Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI
COLLEGE OF ENGINEERING
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423
Minority Status (Hindi Linguistic)

DEPARTMENT OF COMPUTER ENGINEERING

Software Engineering Lab. (CSL501)

Subject Code: CSL501

Conducted by: Mrs. Manasi Vishal Chouk

Academic Year: 2022-23

Semester: V

Branch: Computer Engineering

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI
COLLEGE OF ENGINEERING
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423
Minority Status (Hindi Linguistic)

DEPARTMENT OF COMPUTER ENGINEERING

A Laboratory Journal for

Software Engineering Lab. (CSL501)

ACADEMIC YEAR: 2024-2025

Course Name: Software Engineering Lab. **Course Code:** CSL501

Name:_____

Semester: V (Fifth) **Roll No. :**_____

DEPARTMENT OF COMPUTER ENGINEERING

VISION AND MISSION

Institution's

Vision	To be a world class institute and a front runner in educational and socioeconomic development of the nation by providing high quality technical education to students from all sections of society.
Mission	To provide superior learning experiences in a caring and conducive environment so as to empower students to be successful in life & contribute positively to society.
Quality Policy	We, at SHREE L. R. TIWARI COLLEGE OF ENGINEERING, shall dedicate and strive hard to continuously achieve academic excellence in the field of Engineering and to produce the most competent Engineers through objective & innovative teaching methods, consistent updating of facilities, welfare & quality improvement of the faculty & a system of continual process improvement.

Computer Engineering Department's

Vision	To be a department of high repute focused on quality education, training and skill development in the field of computer engineering to prepare professionals and entrepreneurs of high caliber with human values to serve our nation and globe.
Mission	**M1:** To provide fertile academic environment for the development of skilled professionals and empowered with knowledge, skills, values, and confidence to take the leadership role and to bridge the gap between industry institute and society in the field of Computer engineering. **M2:** To promote caring and interactive teaching practices in a rejoicing learning ambience with richly supported modern educational tools and techniques. **M3:** To enhance and revitalize research culture to provide practical exposure and to establish synergy between teaching and research and make it an enabler for speedy progress. **M4:** To pursue intensification of soft skills and personality development through interplay of achievers of all segments of our society. **M5:** To provide human values to students by promoting lifelong learning ability.
Program Educational Objectives	**PEO-1:** To prepare the Learner with a sound foundation in the mathematical, scientific and engineering fundamentals **PEO-2:** To motivate the Learner in the art of self-learning and to use modern tools for solving real life problems **PEO-3:** To encourage, motivate and prepare the Learner's for Lifelong learning **PEO-4:** To promote student awareness and commitment to lifelong learning and professional ethics during the course of professional practice. **PEO-5:** To inculcate professional and ethical attitude, good leadership qualities and commitment to social responsibilities in the Learner's thought process

Student's Signature

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI
COLLEGE OF ENGINEERING
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423
Minority Status (Hindi Linguistic)

DEPARTMENT OF COMPUTER ENGINEERING

Certificate

This is to certify that Mr. /Ms. _____

Class _____ *Roll No.* _____ *Exam Seat No.* _____ *of* ***Fifth Semester of Degree in Computer Engineering*** *has completed the required number of Practical's / Term Work / Session in the subject* ***Software Engineering Lab*** *from the* ***Department of Computer Engineering*** *during the academic year of* ***2024 -2025*** *as prescribed in the curriculum.*

Course in-Charge

Head of the Department

Date:

Seal of Institution

INSTRUCTION FOR STUDENTS

Students shall read the points given below for understanding the theoretical concepts and practical applications.

1) Listen carefully to the lecture given by teacher about importance of subject, curriculum philosophy learning structure, skills to be developed, information about equipment, instruments, procedure, method of continuous assessment, tentative plan of work in laboratory and total amount of work to be done in a semester.
2) Student shall undergo study visit of the laboratory for types of equipment, instruments, software to be used, before performing experiments.
3) Read the write up of each experiment to be performed, a day in advance.
4) Organize the work in the group and make a record of all observations.
5) Understand the purpose of experiment and its practical implications.
6) Write the answers of the questions allotted by the teacher during practical hours if possible or afterwards, but immediately.
7) Student should not hesitate to ask any difficulty faced during conduct of practical/exercise.
8) The student shall study all the questions given in the laboratory manual and practice to write the answers to these questions.
9) Student shall develop maintenance skills as expected by the industries.
10) Student should develop the habit of pocket discussion/group discussion related to the experiments/exercises so that exchanges of knowledge/skills could take place.
11) Student shall attempt to develop related hands-on-skills and gain confidence.
12) Student shall focus on development of skills rather than theoretical or codified knowledge.
13) Student shall visit the nearby workshops, workstation, industries, laboratories, technical exhibitions, trade fair etc. even not included in the Lab manual. In short, students should have exposure to the area of work right in the student hood.
14) Student shall insist for the completion of recommended laboratory work, industrial visits, answers to the given questions, etc.
15) Student shall develop the habit of evolving more ideas, innovations, skills etc. those included in the scope of the manual.
16) Student shall refer technical magazines, proceedings of the seminars, refer websites related to the scope of the subjects and update his knowledge and skills.
17) Student should develop the habit of not to depend totally on teachers but to develop self-learning techniques.
18) Student should develop the habit to react with the teacher without hesitation with respect to the academics involved.
19) Student should develop habit to submit the practical's, exercise continuously and progressively on the scheduled dates and should get the assessment done.
20) Student should be well prepared while submitting the write up of the exercise. This will develop the continuity of the studies and he/she will not be over loaded at the end of the term.

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI
COLLEGE OF ENGINEERING
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423
Minority Status (Hindi Linguistic)

GUIDELINES FOR TEACHERS

Teachers shall discuss the following points with students before start of practicals of the subject.

1) Learning Overview: To develop better understanding of importance of the subject. To know related skills to be developed such as Intellectual skills and Motor skills.
2) Learning Structure: In this, topic and sub topics are organized in systematic way so that ultimate purpose of learning the subject is achieved. This is arranged in the form of fact, concept, principle, procedure, application and problem.
3) Know you're Laboratory Work: To understand the layout of laboratory, specifications of equipment/Instruments/Materials, procedure, working in groups, planning time etc. Also to know total amount of work to be done in the laboratory.
4) Teaching shall ensure that required equipment's are in working condition before start of experiment, also keep operating instruction manual available.
5) Explain prior concepts to the students before starting of each experiment.
6) Involve students' activity at the time of conduct of each experiment.
7) While taking reading/observation each student shall be given a chance to perform or observe the experiment.
8) If the experimental set up has variations in the specifications of the equipment, the teachers are advised to make the necessary changes, wherever needed.
9) Teacher shall assess the performance of students continuously as per norms prescribed by university of Mumbai and guidelines provided by IQAC.
10) Teacher should ensure that the respective skills and competencies are developed in the students after the completion of the practical exercise.
11) Teacher is expected to share the skills and competencies are developed in the students.
12) Teacher may provide additional knowledge and skills to the students even though not covered in the manual but are expected from the students by the industries.
13) Teachers shall ensure that industrial visits if recommended in the manual are covered.
14) Teacher may suggest the students to refer additional related literature of the Technical papers/Reference books/Seminar proceedings, etc.
15) During assessment teacher is expected to ask questions to the students to tap their achievements regarding related knowledge and skills so that students can prepare while submitting record of the practical's. Focus should be given on development of enlisted skills rather than theoretical /codified knowledge.
16) Teacher should enlist the skills to be developed in the students that are expected by the industry.
17) Teacher should organize Group discussions /brain storming sessions / Seminars to facilitate the exchange of knowledge amongst the students.
18) Teacher should ensure that revised assessment norms are followed simultaneously and progressively.
19) Teacher should give more focus on hands on skills and should actually share the same.
20) Teacher shall also refer to the circulars related to practical's supervise and assessment for additional guidelines.

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI
COLLEGE OF ENGINEERING
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423
Minority Status (Hindi Linguistic)

DEPARTMENT OF COMPUTER ENGINEERING

RECORD OF PROGRESSIVE ASSESSMENTS

Student Name: _____ **Roll No.: _____(TE CMPN SEM.-V)**
Course Name : Software Engineering Lab. **Course Code: CSL501**

Assessment of Experiments (A)

Sr. no.	Name of Experiments	Page No.	Date of Performance	Date of Submission	Assessment (out of 15)	Teacher's Signature and Remark	CO Covered
1	Application of at least two traditional process models.						CO1
2	Application of the Agile process models.						CO1
3	Preparation of software requirement specification (SRS) document in IEEE format.						CO2
4	Perform Structured data flow analysis.						CO2
5	Use of metrics to estimate the cost						CO2
6	Scheduling & tracking of the project						CO2
7	Write test cases for black box testing.						CO2
8	Write test cases for white box testing.						CO3
9	Preparation of Risk Mitigation, Monitoring and Management Plan (RMMM).						CO3
10	Version controlling of the project.						CO3
Average Marks (Out of 15)							

Assessment of Assignment (B)

Assignment No.	Page No.	Date of Performance	Date of Submission	Assessment (out of 12)	Teacher's Signature and Remark	CO Covered
Assignment No.1						CO1, CO2, CO3, CO4, CO5
Assignment No.2						
Assignment No.3						
Assignment No.4						
Assignment No.5						
Converted Marks (Out of 5) (B)						

Assessments of Attendance (C)

SE Theory Attendance			SEL Practical Attendance			AVG. Attendance % (TH+PR)	Attendance Marks (D) (Out of 5)
TH (out of)	TH attend.	TH %	PR (out of)	PR Attend.	PR %		

Total Term Work Marks: A+B+C =_____(Out of 25)

---------- ----------
Student Signature **Subject In-charge**

DEPARTMENT OF COMPUTER ENGINEERING

Program Outcome (POs & PSOs)

Program Outcomes are the skills and knowledge which the students have at the time of graduation. This will indicate what student can do from subject-wise knowledge acquired during the program .

PO	Graduate Attributes	Description of the Program outcome as defined by the NBA
PO-1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO-2	Problem analysis	Identify, formulate, review research literature, and analyses complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO-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.
PO-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.
PO-5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO-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.
PO-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.
PO-8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO-9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO-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.
PO-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.
PO-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.
Program Specific Outcomes (PSOs) defined by the programme. Baseline-Rational Unified Process(RUP)		
PSO-1	System Inception and Elaboration	Conceptualize the software and/or hardware systems, system components and process/procedures through requirement analysis, modeling/design of the system using various architectural/design patterns, standard notations, procedures and algorithms.
PSO-2	System Construction	Implement the systems, procedures and processes using the state of the art technologies, standards, tolls and programming paradigms.
PSO-3	System Testing and Deployment	Verify and validate the systems, procedures and processes using various testing and verification techniques and tools.
PSO-4	Quality and Maintenance	Manage the quality through various product development strategies under revision, transition and operation through maintainability, flexibility, testability, portability, reusability, interoperability, correctness, reliability, efficiency, integrity and usability to adapt the system to the changing structure and behavior of the systems/environments.

Student's Signature

SHREE L.R. TIWARI COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

Course Exit Form

Academic Year: 2023-2024 **Class: TE**
Course Code: CSL501 **Div.: CMPN**
Course Name: Software Engineering **Sem.: V**
Name of Student: **Roll No.:**

Judge your ability with regard to the following points by putting a **(√),** on the scale of **1 (lowest) to 5 (highest),** based on the knowledge and skills you attained from this course.

