-based Cost Estimation**

The LOC (Line of Code) is a product size metric in software engineering. Here, the number of lines in the code are counted and based on the number of lines the cost is calculated.

### **LOC-based Estimation**

- Different languages lead to different lengths of code
- It is not clear how to count lines of code
- A report, screen, or GUI generator, can generate thousands of lines of code in minutes

- Depending on the application, the complexity of code is different

### **Function Points: FP**

Function Points is used in 2 contexts:

- **Past:** To develop **metrics** from historical data
- **Future:** Use of available metrics to size the s/w of a new project

### **FP-based Estimation**

- Based on FP metric for the size of a product
- Based on the number of inputs (Inp), outputs (Out), inquiries (Inq), master files (Maf), interfaces (Inf)
- Classify each component of the product (Inp, Out, Inq, Maf, Inf) as simple, average, or complex (next slide)

**Conclusion:**

**Sign and Remark:**

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## **EXPERIMENT NO.:06**

**Date of Performance:**

**Date of Submission:**

**Aim:** Scheduling and tracking of the project

**Software Used:** Gantt Project

**Theory:- Project Scheduling**

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following:

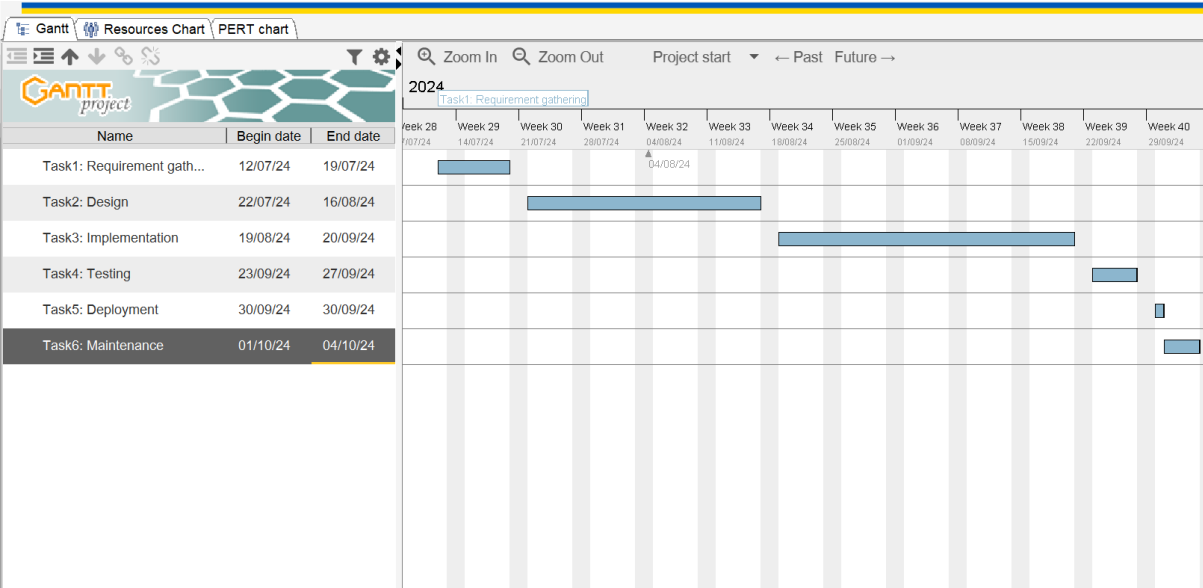
1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

The first method in scheduling a software plan involves identifying all the functions required to complete the project. A good judgment of the intricacies of the project and the development process helps the supervisor to identify the critical role of the project effectively. Next, the large functions are broken down into a valid set of small activities which would be assigned to various engineers. The work breakdown structure formalism supports the manager to breakdown the function systematically after the project manager has broken down the purpose and constructs the work breakdown structure; he has to find the dependency among the activities. Dependency among the various activities determines the order in which the various events would be carried out. If an activity A necessary the results of another activity B, then activity A must be scheduled after activity B. In general, the function dependencies describe a partial ordering among functions, i.e., each service may precede a subset of other functions, but some functions might not have any precedence ordering describe between them (called concurrent function). The dependency among the activities is defined in the pattern of an activity network.

