MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)
DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual For

Software Engineering Lab
Course CODE :CSL501

SE SEM IV (R-2019 C- Scheme)

Prepared By
Dr. Sharmila Gaikwad

2020-2021

DEPARTMENT OF COMUTER ENGINEERING

Juhu Versova Link Road
Andheri(W) Mumbai-53

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)

DEPARTMENT OF COMPUTER ENGINEERING

Vision

To become a leading department committed to nurture student centric learning through outcome and skill based transformative IT education to create Technocrats and leaders for the service of society.

Mission

M1: To shape ourselves into a learning community to flourish leadership, team spirit, ethics, listen and respect each other.

M2: To provide computer educational experience that transforms student through rigorous course work and by providing an understanding of the need of the society and industry.

M3: To educate students to be professionally IT competent for industry and research program by providing industry institute interaction.

M4: To strive for excellence among students by infusing a sense of excitement in Computer innovation, invention, design, creation and entrepreneurship.

M5: To contribute in the service of society by participation of faculty, staff and students in socio-economic and socio-cultural activities.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1. To prepare the Learner with a sound foundation in the mathematical, scientific and engineering fundamentals.

PEO2. To motivate the Learner in the art of self-learning and to use modern tools for solving real life problems.

PEO3. To equip the Learner with broad education necessary to understand the impact of Computer Science and Engineering in a global and social context.

PEO4. To encourage, motivate and prepare the Learners for Life long learning.

PEO5.To inculcate professional and ethical attitude, good leadership qualities and commitment to social responsibilities in the Learners thought process.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion ofthe course, B.E Computer Engineering, the graduates will have the following Program Specific Outcomes:

PSO 1: An ability to apply the knowledge of computer engineering for the design and development of IoT system.

PSO 2: An ability to solve complex computer Engineering problems using latest technical tools to achieve optimized solutions.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)

DEPARTMENT OF COMPUTER ENGINEERING

PROGRAM OUTCOMES (POs)

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 multi-disciplinary 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.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)

DEPARTMENT OF COMPUTER ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before the starting time into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system according to roll number wise.
5. Execute your task in the laboratory and record the results / output in the lab observation notebook and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions.
9. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
10. Students must take the permission of the faculty in case of any urgency to go out.
11. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

OBJECTIVES AND OUTCOMES

Lab Code	Lab Name	Credit
CSL501	Software Engineering Lab	1

Prerequisite: Object Oriented Programming with Java , Python Programming	
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	Identify requirements and apply software process model to selected case study.
2	Develop architectural models for the selected case study.
3	Use computer-aided software engineering (CASE) tools.

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS

i. Intel based desktop PC of 166MHz or faster processor with at least 2GB RAM and 500 GB HDD.

ii. Java, Python Programming, StarUML

USEFUL TEXTBOOKS / REFERECES

Textbooks:

1 Roger Pressman, “Software Engineering: A Practitioner‘s Approach”, 9 th edition , McGraw-Hill Publications, 2019
2 Ian Sommerville, “Software Engineering”, 9 th edition, Pearson Education, 2011
3 Ali Behfrooz and Fredeick J. Hudson, "Software Engineering Fundamentals", Oxford University Press, 1997
4 Grady Booch, James Rambaugh, Ivar Jacobson, “The unified modeling language user guide”, 2 nd edition, Pearson Education, 2005

References:

1 Pankaj Jalote, "An integrated approach to Software Engineering", 3 rd edition, Springer, 2005
2 Rajib Mall, "Fundamentals of Software Engineering", 5 th edition, Prentice Hall India, 2014
3 Jibitesh Mishra and Ashok Mohanty, “Software Engineering”, Pearson , 2011
4 Ugrasen Suman, “Software Engineering – Concepts and Practices”, Cengage Learning, 2013
5 Waman S Jawadekar, “Software Engineering principles and practice”, McGraw Hill Education, 2004

List of Experiments

Sr. No	Name of Experiment	CO	PO
1	Application of at least two traditional process models.	1	1
2	Application of the Agile process models.	1	1
3	Preparation of software requirement specification.(SRS)	1	2
4	Structured data flow analysis	2	2
5	Use of metrics to estimate the cost.	2	3
6	Scheduling and tracking of the project.	3	8
7	Write test cases for black box testing.	3	9
8	Write test cases for white box testing.	3	10
9	Preparation of Risk Mitigation, Monitoring and Management Plan(RMMM).	3	9
10	Version controlling of the project.	3	11

Experiment No.1

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

Theory:

Software development cycle and Software Process Models.
Student should read and observe

What is SDLC?

A software development life cycle (SDLC) is a hypothetical structure which explains all process activities in a software development with its stages and tasks involved in each step from planning to completion of support and maintenance.

Purpose of SDLC:

In development of software, use of Software Development Life Cycle serve the purposes such as:

1. To produce a quality software.
2. To deliver the software in timeline/deadline given by the customer.
3. Software should be developed within the budget mentioned by customer.
4. And the most important is customer's satisfaction about requirementfulfilment. SDLC gives the step by step, disciplined and measurable approach for developing good quality software.

What are the stages of SDLC?

The Software Development Life Cycle includes following phases:

1. Requirement Analysis
2. Planning
3. Design
4. Implementation (Coding)
5. Testing and Maintenance

Software Development Life Cycle (SDLC) has family that contains software process models which follow the framework of SDLC. Software Process Models (SPM) has three main categories those are as follows:

1. Linear Sequential Model
2. Prototyping Model
3. Evolutionary Model

These all types of models follow the common approach of SDLC with slight difference in their execution process.

Sr No.	Name of the Phase	Deliverable it contains
1.	Requirement Analysis	This phase contains SRS (Software Requirement Specification) document. This includes details about problem recognition, evaluation and synthesis, modelling, system specification and review. SRS is used as a tool for the assessment of quality once the software is developed.
2.	Planning	Planning document which has planning about resources, product and processes.
3.	Design	This phase serves a document which gives detail information about- software architecture, data structure, interface and procedure or algorithm details to be used as per the requirement of the project.
4.	Implementation	Here the design is translated into machine readable form that is coding.
5.	Testing	Before release, software must be tested to remove the errors and to be checked for its validity about requirement fulfilment.
6.	Support and maintenance	Is the endless procedure, it has three activities correcting errors, improving for correctness, installation or deployment?

Software Quality Parameters: “Quality is not a feature which can be incorporated once the software is developed”. Care of the quality about software must be taken from the very first stage of the software development process.

Software quality parameters are:

1. Fulfilment of Requirements
2. Reliability
3. Efficiency
4. Flexibility
5. Functionality
6. Ease of Use
7. Low Cost
8. Good Design

The working of linear sequential model, prototyping model and evolutionary model with advantages and disadvantages.

Following are the types of evolutionary software process models.

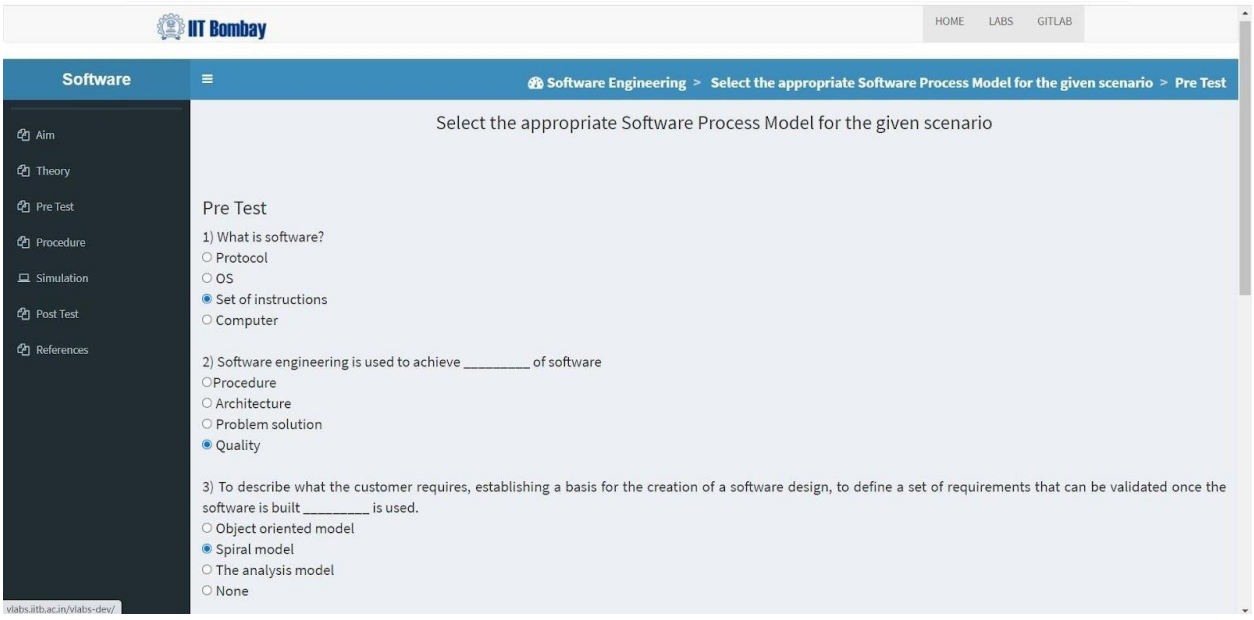
1. The Incremental Model
2. The Spiral Model
3. The WIN Spiral Model
4. The concurrent Development Model

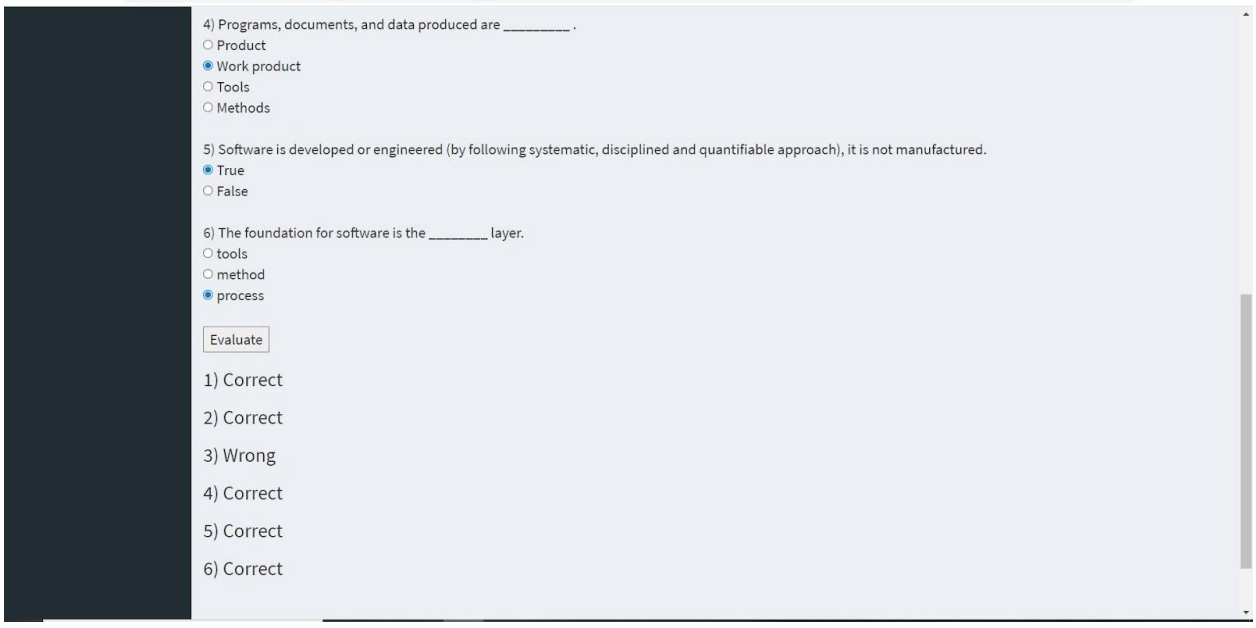
Problem statement:

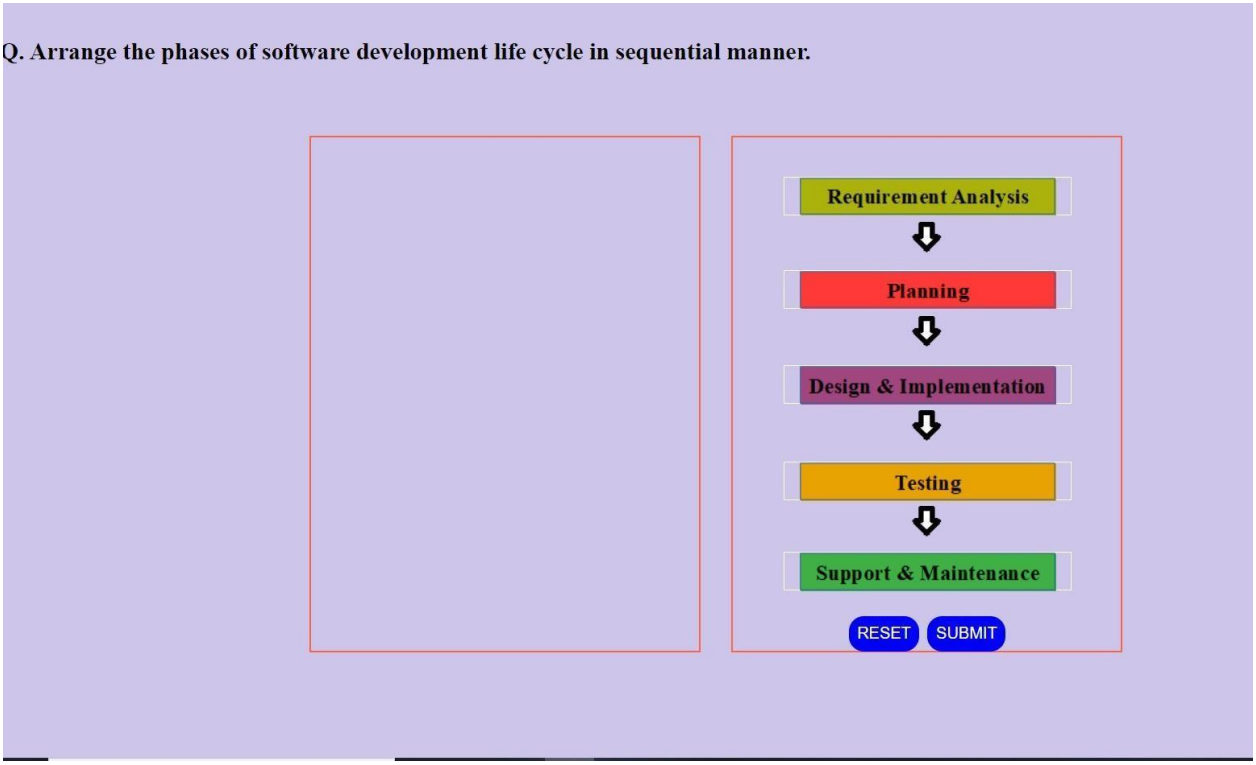
Example CASE Study (The objective of this project is to design a website on pandemic management system which will help people to track necessary services during the Times of pandemic or emergency. The website will track services near your location like hospital available with ICU, Blood bank, ambulance and medical stores.