Sr.	Your ability to	1 Lowest	2	3	4	5 Highest
1	CSC502.1 Identify requirements & assess the process models					
2	CSC502.2 Plan, schedule and track the progress of the projects.					
3	CSC503.3 Design the software projects.					
4	CSC502.4 Identify risks, manage the change to assure quality in software projects					
5	CSC502.5 Do testing of software project					

Signature of the student: **Date:**

Shree Rahul Education Society's (Regd.)
SHREE L. R. TIWARI
COLLEGE OF ENGINEERING
(Approved by AICTE & DTE, Maharashtra State & Affiliated to University of Mumbai)
NAAC Accredited, NBA Accredited Program, ISO 9001:2015 Certified | DTE Code No. : 3423
Minority Status (Hindi Linguistic)

DEPARTMENT OF COMPUTER ENGINEERING

CSL501 Software Engineering Lab

Fifth Semester, 2022-2023 (Odd Semester)

Name of Student :

Roll No.

Batch :

Day / Session :

Venue : Computer Lab 307

Experiment No. :

Title of Experiment :

Date of Conduction :

Date of Submission :

Particulars	Max. Marks	Marks Obtained
Preparedness and Efforts (PE)	**3**	
Knowledge of Tools (KT)	**3**	
Debugging and Results (DR)	**3**	
Documentation (DN)	**3**	
Punctuality & Lab Ethics (PL)	**3**	
Total	**15**	

Grades – Meet Expectations (3 Marks), Moderate Expectations (2 Marks), Below Expectations (1 Mark)

Checked and verified by

Name of Faculty : Mr. Ravindra Sonavane

Signature :

Date :

EXPERIMENT NO. 1

TITLE: Application of at least two traditional process models.

AIM: Application of at least two traditional process models.

OBJECTIVE: To get familiar with traditional process models.

THEORY:

1) Waterfall Model

The Waterfall Model is the earliest **SDLC (System Development Life Cycle**). SDLC is also referred as **application development life cycle** and it is a process for requirement gathering or communication, planning, modeling, construction, deployment and maintenance. Dr.Winston Royce introduced this model in 1970 used it to reduce the high cost of releasing new revisions of product. This model is named "**Waterfall Model**" resembles the flow of river, with each stage processing sequentially. The key feature of this model is that each phase must be completed then only next phase proceed further.

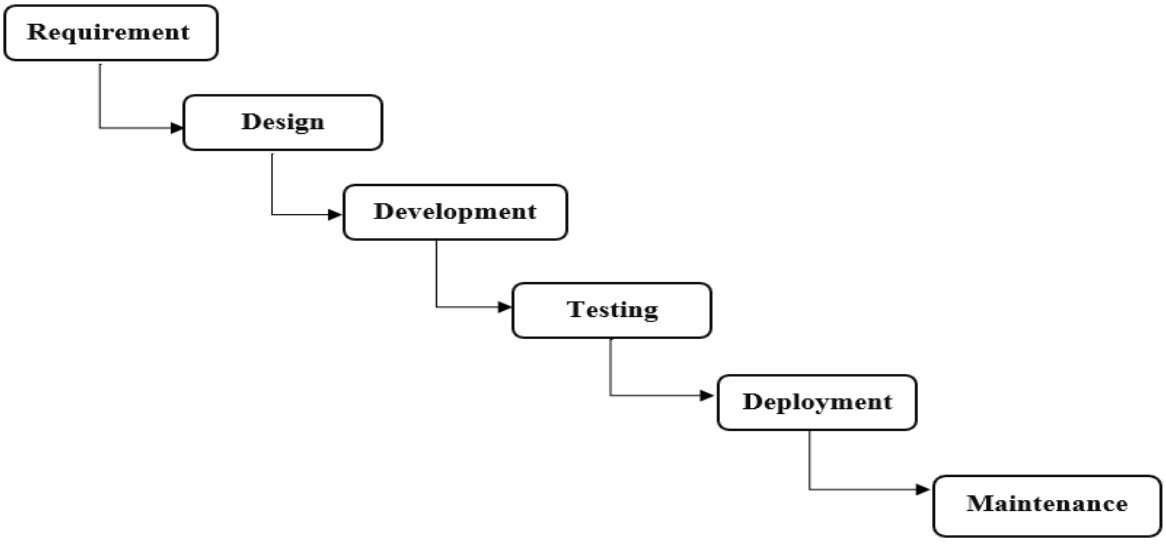


Diagram of Waterfall Model

The Waterfall model consists of six phases are as follows

1) Requirements Gathering and Analysis
2) Design
3) Development
4) Testing
5) Deployment
6) Maintenance

Requirements Gathering and Analysis

This phase is "Requirement Gathering of Communication" which collect requirement from customer and stakeholder through meetings, interviews and survey. The software developer gather this all requirements and make a document known as "SRS" (Software Requirement Specification). The document is created which contained a detailed description of what the system will do in the common language. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. The aim of this phase is to understand the exact requirements of the customer and to document them properly. This stage also involves analysing the requirements to identify risks and documenting strategies.

Design

The design phase focuses on designing architecture, business logic and concepts for the software, user interfaces and features. It aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. All this work is documented as a Software Design Document (SDD).

Development:

In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The unit testing phase aims to check whether each module is working properly or not.

Testing:

Integration of different modules is undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over several steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this. System testing consists of three different kinds of testing activities as described below.

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performs acceptance testing to determine whether to accept the delivered software or reject it.

Deployment & Maintenance:

Integration of different modules is undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over several steps. Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are three types of maintenance.

Corrective Maintenance: This type of maintenance is carried out to correct errors that were not discovered during the product development phase.

Perfective Maintenance: This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request

Adaptive Maintenance: Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Applications of Waterfall Model

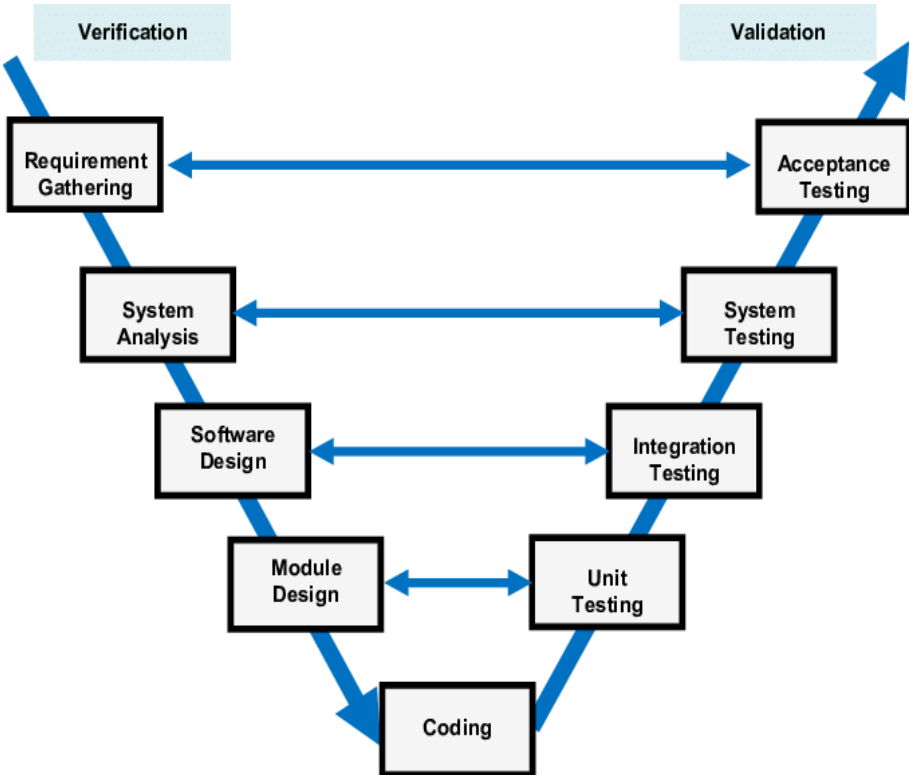
- **Large-scale Software Development Projects:** The Waterfall Model is often used for large-scale software development projects, where a structured and sequential approach is necessary to ensure that the project is completed on time and within budget.
- **Safety-Critical Systems:** The Waterfall Model is often used in the development of safety-critical systems, such as aerospace or medical systems, where the consequences of errors or defects can be severe.
- **Government and Defense Projects:** The Waterfall Model is also commonly used in government and defense projects, where a rigorous and structured approach is necessary to ensure that the project meets all requirements and is delivered on time.
- **Projects with well-defined Requirements:** The Waterfall Model is best suited for projects with well-defined requirements, as the sequential nature of the model requires a clear understanding of the project objectives and scope.
- **Projects with Stable Requirements:** The Waterfall Model is also well-suited for projects with stable requirements, as the linear nature of the model does not allow for changes to be made once a phase has been completed.

2) V Model

Verification Phases:

It involves a static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements are met.

There are several Verification phases in the V-Model:



Business Requirement Analysis:

This is the first step of the designation of the development cycle where product requirement needs to be cured from the customer's perspective. in these phases include proper communication with the customer to understand the requirements of the customers. these are the very important activities that need to be handled properly, as most of the time customers do not know exactly what they want, and they are not sure about it at that time then we use an **acceptance test design** planning which is done at the time of business requirement it will be used as an **input** for acceptance testing.

System Design:

Design of the system will start when the overall we are clear with the product requirements, and then need to design the system completely. This understanding will be at the beginning of complete under the product development process. these will be beneficial for the future execution of test cases.

Architectural Design:

In this stage, architectural specifications are comprehended and designed. Usually, several technical approaches are put out, and the ultimate choice is made after considering both the technical and financial viability. The system architecture is further divided into modules that each handle a distinct function. Another name for this is High-Level Design (HLD).

At this point, the exchange of data and communication between the internal modules and external systems are well understood and defined. During this phase, integration tests can be created and documented using the information provided.

Module Design:

This phase, known as Low-Level Design (LLD), specifies the comprehensive internal design for every system module. Compatibility between the design and other external systems as well as other modules in the system architecture is crucial. Unit tests are a crucial component of any development process since they assist in identifying and eradicating the majority of mistakes and flaws at an early stage. Based on the internal module designs, these unit tests may now be created.

Coding Phase:

The Coding step involves writing the code for the system modules that were created during the Design phase. The system and architectural requirements are used to determine which programming language is most appropriate. The coding standards and principles are followed when performing the coding. Before the final build is checked into the repository, the code undergoes many code reviews and is optimized for optimal performance.

Validation Phases:

It involves dynamic analysis techniques (functional, and non-functional), and testing done by executing code. Validation is the process of evaluating the software after the completion of the development phase to determine whether the software meets the customer's expectations and requirements. So, V-Model contains Verification phases on one side of the Validation phases on the other side. The verification and Validation phases are joined by the coding phase in a V-shape. Thus, it is called V-Model. There are several **Validation** phases in the V-Model:

Unit Testing:

Unit Test Plans are developed during the module design phase. These Unit Test Plans are executed to eliminate bugs in code or unit level.

Integration testing:
After completion of unit testing Integration testing is performed. In integration testing, the modules are integrated and the system is tested. Integration testing is performed in the Architecture design phase. This test verifies the communication of modules among themselves.

System Testing:
System testing tests the complete application with its functionality, inter-dependency, and communication. It tests the functional and non-functional requirements of the developed application.

User Acceptance Testing (UAT):
UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets the user's requirement and the system is ready for use in the real world.

Design Phase:

- **Requirement Analysis:** This phase contains detailed communication with the customer to understand their requirements and expectations. This stage is known as Requirement Gathering.
- **System Design:** This phase contains the system design and the complete hardware and communication setup for developing the product.
- **Architectural Design:** System design is broken down further into modules taking up different functionalities. The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood.
- **Module Design:** In this phase, the system breaks down into small modules. The detailed design of modules is specified, also known as Low-Level Design (LLD).

Testing Phases:

- **Unit Testing:** Unit Test Plans are developed during the module design phase. These Unit Test Plans are executed to eliminate bugs at the code or unit level.
- **Integration testing:** After completion of unit testing Integration testing is performed. In integration testing, the modules are integrated, and the system is tested. Integration testing is performed in the Architecture design phase. This test verifies the communication of modules among themselves.
- **System Testing:** System testing tests the complete application with its functionality, interdependency, and communication. It tests the functional and non-functional requirements of the developed application.
- **User Acceptance Testing (UAT):** UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets the user's requirement and the system is ready for use in the real world.