Once the activity network representation has been processed out, resources are allocated to every activity. Resource allocation is usually done using a Gantt chart. After resource allocation is completed, a PERT chart representation is developed. The PERT chart representation is useful for program monitoring and control. For task scheduling, the project plan needs to decompose the project functions into a set of activities. The time frame when every activity is

to be performed is to be determined. The end of every action is called a milestone. The project manager tracks the function of a project by audit the timely completion of the milestones. If he examines that the milestones start getting delayed, then he has to handle the activities carefully so that the complete deadline can still be met.

Gantt Chart:



Conclusion:

Thus, we have scheduled and tracked our project using gantt chart.

Sign and Remark:

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## EXPERIMENT NO.:07

**Date of Performance:**

**Date of Submission:**

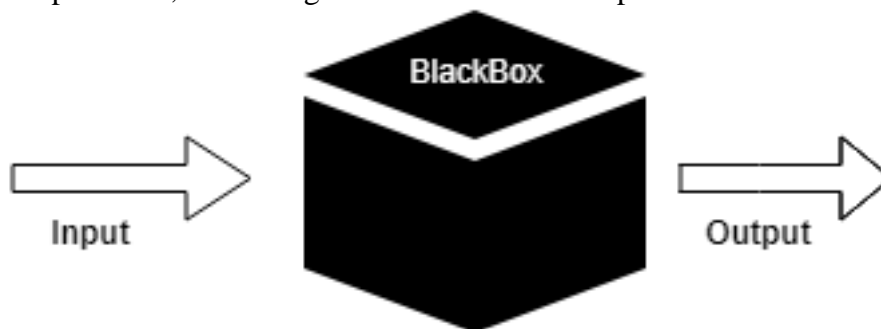
**Aim:** Write test cases for black box testing

**Software Used:** Selenium/GitHub/Jira

### Theory:-

**Black Box Testing** is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.



### Generic steps of black box testing

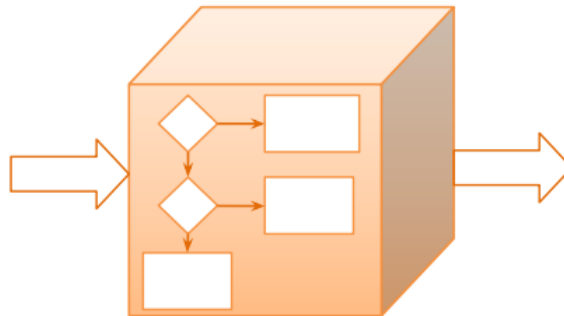
- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

### White Box Testing

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and

security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

The developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and



identify the bugs and sends it to the developer.

### Test procedure

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function. A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique.

### Test cases

Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications. For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team; there is not any involvement of the development team of software.

### Techniques Used in Black Box Testing

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.

State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result. It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

Test Case ID	Test Case Title	Test Scenario	Input Data	Expected Result
TC-001	Validate Language Switch	Test switching between Marathi and English	Select "Marathi" from dropdown	Interface updates to Marathi with all text correctly displayed.
TC-002	Check Functional Buttons	Verify button functionality in both languages	Click "Buy Now" in Hindi	Process of buying done successfully with confirmation in Hindi.
TC-003	UI Consistency	Ensure layout remains consistent	Access site in various languages	Layout is consistent across all languages with no overlapping elements.
TC-004	Validate Translations	Check accuracy of translated content	Navigate to "About Us" page in Bengali	Text is accurately translated to Bengali and contextually appropriate.
TC-005	Error Message Localization	Test error handling in different languages	Enter invalid data in any form	Displays an appropriate error message in the selected language.
TC-006	Performance Check	Assess loading times for different languages	Access homepage in Tamil	Page loads within acceptable time limits (e.g., < 3 seconds).



TC-007	Accessibility Testing	Verify screen reader compatibility	Use screen reader with the site in Kannada	Screen reader reads all content correctly in Kannada.
TC-008	Usability Feedback	Gather user feedback on multilingual interface	User feedback survey	Positive feedback regarding ease of navigation in all languages.
TC-009	Cross-Browser Compatibility	Check application across browsers	Access site in Chrome, Firefox, and Safari	Site functions correctly in all tested browsers without issues in all languages.