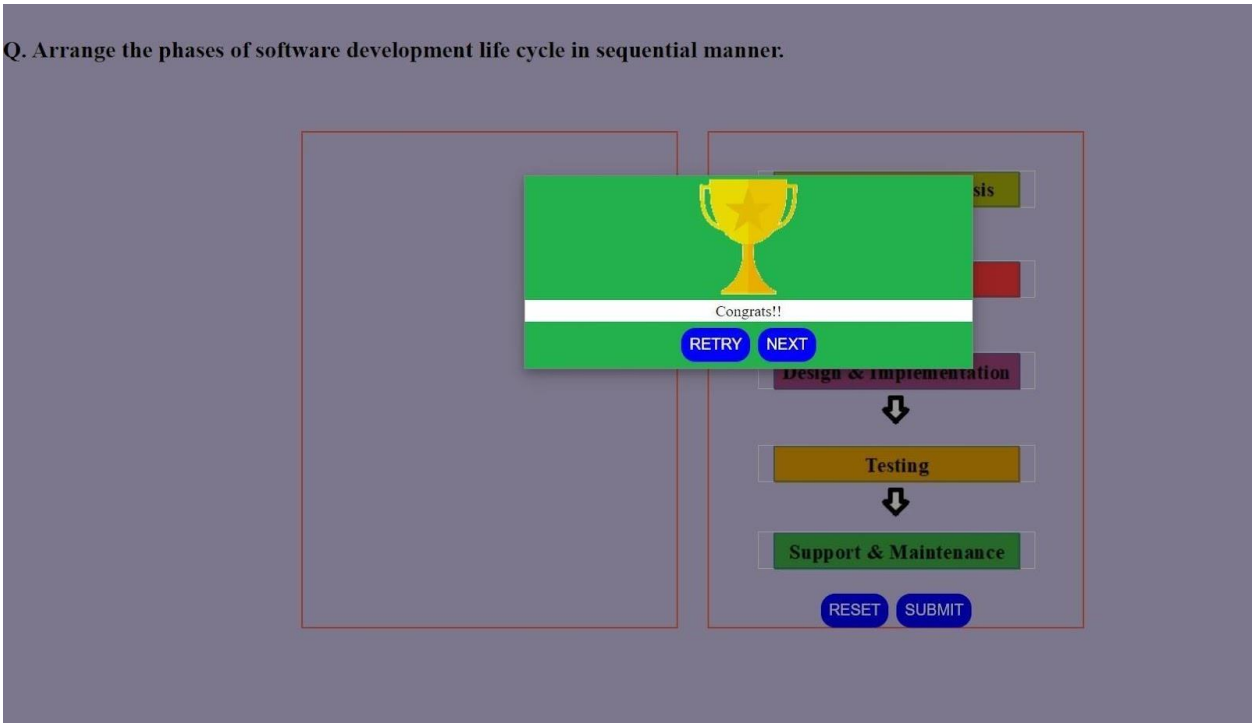
In the current senecio we all are facing a global pandemic namely covid-19 which had made massive impact on everyone's life. The government bodies and people's now have realised that they need to be prepared from such conditions in future, keeping this demand in mind we're designing this pandemic management system.)

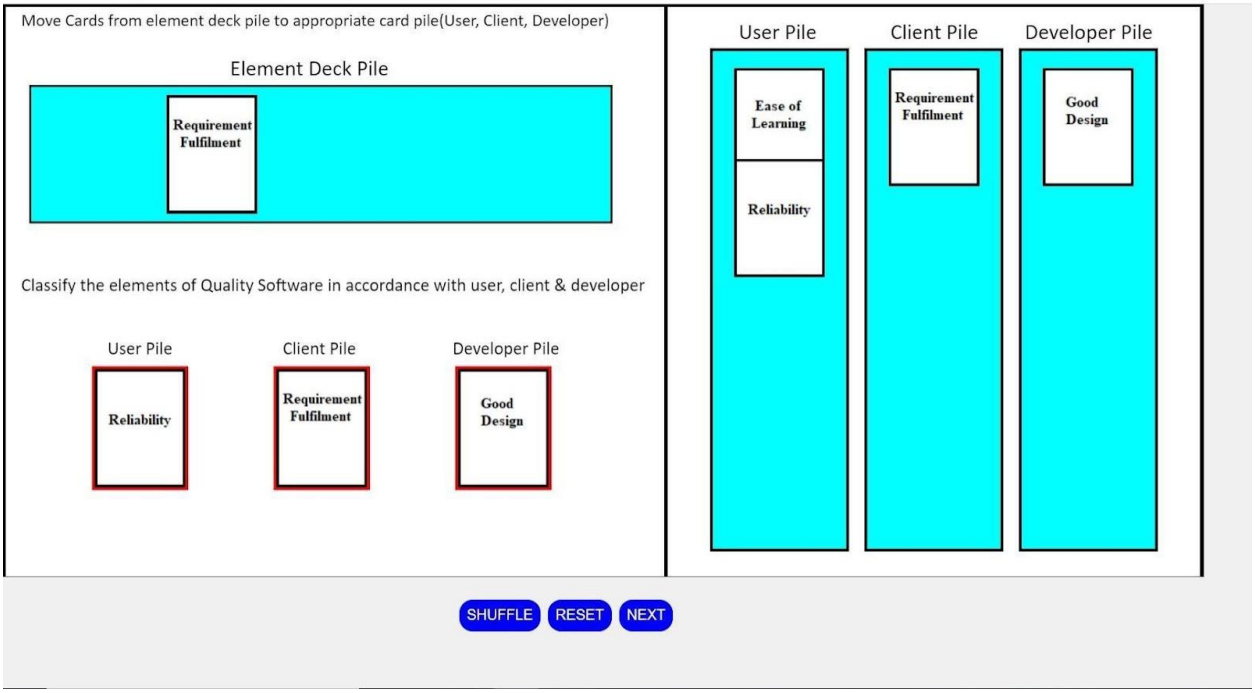
Output for IIT Virtual LAbs:

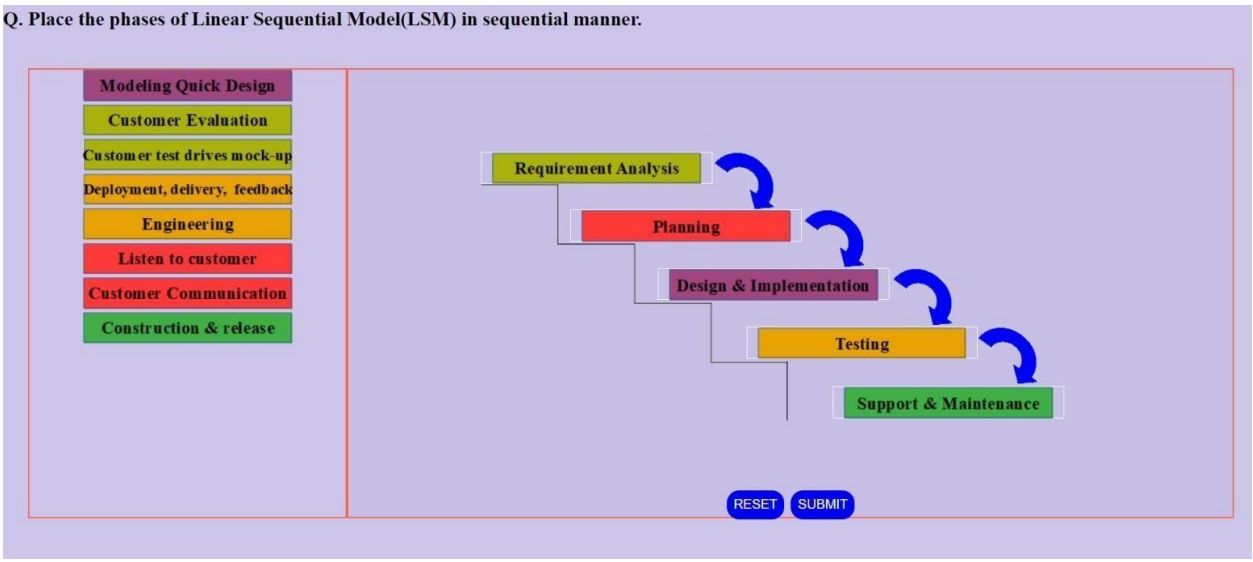


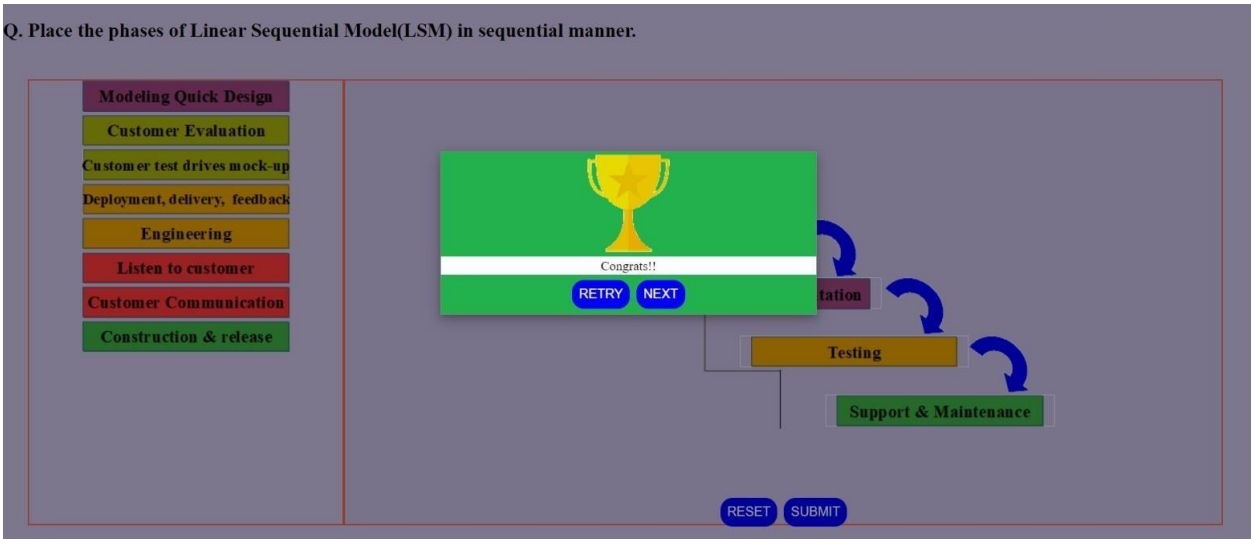


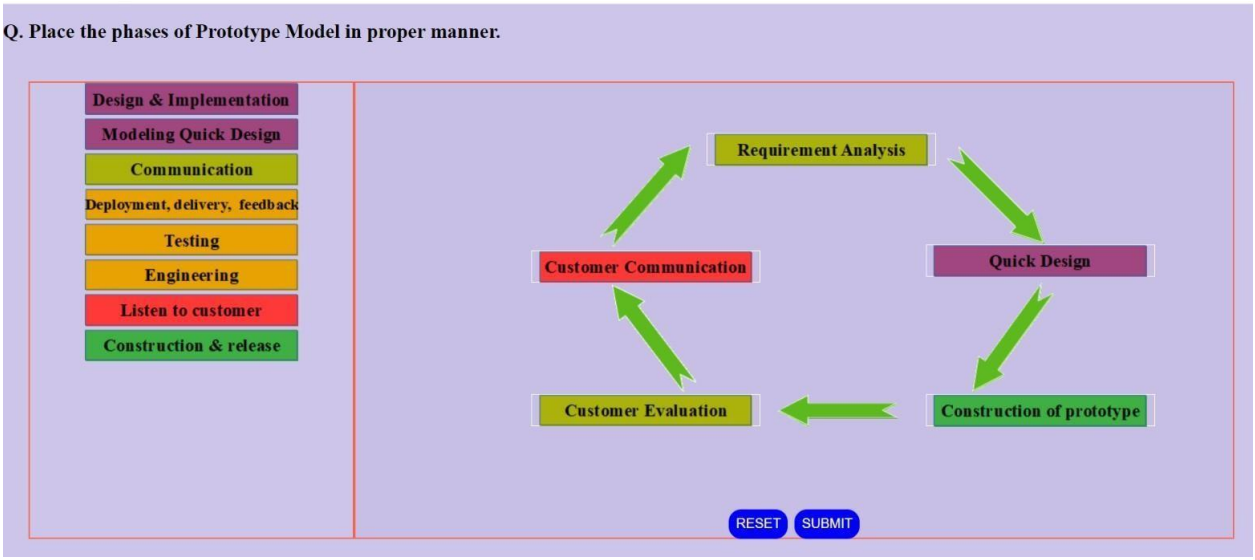

Q. Arrange the phases of software development life cycle in sequential manner.
Requirement Analysis
Planning
Design & Implementation
Testing
Support & Maintenance
RESET
SUBMIT

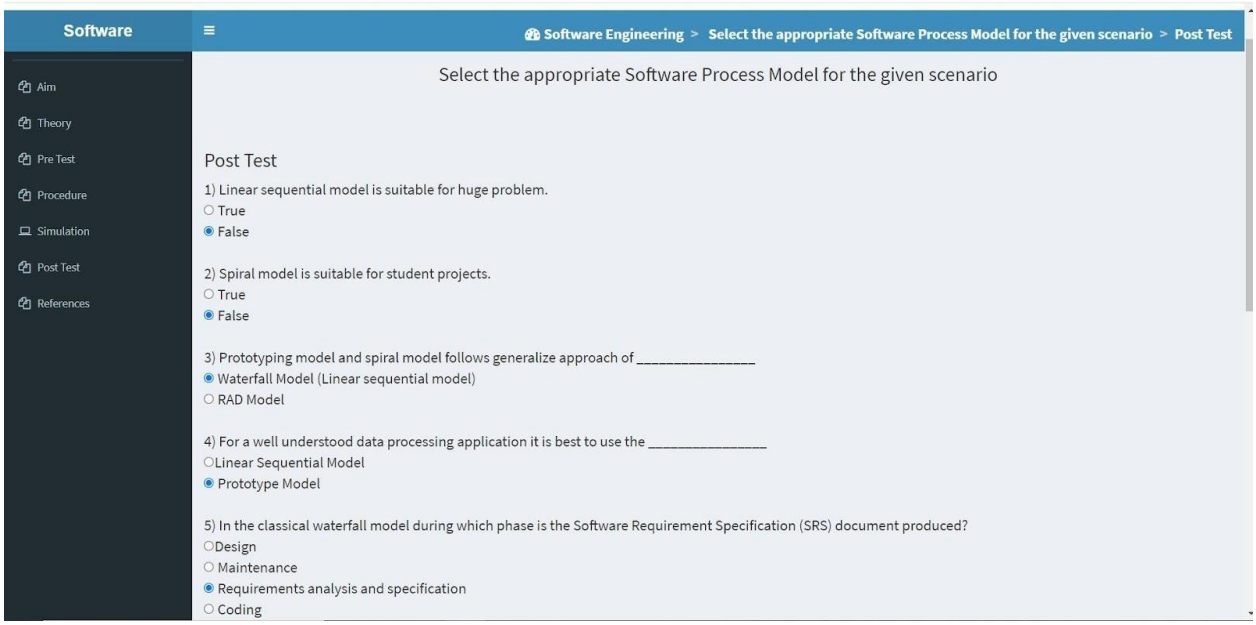

Q. Arrange the phases of software development life cycle in sequential manner.
Congrats!!
RETRY
NEXT
Testing
Support & Maintenance
RESET
SUBMIT

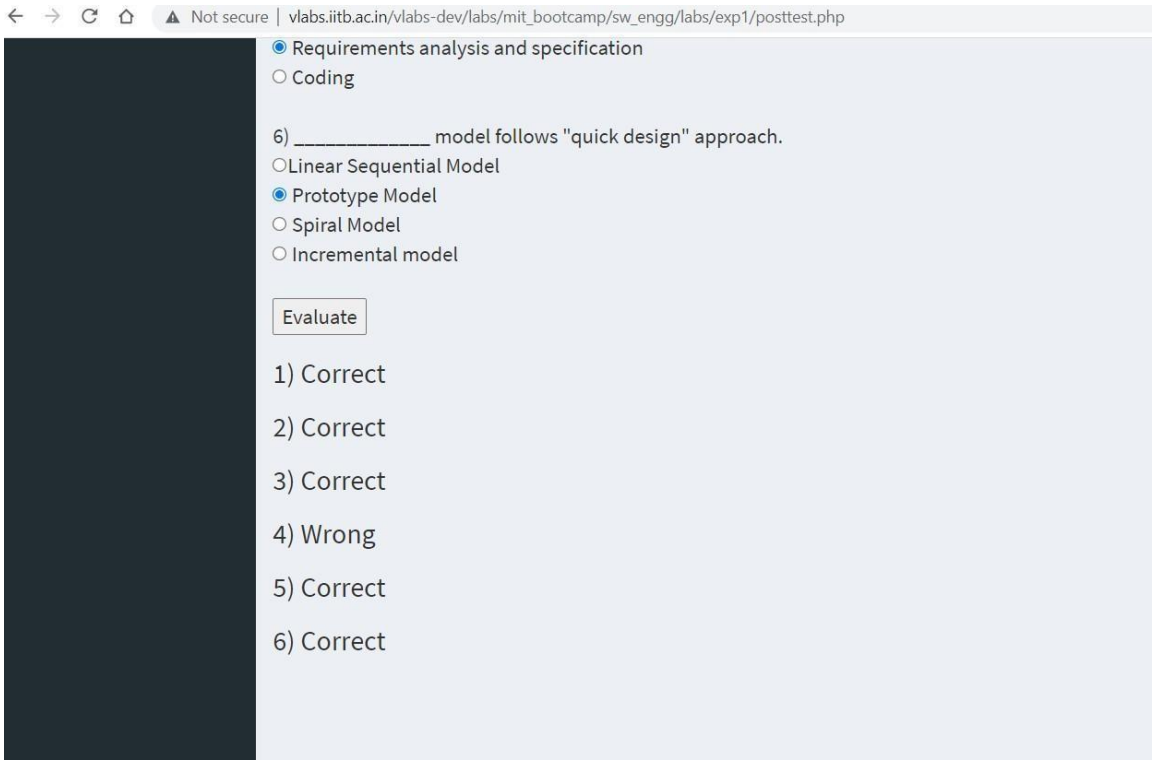

Move Cards from element deck pile to appropriate card pile(User, Client, Developer)
Element Deck Pile
Requirement Fulfilment
Classify the elements of Quality Software in accordance with user, client & developer
User Pile
Client Pile
Developer Pile
Reliability
Requirement Fulfilment
Good Design
User Pile
Client Pile
Developer Pile
Ease of Learning
Reliability
Requirement Fulfilment
Good Design
SHUFFLE
RESET
NEXT


Q. Place the phases of Linear Sequential Model(LSM) in sequential manner.
Modeling Quick Design
Customer Evaluation
Customer test drives mock-up
Deployment, delivery, feedback
Engineering
Listen to customer
Customer Communication
Construction & release
Requirement Analysis
Planning
Design & Implementation
Testing
Support & Maintenance
RESET
SUBMIT


Q. Place the phases of Linear Sequential Model(LSM) in sequential manner.
Modeling Quick Design
Customer Evaluation
Customer test drives mock-up
Deployment, delivery, feedback
Engineering
Listen to customer
Customer Communication
Construction & release
Congrats!!
RETRY
NEXT
Testing
Support & Maintenance
RESET
SUBMIT


Q. Place the phases of Prototype Model in proper manner.
Design & Implementation
Modeling Quick Design
Communication
Deployment, delivery, feedback
Testing
Engineering
Listen to customer
Construction & release
Requirement Analysis
Quick Design
Construction of prototype
Customer Evaluation
Customer Communication
RESET
SUBMIT





Conclusion: In this experiment we student the **(**Linear Sequential model, Prototyping model) and traditional model and done the simulation on the different models.