Conclusion

The Waterfall Model has greatly influenced conventional software development processes. This methodical, sequential technique provides an easily understood and applied structured framework. Project teams have a clear roadmap due to the model's methodical evolution through the phases of requirements, design, implementation, testing, deployment, and maintenance. A scientific and organized approach to the Software Development Life Cycle (SDLC) is provided by the Software Engineering V-Model. The team's expertise with the selected methodology, the unique features of the project, and the nature of the requirements should all be taken into consideration when selecting any SDLC models, including the V-Model.

EXPERIMENT NO. 2

TITLE: Application of Agile process models.

AIM: Application of Agile process models.

OBJECTIVE: To get familiar with application of Agile process models

What is Agile?

In software development life cycle, there are two main considerations, one is to emphasize on process and the other is the quality of the software and process itself. Agile software processes is an iterative and incremental based development, where requirements are changeable according to customer needs. It helps in adaptive planning, iterative development and time boxing. It is a theoretical framework that promotes foreseen interactions throughout the development cycle. There are several SDLC models like spiral, waterfall, RAD which has their own advantages. SDLC is a framework that describes the activities performed at each stage of a software development life cycle. The software development activities such as planning, analysis, design, coding, testing and maintenance which need to be performed according to the demand of the customer. It depends on the various applications to choose the specific model. In this paper, however, we will study the agile processes and its methodologies. Agile process is itself a software development process. Agile process is an iterative approach in which customer satisfaction is at highest priority as the customer has direct involvement in evaluating the software.

The agile process follows the software development life cycle which includes requirements gathering, analysis, design, coding, testing and delivers partially implemented software and waits for the customer feedback. In the whole process, customer satisfaction is at highest priority with faster development time. The following Figure. 1, depicts the software development life cycle of Agile Process

*Characteristics Of Agile Projects**:***

Agile process requires less planning and it divides the tasks into small increments. Agile process is meant for short term projects with an effort of team work that follows the software development life cycle. Software development life cycle includes the following phases 1. Requirements gathering, 2. Analysis, 3. Design, 4. Coding, 5. Testing, 6. Maintenance. The involvement of software team management with customers reduces the risks associated with the software. This agile process is an iterative process in which changes can be made according to the customer satisfaction. In agile process new features can be added easily by using multiple iterations.

1. Iterative The main objective of agile software processes is satisfaction of customers, so it focuses on single requirement with multiple iterations.

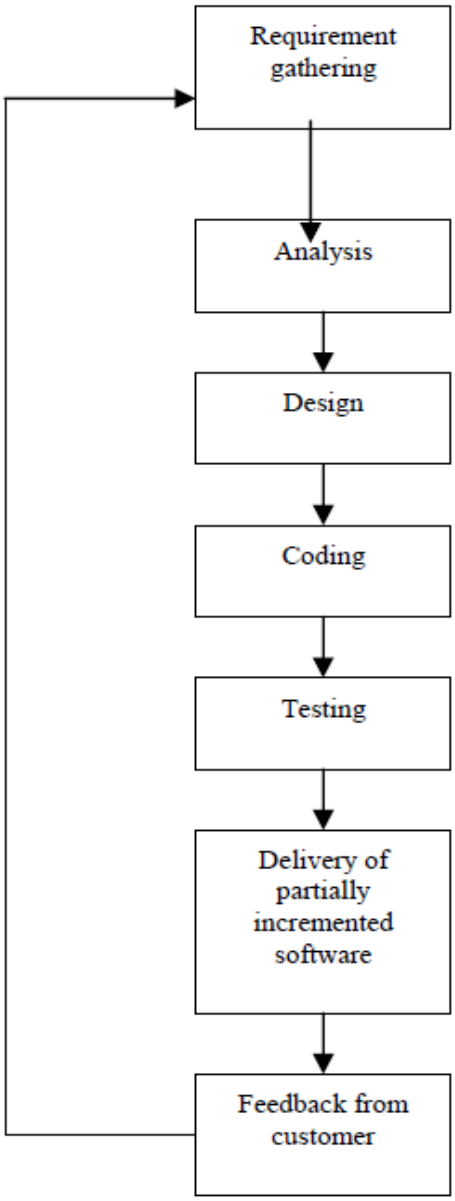


Figure 1. Phases of Agile Process.

2. **Modularity:** Agile process decomposes the complete system into manageable pieces called modules. Modularity plays a major role in software development processes.

3. **Time Boxing** As agile process is iterative in nature, it requires the time limits on each module with respective cycle

4. **Parsimony:** In agile processes parsimony is required to mitigate risks and achieve the goals by minimal number of modules.

5. **Incremental**: As the agile process is iterative in nature, it requires the system to be developed in increments, each increment is independent of others, and at last all increments are integrated into complete system.

6. **Adaptive:** Due to the iterative nature of agile process new risks may occurs. The adaptive characteristic of agile process allows adapting the processes to attack the new risks and allows changes in the real time requirements.

7. **Convergent:** All the risks associated with each increment are convergent in agile process by using iterative and incremental approach.

8**. Collaborative:** As agile process is modular in nature; it needs a good communication among software development team. Different modules need to be integrated at the end of the software development process.

9. **People Oriented**: In the agile processes customer satisfaction is the first priority over the technology and process. A good software development team increases the performance and productivity of the software

What are the Methodologies?

There are several methodologies through which we can implement Agile Projects. Here we have discussed three methodologies which are most widely used in Industry. The agile methods are focused on different aspects of the software development life cycle. Some focus on the practices (extreme programming, pair programming), while others focus on managing the software projects (the scrum approach).

A. Extreme Programming (XP)

XP is the most successful method of developing agile software because of its focus on customer satisfaction. XP requires maximum customer interaction to develop the software. It divides the entire software development life cycle into several number of short development cycles. It welcomes and incorporates changes or requirements from the customers at any phase of the development life cycle.

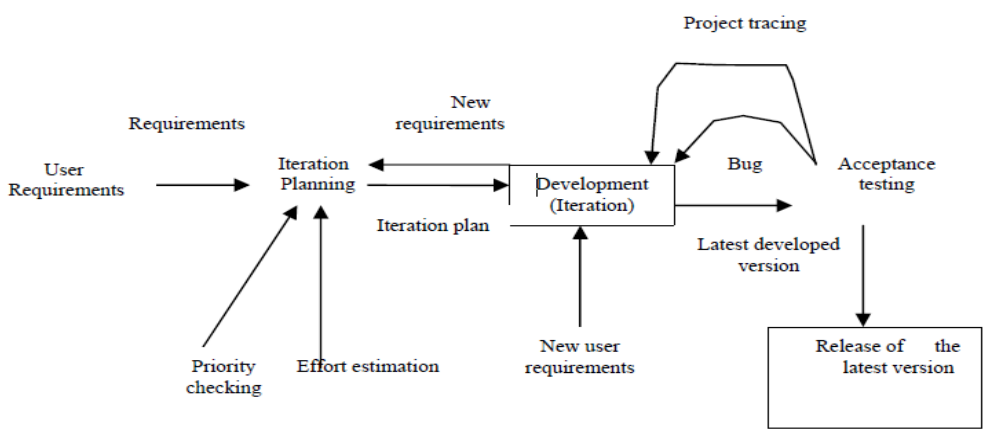


Figure 2 : Method of Developing Agile Processes using Extreme Programming

The above diagram shows the complete method of developing agile process using XP method. Extreme programming starts with collecting user requirements. Depending upon these requirements the whole development process is divided into several small no of cycles. So, the next phase is iteration planning i.e. deciding the no of cycles, prioritizing the requirements and estimating the amount of effort required to implement each cycle. Now each iteration is developed using pair programming.

During the development phase new user requirements may come and the iteration plan should be adjusted according to that. Next step is to test the latest developed version for bugs, if detected; the bugs will be removed in the next iteration. After every acceptance testing project tracing should be done in which feedback is taken from the project that how much job has already been done.

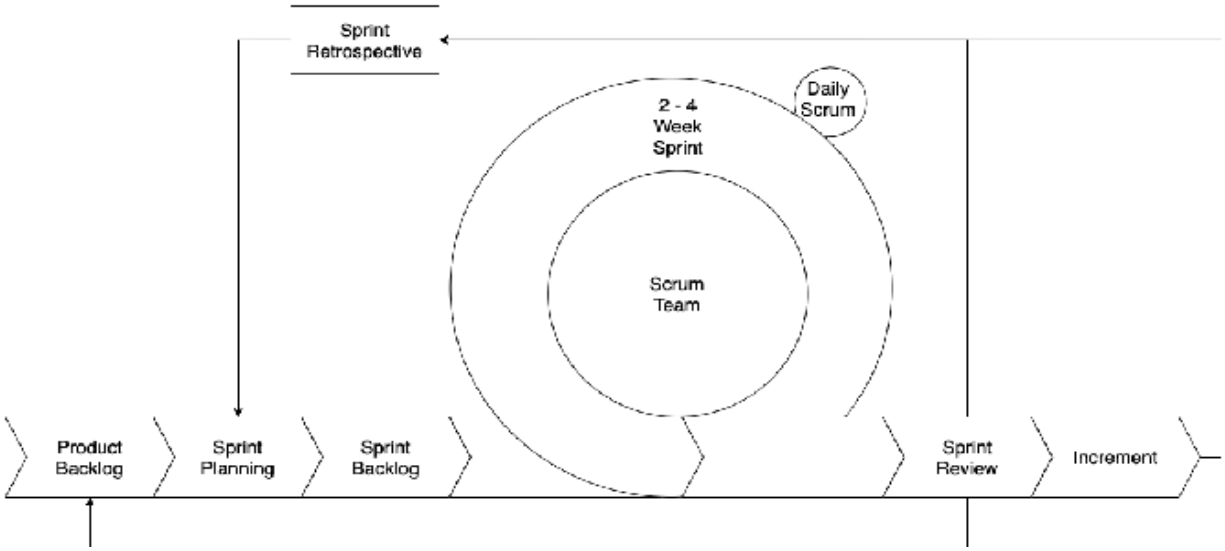
XP has introduced many new things for developers like pair programming, extensive code review, code refactoring and open workspace

B. Scrum

Scrum is another popular method of agile development through which productivity becomes very high. It is basically based on incremental software development process. In scrum method the entire development cycle is divided into a series of iteration where each iteration is called as a sprint. Maximum duration of a sprint is 30 days.

The method starts with collecting user requirements but it is not expected that all the requirements should come out from the user at the beginning. User can change their mind at any time during development; they can add new features, remove or update some existing features. Next phase is to prioritize the requirements and the list is known as product backlog. A proper planning for sprint should be done i.e. how many sprints are needed to develop the software, duration of the sprint, and what are the requirements from the product backlog should be implemented in each sprint. This particular list is known as sprint backlog. During each sprint one sprint meeting is held every day to take the feedback how much work has been done. After each sprint review is taken to determine whether all the requirements for that particular sprint have already been implemented or not and to decide the requirements should be implemented at the next sprint. After each sprint we get a working increment of the software.

Scrum is one of the most fashionable ways to instrument Agile. Scrum is an iterative development model which follows a set of roles, responsibilities, and meetings which hardly change. Usually, sprints last one or two weeks and allow the team to deliver software regularly. Scrum is a very well-known Agile methodology due to its simplicity, productivity, and the ability to act as a framework for all the various practices promoted by Agile methodologies. Scrum relies on time-bound sprints like other agile methodologies. However, Scrum is a bit more prescriptive in structuring your sprints. Each sprint in Scrum features four "ceremonies" that help teams move forward



1. Sprint planning is a team meeting to decide what to include in the current sprint. Once it is decided what to include in the sprint, nothing else can be added except the team
2. Sprint demo is a sharing meeting where the team shows off what they have shipped.
3. Daily Stand-up is a regular 10-15 meeting to sync up and talk about progress and roadblocks.
4. A retrospective is a review of the results of the previous sprint to tweak the process if required

C. Crystal Clear

Crystal Clear development is part of the Crystal family of methodologies. Crystal Clear can be used with six to eight developers. It focuses on the people rather than processes or artifacts. Crystal Clear requires the following: frequent delivery of usable code to users, reflective improvement, and osmotic communication, preferably by being co-located

CONCLUSION

In this, we have discussed the software development life cycle models, the characteristics of agile process, and spiral model, methodologies of agile process, advantages and disadvantages. In the comparative study of agile software development with other software development models we conclude that agile project is much better than other software development process in terms of productivity, performance, faster time cycles, risk analysis. Agile processes are implemented in important applications such as web based, testing tools, etc.