**Conclusion:** Thus studied black box testing for Web Translating Tool.

**Sign and Remark:**

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## EXPERIMENT NO.:08

**Date of Performance:**

**Date of Submission:**

**Aim:** Write test cases for white box testing

**Software Used:** Selenium/GitHub/Jira

### **Theory:-**

**White Box Testing** is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing and Glass box testing.

White box testing which also known as glass box is **testing, structural testing, clear box testing, open box testing and transparent box testing**. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

Here, the test engineers will not include in fixing the defects for the following reasons:

- Fixing the bug might interrupt the other features. Therefore, the test engineer should always find the bugs, and developers should still be doing the bug fixes.
- If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.

### Techniques Used in White Box Testing

Data Flow Testing	Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.
Control Flow Testing	Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.
Branch Testing	Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.
Statement Testing	Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code, out of total statements present in the source code.
Decision Testing	This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Test Case ID	Test Case Title	Description	Expected Result
WTC-001	Code Path Coverage	Ensure all code paths for language switching are tested	All relevant code paths executed with no errors.
WTC-002	Validate Translation Logic	Validate logic for fetching translations	Correct translations returned for Marathi without errors.
WTC-003	Conditional Statements Testing	Test conditional statements in language selection	The correct language displayed based on user selection.
WTC-004	Input Validation Logic	Check input validation for forms in multiple languages	Proper error messages displayed in the selected language for invalid inputs.
WTC-005	Performance Profiling	Assess performance of translation retrieval	Loading time remains within acceptable limits for all languages.
WTC-006	Security Testing	Test for SQL injection vulnerabilities in language input	Page loads within acceptable time limits (e.g., < 3 seconds).
WTC-007	Loop and Iteration Testing	Validate loops in language switching functionality	Application handles multiple switches without performance degradation.
WTC-008	Resource Management	Check memory usage for loading translations	Memory usage remains optimal without leaks or crashes.
WTC-009	Exception Handling Verification	Verify exception handling for unavailable translations	Application handles errors gracefully with a fallback message.

**Conclusion:** Thus studied white box testing for Web Translating Tool.

**Sign and Remark:**

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## **EXPERIMENT NO.:09**

**Date of Performance:**

**Date of Submission:**

**Aim:** Preparation of Risk Mitigation, Monitoring and Management plan (RMMM.)

**Software Used:** Ms Word

**Theory:-**

RMMM Plan:

A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan.

In some software teams, risk is documented with the help of a Risk Information Sheet (RIS). This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis. After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

### **Risk Mitigation:**

It is an activity used to avoid problems (Risk Avoidance).

Steps for mitigating the risks as follows.

1. Finding out the risk.
2. Removing causes that are the reason for risk creation.
3. Controlling the corresponding documents from time to time.
4. Conducting timely reviews to speed up the work.

### **Risk Monitoring:**

It is an activity used for project tracking.

It has the following primary objectives as follows.

To check if predicted risks occur or not.

1. To ensure proper application of risk aversion steps defined for risk.
2. To collect data for future risk analysis.
3. To allocate what problems are caused by which risks throughout the project.

### **Risk Management and planning:**

It assumes that the mitigation activity failed and the risk is a reality. This task is done by Project manager when risk becomes reality and causes severe problems. If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks. This shows that the response that will be taken for each risk by a manager. The main objective of the risk management plan is the risk register. This risk register describes and focuses on the predicted threats to a software project.

### **Steps for Risk Management**

1. Identify possible risks and recognize what can go wrong

2. Analyse each risk to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur
3. Rank the risks by probability and impact. Impact may be negligible, marginal, critical, and catastrophic.
4. Develop a contingency plan to manage those risks having high probability and high impact

### Risk Table

Risks ID	Risks	Category	Probability	Impact	Risk Management
R001	Delays in translation accuracy	Technical	Medium	2 (Critical)	Implement a review process with linguistic experts; schedule regular check-ins.
R002	Compatibility issues across browsers	Technical	Low	2 (Critical)	Perform cross-browser testing early in development; document known issues.
R003	User interface inconsistencies	Usability	Medium	3 (Marginal)	Create a UI style guide; conduct user testing sessions for feedback.
R004	Security vulnerabilities	Security	Low	1(Catastrophic)	Conduct regular security audits; implement best practices for secure coding.
R005	Performance degradation under load	Performance	Medium	2 (Critical)	Optimize code and resources; stress-test the application before launch.
R006	Cultural misinterpretation in translation	Content	Medium	3 (Marginal)	Involve native speakers in the review process; create a glossary of terms.
R007	Lack of resources for extensive testing	Resource	High	3 (Marginal)	Allocate budget for additional testers; prioritize test scenarios based on risk.
R008	Inadequate error handling	Technical	Low	2 (Critical)	Implement robust error logging; conduct thorough error-handling reviews.
R009	User resistance to new languages	Human Factors	Medium	3 (Marginal)	Provide clear communication about benefits; offer training sessions.
R0010	Insufficient training for team members	Organizational	Medium	3 (Marginal)	Develop a comprehensive training program; schedule regular knowledge-sharing sessions.