EXPERIMENT 2

Aim :Application of the Agile process models.

Theory :

Agile process model:

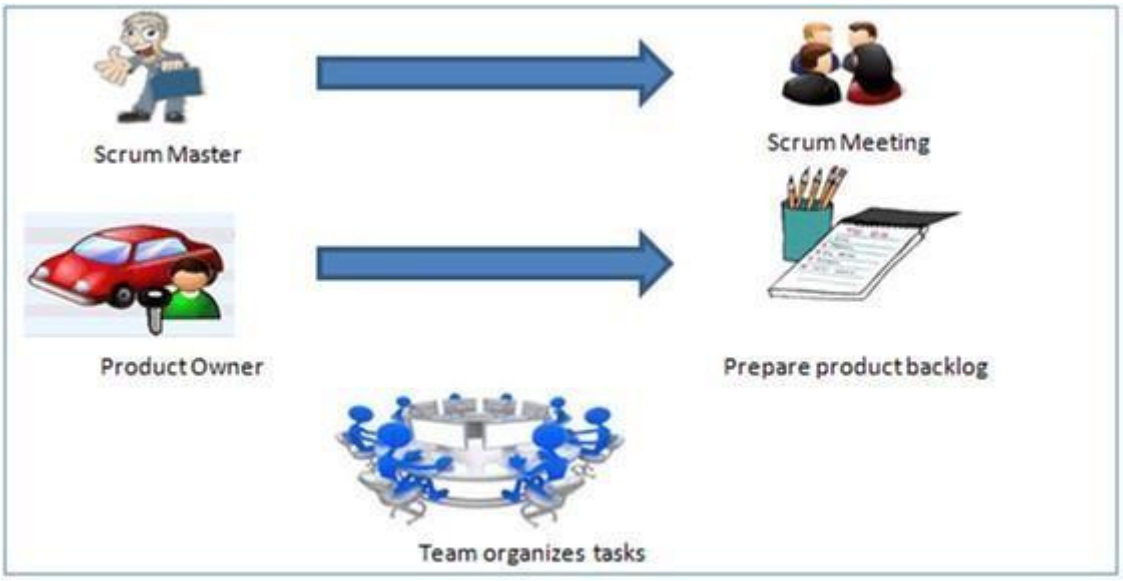
- "Agile process model" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans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

A few Agile SDLC models are given below:

- Crystal
- Atern
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process

1. **Scrum**

SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment. Basically, Scrum is derived from activity that occurs during a rugby match. Scrum believes in empowering the development team and advocates working in small teams. Agile and Scrum consist of three roles, and their responsibilities are explained as follows:

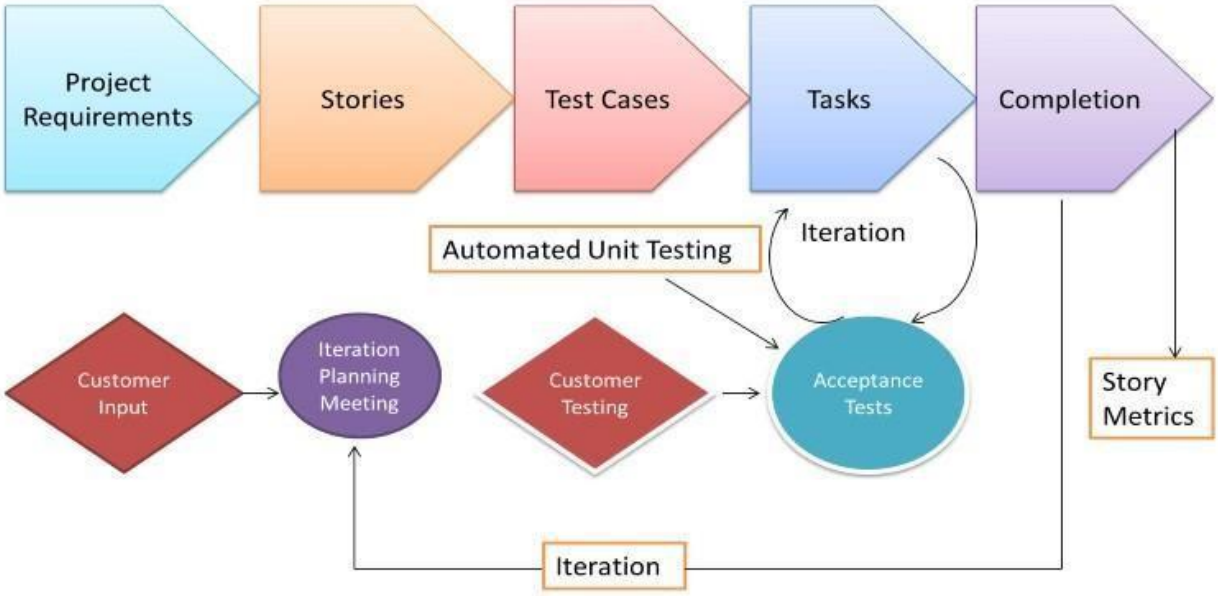


- Scrum Master
    - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- Product owner
    - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- Scrum Team
    - Team manages its own work and organizes the work to complete the sprint or cycle

2. <u>Extreme Programming (XP)</u>

Extreme Programming technique is very helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system. It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and

also introduces a checkpoint where any customer requirements can be easily implemented. The XP develops software keeping customer in the target.



Extreme Programming (XP)

- **Phases of eXtreme programming:**

There are 6 phases available in Agile XP method, and those are explained as follows:

Planning

- Identification of stakeholders and sponsors
- Infrastructure Requirements
- Security related information and gathering
- Service Level Agreements and its conditions

Analysis

- Capturing of Stories in Parking lot
- Prioritize stories in Parking lot
- Scrubbing of stories for estimation
- Define Iteration SPAN(Time)
- Resource planning for both Development and QA teams

Design

- Break down of tasks
- Test Scenario preparation for each task
- Regression Automation Framework

Execution

- Coding
- Unit Testing
- Execution of Manual test scenarios
- Defect Report generation
- Conversion of Manual to Automation regression test cases
- Mid Iteration review
- End of Iteration review

Wrapping

- Small Releases
- Regression Testing
- Demos and reviews
- Develop new stories based on the need
- Process Improvements based on end of iteration review comments

Closure

- Pilot Launch
- Training
- Production Launch
- SLA Guarantee assurance
- Review SOA strategy
- Production Support

3. Crystal Methodologies

Crystal Methodology is based on three concepts

1. Chartering: Various activities involved in this phase are creating a development team, performing a preliminary feasibility analysis, developing an initial plan and fine-tuning the development methodology

2. Cyclic delivery: The main development phase consists of two or more delivery cycles, during which theTeam updates and refines the release plan

3. Implements a subset of the requirements through one or more programtest integrate iterations

4. Integrated product is delivered to real users

5. Review of the project plan and adopted development methodology

6. Wrap Up: The activities performed in this phase are deployment into the user environment, post- deployment reviews and reflections are performed.

4. Dynamic Software Development Method (DSDM)

DSDM is a Rapid Application Development (RAD) approach to software development and provides an agile project delivery framework. The important aspect of DSDM is that the users are required to be involved actively, and the teams are given the power to make decisions. Frequent delivery of product becomes the active focus with DSDM. The techniques used in DSDM are

1. Time Boxing
2. MoSCoW Rules
3. Prototyping

The DSDM project consists of 7 phases

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration

6. Implementation
7. Post-project

5. Feature Driven Development (FDD)

This method is focused around "designing & building" features. Unlike other Agile methods in software engineering, FDD describes very specific and short phases of work that has to be accomplished separately per feature. It includes domain walkthrough, design inspection, promote to build, code inspection and design.
FDD develops product keeping following things in the target

1. Domain object Modeling
2. Development by feature
3. Component/ Class Ownership
4. Feature Teams
5. Inspections
6. Configuration Management
7. Regular Builds
8. Visibility of progress and results

6. Lean Software Development

Lean software development method is based on the principle "Just in time production". It aims at increasing speed of software development and decreasing cost. Lean development can be summarized in seven steps.

1. Eliminating Waste

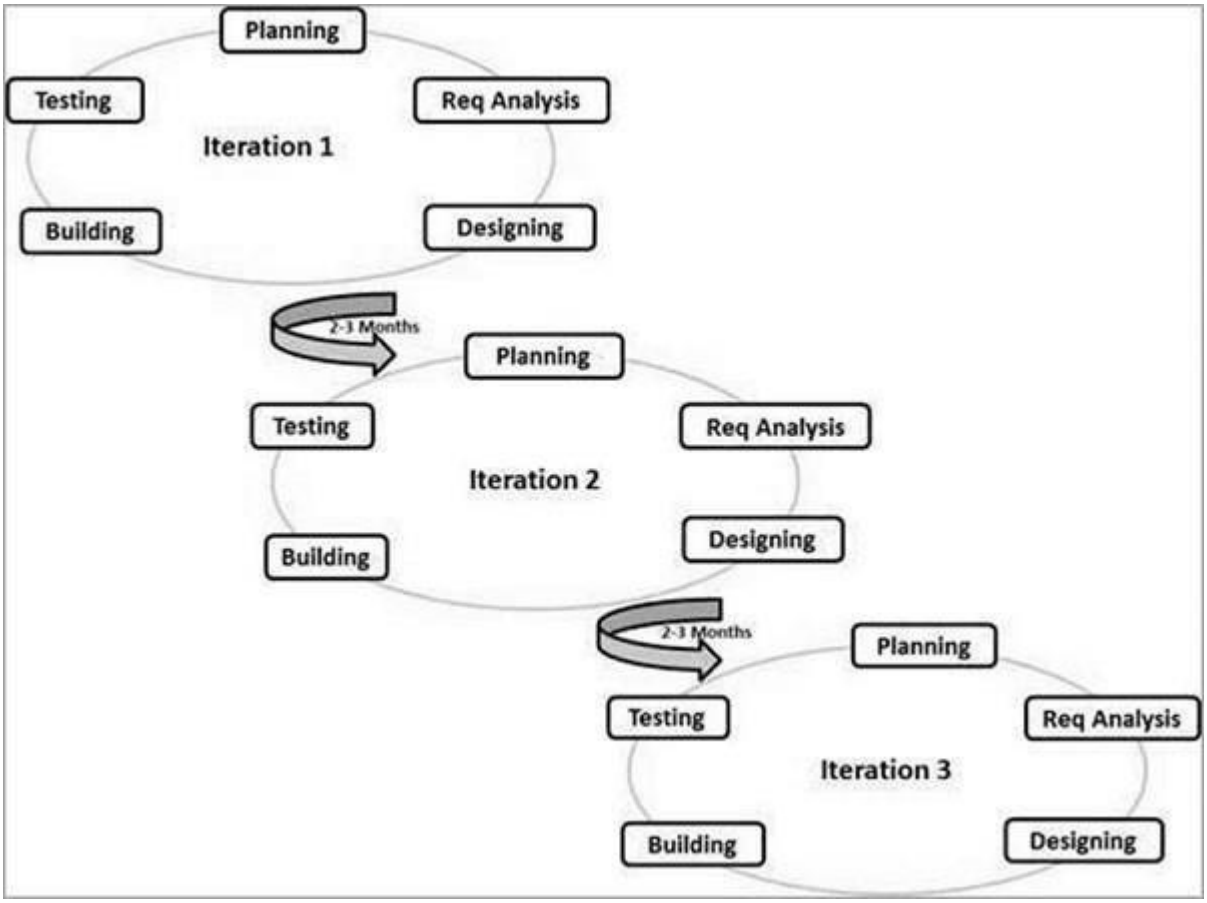
2. Amplifying learning
3. Defer commitment (deciding as late as possible)
4. Early delivery
5. Empowering the team
6. Building Integrity
7. Optimize the whole

7. Kanban

Kanban originally emerged from a Japanese word that means, a card containing all the information needed to be done on the product at each stage along its path to completion. This framework or method is quite adopted in software testing method especially in Agile concepts.

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

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



The advantages of the Agile Model are as follows −

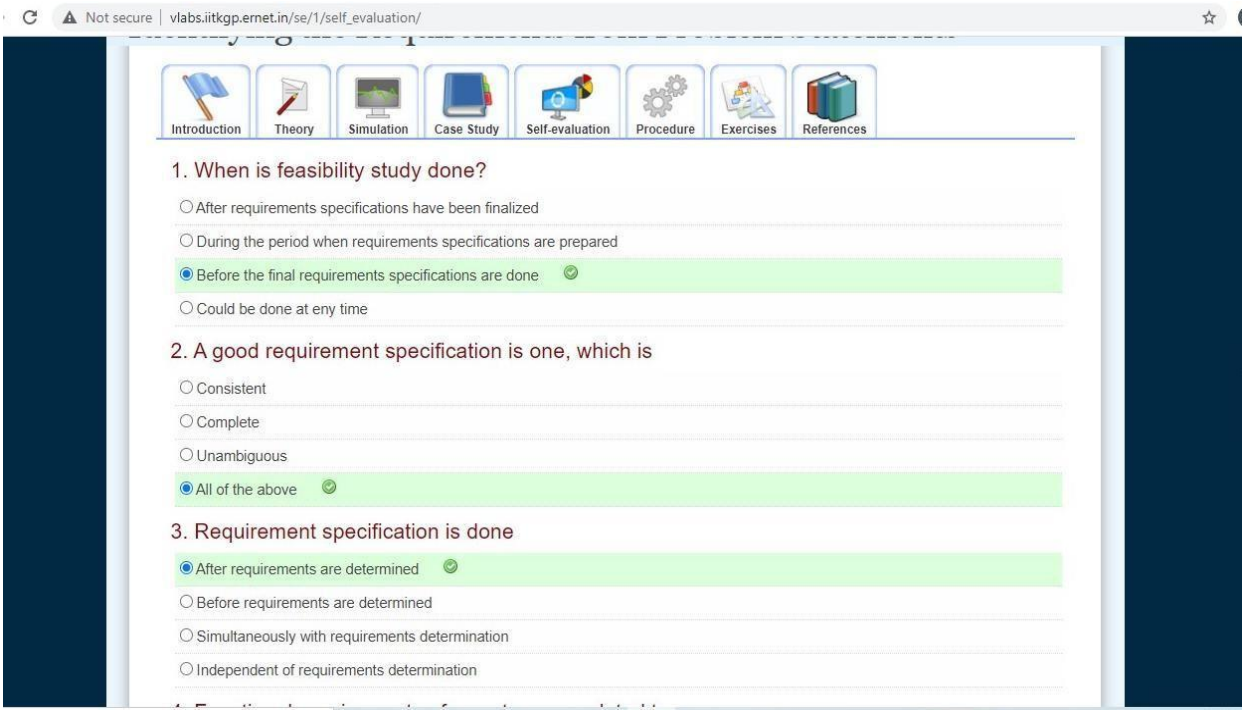
- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum. ● Suitable for fixed or changing requirements ● Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.