EXPERIMENT NO. 3

TITLE: Preparation of Software Requirement Specification (SRS) document in IEEE format.

AIM: To Prepare SRS as per IEEE format.

OBJECTIVE: To get familiar with preparing a document which is used before starting the project.

THEORY:

1) **What is SRS?**

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to anyactual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS also functions as a blueprint for completing a project with as little cost growth as possible. SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

2) **Goals of SRS?**

Some of the goals an SRS should achieve are to: -

i. Provide feedback to the customer, ensuring that the IT company understands the issues the softwaresystem should solve and how to address those issues.
ii. Help to break a problem down into smaller components just by writing down the requirements.
iii. Speed up the testing and validation processes.
iv. Facilitate reviews.

SRS are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirementsare needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written afterthe requirements have been gathered and analyzed.

3) **Points to be addressed by SRS**

The basic issues that the SRS shall address are the following:

- Functionality: What is the software supposed to do?

- External interfaces: How does the software interact with people, the system's hardware, other hardware, and other software?

- Performance: What is the speed, availability, response time, recovery time of varioussoftware functions, etc.?
- Attributes: What are the portability, correctness, maintainability, security, etc. considerations?
- Design constraints imposed on an implementation: Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.

1) Characteristics of a Good SRS

An SRS should be: -

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Software Requirements Specification

for

Quiz Application using AWT

Prepared by:

----

05 – August – 2022
Name:

DEPARTMENT OF COMPUTER ENGINEERING

SHREE L. R. TIWARI COLLEGE OF ENGINEERING

KANAKIA PARK, MIRA ROAD (E), THANE – 401 107, MAHARASHTRA.

University of Mumbai

(AY 2024-25)

Table of Contents

Roll No.: Roll No.:

1. Introduction

1.1 Purpose

We have created an application using AWT that applies the basics principles of Java Programming and gives us real life experience of modern tools used in the industry.

1.2 Intended Audience and Reading Suggestions

Document is intended for developers, project managers, marketing staff, users, testers, and documentation writers.

We suggest reading this document, beginning with the overview sections and proceeding through the sections that are most pertinent to developers.

1.3 Contact information/SRS team members

49 – Saitwadekar Valay Angar – 7666394676
56 – Singh Anuj Pravin – 7666177461

1.4 References

https://www.eclipse.org/

https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

2. Overall Description

2.1 Product perspective

It is the implementation of the AWT Library. It is a self – contained standalone software.
The diagram shows the basic major components of the overall system.

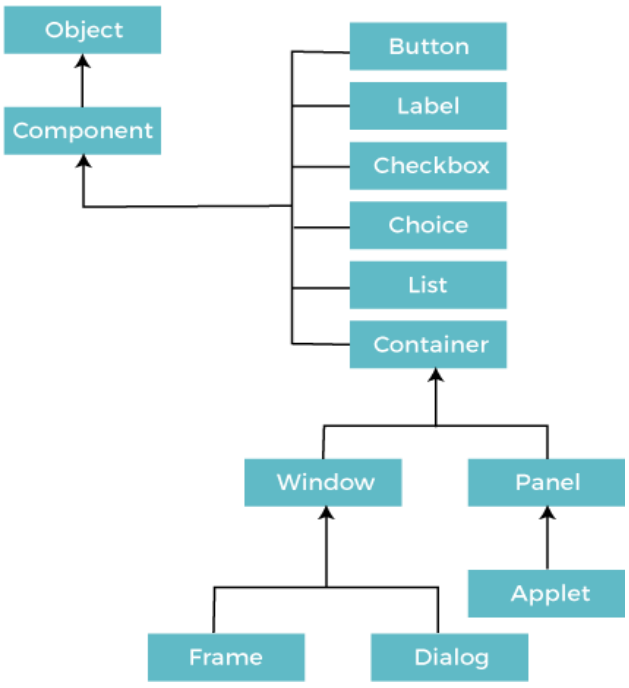


Figure 1 – Major Components

2.2 Product functions

There are 2 functions of this product:

- To create a quiz.
- To attempt a quiz.

2.3 User classes and characteristics

- AWT (GUI) – Majority usage
- Applet (Dynamic Application) – Majority usage
- Swing (Framework) – Majority usage

2.4 Operating environment

Java Interpreter on any PC with 32KB RAM and single core processor on any OS (As Java is Platform Independent).

2.5 Design/implementation constraints

The GUI can't be modified much due to the Java GUI Library being used.

2.6 Assumptions and dependencies

The user setting the questions needs some basic knowledge on simple programming. The user doesn't require an online MCQ based application.

3. External Interface Requirements

3.1 User interfaces

Buttons for navigation, Radio buttons for choice selection, use of Tab key for navigation applicable, Textboxes for Data Entry

3.2 Hardware interfaces

Standard Mouse and Keyboard

3.3 Software interfaces

N/A

3.4 Communication protocols and interfaces

N/A

4. System Features

The following features need to be provided to meet the functional requirements of this application under development:

4.1 Bookmarking a Question

4.1.1 Description and priority

This feature needs to be implemented and has a high priority as it will be essential for the users to come back to the questions which have been marked for review without wasting a lot of time.

4.1.2 Action/result

When the Bookmark button is pressed, the question number is saved and an additional button is shown at the side of the screen for easy access. When the new button is clicked, the question marked for review is shown.

4.1.3 Functional requirements

Nil, already available in the library itself.

4.2 Displaying Correct Answers

4.2.1 Description and priority

This button will be used to show the user the score after completion of the quiz, which will help them evaluate themselves instantly.

4.2.2 Action/result

When the Result button will be clicked, a dialog box will be shown displaying the number of current answers attempted.

4.2.3 Functional requirements

Nil, already available in the library itself.

5. Other Nonfunctional Requirements

5.1 Performance requirements

A High DPI Mouse for accuracy while pressing the Radio Buttons, More RAM and CPU Cores for Faster initial launch of the software.

5.2 Safety requirements

N/A

5.3 Security requirements

N/A

5.4 Software quality attributes

The user setting the questions must have basic Java programming knowledge to maintain the quality of the software.

6. Other Requirements

Appendix A: Terminology/Glossary/Definitions list

GUI – Graphical User Interface

AWT – Abstract Window Toolkit

DPI – Dots per Inch

MCQ – Multiple Choice Questions

OS – Operating System

RAM – Random Access Module

Appendix B: To be determined

Nil

CONCLUSION: Hence, we have prepared an SRS document in IEEE f

EXPERIMENT NO. 4

TITLE: Perform Structured data flow analysis.

AIM: To Perform Structured data flow analysis.

OBJECTIVE:

• The basic goal of SA/SD is to improve quality and reduce the risk of system failure.

• It establishes concrete management specifications and documentation.

• It focuses on the solidity, pliability, and maintainability of the system.

THEORY:

Structured analysis is a software engineering technique that uses graphical diagrams to develop and portray system specifications that are easily understood by users. These diagrams describe the steps that need to occur and the data required to meet the design function of a particular software. This type of analysis mainly focuses on logical systems and functions, and aims to convert business requirements into computer programs and hardware specifications. Structured analysis is a fundamental aspect of system analysis.

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.

Basically, the approach of SA/SD is based on the Data Flow Diagram. It is easy to understand SA/SD but it focuses on well-defined system boundary whereas the JSD approach is too complex and does not have any graphical representation.

SA/SD is combined known as SAD and it mainly focuses on the following 3 points:

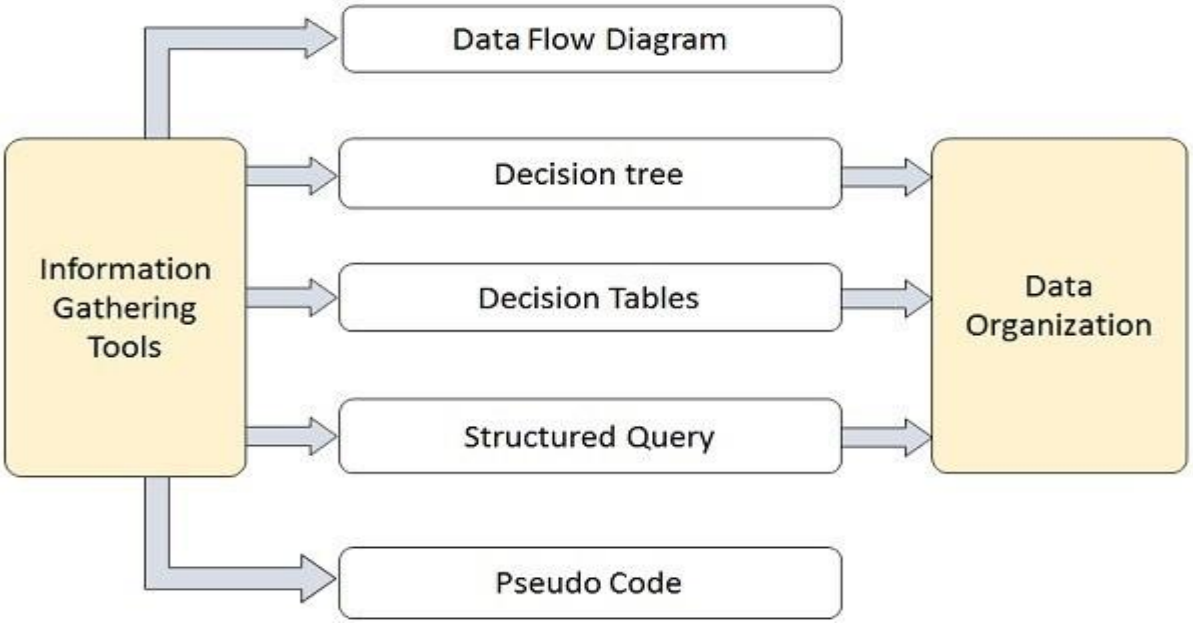
1. System
2. Process
3. Technology

SA/SD involves 2 phases:

1. Analysis Phase: It uses Data Flow Diagram, Data Dictionary, State Transition diagram and ER

diagram.

2. Design Phase: It uses Structured Query and Pseudo Code



Analysis Phase:

Analysis Phase involves data flow diagram, data dictionary, state transition diagram, and entity-relationship diagram.

a) Data Flow Diagram:

In the data flow diagram, the model describes how the data flows through the system. We can incorporate the Boolean operators and & or link data flow when more than one data flow may be input or output from a process.

For example, if we have to choose between two paths of a process we can add an operator or and if two data flows are necessary for a process we can add an operator. The input of the process "check-order" needs the credit information and order information whereas the output of the process would be a cash-order or a good-credit-order.

b) Data Dictionary:

The content that is not described in the DFD is described in the data dictionary. It defines the data store and relevant meaning. A physical data dictionary for data elements that flow between processes, between entities, and between processes and entities may be included. This would also include descriptions of data elements that flow external to the data stores.

A logical data dictionary may also be included for each such data element. All system names, whether they are names of entities, types, relations, attributes, or services, should be entered in the dictionary.

c) State Transition Diagram:

State transition diagram is similar to the dynamic model. It specifies how much time the function will take to execute and data access triggered by events. It also describes all of the states that an object can have, the events under which an object changes state, the conditions that must be fulfilled before the transition will occur and the activities were undertaken during the life of an object.

d) ER Diagram:

ER diagram specifies the relationship between data store. It is basically used in database design. It basically describes the relationship between different entities.