Impact Values:            1 – Catastrophic            2 – Critical            3 – Marginal    4 – Negligible

**Conclusion:** Thus, we have identified and assessed the potential risks associated with the Web Translating Tool, enabling us to implement effective mitigation strategies for a successful project outcome.

**Sign and Remark:**

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## EXPERIMENT NO.:10

**Date of Performance:**

**Date of Submission:**

**Aim: Version control of the project**

**Software Used: GitHub**

### **Theory:-**

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done to the code.

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes in some specific kind of functionality/features. So in order to contribute to the product, they made modifications in the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage (track) all the changes that have been made to the source code along with the information like who made and what change has been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

### **Benefits of the version control system:**

- a) Enhances the project development speed by providing efficient collaboration,
- b) Leverages the productivity, expedite product delivery, and skills of the employees through better communication and assistance,
- c) Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
- d) Employees or contributor of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,
- e) For each different contributor of the project a different working copy is maintained and not merged to the main file unless the working copy is validated. A most popular example is **Git, Helix core, Microsoft TFS**,
- f) Helps in recovery in case of any disaster or contingent situation,
- g) Informs us about Who, What, When, Why changes have been made.

### **Use of Version Control System:**

- **A repository:** It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
- **Copy of Work (sometimes called as checkout):** It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.




# Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*

 SujalSongire ▾

Repository name \*

/ E-commerce-application

✔ E-commerce-application is available.

Great repository names are short and memorable. Need inspiration? How about [verbose-barnacle](#) ?

Description (optional)

☐

Public

Anyone on the internet can see this repository. You choose who can commit.

☒

Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐

Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)



You are creating a private repository in your personal account.

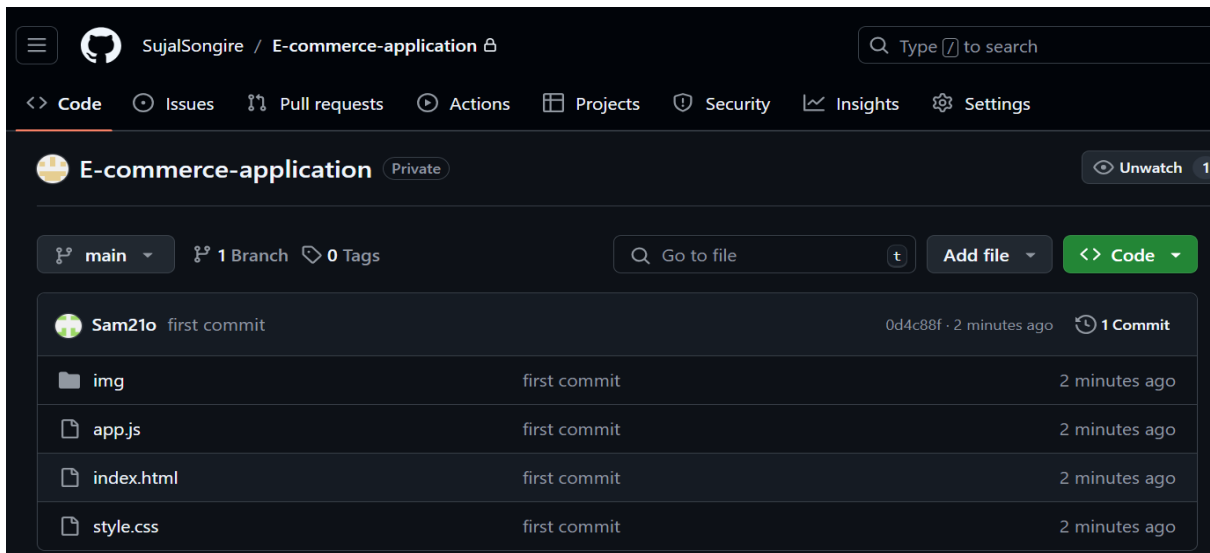
Create repository

```
PS C:\Users\sujal\Desktop\project> git init
Initialized empty Git repository in C:/Users/sujal/Desktop/project/.git/
PS C:\Users\sujal\Desktop\project> git add .
warning: in the working copy of 'app.js', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'style.css', LF will be replaced by CRLF the next time Git touches it
PS C:\Users\sujal\Desktop\project> git commit -m "first commit"
[master (root-commit) 0d4c88f] first commit
25 files changed, 1321 insertions(+)
```

```

PS C:\Users\sujal\Desktop\project> git branch -M main
PS C:\Users\sujal\Desktop\project> git remote add origin https://github.com/SujalSongire/E-commerce-application.git
PS C:\Users\sujal\Desktop\project> git push -u origin main
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 12 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (28/28), 3.85 MiB | 1.38 MiB/s, done.
Total 28 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/SujalSongire/E-commerce-application.git
 * [new branch]    main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\sujal\Desktop\project>