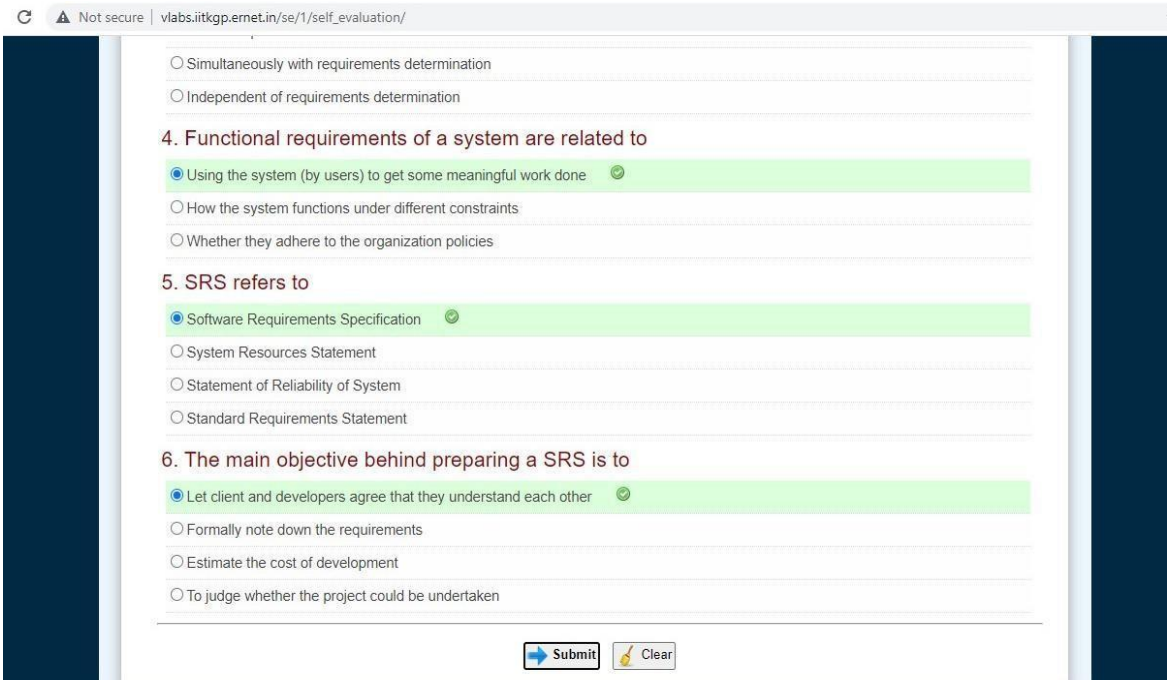
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

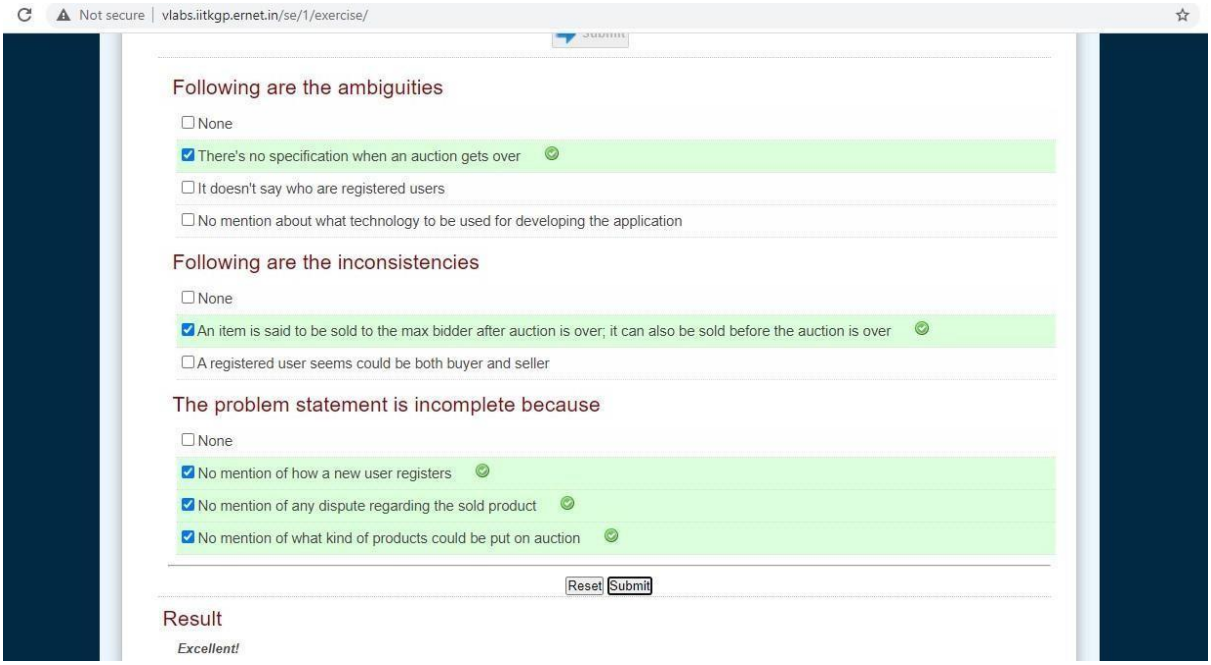
The disadvantages of the Agile Model are as follows −

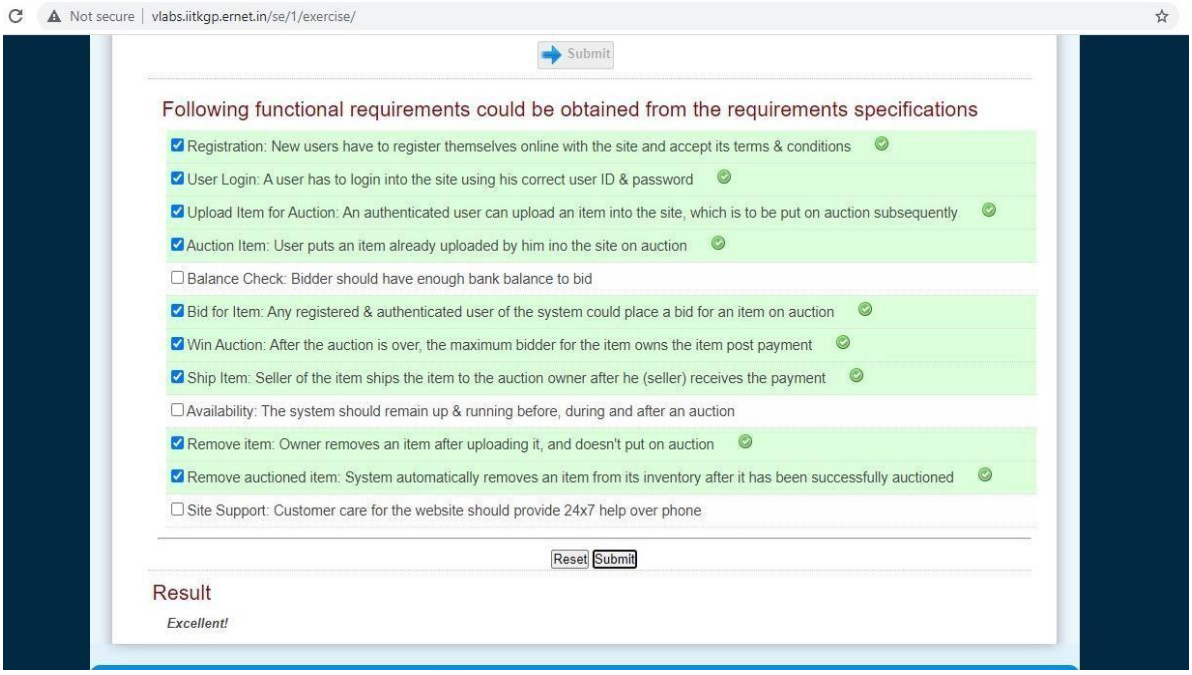
- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

APPLICATION (PANDEMIC MANAGEMENT SYSTEM) :The world is generally not prepared for severe pandemic and disaster. The pandemic management system would help to improve the pandemic preparedness. The main objectives of the pandemic management system is to create an integrated management system which is user friendly in use and to give direct access to emergency services like ambulance, blood banks and isolation centers.


Not secure | vlabs.iitkgp.ernet.in/se/1/self_evaluation/
Introduction
Theory
Simulation
Case Study
Self-evaluation
Procedure
Exercises
References
1. When is feasibility study done?
After requirements specifications have been finalized
During the period when requirements specifications are prepared
Before the final requirements specifications are done
Could be done at eny time
2. A good requirement specification is one, which is
Consistent
Complete
Unambiguous
All of the above
3. Requirement specification is done
After requirements are determined
Before requirements are determined
Simultaneously with requirements determination
Independent of requirements determination


Not secure | vlabs.iitkgp.ernet.in/se/1/self_evaluation/
Simultaneously with requirements determination
Independent of requirements determination
4. Functional requirements of a system are related to
Using the system (by users) to get some meaningful work done
How the system functions under different constraints
Whether they adhere to the organization policies
5. SRS refers to
Software Requirements Specification
System Resources Statement
Statement of Reliability of System
Standard Requirements Statement
6. The main objective behind preparing a SRS is to
Let client and developers agree that they understand each other
Formally note down the requirements
Estimate the cost of development
To judge whether the project could be undertaken
Submit
Clear





Conclusion : Thus, we have an understanding of agile process models.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
(Permanently Affiliated to University of Mumbai)
DEPARTMENT OF COMPUTER ENGINEERING

EXPERIMENT 3

Aim :Preparation of software requirement specification.(SRS)
IEEE format

Software Requirements Specification for

(Example CASE Study System)

Version 1.0 approved

Prepared by

Organisation

Rajiv Gandhi Institute of Technology

Date:

Table of Contents
Revision History

1. Introduction

1. Purpose

(Human societies and civilizations appear to have difficulty in mobilizing a collective will to avoid the high-impact threats such as pandemics. So, according this project argues on pandemic preparedness and response, primarily focused on the health sector, while necessary, is not sufficient to prevent and manage multiple risks to global health and human security. Instead, this paper proposes a systematic approach for an integrated risk-management system to prevent and mitigate complex socio-economic risks and impacts of severe pandemics)

2. Document Conventions

This document uses the following Convention. DB – Database HD-Hospital details

3. Intended Audience and Reading Suggestions.

This Project is a prototype for the pandemic Management System and it is restricted to college premises. This has been implemented under the guidance of college professors. This project is useful for student, teachers as well as external exam cell portals.

4. Product Scope

The scope of the project is the system on which the software is installed, i.e. the project is developed as a third party general-purpose software, and it will work for a particular institute which can efficiently generate optimal solutions.

5. References

1Salvatore Ammirato, Roberto Linzalone&Alberto M. Felicetti"Knowledge management in pandemics. A critical literature review",2020[Online]. Available :https://www.tandfonline.com/doi/full/10.1080/14778238.2020.1801364

2. Michele Augusto Riva, Marta Benedetti & Giancarlo Cesana"Pandemic Fear and Literature",2017[Online]. Available :https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4193163/

3. Roubini, N. Coronavirus pandemic has delivered the fastest, deepest economic shock in history. The Guardian, 25 March 2020.

4. Donthu, N.; Gustafsson, A. Effects of COVID-19 on business and research. J. Bus. Res. 2020, 117, 284–289. [CrossRef] [PubMed]

5. Painter, M.; Qiu, T. Political Beliefs affect Compliance with COVID-19 social Distancing Orders.SSRN Electron. J. 2020. [CrossRef] Koren, M.; Pet˝o, R. Business disruptions from social distancing. PLoS ONE 2020, 15, e0239113.

6. Ozili, P.; Arun, T. Spillover of COVID-19: Impact on the Global Economy; University Library of Munich: Munich, Germany, 2020.

7. Gössling, S.; Scott, D.; Hall, C.M. Pandemics, tourism and global change: A rapid assessment of COVID-19. J. Sustain. Tour. 2020, 29, 1–20. [CrossRef]

8. Gray, R.S. Agriculture, transportation, and the COVID-19 crisis. Can. J. Agric. Econ. Can.d'agroeconomie 2020, 68, 239–243. [CrossRef]

9. Ratten, V. Coronavirus disease (COVID-19) and sport entrepreneurship. Int. J. Entrep. Behav. Res. 2020, 26, 1379–1388. [CrossRef]

10. Hang, H.; Aroean, L.; Chen, Z. Building emotional attachment during COVID19. Ann. Tour. Res. 2020, 83, 103006. [CrossRef] [PubMed]

2. Overall Description

1. Product Perspective
2. Product Functions
3. User Classes Diagram

4. Operating Environment

3. External Interface Requirements

1. User interface

2. Hardware Interface

3. Software Interface

4. Communication Interface

4. System features

1. System features 1

5. Other Non-functional Requirements

6. Other Requirements None.

No other additional requirements are needed apart from the requirements specified in SRS.

Appendix A: Glossary

PMS pandemic management system

Databaseis the collection of information altogether stored in structured format from where the data (information) about the respective entities can be extracted.

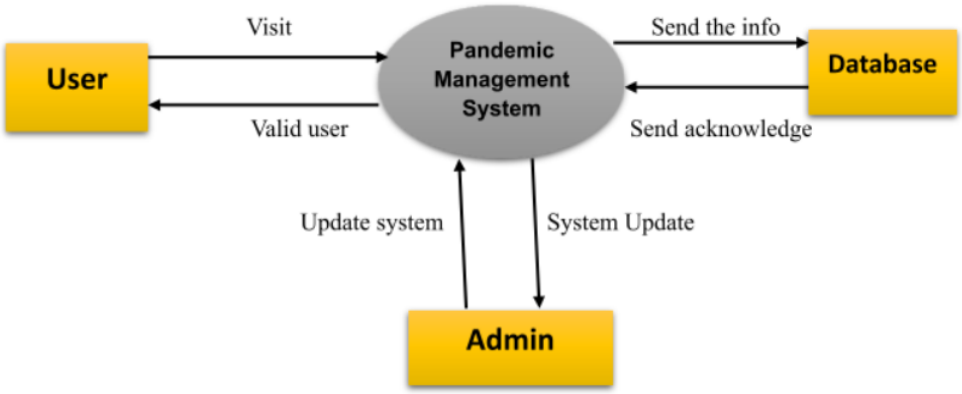
DFDstands for Data Flow Diagram, comes under the category of UML diagrams which defines the processes flow at different levels.

UML stands for Unified Modelling Language is a general-purpose modelling language used widely in the field of software engineering through which the design of a system can be under- stood thoroughly.

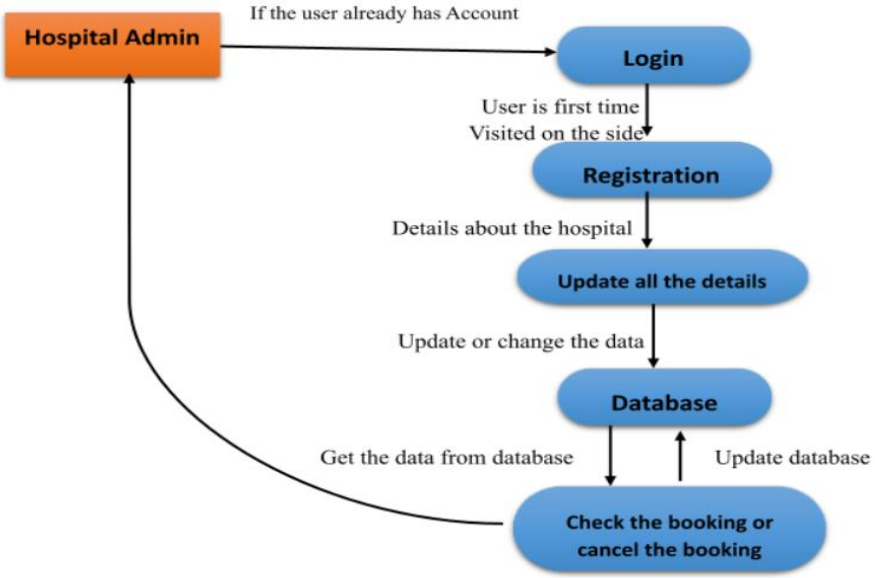
Use Caseis a Diagram which defines the interactions between external entities and the system.