1. Design Phase:

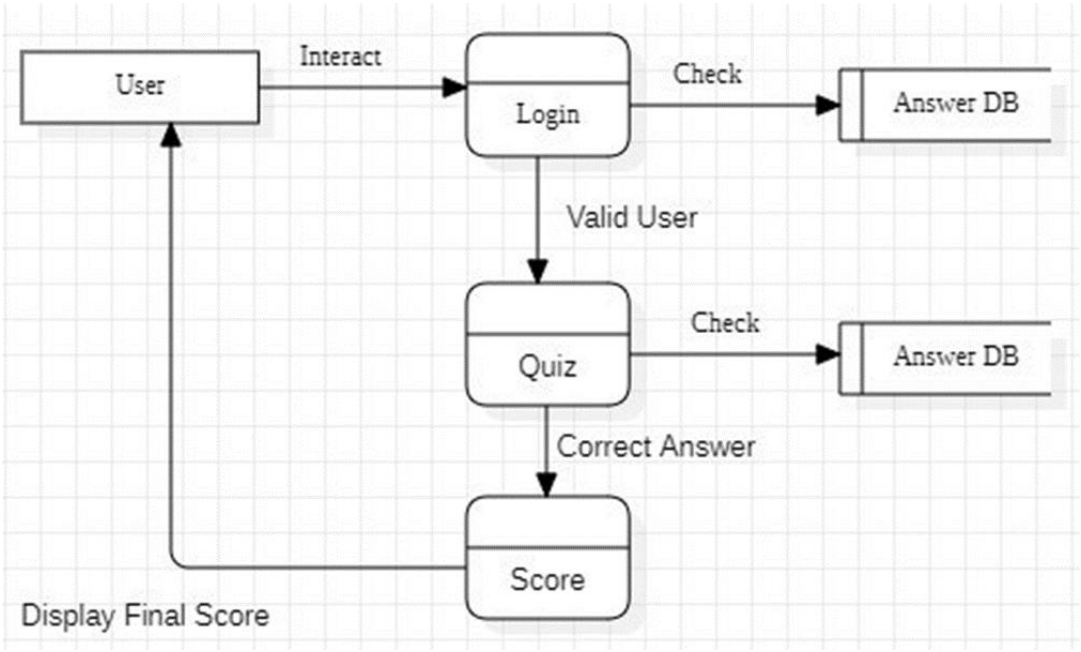
Design Phase involves structure chart and pseudocode.

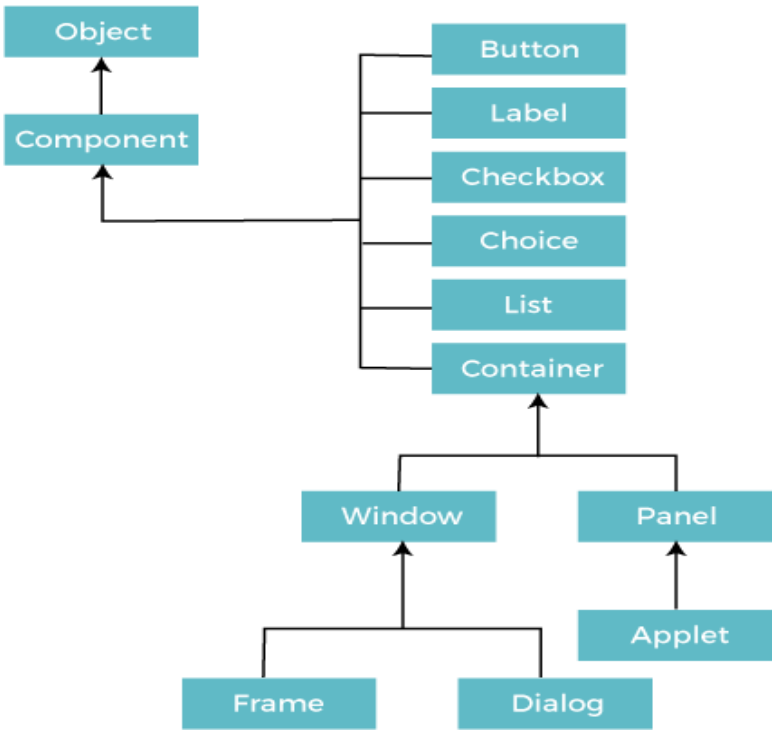
a) Structure Chart:

It is created by the data flow diagram. Structure Chart specifies how DFS's processes are grouped into tasks and allocate to the CPU. The structured chart does not show the working and internal structure of the processes or modules and does not show the relationship between data or data-flows. Similar to other SASD tools, it is time and cost-independent and there is no error-checking technique associated with this tool. The modules of a structured chart are arranged arbitrarily and any process from a DFD can be chosen as the central transform depending on the analysts' own perception. The structured chart is difficult toamend, verify, maintain, and check for completeness and consistency.

b) Pseudo Code:

It is the actual implementation of the system. It is an informal way of programming that doesn't require any specific programming language or technology.





There are 2 functions of this product:

- To create a quiz.
- To attempt a quiz.

The User interacts with Login UI and then attempt the quiz while the functions will validation from the Database.

CONCLUSION: Hence, we have studied structured data flow analysis using Star UML tool

EXPERIMENT NO. 5

TITLE: Use of metrics to estimate the cost of Software Project.

AIM: To us metrics to estimate the cost of Software Project.

OBJECTIVE: Categorize projects using COCOMO, and estimate effort and development time required for a project.

PRE – CONCEPT: Types of software such as Organic, Semidetached, Embedded.

NEW CONCEPT: Cost Estimation using Basic COCOMO Model and Intermediate COCOMO Model.

THEORY:

Project Estimation Techniques:

A software project is not just about writing a few hundred lines of source code to achieve a particular objective. The scope of a software project is comparatively quite large, and such a project could take several years to complete. However, the phrase "quite large" could only give some (possibly vague) qualitative information.

As in any other science and engineering discipline, one would be interested to measure how complex a project is. One of the major activities of the project planning phase, therefore, is to estimate various project parameters in order to take proper decisions. Some important project parameters that are estimated include:

Project size: What would be the size of the code written say, in number of lines, files, modules?

Cost: How much would it cost to develop a software? A software may be just pieces of code, but one has to pay to the managers, developers, and other project personnel.

Duration: How long would it be before the software is delivered to the clients?

Effort: How much effort from the team members would be required to create the software?

In this experiment we will focus on COCOMO method for estimating project metrics.

COCOMO: COCOMO (Constructive Cost Model) was proposed by Boehm. According to him, there could be three categories of software projects: organic, semidetached, and embedded. The classification is done considering the characteristics of the software, the development team and environment. These product classes typically correspond to application, utility and system programs, respectively. Data processing programs could be considered as application programs. Compilers, linkers, are examples of utility programs. Operating systems, real-time system programs are examples of system programs. One could easily apprehend that it would take much more time and effort to develop an OS than an attendance management system.

The concept of organic, semidetached, and embedded systems is described below:

Organic: A development project is said to be of organic type, if the project deals with developing a well understood application, the development team is small, the team members have prior experience in working

with similar types of projects.

Semidetached: A development project can be categorized as semidetached type, if the team consists of some experienced as well as inexperienced staff, Team members may have some experience on the type of system to be developed.

Embedded: Embedded type of development project are those, which aims to develop software strongly relatedto machine hardware, Team size is usually large.

Boehm suggested that estimation of project parameters should be done through three stages:

1) Basic COCOMO Model

The basic COCOMO model helps to obtain a rough estimate of the project parameters. It estimates effort and time required for development in the following way:

Effort = a * (KDSI) b PM

T dev = 2.5 * (Effort)c Months

Where;

KDSI is the estimated size of the software expressed in Kilo Delivered Source Instructions a, b, c are constants determined by the category of software project.

Effort denotes the total effort required for the software development, expressed in person months (PM).

Tdev denotes the estimated time required to develop the software (expressed in months).

The value of the constants a, b, c is 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 allowed for a quick and rough estimate, but it resulted in a lack of accuracy. Boehm introduced an additional set of 15 predictors called "cost drivers" in the intermediate model to take account of the software development environment. Cost drivers are used to adjust the nominal cost of the project to the actual project environment, hence increasing the accuracy of the estimate.

Boehm suggested that estimation of project parameters should be done through three stages:

3) Basic COCOMO Model

The basic COCOMO model helps to obtain a rough estimate of the project parameters. It estimates effort and

time required for development in the following way:

Effort = a * (KDSI) b PM

T dev = 2.5 * (Effort)c Months

Where;

KDSI is the estimated size of the software expressed in Kilo Delivered Source Instructions a, b, c are constants determined by the category of software project.

Effort denotes the total effort required for the software development, expressed in person months (PM).

Tdev denotes the estimated time required to develop the software (expressed in months).

The value of the constants a, b, c is 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

4) Intermediate COCOMO Model

The basic COCOMO model allowed for a quick and rough estimate, but it resulted in a lack of accuracy. Boehm introduced an additional set of 15 predictors called "cost drivers" in the intermediate model to take account of the software development environment. Cost drivers are used to adjust the nominal cost of the project to the actual project environment, hence increasing the accuracy of the estimate. The cost drivers are grouped into four categories:

1. Product attributes

(a) Required software reliability

(b) Size of application database

(c) Complexity of the product

2. Hardware attributes

(a) Run-time performance constraints

(b) Memory constraints

(c) Volatility of the virtual machine environment

(d) Required turnabout time

3. Personnel attributes

(a) Analyst capability

(b) Software engineering capability

(c) Applications experience

(d) Virtual machine experience

(e) Programming language experience

4. Project attributes

(a) Use of software tools

(b) Application of software engineering methods

(c) Required development schedule

Each cost driver is rated for a given project environment. The rating uses a scale very low, low, nominal, high, very high, extra high which describes to what extends the cost driver applies to the project being estimated. Table given below, gives the multiplier values for the 15 cost drivers and their rating provided by Boehm.

The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4. The Intermediate COCOMO model equations take the form:

$E = a_i(KLoC)^b \bullet (EAF)$

$D = c_i(E)^d_i$

	Ratings					
Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Where co-efficients a_i and b_i are given below in table:

Software project	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

And c_i and d_i uses in the same way as in the Basic COCOMO.

5) Detailed COCOMO Model:

Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost drivers effect on each method of the software engineering process. The detailed model uses various effort multipliersfor 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

The effort is determined as a function of program estimate, and a set of cost drivers are given according toevery phase of the software lifecycle.

Advantages of COCOMO Model:

COCOMO is a simple model, and should help one to understand the concept of project metrics estimation.

Drawbacks of COCOMO Model:

COCOMO uses KDSI, which is not a proper measure of a program's size. Indeed, estimating the size of software is a difficult task, and any slight miscalculation could cause a large deviation in subsequent project estimates. Moreover, COCOMO was proposed in 1981 keeping the waterfall model of project life

cycle in mind [2]. It fails to address other popular approaches like prototype, incremental, spiral, agile models. Moreover, in present day a software project may not necessarily consist of coding of every bit of functionality. Rather, existing software components are often used and glued together towards the development of a new software. COCOMO is not suitable in such cases.

OUTPUT:

Our System Follows Intermediate Organic COCOMO Model as there are

500Kloc.Formula:$DE = EAF * 3.2 * (Size[Kloc])^{1.05}$

$$DE = EAF * 3.2 * 500^{1.05}$$

Software Product	a	b	Total
Orgamic	2.4	1.05	3.45
Semi-Detached	3	1.12	4.12
Embeded	3.6	1.2	4.8

Product attributes	Very Low	Low	Nominal	High	Very High	Extra High
Required software reliability	0.75	0.88	1	1.15	1.4	
Size of application database	0.94	1	1.08	1.16		
Complexity of the product	0.7	0.85	1	1.15	1.3	1.65

CONCLUSION: Hence, we have studied project cost and efforts estimation using COCOMO model.