```



```

PS C:\Users\sujal\Desktop\project> git add .
PS C:\Users\sujal\Desktop\project> git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
PS C:\Users\sujal\Desktop\project>

```

**Conclusion:** In conclusion, implementing version control for the Web Translating Tool enhances collaboration, streamlines development processes, and ensures the integrity and traceability of code changes, ultimately leading to a more robust and maintainable application.

### Sign and Remark:

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	

## **EXPERIMENT NO.:11**

**Date of Performance:**

**Date of Submission:**

**Aim:** The context diagram of mess management

### **Lab Objectives:**

- 1 To solve real life problems by applying software engineering principles
- 2 To impart state-of-the-art knowledge on Software Engineering
- **Lab Outcomes:**
- On successful completion of laboratory experiments, learners will be able to:
  - 1. Develop architectural models for the selected case study.
  - 2. Use computer-aided software engineering (CASE) tools.

### **Software Used: Star UML**

Theory: Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes –

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

### **Structured Analysis Tools**

During Structured Analysis, various tools and techniques are used for system development. They are –

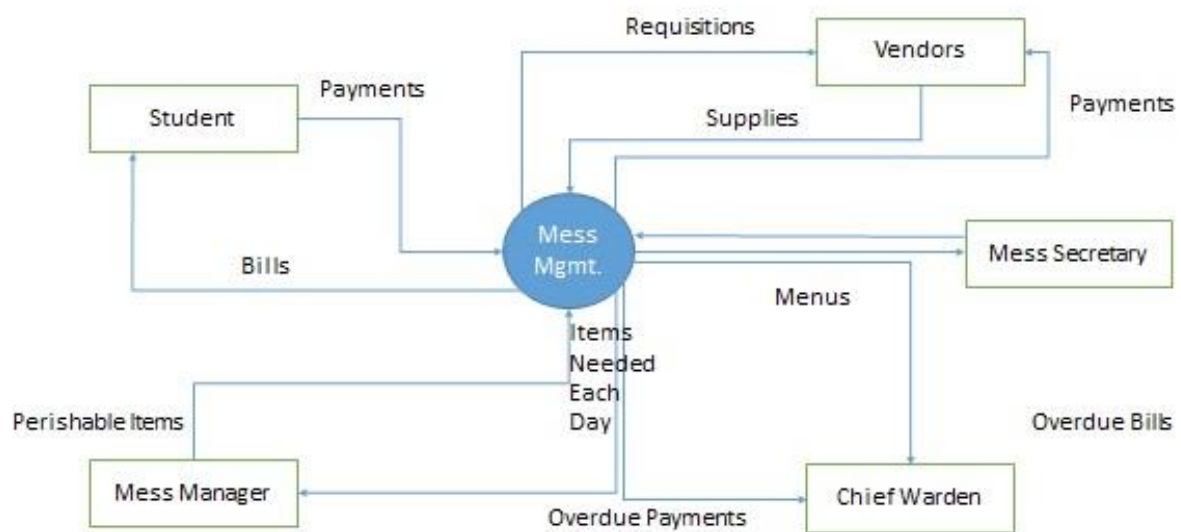
- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables

- Structured English
- Pseudo code

## Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



## Used Case Diagram

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

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

**UML use case diagrams are ideal for:**

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case

**Conclusion:**

**Sign and Remark:**

<b>R1</b>	<b>R2</b>	<b>R3</b>	<b>Total Marks</b>	<b>Signature</b>
<b>(5)</b>	<b>(5)</b>	<b>(5)</b>	<b>(15)</b>	