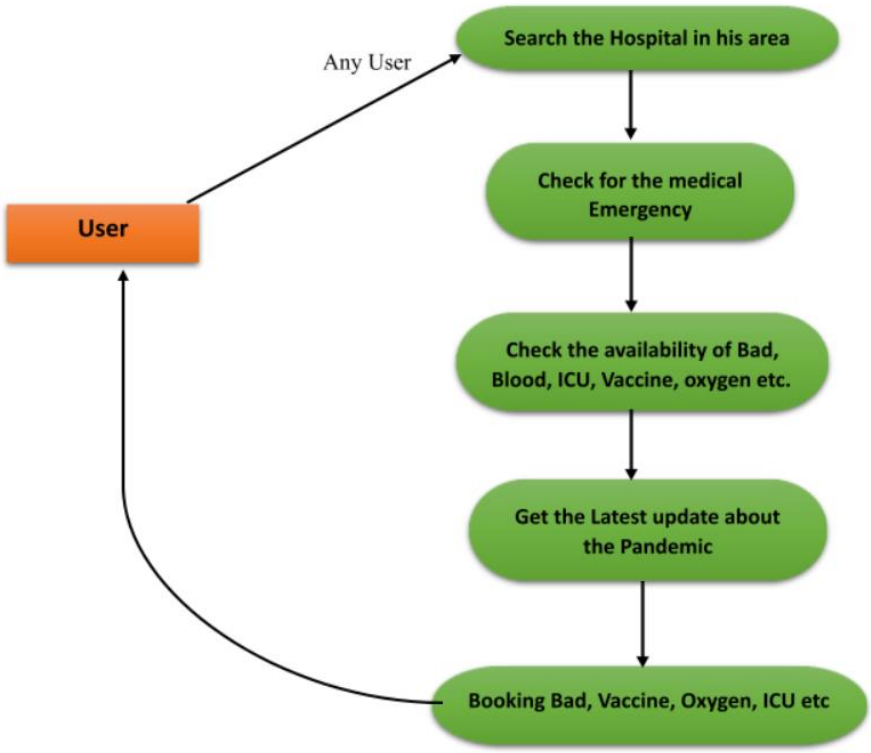
DATA FLOW DIAGRAM

DFD Level 0



DFD Level 1





Appendix C: To Be Determined List

1. J. Ren and X. Gao, "Situation assessment model for uav disaster relief in the city," in 2011 International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping, 2011, pp.1–6.

2. A. Brighente, F. Formaggio, G. M. Di Nunzio, and S. Tomasin "Machine learning for in-region location verification in wireless networks," IEEE Journal on Selected Areas in Communications, vol. 37, no. 11, pp. 2490–2502, 2019.

2. J. Akshya and P. L. K. Priyadarsini, "A hybrid machine learning approach for classifying aerial images of flood-hit areas," in 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), 2019, pp. 1–5.

4. K. Mori, T. Wada, and K. Ohtsuki, "A new disaster recognition algorithm based on svm for eress: Buffering and bagging-svm," in 2016 45th International Conference on Parallel Processing Workshops (ICPPW), 2016, pp. 22–30.

Experiment No.4

Aim: - Structured Dataflow Analysis Theory:

Requirements:

Hardware requirements

Software requirements

What is Structured Analysis?

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
- Pseudocode

Data Flow Diagrams (DFD)

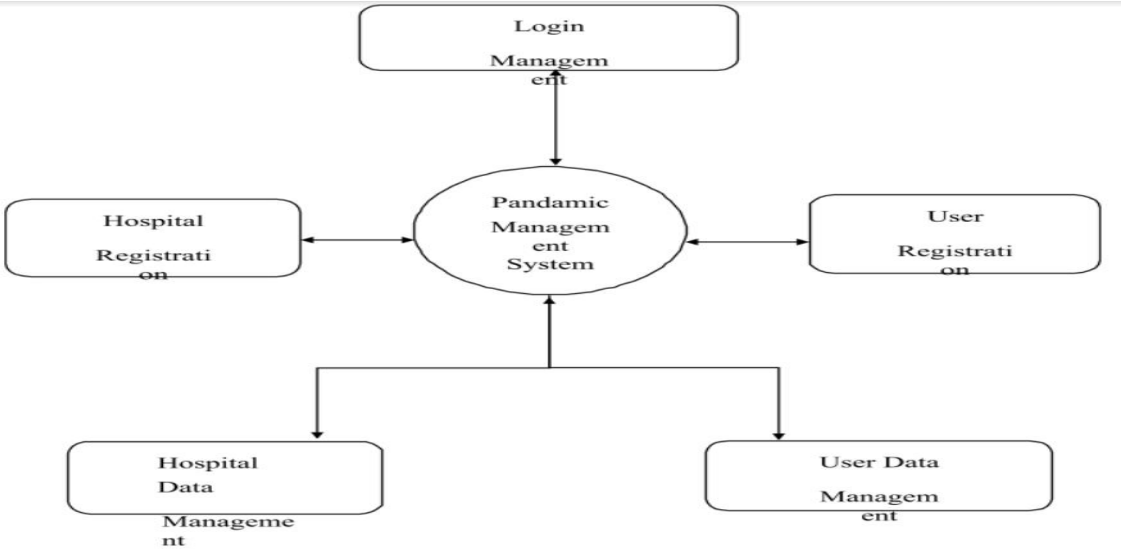
It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

- 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.

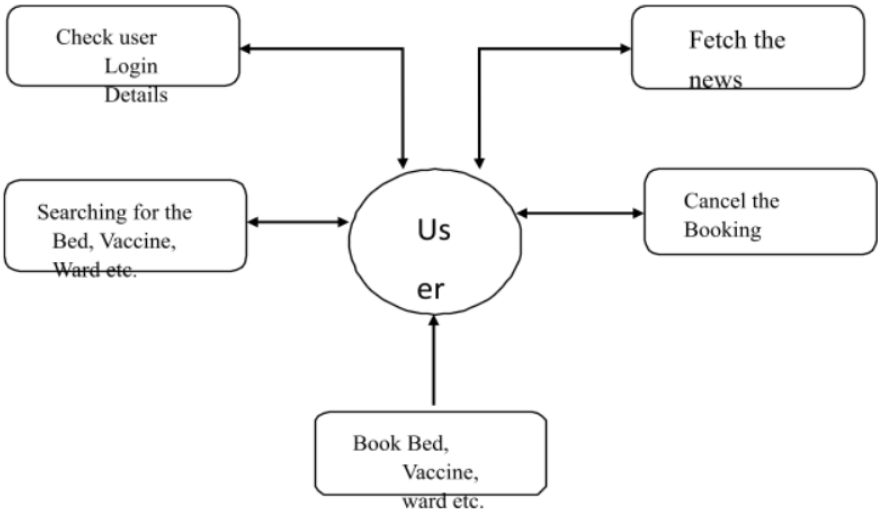
Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

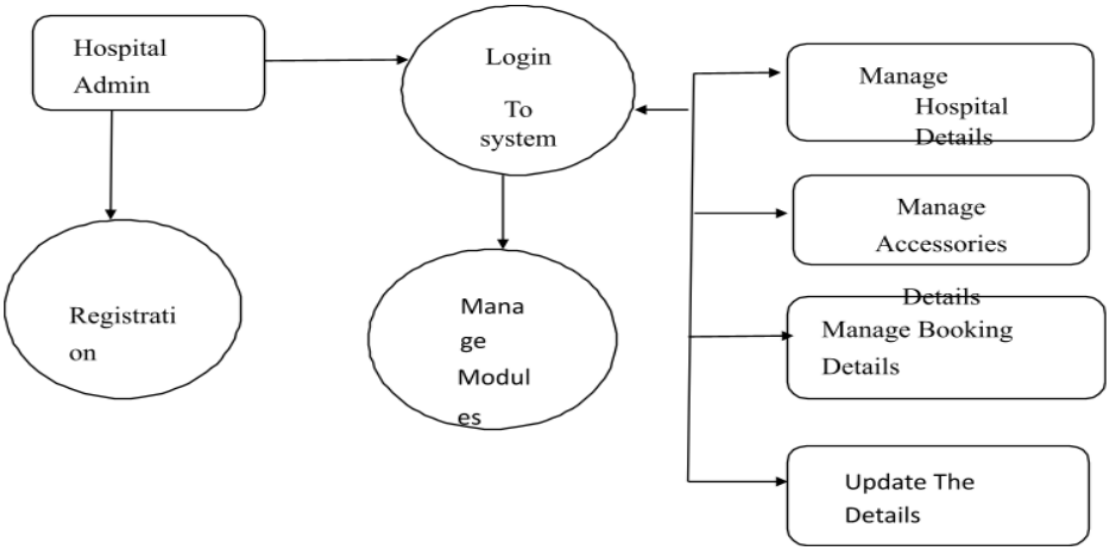
The following table shows the symbols used in designing a DFD and their significance



Zero Level DFD – Pandemic Management System



First Level DFD – Pandemic Management System



Second Level DFD Pandemic Management System

Conclusion: Thus, we have an understood the Structured flow analysis. It requires basic understanding to implement the DFD. Hence, we implemented successfully.

Experiment No.5

Aim: - Use of Metrics to estimate the cost.

Theory: One of the major activities of the project planning phase, therefore, is to estimate various project parameters in order to make proper decisions. Some important project parameters that are estimated include:

1. Project size
2. Cost
3. Duration
4. Effort

COCOMO: Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semi detached
3. Embedded

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate model
3. Detailed Model

1. **Basic COCOMO Model**: The The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

Where

Effort=a_1*(KLOC) a_2 PM Tdev=b_1*(efforts)b_2 Months

KLOC is the estimated size of the software product indicate in Kilo Lines of Code, a1,a2,b1,b2 are constants for each group of software products,
Tdevis the estimated time to develop the software, expressed in months,

- The value of the constants a, b, c are given below:

Software project	*a*	*b*	*c*
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

2. Intermediate COCOMO Model

The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering

Intermediate COCOMO equation:

$$E=a_i (KLOC)\ b_i * EAF$$

$$D=c_i (E)^{d_i}$$

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. **Detailed COCOMO Model:** Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver's effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then
sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

Halstead's Software Metrics

According to Halstead's "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either

operators or operands." In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

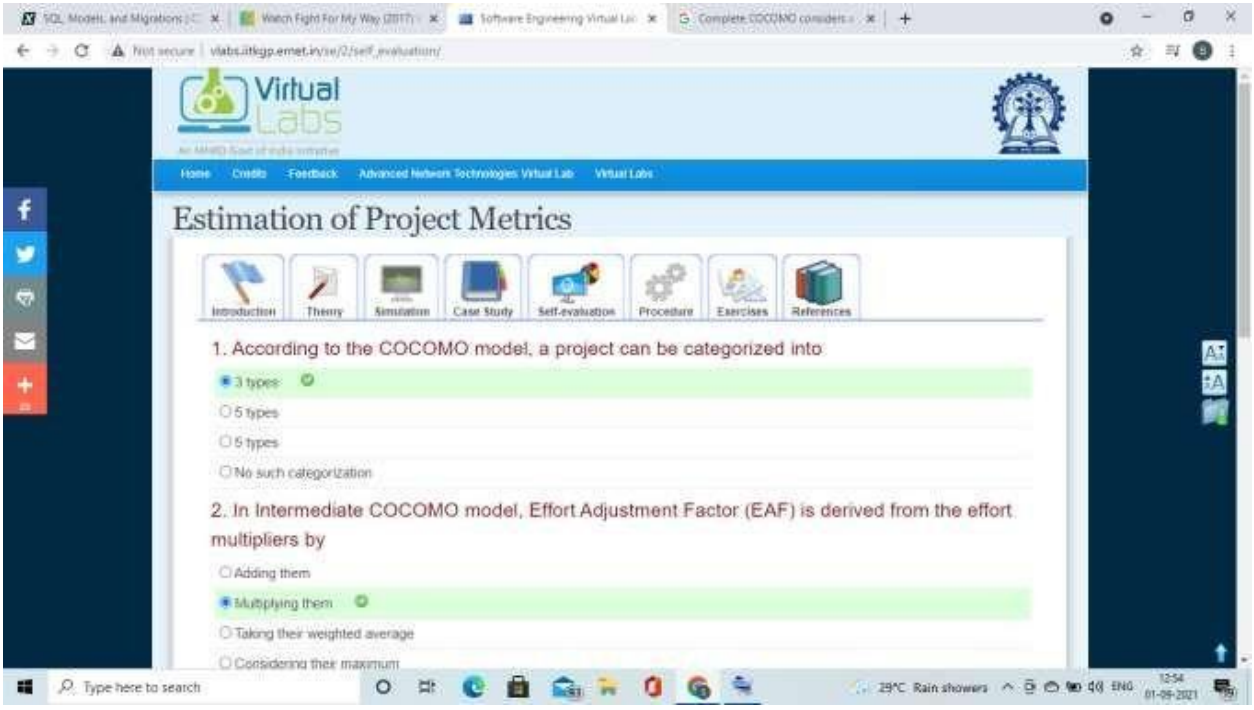
The basic measures are

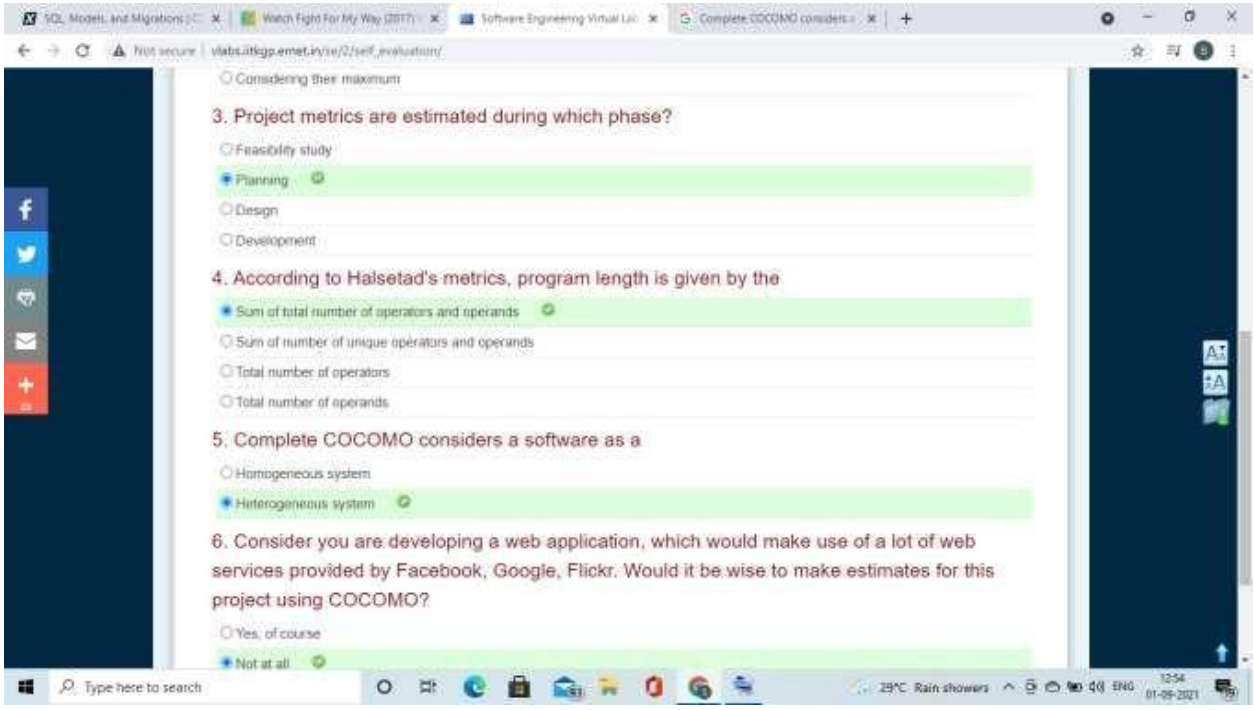
n1 = count of unique
operators. n2 = count of
unique operands.
N1 = count of total occurrences of operators.