EXPERIMENT NO. 6

TITLE: Scheduling & tracking of the Project.

AIM: To prepare a Scheduling & tracking of the Project Using Gantt Chart

OBJECTIVE:

- Better utilization of resources and optimal usage of allotted time for a project
- To check the feasibility of schedule and user requirement
- To determine the project constraints
- To identify roles and responsibility of team members

THEORY:

Time Scheduling:

Project Scheduling:

Scheduling for software development projects can be viewed from two different perspectives:

1. An end data for release of computer-based system has already been established. The software organizationis constrained distribute effort within the prescribed time frame.
2. The second view of software schedules assume that rough chronological bound have been discussed butthat end is set by the software engineering organization.

We scheduled our project on 11 July 2022. In 12 weeks, we carried out each and every process regarding software project. For requirement gathering we spent 1 week. After performing requirement gathering, we took 4 days for requirement analysis. For object-oriented analysis design, coding, testing we required a total of 4 - 5 weeks. Thus, for the completion of the whole project we required 3 months.

Tracking:

The project schedule provides a road map for a software project manager. If it has been properly developed, the project schedule defines the tasks and milestones that must tracked and controlled as the project proceeds.

Planner is a project management tool that allows the user to manage several aspects of a project, including among others planning of tasks using Gantt charts and allocation of resources.

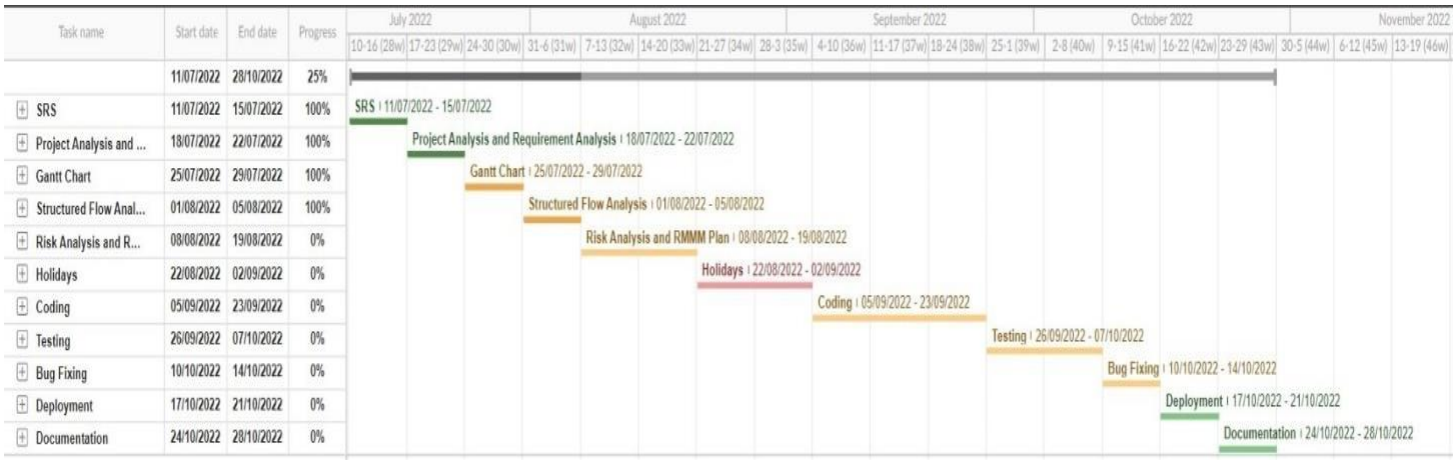
It is an excellent tool for managing the project schedule and resources, but in order to manage projects consistently and successfully, it's important to realize that Planner is a tool that should be used as part of a Project Management Methodology.

It's a tool that exists to support the process of managing a project, with the goal of delivering the goods on time and within budget.

OUTPUT:

When a new project is created, the first screen the user sees is the Gantt Chart for the new project. A Gantt chart is a graphical display of all the tasks that a project is composed of. For each task there is a graphical representation of the length of time it is planned to take. Use the "Insert task" icon from the toolbar on this screen to enter tasks for the new project. In the Gantt view, you may specify a name, a duration, and the amount of work effort required to complete the task. The difference between duration and work effort is that duration is the amount of calendar time allocated, whereas work effort is the amount of actual time it takes tocomplete the task (i.e. you have 3 days (duration) to write a letter, but it only takes on hour (work effort) to actually do it. All durations are entered in days, with fractions permitted (i.e. 3.4 days).

The left toolbar includes the "Tasks" icon. Click on it to display the Task view. Tasks are also shown on the Gantt view, but the Task view shows the task name, start date, finish date, and duration.



CONCLUSION: Hence, we have studied what is a Gantt Chart and learned how to prepare a project planusing Gantt Chart.

EXPERIMENT NO. 7

TITLE: Write a test cases for black box testing.

AIM: To perform black box testing for given scenario.

OBJECTIVE: To affirm the quality of software and deliver an error-free application.

THEORY:

When the tester has no idea of the internal working of the system which he is testing, that approach is called black box testing. In this case, the system under test is viewed as a "black box".

How to write Test Cases for Black Box Testing?

- The tester examines requirements and specifications of the system.
- The tester explores the system's UI and functionality to understand how the processes on the system are expected to work.
- Tester designs test cases with valid inputs and the corresponding expected outputs.
- Tester also includes some negative test cases with invalid inputs and expected outputs (error messages/program termination) as applicable.

Techniques of Black Box Testing

In case of black box testing, inputs to the test cases are the driving factor. Any one of the three techniques discussed below can be used to choose the inputs during the black box testing process

- **Boundary Value Analysis**: This approach is focused on testing the boundary values associated with the system. This approach aims at testing the boundaries of the input domain that have the highest probability of giving erroneous outputs.
- **Equivalence Class Partitioning**: In this approach, a limited set of functions is identified along with its corresponding valid and invalid inputs and expected outputs. This approach aims at identifying classes of errors and therefore reducing the number of test cases required.
- **Error Guessing**: An experienced tester most often uses this approach to first identify the defects and then develop corresponding test cases.

Black Box Testing Example
In this technique, we do not use the code to determine a test suite; rather, knowing the problem that we're trying to solve, we come up with four types of test data:

1. Easy-to-compute data
2. Typical data
3. Boundary / extreme data
4. Bogus data

For example, suppose we are testing a function that uses the quadratic formula to determine the two roots of a second-degree polynomial ax2+bx+c. For simplicity, assume that we are going to work only with real numbers, and print an error message if it turns out that the two roots are complex numbers (numbers involving the square root of a negative number).

We can come up with test data for each of the four cases, based on values of the polynomial's discriminant (b^2-4ac):

Easy data (discriminant is a perfect square):

a	b	c	Roots
1	2	1	-1, -1
1	3	2	-1, -2

Typical data (discriminant is positive):

a	b	c	Roots
1	4	1	-3.73205, -0.267949
2	4	1	-1.70711, -0.292893

Boundary / extreme data (discriminant is zero):

a	b	c	Roots
2	-4	2	1, 1
2	-8	8	2, 2

Bogus data (discriminant is negative, or a is zero):

a	b	c	Roots
1	1	1	square root of negative number
0	1	1	division by zero

DEPARTMENT OF COMPUTER ENGINEERING

OUTPUT:

Test Case ID	V_017	**Test Case Description**	Test the Login Functionality in Banking		
Created By	Valay	**Reviewed By**	Anuj	**Version**	1.0

QA Tester's Log Review comments from Bill incorporate in version 2.0

Tester's Name	Valay	**Date Tested**	20/09/2022	**Test Case (Pass/Fail/Not Executed)**	Pass

Sr. No.	**Prerequisites:**
1	Access to Login Screen
2	
3	
4	

Sr. No.	**Test Data**
1	User ID = ValayS
2	Password = v@l@y
3	
4	

Test Scenario Verify on entering valid User ID and Password, the customer can login

Step No.	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Navigate to open the application	Application should open	As Expected	Pass
2	Enter User ID & Password	Credential can be entered	As Expected	Pass
3	Click Submit	Customer is logged in	As Expected	Pass

CONCLUSION: Hence, Black box testing focuses on results and studied as the functionality of the system from a user objective.

EXPERIMENT NO. 8

TITLE: Write test cases for white box testing.

AIM: To develop test cases for white box testing.

OBJECTIVE:

- In the context of white box testing strategy, differentiate between stronger testing and complementary testing.
- Design statement coverage test cases for a code segment.
- Design branch coverage test cases for a code segment.
- Design condition coverage test cases for a code segment.
- Design path coverage test cases for a code segment.
- Draw control flow graph for any program.
- Identify the linear independent paths.
- Compute cyclomatic complexity from any control flow graph.

THEORY:

WHITE BOX TESTING

One white-box testing strategy is said to be stronger than another strategy, if all types of errors detected by the first testing strategy is also detected by the second testing strategy, and the second testing strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, then they are called complementary. The concepts of stronger and complementary testing are schematically illustrated in fig. 6.1

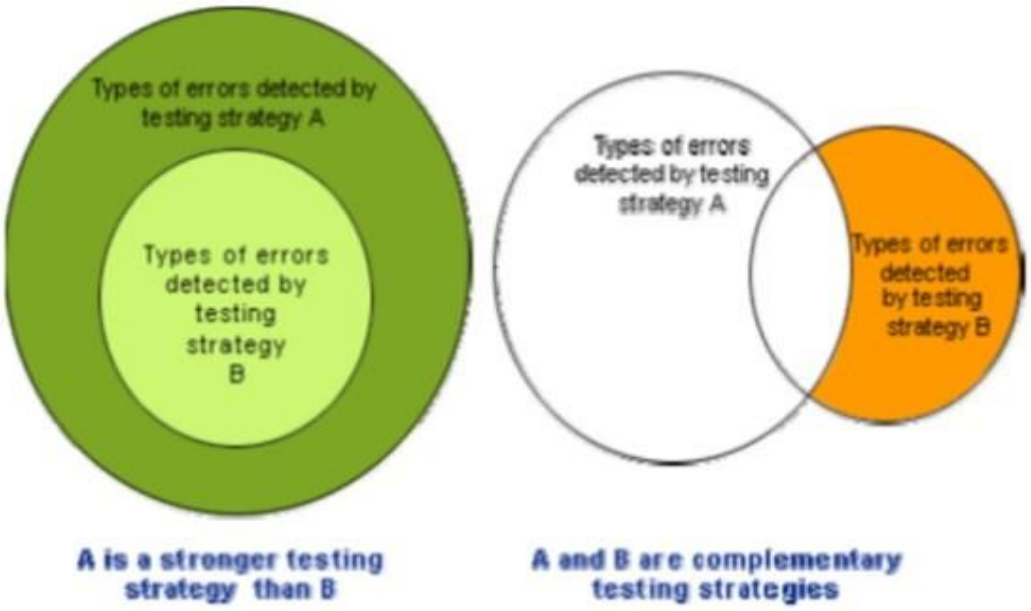


Fig: 8.1: Stronger and Complimentary Testing Strategies

Statement coverage

The statement coverage strategy aims to design test cases so that every statement in a program is executed at least once. The principal idea governing the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement. Unless a statement is executed, it is very difficult to observe whether it causes failure due to some illegal memory access, wrong result computation, etc. However, executing some statement once and observing that it behaves properly for that input value is no guarantee that it will behave correctly for all input values. In the following, designing of test cases using the statement coverage strategy have been shown.

Algorithm: Algorithm to check whether the entered number is a palindrome or not:

Algorithm:

1. Start
2. Create an instance of the Scanner class and declare a variable.
3. Ask the user to initialize the variable.
4. Declare a variable to store the reverse number and initialize it to 0.
5. Use a while loop for the same.
6. Check whether the reversed number is the same as the original number or not.
7. If the same, then print it as a palindrome number.
8. If not the same, then print it as not a palindrome number.
9. Display the result.
10. Stop.