## **EXPERIMENT NO.:11**

**Date of Performance:**

**Date of Submission:**

**Aim:** Activity Diagram for Online Shopping System

**THEORY:** An activity diagram shows the flow from activity to activity. An activity is an ongoing non atomic execution within a state machine .Activities ultimately results in some action, which is made up of executable atomic computations. We can use these diagrams to model the dynamic aspects of a system. Activity diagram is basically a flow chart to represent the flow form one activity to another. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow by using elements like fork, join etc.

### **Fork**

A fork represents the splitting of a single flow of control into two or more concurrent Flow of control. A fork may have one incoming transition and two or more outgoing transitions, each of which represents an independent flow of control. Below fork the activities associated with each of these path continues in parallel.

### **Join**

A join represents the synchronization of two or more concurrent flows of control. A join may have two or more incoming transition and one outgoing transition. Above the join the activities associated with each of these paths continues in parallel.

### **Branching**

A branch specifies alternate paths takes based on some Boolean expression Branch is represented by diamond Branch may have one incoming transition and two or more outgoing one on each outgoing transition, you place a Boolean expression shouldn't overlap but they should cover all possibilities.

### **Swim lane:**

Swim lanes are useful when we model workflows of business processes to partition the activity states on an activity diagram into groups. Each group representing the business organization responsible for those activities, these groups are called Swim lanes.

### **Procedure:-**

Step1: First initial state is created.

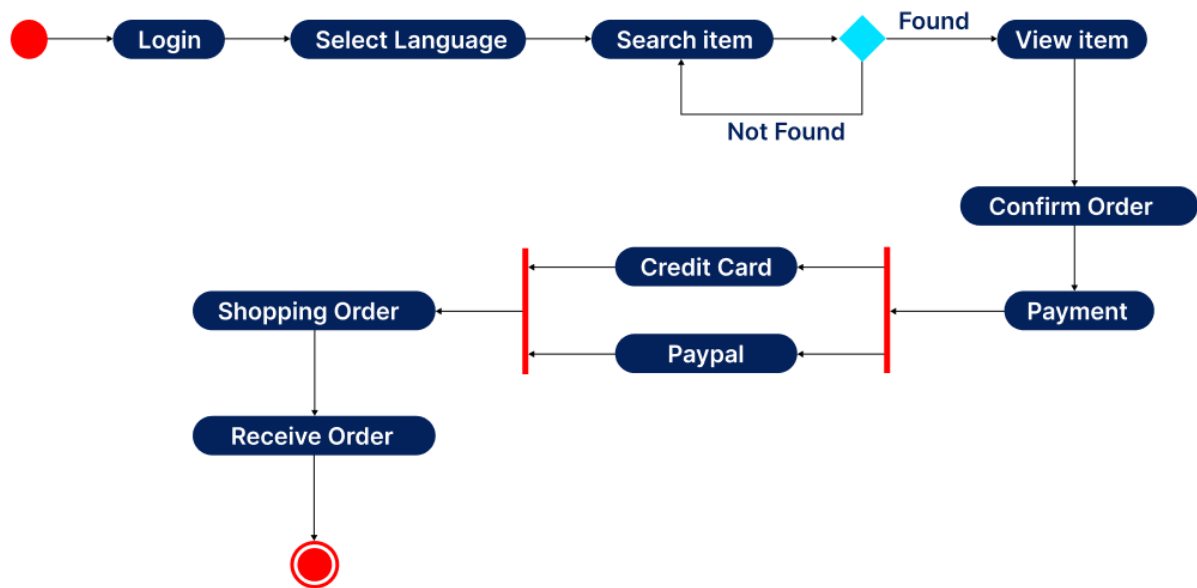
Step2: After that it goes to the action state insert card.

Step3: Next it undergoes transition to the state enter pin

Step4: In this way it undergoes transitions to the various states.

Step5: Use forking and joining wherever necessary.

### DIAGRAM:



**Conclusion:** Thus, we studied the purchasing process of the eCommerce website, illustrating the user journey through various activities from browsing products to receiving order confirmation.

### Sign and Remark:

R1	R2	R3	Total Marks	Signature
(5)	(5)	(5)	(15)	