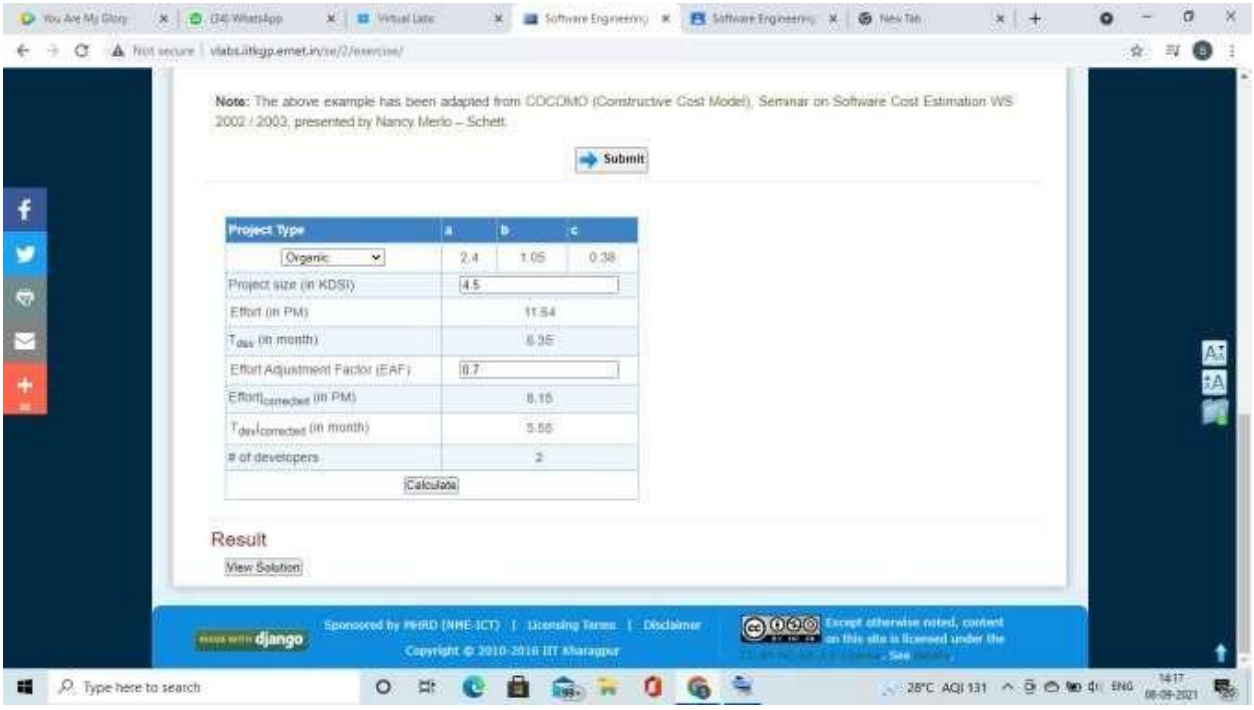
N2 = count of total occurrence of operands

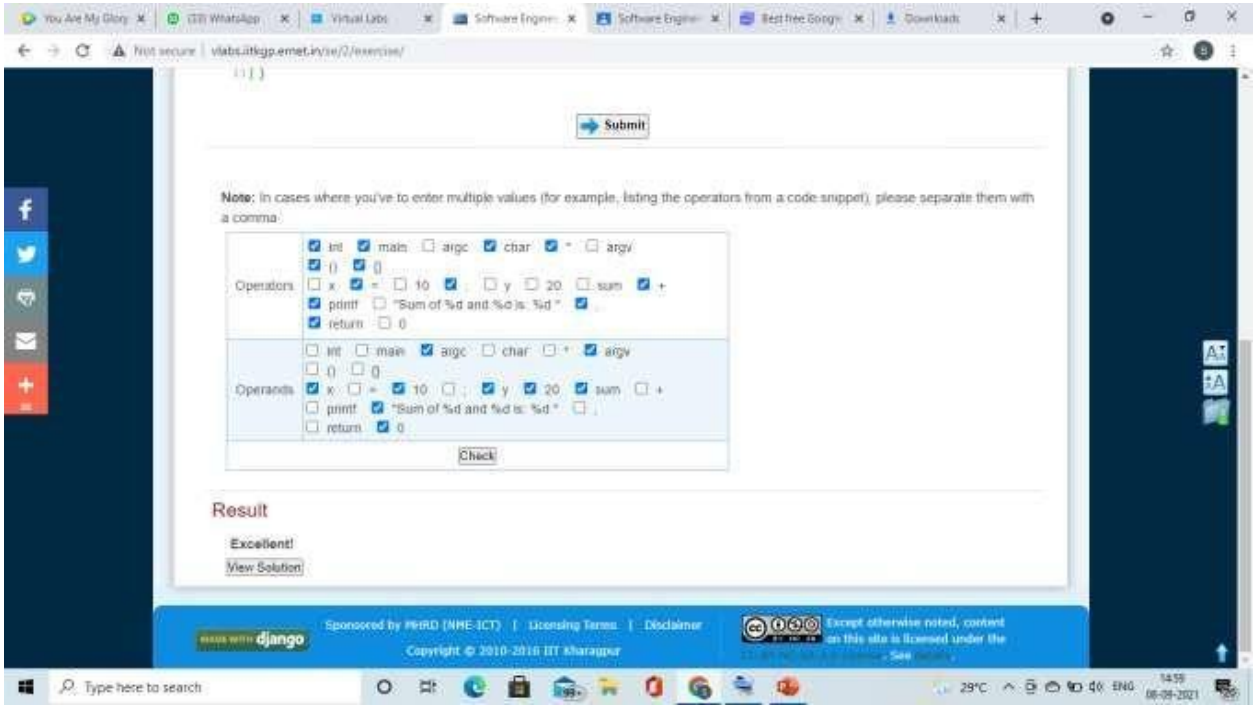
Using the above parameters, one computes

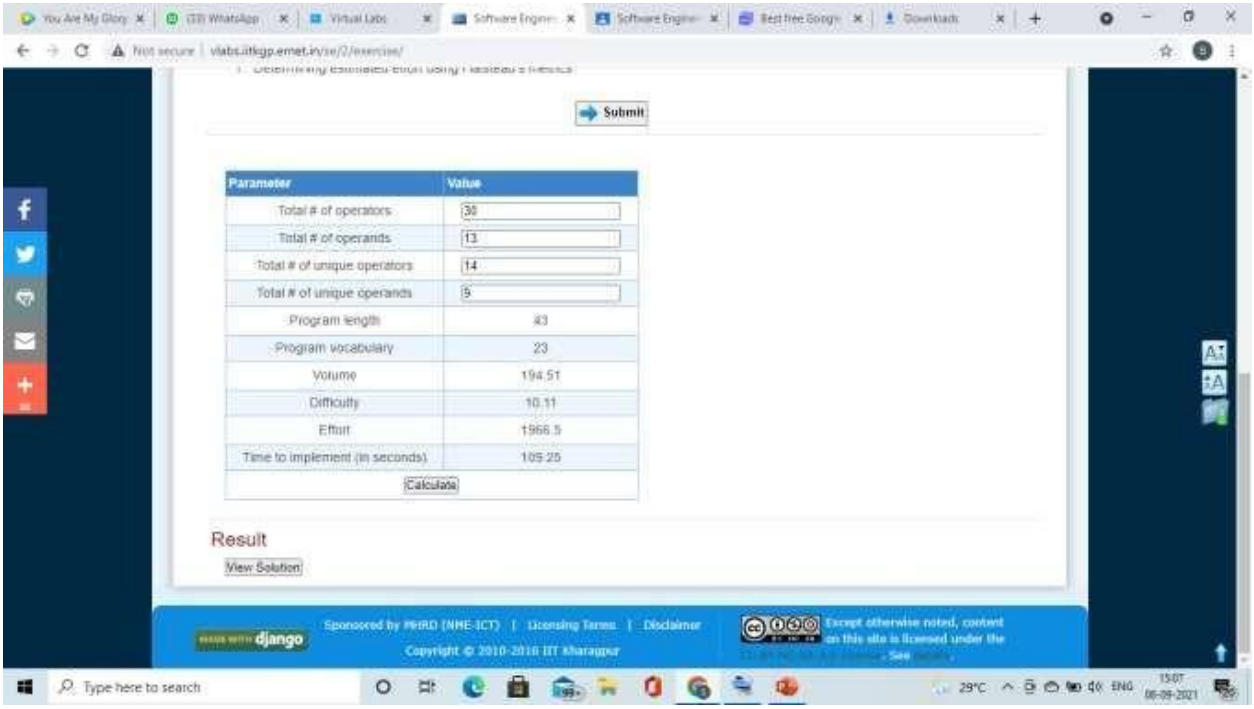
1. Program length : N=N1+N2
2. Program Vocabulary: n=n1+n2
3. Volume: V=N*log n
4. Difficulty: D=(n1*N2)/(2*n2)
5. Effort: E=D*V
6. Time to implement: T=E/18


Virtual Labs
Home
Credits
Feedback
Advanced Network Technologies Virtual Lab
Virtual Labs
Estimation of Project Metrics
Introduction
Theory
Simulation
Case Study
Self-evaluation
Procedure
Exercises
References
1. According to the COCOMO model, a project can be categorized into
3 types
5 types
5 types
No such categorization
2. In Intermediate COCOMO model, Effort Adjustment Factor (EAF) is derived from the effort multipliers by
Adding them
Multiplying them
Taking their weighted average
Considering their maximum


Considering their maximum
3. Project metrics are estimated during which phase?
Feasibility study
Planning
Design
Development
4. According to Halsetad's metrics, program length is given by the
Sum of total number of operators and operands
Sum of number of unique operators and operands
Total number of operators
Total number of operands
5. Complete COCOMO considers a software as a
Homogeneous system
Heterogeneous system
6. Consider you are developing a web application, which would make use of a lot of web services provided by Facebook, Google, Flickr. Would it be wise to make estimates for this project using COCOMO?
Yes, of course
Not at all


Note: The above example has been adapted from COCOMO (Constructive Cost Model), Seminar on Software Cost Estimation WS 2002 / 2003, presented by Nancy Merlo – Schett
Submit
Project Type
a
b
c
Organic
2.4
1.05
0.38
Project size (in KDSI)
4.5
Effort (in PM)
11.54
Tdev (in month)
6.35
Effort Adjustment Factor (EAF)
0.7
Effortcorrected (in PM)
8.15
Tdevcorrected (in month)
5.55
# of developers
2
Calculate
Result
View Solution
Sponsored by MHRD (NME-ICT) | Licensing Terms | Disclaimer
Copyright © 2010-2016 IIT Kharagpur
Except otherwise noted, content on this site is licensed under the


Submit
Note: In cases where you've to enter multiple values (for example, listing the operators from a code snippet), please separate them with a comma
Operators
Operands
Check
Result
Excellent!
View Solution
Sponsored by MHRD (NME-ICT) | Licensing Terms | Disclaimer
Copyright © 2010-2016 IIT Kharagpur
Except otherwise noted, content on this site is licensed under the



CONCLUSION: Thus, we have estimated the cost using metrics.

Experiment No.6

Aim: -Scheduling and Tracking of the Project.

Theory: SCHEDULING

Basic Principles for Project Scheduling:

- **Compartmentalization**
  The project must be compartmentalized into a number of manageable activities, actions, and tasks; both the product and the process are decomposed.

- **Interdependency**
  The interdependency of each compartmentalized activity, action, or task must be determined Some tasks must occur in sequence while others can occur in parallel Some actions or activities cannot commence until the work product produced by another is available

- **Time Allocation**
  Each task to be scheduled must be allocated some number of work units In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies Start and stop dates are also established based on whether work will be conducted on a fulltime or part-time basis.

- **Effort Validation**
  Every project has a defined number of people on the team As time allocation occurs, the project manager must ensure that no more than the allocated number of people have been scheduled at any given time.

- **Defined responsibilities**
  Every task that is scheduled should be assigned to a specific team member.

- **Defined outcomes**
  Every task that is scheduled should have a defined outcomes for software projects such as a work product or part of work product work product are often combined in deliverables.

- **Defined milestones**

Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

Effort Applied vs. Delivery Time

- There is a nonlinear relationship between effort applied and delivery time (Ref: PutnamNorden-Rayleigh Curve) effort increase rapidly time is reduced.
- Also, delaying project delivery can reduce costs significantly as shown in the equation $E = L^3/(P^3t^4)$ and in the curve below.

E = development effort in person-months

L = source lines of code delivered

P = productivity parameter (ranging from 2000 to 12000) t = project duration in calendar months

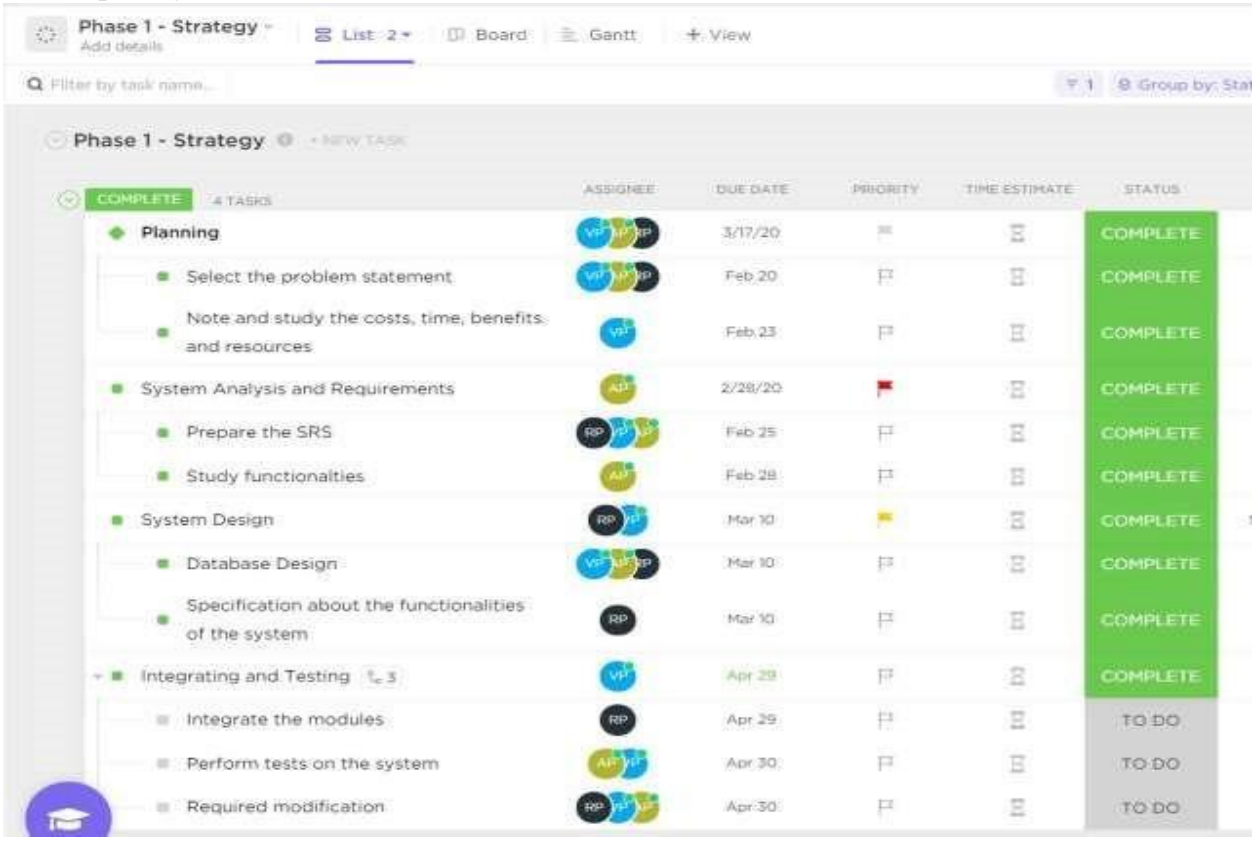
40-20-40 Distribution of Effort

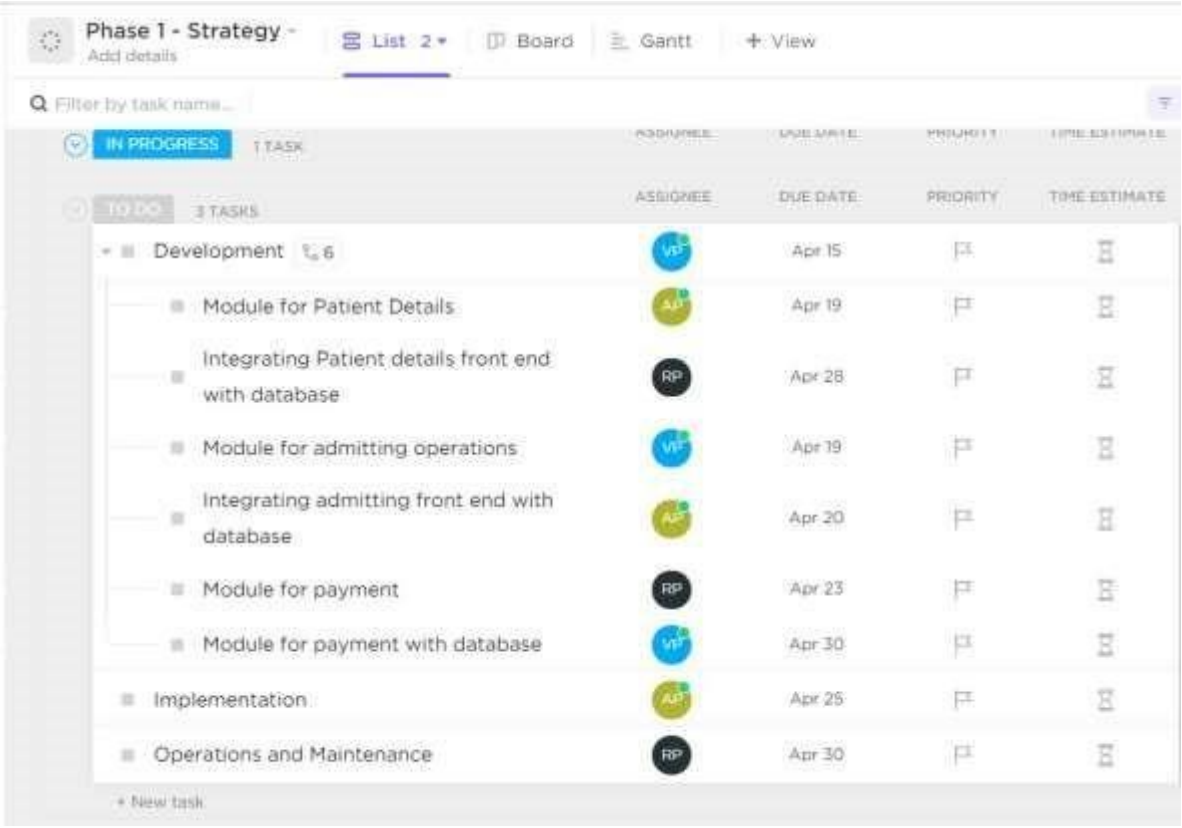
- A recommended distribution of effort across the software process is 40% (analysis and design), 20% (coding), and 40% (testing)
- Work expended on project planning rarely accounts for more than 2 - 3% of the total effort
- Requirement's analysis may comprise 10 - 25% Effort spent on prototyping and project complexity may increase this
- Software design normally needs 20 – 25%
- Coding should need only 15 - 20% based on the effort applied to software design
- Testing and subsequent debugging can account for 30 - 40% Safety or security-related software requires more time for testing
- Example: 100-day project

TASKS

Defining a Task Set

- A task set is the work breakdown structure for the project
- No single task set is appropriate for all projects and process models. It varies depending on the project type and the degree of rigor (based on influential factors) with which the team plans to work
- The task set should provide enough discipline to achieve high software quality.





Purpose of a Task Network

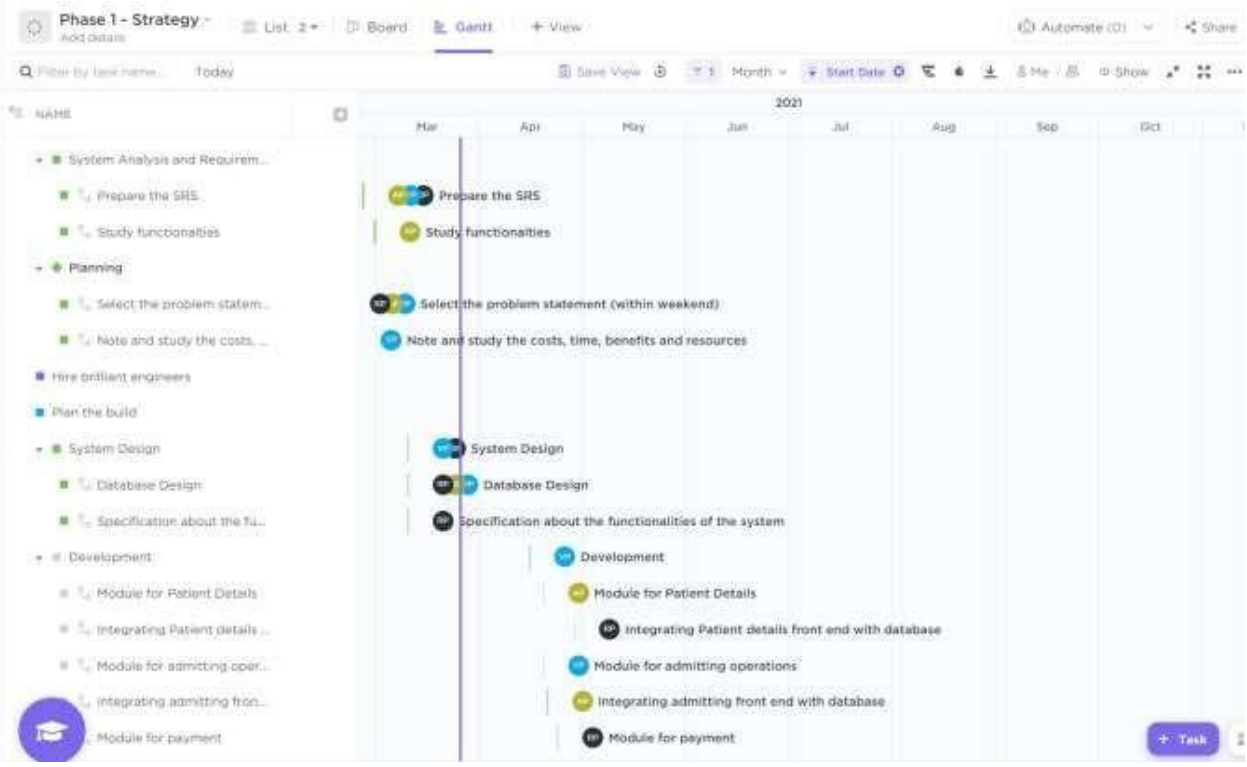
- Also called an activity network.
- It is a graphic representation of the task flow for a project.
- It depicts task length, sequence, concurrency and dependency.
- Points out-liner dependencies to help the manager ensure continuous progress toward project completion.
- The critical path.

A single path leading from start to finish in a task network it contains the sequence of tasks that must be completed on schedule it also determines the minimum duration of the project.

TIMELINE CHART

Mechanics of a timeline chart

- Also called a Gantt chart; invented by Henry Gantt, industrial engineer, 1917
- All project tasks are listed in the far-left column
- The next few columns may list the following for each task: projected start date, projected stop date, projected duration, actual start date, actual stop date, actual duration, task inter-dependencies (i.e., predecessors)
- To the far right are columns representing dates on a calendar
- The length of a horizontal bar on the calendar indicates the duration of the task
- When multiple bars occur at the same time interval on the calendar, this implies task concurrency
- A diamond in the calendar area of a specific task indicates that the task is a milestone; a milestone has a time duration of zero



Methods for Tracking the Schedule Qualitative approaches

Conduct periodic project status meetings in which each team member reports progress and problems Evaluate the results of all reviews conducted throughout the software engineering process Determine whether formal project milestones (i.e., diamonds) have been accomplished by the scheduled date Compare actual start date to planned start date for each project task listed in the timeline chart meet informally with the software engineering team to obtain their subjective assessment of progress to date and problems on the horizon

- Quantitative approach

Use earned value analysis to assess progress quantitatively

Conclusion: Thus, project management tool was used to schedule a project plan.

Experiment No.7

Aim: -Write test cases for black box testing.

Theory:

Black Box Testing

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.

The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Software Testing

Testing software is an important part of the development life cycle of a software. It is an expensive activity. Hence, appropriate testing methods are necessary for ensuring the reliability of a program. According to the ANSI/IEEE 1059 standard, the definition of testing is the process of analyzing a software item, to detect the differences between existing and required conditions i.e. defects/errors/bugs and to evaluate the features of the software item.

The purpose of testing is to verify and validate a software and to find the defects present in a software. The purpose of finding those problems is to get them fixed.

Verification is the checking or we can say the testing of software for consistency and conformance by evaluating the results against pre-specified requirements.

Validation looks at the systems correctness, i.e. the process of checking that what has been specified is what the user actually wanted.

Defect is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

Standards for Software Test Documentation

IEEE 829-1998 is known as the 829 Standard for Software Test Documentation. It is an IEEE standard that specifies the form of a set of documents for use in software testing[i]. There are other different standards discussed below.

IEEE 1008, a standard for unit testing

IEEE 1012, a standard for Software Verification and Validation

IEEE 1028, a standard for software inspections

IEEE 1044, a standard for the classification of software anomalies

IEEE 1044-1, a guide to the classification of software anomalies

IEEE 830, a guide for developing system requirements specifications

IEEE 730, a standard for software quality assurance plans

IEEE 1061, a standard for software quality metrics and methodology

IEEE 12207, a standard for software life cycle processes and life cycle data

BS 7925-1, a vocabulary of terms used in software testing

BS 7925-2, a standard for software component testing

Testing Frameworks

Following are the different testing frameworks:

jUnit - for Java unit test[ii]

Selenium - is a suite of tools for automating web applications for software testing purposes, plugin for Firefox[iii]

HP QC - is the HP Web-based test management tool. It familiarizes with the process of defining releases, specifying requirements, planning tests, executing tests, tracking defects, alerting on changes, and analyzing results. It also shows how to customize project[iv]

IBM Rational - Rational software has a solution to support business sector for designing, implementing and testing software[v]

Need for Software Testing

There are many reasons for why we should test software, such as:

Software testing identifies the software faults. The removal of faults helps reduce the number of system failures. Reducing failures improves the reliability and the quality of the systems.

Software testing can also improves the other system qualities such as maintainability, usability, and testability.

In order to meet the condition that the last few years of the 20th century systems had to be shown to be free from the 'millennium bug'.

In order to meet the different legal requirements.

In order to meet industry specific standards such as the Aerospace, Missile and Railway Signaling standards.

Test Cases and Test Suite

A test case describes an input descriptions and an expected output descriptions. Input are of two types: preconditions (circumstances that hold prior to test case execution) and the actual inputs that are identified by some testing methods. The set of test cases is called a test suite. We may have a test suite of all possible test cases.

Types of Software Testing

Testing is done in every stage of software development life cycle, but the testing done at each level of software development is different in nature and has different objectives. There are different types of testing, such as stress testing, volume testing, configuration testing, compatibility testing, recovery testing, maintenance testing,

documentation testing, and usability testing. Software testing are mainlyof following types[1]

Unit Testing

Integration Testing

System Testing

Unit Testing

Unit testing is done at the lowest level. It tests the basic unit of software, that is the smallest testable piece of software. The individual component or unit of a program are tested in unit testing. Unit testing are of two types.

Black box testing: This is also known as functional testing, where the test cases are designed based on input output values only. There are many types of Black Box Testing but following are the prominent ones.

- **Equivalence class partitioning**: In this approach, the domain of input values to a program is divided into a set of equivalence classes. e.g. Consider a software program that computes whether an integer number is even or not that is in the range of 0 to 10. Determine the equivalence class test suite. There are three equivalence classes for this program. - The set of negative integer - The integers in the range 0 to 10 - The integer larger than 10
- **Boundary value analysis:** In this approach, while designing the test cases, the values at boundaries of different equivalence classes are taken into consideration. e.g. In the above given example as in equivalence class partitioning, a boundary values based test suite is { 0, -1, 10, 11 }

White box testing: It is also known as structural testing. In this testing, test cases are designed on the basis of examination of the code. This testing is performed based on the knowledge of how the system is implemented. It includes analysing data flow, control flow, information flow, coding practices, exception and error handling within the system, to test the intended and unintended software behaviour. White box testing can be performed to validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities. This testing requires access to the source code. Though white box testing can be performed any time in the life cycle after the code is developed, but it is a good practice to perform white box testing during the unit testing phase.

Integration Testing

Integration testing is performed when two or more tested units are combined into a larger structure. The main objective of this testing is to check whether the different modules of a program interface with each other properly or not. This testing is mainly of two types:

Top-down approach

Bottom-up approach

In bottom-up approach, each subsystem is tested separately and then the full system is tested. But the top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level 'skeleton' has been tested, the immediately subroutines of the 'skeleton' are combined with it and tested.

System Testing

System testing tends to affirm the end-to-end quality of the entire system. System testing is often based on the functional / requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability are also checked. There are three types of system testing

Alpha testing is done by the developers who develop the software. This testing is also done by the client or an outsider with the presence of developer or we can say tester.

Beta testing is done by very few number of end users before the delivery, where the change requests are fixed, if the user gives any feedback or reports any type of defect.

User Acceptance testing is also another level of the system testing process where the system is tested for acceptability. This test evaluates the system's compliance with the client requirements and assess whether it is acceptable for software delivery

An error correction may introduce new errors. Therefore, after every round of error-fixing, another testing is carried out, i.e. called regression testing. Regression testing does not belong to either unit testing, integration testing, or system testing, instead, it is a separate dimension to these three forms of testing.

Regression Testing

The purpose of regression testing is to ensure that bug fixes and new functionality introduced in a software do not adversely affect the unmodified parts of the program[2]. Regression testing is an important activity at both testing and maintenance phases. When a piece of software is modified, it is necessary to ensure that the quality

of the software is preserved. To this end, regression testing is to retest the software using the test cases selected from the original test suite.

Example

Write a program to calculate the square of a number in the range 1-100

```
#include <stdio.h>
int
Department of Computer Engineering
main()
{
 int n, res;
 printf("Enter a number: ");
 scanf("%d", &n);
 if (n >= 1 && n <= 100)
 {
res = n * n;
printf("\n Square of %d is %d\n", n, res);
 }
 else if (n<= 0 || n > 100)
printf("Beyond the range");
 return 0;
}
```

Output

Inputs	Outputs
I1 : -2	O1 : Beyond the range
I2 : 0	O2 : Beyond the range
I3 : 1	O3 : Square of 1 is 1
I4 : 100	O4 : Square of 100 is 10000
I5 : 101	O5 : Beyond the range
I6 : 4	O6 : Square of 4 is 16
I7 : 62	O7 : Square of 62 is 3844

Test Cases

T1 : {I1 ,O1}

T2 : {I2 ,O2}

T3 : {I3, O3}

T4 : {I4, O4}

T5 : {I5, O5}

T6 : {I6, O6}

T7 : {I7, O7}

As already discussed under the theory section, test case preparation could begin right after requirements identification stage. It is desirable (and advisable) to create a Requirements Traceability Matrix (RTM) showing a mapping from individual requirement to test case(s). A simplified form of the RTM is shown in table 1 (the numbers shown in this table are arbitrary; not specific to LIS).

Requirement #	Test Case #
R1	TC1
R2	TC2, TC3, TC4
R3	TC5
R4	TC6

Table 1: Simplified Mapping from requirements to test cases

Table 1 states which test case should help us to verify that a specified feature has been implemented and working correctly. For instance, if test case # TC6 fails, that would indicate requirement # R4 has not fully realized yet. Note that it is possible that a particular requirement might need multiple test cases to verify whether it has been implemented correctly.

To be specific to our problem, let us see how we can design test cases to verify the "User Login" feature. The simplest scenario is when both user name and password have been typed in correctly. The outcome will be that the user could then avail all features of LIS. However, there could be multiple unsuccessful conditions:

- User ID is wrong
- Password is wrong
- User ID & password are wrong
- Wrong password given twice consecutively
- Wrong password given thrice consecutively
- Wrong password given thrice consecutively, and security question answered correctly
- Wrong password given thrice consecutively, and security question answered incorrectly

We would create test case for the above stated login scenarios. These test cases together would constitute a test suite to verify the concerned requirement. Table 2 shows the details of this test suite.

Department of Computer Engineering

TC5	Verify that user already registered with the LIS is unable to login with incorrect password given thrice consecutively	TC4	This test case is executed after execution of TC4 before executing any other test case	User is not logged in	1. Type in employee ID as *149405* 2. Type in password *whatever3* 3. Click on the 'Login' button	The "Login" dialog is shown with a *"Login failed! Check your user ID and password"* message; the security question and input box for the answer are displayed
TC6	Verify that a registered user can login after three consecutive failures by correctly answering the security question	TC5	This test case is executed after execution of TC6 before executing any other test case. Answer to the security question is *my_answer*.	Email sent containing new password. The email is expected to be received within 2 minute.	1. Type in the answer as *my_answer* 2. Click on the 'Email Password' button	Login dialog is displayed; an email containing the new password is received
TC7	Verify that a registered user's account is blocked after three consecutive failures and answering the security question incorrectly		Execute the test cases TC3, TC4, and TC5 once again (in order) before executing this test case	User account has been blocked	1. Type in the answer as *not_my_answer* 2. Click on the 'Email Password' button	The message *"Your account has been blocked! Please contact the administrator."* appears

Conclusion: In this experiment we learn about types of testing and we implement the black box testing successfully**.**

Experiment No.8

Aim: -Write test cases for White box testing

Theory:

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.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user type perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

Software Testing

Testing software is an important part of the development life cycle of a software. It is an expensive activity. Hence, appropriate testing methods are necessary for ensuring the reliability of a program. According to the ANSI/IEEE 1059 standard, the definition of testing is the process of analyzing a software item, to detect the differences between existing and required conditions i.e. defects/errors/bugs and to evaluate the features of the software item.

- The purpose of testing is to verify and validate a software and to find the defects present in a software. The purpose of finding those problems is to get them fixed.
- Verification is the checking or we can say the testing of software for consistency and conformance by evaluating the results against pre-specified requirements.

- Validation looks at the systems correctness, i.e. the process of checking that what has been specified is what the user actually wanted.
- Defect is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

Need for Software Testing

There are many reasons for why we should test software, such as:

- Software testing identifies the software faults. The removal of faults helps reduce the number of system failures. Reducing failures improves the reliability and the quality of the systems.
- Software testing can also improve the other system qualities such as maintainability, usability, and testability.
- In order to meet the condition that the last few years of the 20th century systems had to be shown to be free from the 'millennium bug'.
- In order to meet the different legal requirements.
- In order to meet industry specific standards such as the Aerospace, Missile and Railway Signalling standards.

Test Cases and Test Suite

A test case describes an input description and an expected output description. Input are of two types: preconditions (circumstances that hold prior to test case execution) and the actual inputs that are identified by some testing methods. The set of test cases is called a test suite. We may have a test suite of all possible test cases.

White box testing

It refers to a scenario where (as opposed to **black box testing**), the tester deeply understands the inner workings of the system or system component being tested. Gaining a deep understanding of the system or component is possible when the tester understands these at program- or code-level. So almost all the time, the tester needs to either understand or have access to the source code that makes up the system – usually in the form of specification documents.

Armed with the level of technical detail that is normally visible only to a developer, a Tester will then be able to design and execute test cases that cover all possible scenarios and conditions that the system component is designed to handle. We'll see how this is done in our example later. By performing testing at the most granular level

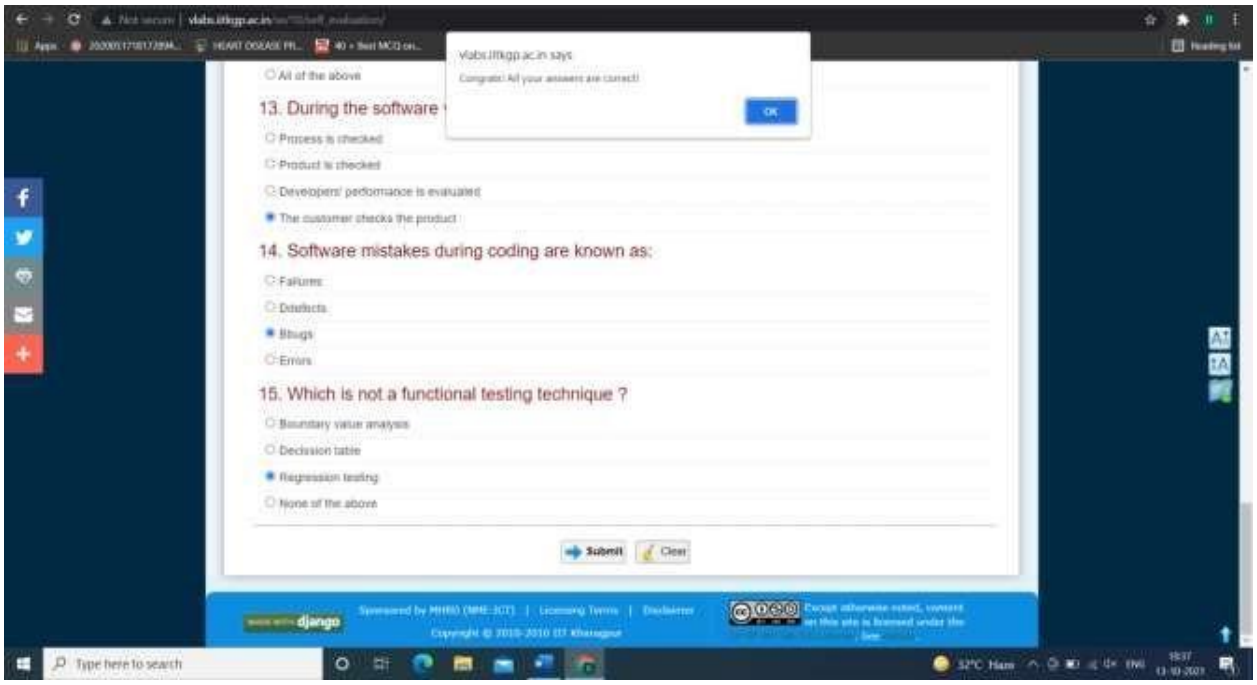
of the system, you are able to build a robust system that works exactly as expected, and ensure it will not throw up any surprises whatsoever.

The key principles that assist you in executing white box tests successfully are:

Statement Coverage – ensure every single line of code is tested.
Statement Coverage = (Number of statements executed / Total number of statements executable) x 100%

Branch Coverage – ensure every branch (e.g. true or false) is tested.
Branch Coverage = (Number of decisions outcomes tested / Total number of decision outcomes) x 100%

Path Coverage – ensure all possible paths are tested.
Path Coverage = (Number of paths executed / Total number of paths) x 100%



Conclusion: Hence we study about the white box testing.

Experiment No.9

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

Theory:

Risk Mitigation: It is an activity used to avoid problems (Risk Avoidance). Steps for mitigating the risks as follows.

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

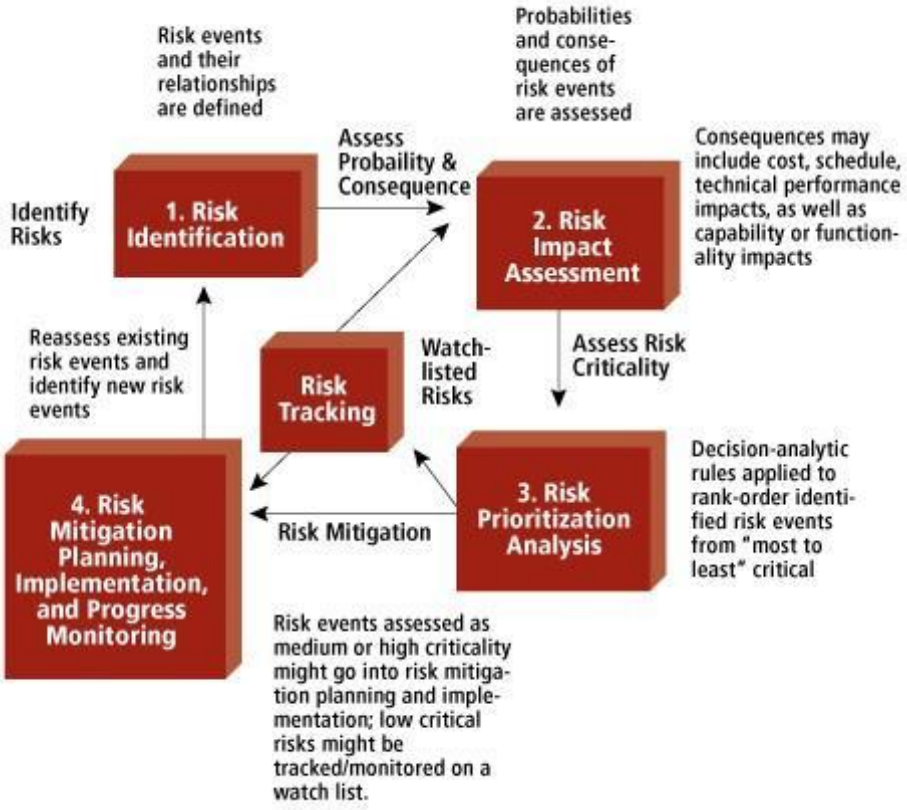
Product Size Risks:

- Estimated size in lines of code (LOC) : Credit Card Fraud Detection will have an estimated 17,555 lines of code.
- Degree of confidence in estimated size: We are highly confident in our estimated size.
- Estimated size in number of programs, files, and transactions:
- We estimate 2 programs.
- We estimate 10 large files for the engine, 5 large files for the user interface.
- We estimate 40 or more transactions for the engine, and 20 transactions for the user-interface.
- Size of database created or used: The size of the database that we will use will be an estimated 7 tables. The number of fields will vary per table and will have an overall average of 8 fields per table.

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.
- To ensure proper application of risk aversion steps defined for
- risk.
- To collect data for future risk analysis.
- 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 areality. This task is done by Project manager when riskbecomes reality and causes severe problems. If the projectmanager effectively uses project mitigation to remove riskssuccessfully 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.



Risk mitigation handling options include:

1. Assume/Accept: Acknowledge the existence of a particular risk, and make a deliberate decision to accept it without engaging in special efforts to control it. Approval of project or program leaders is required.

2. Avoid: Adjust program requirements or constraints to eliminate or reduce the risk. This adjustment could be accommodated by a change in funding, schedule, or technical requirements.
3. Control: Implement actions to minimize the impact or likelihood of the risk.
4. Transfer: Reassign organizational accountability, responsibility, and authority to another stakeholder willing to accept the risk.
5. Watch/Monitor: Monitor the environment for changes that affect the nature and/or the impact of the risk.

CONCLUSION: Thus, we have prepared a Risk Mitigation, Monitoring and Management Plan (RMMM).

Experiment No.10

Aim: To change specifications and make different versions using any SCM Tool.

Theory:

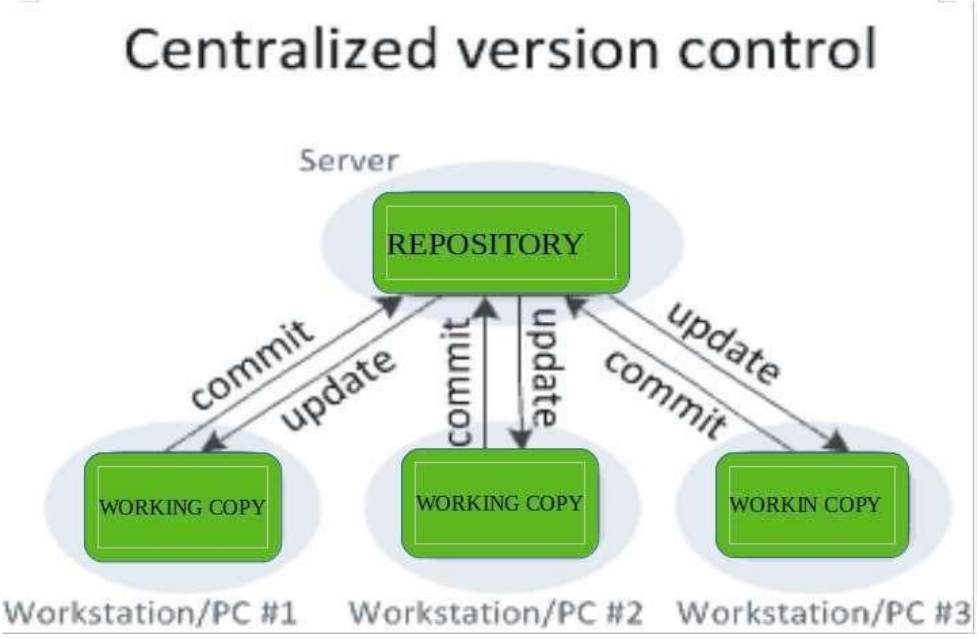
Software Configuration Management (SCM): The primary focus of SCM is to identify and control major software changes, ensure that change is being properly implemented, and report changes to any other personnel or clients who may have an interest.

The objective of SCM is to limit the impact changes may have on the entire system. This will help to eliminate unnecessary changes, and to monitor and control any necessary changes. This allows software development to continue, despite large and/or insignificant changes without significant backtracking, lessening development time and resulting in a higher-quality product.

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.



SCM Process: It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

1. Identification of objects in the software configuration:

a. Description:

- The SCM leader will analyse all current design specifications and break down the software into subsystems.
- All subsystems will consist of major software functions or interface components.
- Any submitted changes will be connected to its corresponding subsystem, which will be traced backwards through the system to determine its impact.

b. Work Products:

- An SCI document will contain the breakdown of subsystems and how they are interrelated.
- Any approved changes will be returned to the software engineer with a change approval sheet, a listing of all possible affected subsystems, and any additional information the software engineer may need before he begins changing code.

2. Version Control:

a. Description:

- Whenever the system or a subsystem is updated, the program build number (version number) will be updated to reflect the change.
- The version number will follow a standard x.x.x input mask (for example, version

1.2.7), with each digit placed corresponding to an increasing severity of change. -The hundredths place (x.x.x) will reflect very minor changes to the software. -The tenths place (x.x.x) will reflect more substantial software changes.

-The one place (x.x.x) will reflect severe changes to the software. Version control will be done by hand. No source code tracking / version control tools will be used.

b. Minor Version Change

c. Substantial Version Change

3. Change Control:

- Software engineers will submit a change request to the SCM leader.
- The SCM leader will then analyze the request, using the SCI document, the project design document, and the current prototype of the software.
- He will base his decision on how severely the change will impact the entire system and, more importantly, on the corresponding subsystem.
- Once his decision has been made, he must submit the change to the software engineer of his choice, as well as updating the SCI document to accommodate the change, and the SCM library to record the change request and decision.

4. Configuration Audit

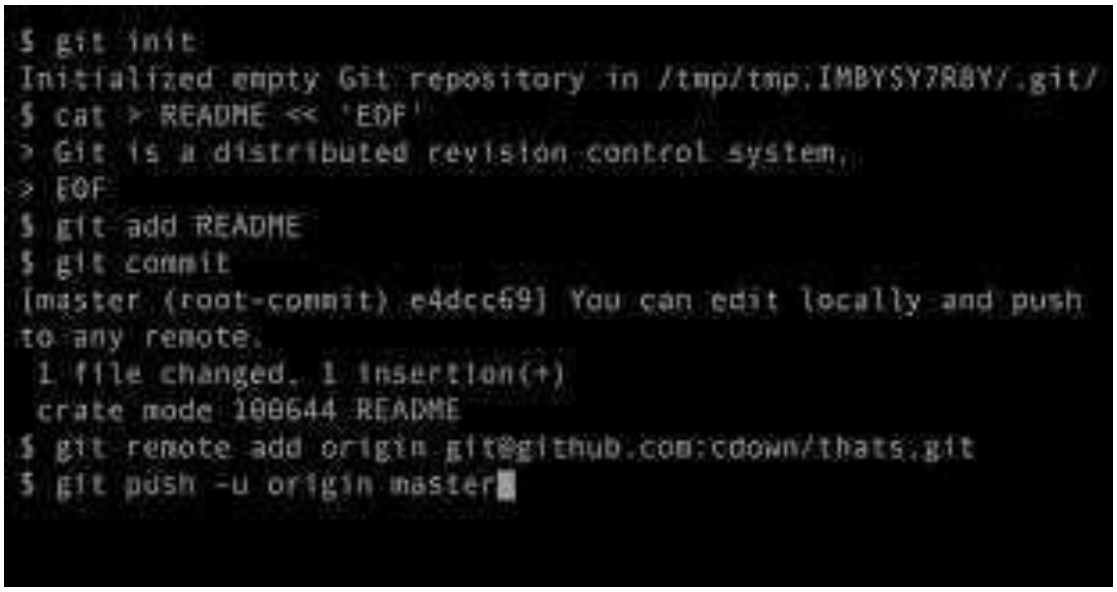
5. Status Reporting

GitHub: GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

Step 1. Create a
Repository: git remote

add origin “url” Step
2. Create a Branch: git
branch -M main

Step 3. Make and commit changes:
git commit -m "your
message" git push -u origin
main



index.html

git_test > index.html > ...

```
1   <!DOCTYPE html>
2   <html lang="en">
3
4   <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>Git Test</title>
9   </head>
10
11  <body>
12    <!-- the following line has been modified -->
13    <h1>This is an Updated GIT test</h1>
14
15    <!-- the following line has been removed -->
16
17    <!-- the following line has been added -->
18    <h2>This is a new line of code</h2>
19  </body>
20
21  </html>
```

CONCLUSION: Thus, we have made different versions.