By choosing the test set **{(x = 2001), (x = 1001)}**, we can exercise the program such that all statements are executed at least once.

Branch coverage

In the branch coverage-based testing strategy, test cases are designed to make each branch condition to assume true and false values in turn. Branch testing is also known as **edge testing** as in this testing scheme, each edge of a program's control flow graph is traversed at least once.

It is obvious that branch testing guarantees statement coverage and thus is a stronger testing strategy compared to the statement coverage-based testing. For palindrome checking algorithm, the test cases for branch coverage can be **{(x = 2001), (x = 1001)}.**

Condition coverage

In this structural testing, test cases are designed to make each component of a composite conditional expression to assume both true and false values. For example, in the conditional expression (c1.or.c2), the components c1 and c2 are each made to assume both true and false values. Branch testing is probably the simplest condition testing strategy where only the compound conditions appearing in the different branch statements are made to assume the true and false values. Thus, condition testing is a stronger testing strategy

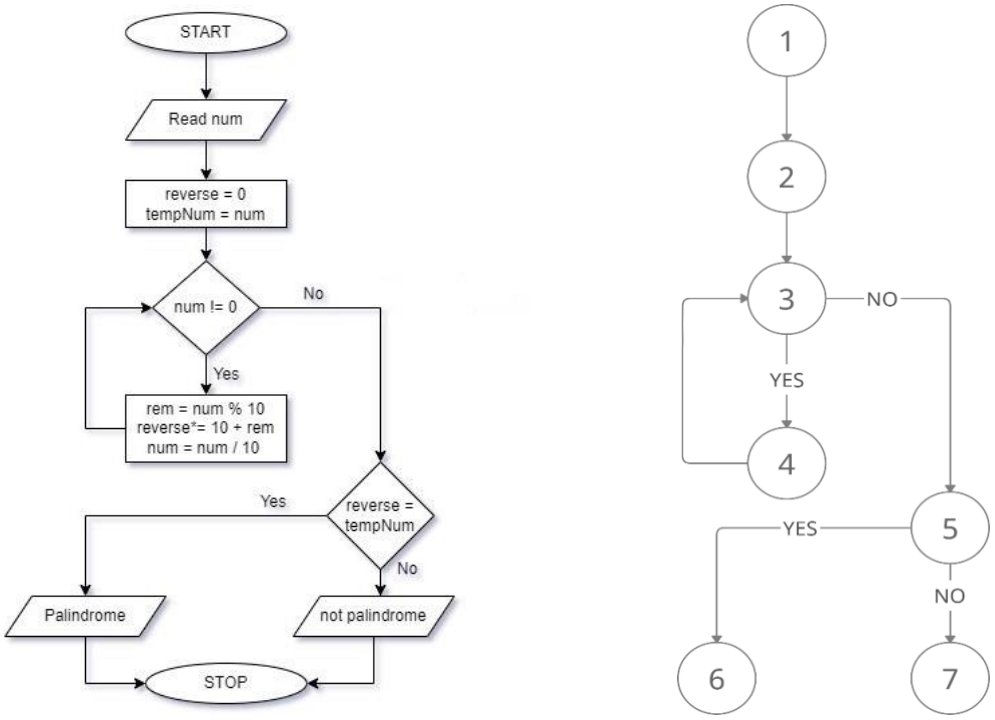
than branch testing and branch testing is stronger testing strategy than the statement coverage-based testing. Thus, for condition coverage, the number of test cases increases exponentially with the number of component conditions. Therefore, a condition coverage-based testing technique is practical only if n (the number of conditions) is small.

Path coverage

The path coverage-based testing strategy requires us to design test cases such that all linearly independent paths in the program are executed at least once. A linearly independent path can be defined in terms of the control flow graph **(CFG) of a program.**

Control Flow Graph (CFG)

A control flow graph describes the sequence in which the different instructions of a program get executed. In other words, a control flow graph describes how the control flows through the program.



In order to draw the control flow graph of a program, all the statements of a program must be numbered first. The different numbered statements serve as nodes of the control flow graph. An edge from one node to another node exists if the execution of the statement representing the first node can result in the transfer of control to the other node.

The CFG for any program can be easily drawn by knowing how to represent the sequence, selection, and iteration type of statements in the CFG. After all, a program is made up from these types of statements. Figure above summarizes how the CFG for these two types of statements can be drawn. It is important to note that for the iteration type of constructs such as the while construct, the loop condition is tested only at the beginning of the loop and therefore the control flow from the last statement of the loop is always to the top of the loop. Using these basic ideas, the CFG of palindrome checking algorithm can be drawn as shown in figure above.

Path

A path through a program is a node and edge sequence from the starting node to a terminal node of the control flow graph of a program. There can be more than one terminal node in a program. Writing test cases to cover all the paths of a typical program is impractical. For this reason, the path-coverage testing does not require coverage of all paths but only coverage of linearly independent paths.

Linearly independent path

A linearly independent path is any path through the program that introduces at least one new edge that is not included in any other linearly independent paths. If a path has one new node compared to all other linearly independent paths, then the path is also linearly independent. This is because, any path having a new node automatically implies that it has a new edge. Thus, a path that is sub path of another path is not considered to be a linearly independent path.

Control flow graph

In order to understand the path coverage-based testing strategy, it is very much necessary to understand the control flow graph (CFG) of a program. Control flow graph (CFG) of a program has been discussed earlier.

Linearly independent path

The path-coverage testing does not require coverage of all paths but only coverage of linearly independent paths. Linearly independent paths have been discussed earlier.

Cyclomatic complexity

For more complicated programs it is not easy to determine the number of independent paths of the program. McCabe's cyclomatic complexity defines an upper bound for the number of linearly independent paths through a program. Also, the McCabe's cyclomatic complexity is very simple to compute. Thus, the McCabe's cyclomatic complexity metric provides a practical way of determining the maximum number of linearly independent paths in a program. Though the McCabe's metric does not directly identify the linearly independent paths, but it informs approximately how many paths to look for.

There are **three** different ways to compute the cyclomatic complexity. The answers computed by the three methods are guaranteed to agree.

Method 1:

Given a control flow graph G of a program, the cyclomatic complexity V(G) can be computed as:

V(G) = E – N + 2

where N is the number of nodes of the control flow graph and E is the number of edges in the control flow graph.

For the CFG of example shown in figure above, E = 7 and N = 6. Therefore, the cyclomatic complexity = 7 – 6 + 2 = **3**.

Method 2:

An alternative way of computing the cyclomatic complexity of a program from an inspection of its control flow graph is as follows:

V(G) = Total number of bounded areas + 1

In the program's control flow graph G, any region enclosed by nodes and edges can be called as a bounded area. This is an easy way to determine the McCabe's cyclomatic complexity. But, what if the graph G is not planar, i.e. however you draw the graph, two or more edges intersect? Actually, it can be shown that structured programs always yield planar graphs. But, presence of GOTO's can easily add intersecting edges. Therefore, for non-structured programs, this way of computing the McCabe's cyclomatic complexity cannot be used.

The number of bounded areas increases with the number of decision paths and loops. Therefore, the McCabe's metric provides a quantitative measure of testing difficulty and the ultimate reliability. For the CFG example shown in figure, from a visual examination of the CFG the number of bounded areas is 2. Therefore, the cyclomatic complexity, computing with this method is also **2 + 1 = 3**. This method provides a very easy way of computing the cyclomatic complexity of CFGs, just from a visual examination of the CFG. On the other hand, the other method of computing CFGs is more amenable to automation, i.e. it can be easily coded into a program which can be used to determine the cyclomatic complexities of arbitrary CFGs.

Method 3:

The cyclomatic complexity of a program can also be easily computed by computing the number of decision statements of the program. If N is the number of decision statement of a program, then the McCabe's metric is equal to N+1.

(2 + 1 = 3) (For Above Example)

Data flow-based testing

Data flow-based testing method selects test paths of a program according to the locations of the definitions and uses of different variables in a program.

For a statement numbered S, let

DEF(S) = {X/statement S contains a definition of X}, and

USES(S) = {X/statement S contains a use of X}

For the statement S: a = b + c; DEF(S) = {a}. USES(S) = {b, c}. The definition of variable X at statement S is said to be live at statement S1, if there exists a path from statement S to statement S1 which does not contain any definition of X.

The definition-use chain (or DU chain) of a variable X is of form [X, S, S1], where S and S1 are statement numbers, such that X ∈ DEF(S) and X ∈ USES(S1), and the definition of X in the statement S is live at statement S1. One simple data flow testing strategy is to require that every DU chain be covered at least once. Data flow testing strategies are useful for selecting test paths of a program containing nested if and loop statements.

Advantages of White Box Testing:

- Forces test developer to reason carefully about implementation.
- Reveals errors in "hidden" code.
- Spots the Dead Code or other issues with respect to best programming practices.

Disadvantages of White Box Testing:

- Expensive as one has to spend both time and money to perform white box testing.
- Every possibility that few lines of code are missed accidentally.

In-depth knowledge about the programming language is necessary to perform white box testing.

PROGRAM:

```
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner s = new Scanner(System.in);
       System.out.println("Enter the number: ");
       int num = s.nextInt();
       int r,sum = 0;
       int temp = num;
       while(num > 0)
       {
       r = num%10;
       sum = (sum*10) + r;
       num = num/10;
       }
       if(temp == sum)
       System.out.println("The entered number "+temp+" is a palindrome number.");
       else
       System.out.println("The entered number "+temp+" is not a palindrome number.");
   }

}
```

OUTPUT:

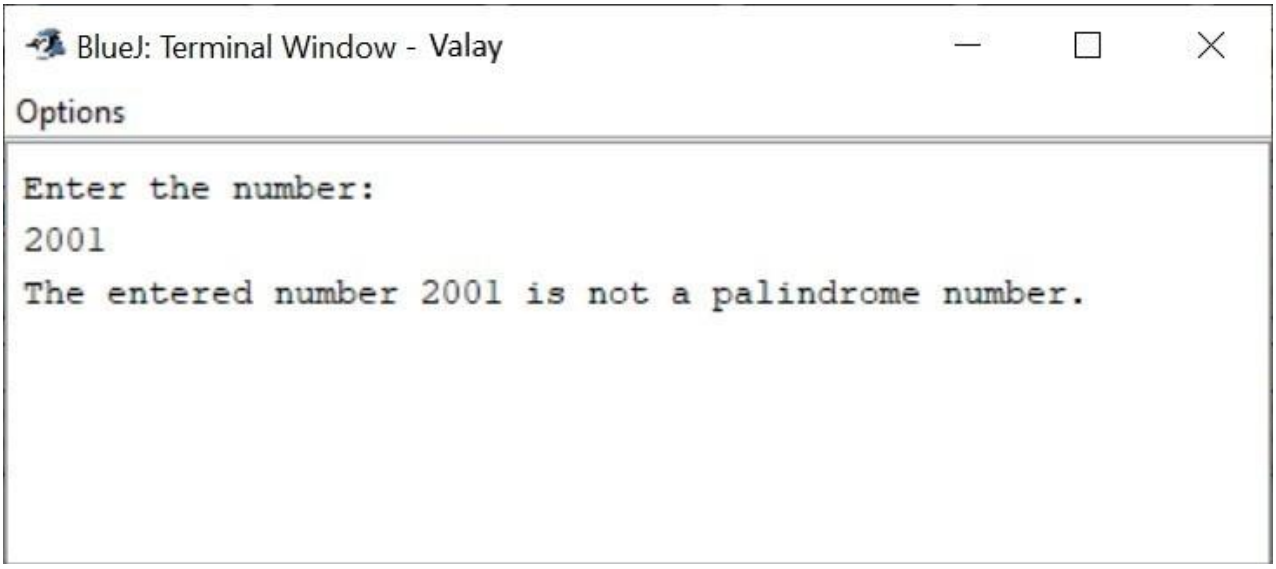
Statement Coverage – we would need only one test case to check all the lines of code. If we consider **TestCase_01 to be (num = 2001) and TestCase_02 to be (num = 1001)** then all the lines of code will be executed.

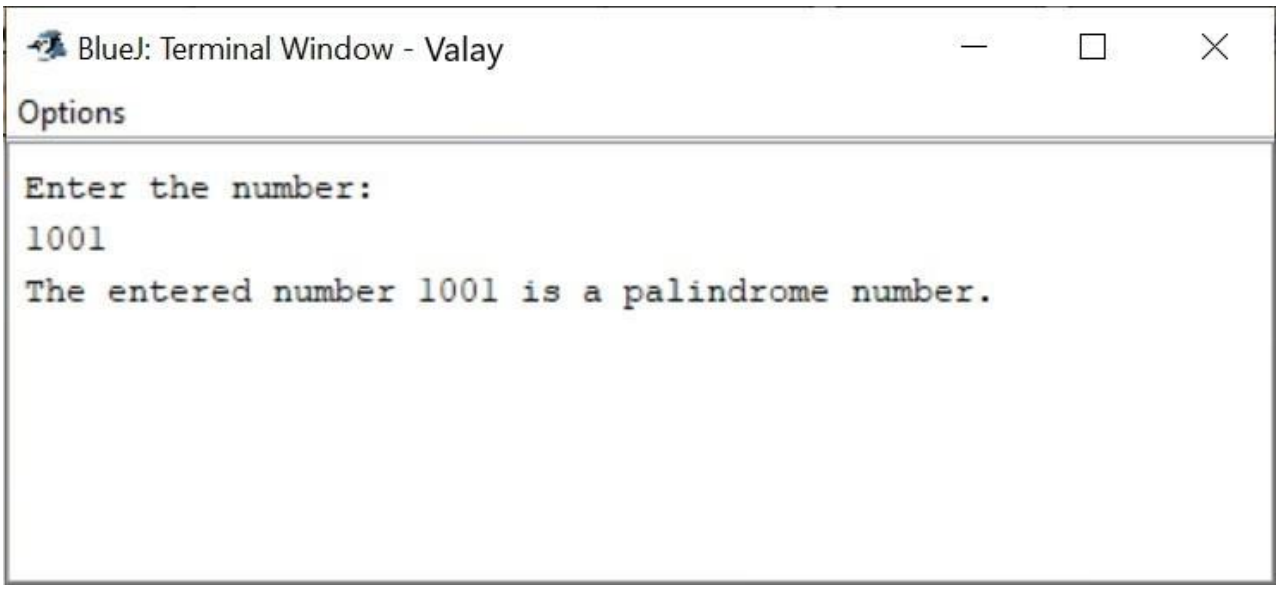
Branch coverage - we would require same set of test cases **TestCase_01 to be (num = 2001) and TestCase_02 to be (num = 1001,** to complete testing of this pseudo code.With this, we can see that each and every line of code is executed at least once.

Path Coverage- It is used to test the complex code snippets, which basically involves loop statements or combination of loops and decision statements. So, in order to have the full coverage, we would need many test cases (But as the code is a simple one so the test cases used in Branch Coverage are sufficient)

(CFG IS SHOWN IN THEORY SECTION)

TestCase_01 to be (num = 2001) and TestCase_02 to be (num = 1001). With this, we can see that each and every line of code is executed at least once.





CONCLUSION: White box testing is used for security and effective. It should follow a risk-based approach to balance the testing effort with consequences of software failure.

EXPERIMENT NO. 9

TITLE: Preparation of Risk Mitigation, Monitoring Analysis and Management Plan (RMMM)

AIM: To Perform Risk Analysis and Prepare Risk Assessment Template.

OBJECTIVE: To Study RIS and CMMM plan.

THEORY:

Risk Analysis:

There are quite different types of risk analysis that can be used. Basically, risk analysis is used to identify the high risk elements of a project in software engineering. Also, it provides ways of detailing the impact of risk mitigation strategies. Risk analysis has also been found to be most important in the software design phase to evaluate criticality of the system, where risks are analyzed and necessary counter measures are introduced. The main purpose of risk analysis is to understand risks in better ways and to verify and correct attributes. A successful risk analysis includes important elements like problem definition, problem formulation, data collection.

Risk Assessment:

Risk assessment is another important case that integrates risk management and risk analysis. There are many risk assessment methodologies that focus on different types of risks. Risk assessment requires correct explanations of the target system and all security features. It is important that a risk referent levels like performance, cost, support and schedule must be defined properly for risk assessment to be useful.

Risk Classification:

The key purpose of classifying risk is to get a collective viewpoint on a group of factors. These are the types of factors which will help project managers to identify the group that contributes the maximum risk. A best and most scientific way of approaching risks is to classify them based on risk attributes. Risk classification is considered as an economical way of analyzing risks and their causes by grouping similar risks together into classes. Software risks could be classified as internal or external. Those risks that come from risk factors within the organization are called internal risks whereas the external risks come from out of the organization and are difficult to control. Internal risks are project risks, process risks, and product risks. External risks are generally business with the vendor, technical risks, customers' satisfaction, political stability and so on. In general, there are many risks in the software engineering which is very difficult or impossible to identify all of them. Some of most important risks in software engineering project are categorized as software requirement

risks, software cost risks, software scheduling risk, software quality risks, and software business risks. These risks are explained detail below.

Software Requirement Risks:

- Lack of analysis for change of requirements.
- Change extension of requirements
- Lack of report for requirements
- Poor definition of requirements
- Ambiguity of requirements
- Change of requirements
- Inadequate of requirements
- Impossible requirements
- Invalid requirements

Software Cost Risks:

- Lack of good estimation in projects
- Unrealistic schedule
- The hardware does not work well

Risk identification:

- Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading etc.)
- By identifying known and predictable risks the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

Types of risks:

- **Generic Risks**: Potential threats to every software product.
- **Product-Specific Risks:** Any special characteristics of the product that may threaten the project plan

- **Product size:** Risks associated with the overall size of the software to be built or modified
- **Business impact:** Risks associated with constraints imposed by management or the marketplace.

- **Customer characteristics:** Risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
- **Staff size and experience:** Risks associated with the overall technical and project experience of the software engineers who will do the work
- **Process definition:** Risks associated with the degree to which the software process has been defined and is followed by the development organization
- **Development environment:** Risks associated with the availability and quality of the tools to be used to build the product
- **Technology to be built:** Risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system

Risk Projection:

- Also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur.
- The project planner, along with other managers and technical staff, performs four risk projection activities:

1) Establish a scale that reflects the perceived likelihood of a risk,
2) Delineate the consequences of the risk,
3) Estimate the impact of the risk on the project and the product, and
4) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

RISK ID	RISK	RISK TYPE	PROBABILITY (%)	IMPACT
1	Server Down	Technical	25	1
2	Login problem	Technical	20	4
3	Database Failure	Technical	35	1
4	Memory Problem	Technical	10	2
5	Software Budget Risk	Business	40	3
6	Resource Risk	Technical	60	1
7	No Market Response	Market	30	2
8	Concurrency Problem	Technical	60	1
9	Database Not Updated	Technical	25	2
10	No Periodic Backup	Technical	15	1

DEPARTMENT OF COMPUTER ENGINEERING

RISK NATURE	RISK IMPACT VALUE
Catastrophic	1
Critical	2
Marginal	3
Negligible	4

RISK ID	RISK	RISK TYPE	PROBABILITY (%)	IMPACT
6	Resource Risk	Technical	60	1
8	Concurrency Problem	Technical	60	1
5	Software Budget Risk	Business	40	3
3	Database failure	Technical	35	1
7	No Market Response	Market	30	2
1	Server Down	Technical	25	1
9	Database Not Updated	Technical	25	2
2	Login Problem	Technical	20	4
10	No Periodic Backup	Technical	15	1
4	Memory Problem	Technical	10	2

Risk Mitigation, Monitoring and Management (RMMM) Plan:

Risk id: 6	*Resource Risk*	**Probability:** 60%	**Impact: 1**
Description: Work allotted to a particular group is not done up to the mark or in time limit given to them and because of this even condition like deadlock may rise up.			
Refinement/Context: Miscommunication among staff and inexperienced staff.			
Mitigation and Monitoring: Check the status of the work done by the team periodically and make moves accordingly.			
Management: Experienced and skilled staff should be recruited.			

Risk id: 8	*Concurrency problem*	**Probability:** 60 %	**Impact:** 1
Description: When multiple users try to purchase same product at a time and this may lead to inconsistent state of the software			
Refinement/Context: Limited stock of the particular product and not keeping a check on the quantity of product being sold the most.			
Mitigation and Monitoring: Timely check which products are being sold and accordingly increase the availability.			
Management: Give priority to the older customer			

Risk id: 5	*Software Budget Risk*	**Probability:** 40 %	**Impact:** 3
Description: Inaccurate estimation done for various requirements and softwares for the project as well as for the technical staff needed.			
Refinement/Context: Lack of capital as compared to the required investment. Sudden and unexpected increase in the requirement of capital.			
Mitigation and Monitoring: Train the available staff. Checkout if sudden requirement is important and necessary and make the arrangements.			
Management: Increase the capital in advance as well as while recruiting select experienced staff.			

Risk id: 3	*Database Failure*	**Probability:** 35 %	**Impact: 1**
Description: Database contains personal information about customers and all staff. It also contains billing related information of all the customers.			
Refinement/Context: Unauthorized users accessing database.			
Mitigation and Monitoring: Providing access to authorized users by means of password login.			
Management: Use backup database in place of original one.			

Risk id: 7	*No Market Response*	**Probability: 30 %**	**Impact: 2**
Description: Site is not known to many people due to improper marketing or customer is not happy or satisfied with the products. May be site is not user friendly.			
Refinement/Context: Keeping Old fashioned products. Site is not user friendly.			
Mitigation and Monitoring: Be at par with the market price and customer's demands. Make proper marketing strategy.			
Management: Provide discounts to attract customer's attention. Keep updating new arrivals. Make the site user Friendly.			

OUTPUT:

RISK ID	RISK	RISK TYPE	PROBABILITY (%)	IMPACT
3	Software Budget Risk	Business	40	3
2	Database failure	Technical	35	1
1	No Market Response	Market	30	2

DEPARTMENT OF COMPUTER ENGINEERING

Risk id: 1	No Market Response	**Probability: 30 %**	**Impact: 2**
Description: System is not known to many people due to improper marketing or customer is not happy or satisfied with the product. May be UI is not user-friendly.			
Refinement/Context: Site is not user friendly.			
Mitigation and Monitoring: Be at par with the market price and customer's demands. Make proper marketing strategy.			
Management: Provide discounts to attract customer's attention. Make the site user Friendly.			

Risk id: 2	Database Failure	**Probability:** 35 %	**Impact: 1**
Description: Database contains personal information about customers and all staff. It also contains billing related information of all the customers.			
Refinement/Context: Unauthorized users accessing database.			
Mitigation and Monitoring: Providing access to authorized users by means of password login.			
Management: Use backup database in place of original one.			

Risk id: 3	Software Budget Risk	**Probability:** 40 %	**Impact:** 3
Description: Inaccurate estimation done for various requirements and softwares for the project as well as for the technical staff needed.			
Refinement/Context: Lack of capital as compared to the required investment. Sudden and unexpected increase in the requirement of capital.			
Mitigation and Monitoring: Train the available staff. Checkout if sudden requirement is important and necessary and make the arrangements.			

CONCLUSION: Hence, we have studied Perform Risk Mitigation, Monitoring Analysis and Management Plan (RMMM)

EXPERIMENT NO. 10

TITLE: Version control of the Project

AIM: To Perform Version control of the Project

OBJECTIVE: To Study Version control of the Project.

THEORY: