D Y PATIL
RAMRAO ADIK INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Department of Computer Engineering

Lab Manual

Third Year Semester-VI

Subject: Software Engineering

Institutional Vision, Mission and Quality Policy

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

RAIT's firm belief in new form of engineering education that lays equal stress on academics and leadership building extracurricular skills has been a major contribution to the success of RAIT as one of the most reputed institution of higher learning. The challenges faced by our country and world in the 21 Century needs a whole new range of thought and action leaders, which a conventional educational system in engineering disciplines are ill equipped to produce. Our reputation in providing good engineering education with additional life skills ensure that high grade and highly motivated students join us. Our laboratories and practical sessions reflect the latest that is being followed in the Industry. The project works and summer projects make our students adept at handling the real life problems and be Industry ready. Our students are well placed in the Industry and their performance makes reputed companies visit us with renewed demands and vigour.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

RAIT's Mission is to produce engineering and technology professionals who are innovative and inspiring thought leaders, adept at solving problems faced by our nation and world by providing quality education.

The Institute is working closely with all stake holders like industry, academia to foster knowledge generation, acquisition, dissemination using best available resources to address the great challenges being faced by our country and World. RAIT is fully dedicated to provide its students skills that make them leaders and solution providers and are Industry ready when they graduate from the Institution.

Quality Policy

ज्ञानधीनं जगत् सर्वम।

Knowledge is supreme.

Our Quality Policy

Itisourearnestendeavourtoproducehighqualityengineeringprofessionalswhoare innovative and inspiring, thought and action leaders, competent to solve problems facedbysociety,nationandworldatlargebystrivingtowardsveryhighstandardsin learning,teaching and training methodologies.

Our Motto: If it is not of quality, it is NOT RAIT!

Departmental Vision, Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering. To provide the diverse platforms of sports, technical, cocurricular and extracurricular activities for the overall development of student with ethical attitude. To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service. To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental Program Educational Objectives (PEOs)

1. **Learn and Integrate**

   To provide Computer Engineering students with a strong foundation in the mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze engineering problems and to prepare them for graduate studies.

2. **Think and Create**

   To develop an ability to analyze the requirements of the software and hardware, understand the technical specifications, create a model, design, implement and verify a computing system to meet specified requirements while considering real-world constraints to solve real world problems.

3. **Broad Base**

   To provide broad education necessary to understand the science of computer engineering and the impact of it in a global and social context.

4. **Techno-leader**

   To provide exposure to emerging cutting edge technologies, adequate training & opportunities to work as teams on multidisciplinary projects with effective communication skills and leadership qualities.

5. **Practice citizenship**

   To provide knowledge of professional and ethical responsibility and to contribute to society through active engagement with professional societies, schools, civic organizations or other community activities.

6. **Clarify Purpose and Perspective**

   To provide strong in-depth education through electives and to promote student awareness on the life-long learning to adapt to innovation and change, and to be successful in their professional work or graduate studies.

Departmental Program Outcomes (POs)

PO1 Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2**.** Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3**.** 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.

PO4. 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.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6**.** 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.

PO7**.** 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.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. 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.

PO11. 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.

PO12. 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)

PSO1: To build competencies towards problem solving with an ability to understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.

PSO2: To build appreciation and knowledge acquiring of current computer techniques with an ability to use skills and tools necessary for computing practice.

PSO3: To be able to match the industry requirements in the area of computer science and engineering. To equip skills to adopt and imbibe new technologies.

Index

List of Experiments

Sr. No.	Experiments Name
1	Prepare detailed Problem statement of problem for the selected/ allotted mini project and identify suitable process model for the same with justification
2	Develop Software Requirement Specification (SRS) document in IEEE format for the project.
3	Use project management tool to prepare schedule of the project.
4	Identify Scenarios & Develop UML use case & class Diagram for the project.
5	Draw DFD (up to 2 levels) and prepare Data dictionary for the project
6	Develop Activity diagram for the project
7	Develop Sequence & collaboration diagram for the project
8	Compute Cohesion & coupling for given case study
9	Prepare RMMM plan for the project
10	Change specification & make different versions using any SCM Tool
11	Develop test cases for the project using white box testing
12	Installation of JIRA agile project management tool

Course Objectives, Course Outcome & Experiment Plan.

Course Objectives:

1.	To provide the knowledge of software engineering discipline.
2.	To apply analysis, design and testing principles to software project development.
3.	To demonstrate and evaluate real time projects with respect to software engineering principles.

Course Outcomes:

CO1	Select case study & apply process model to selected case study.
CO2	Identify requirements and analyze models for the selected case study using UML modeling**.**
CO3	Use software engineering tools to plan, schedule & track the progress of the project
CO4	Apply Design concepts for the selected case study
CO5	**To analyze risk & manage the change in design.**
CO6	**To validate the software project**

Module No.	Week No.	Experiments Name	Course Outcome	Weightage

1		Prepare detailed Problem statement of problem for the selected/ allotted mini project and identify suitable process model for the same with justification	CO1	5
2		Develop Software Requirement Specification (SRS) document in IEEE format for the project.	CO2	2
3		Use project management tool to prepare schedule of the project.	CO3	10
4		Identify Scenarios & Develop UML use case & class Diagram for the project.	CO2	2
5		Draw DFD (up to 2 levels) and prepare Data dictionary for the project	CO2	2
6		Develop Activity diagram for the project	CO2	2
7		Develop Sequence & collaboration diagram for the project	CO2	2
8		Compute Cohesion & coupling for given case study	CO4	10
9		Prepare RMMM plan for the project	CO5	5
10		Change specification & make different versions using any SCM Tool	CO5	5
11		Develop test cases for the project using white box testing	CO6	10
12		Study & Installation of JIRA agile project management tool	CO1	5

Mapping Course Outcomes (CO) - Program Outcomes (PO)

Subject Weight	Course Outcomes	Contribution to Program outcomes											
		PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
TH 20%	Select case study & apply process model to selected case study.	2	1	1	1		1	1			1	1	1
	Identify requirements and analyze models for the selected case study using UML modeling.		2	2	1	1				1	1	1	1
	Use software engineering tools to plan, schedule & track the progress of the project			1		2	1	1	1	1	1	1	1
	Apply Design concepts for the selected case study		1	1	1	3				1	1	1	1
	To analyze risk & manage the change in design.	1	1	1	1	2				1	1	1	1
	To validate the software project		2	2	1	2	1					1	1

Subject Weight	Course Outcomes	Contribution to Program outcomes											
		PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
PR 80%	Select case study & apply process model to selected case study.	2	1	1	1		1	1			1	1	1
	Identify requirements and analyze models for the selected case study using UML modeling.		2	2	1	1				1	1	1	1
	Use software engineering tools to plan, schedule & track the progress of the project			1		2	1	1	1	1	1	1	1
	Apply Design concepts for the selected case study		1	1	1	3				1	1	1	1
	To analyze risk & manage the change in design.	1	1	1	1	2				1	1	1	1
	To validate the software project		2	2	1	2	1					1	1

Mapping Course Outcomes (CO) – Program Specific Outcomes (PSO)

Course Outcomes		Contribution to Program Specific outcomes		
		PSO1	PSO2	PSO3
CO1	Select case study & apply process model to selected case study.	2		2
CO2	Identify requirements and analyze models for the selected case study using UML modeling.	3	1	2
CO3	Use software engineering tools to plan, schedule & track the progress of the project	1	3	2
CO4	Apply Design concepts for the selected case study	3	3	2
CO5	**To analyze risk & manage the change in design.**	2	2	1
CO6	**To validate the software project**	3	2	1

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
CSL601	Software Engineering		02	--		01	--	01

Course Code	Course Name	Examination Scheme		
		Term Work	Oral	Total
CSL601	Software Engineering	25	25	50

Term Work:

Term work (25 Marks) shall consist of

- Laboratory work ….. 15 marks
- Two assignments … 05 marks
- Attendance (theory and practical) ….. 05 marks

Practical & Oral:

Oral examination will be conducted based on syllabus.

Experiment No. : 1

Prepare detailed statement of problem for the selected / allotted mini project and identify suitable process model for the same with justification.

Experiment No. 1

1. **Aim: Prepare detailed statement of problem for the selected / allotted mini project and identify suitable process model for the same with justification.**

2. **Objectives:** From this experiment, the student will be able to
   - To understand the requirements and prepare detailed statement of the problem.
   - To learn how documentation is prepared according to requirements and identify suitable process model for the problem.

3. **Outcomes:** Students will demonstrate the basic knowledge of software process models and software processes.

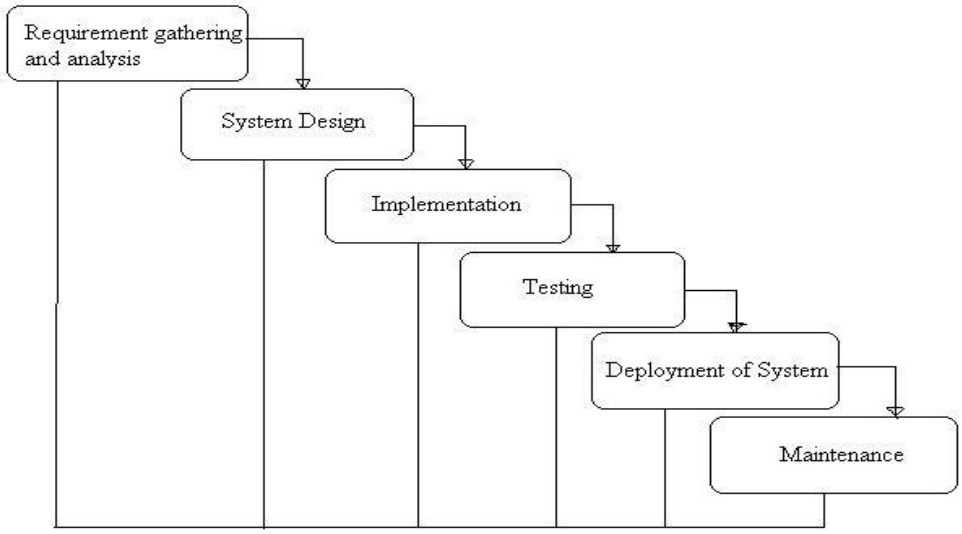
4. **Hardware/ Software Required:** Any text editor, Open source tool

5. **Theory:**

   A **problem statement** is a concise description of an issue to be addressed or a condition to be improved upon. It identifies the gap between the current (problem) state and desired (goal) state of a process or product. The main purpose of the problem statement is to identify and explain the problem. This includes describing the existing environment, where the problem occurs, and what impacts it has on users, finances, and ancillary activities.

   A **Software Process** is: "a set of activities, practices and transformations that people use to develop and maintain software and the associated products, e.g., project plans, design documents, code, test cases and user manual". A number of software processes are available. However, no single software process works well for every project. Each process works best in certain environments. Examples of the available software process models include: the Waterfall model (the Linear model), the evolutionary development, the formal systems development, and reuse-based (component-based) development. Other process models support iteration; these include: the Incremental model, the Spiral model, and Extreme Programming.

   1. Waterfall Process Model

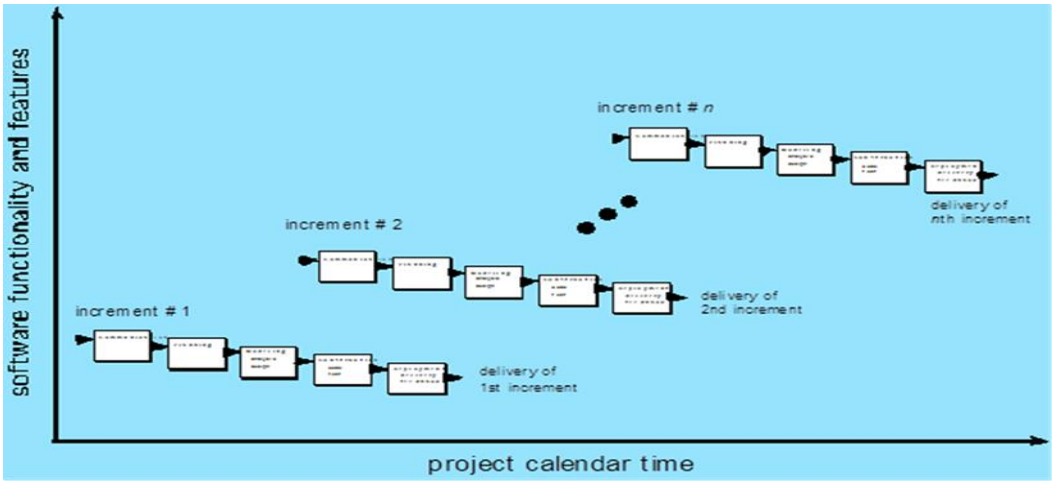


The waterfall model was the first software process model. It is also referred to as a linear-sequential life cycle model. The principal stages of the model represent the fundamental development activities:

1. Requirements analysis and definition
2. System and software design
3. Implementation and unit testing
4. Integration and system testing
5. Deployment and maintenance

2. Incremental Process Models

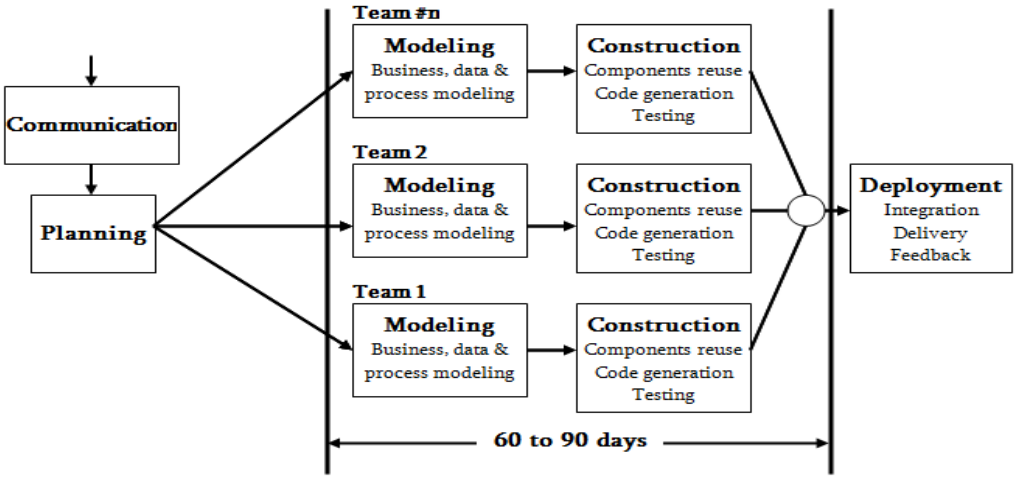
The waterfall model of development requires defining the requirements for a system before design begins. On contrary, an evolutionary development allows requirements to change but it leads to software that may be poorly structured and difficult to understand and maintain. Incremental delivery is an approach that combines the advantages of these two models. In an incremental development process, customers identify the services to be provided by the software system. They decide which subset of the services is most important and which are least important to them. A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality. The allocation of services to increments depends on the priority of service. The highest priority services are delivered first. Once the system increments have been identified, the requirements for first increment are defined in detail, and that increment is developed using the best suited software process.



3. RAD Model

Rapid Application Development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

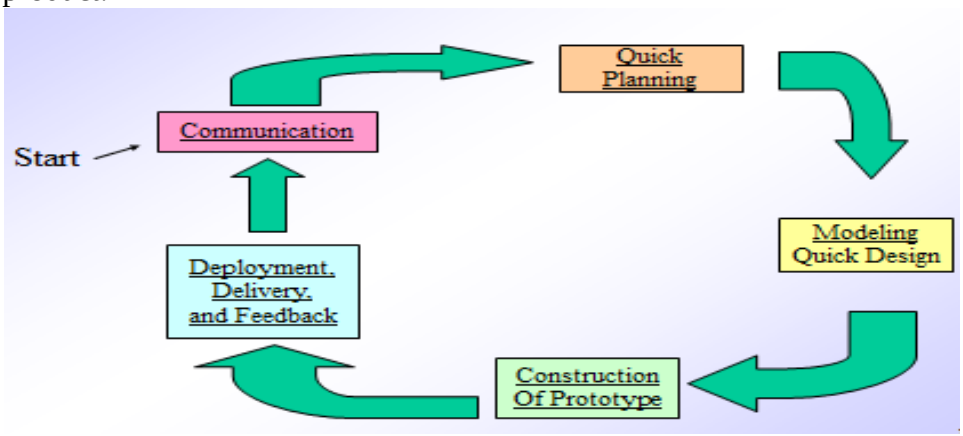


Evolutionary Process Models

Software evolves over a period of time; business and product requirements often change as development proceeds, making a straight-line path to an end product unrealistic. Software Engineering needs a process model that has been explicitly designed to accommodate a product that evolves over time. Evolutionary process models are iterative. They produce increasingly more complete versions of the Software with each iteration.

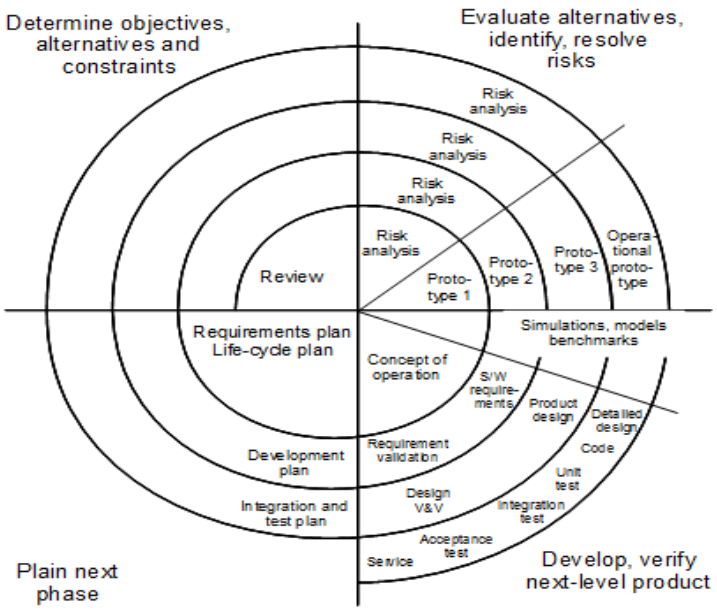
4. Prototyping

The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models).This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.



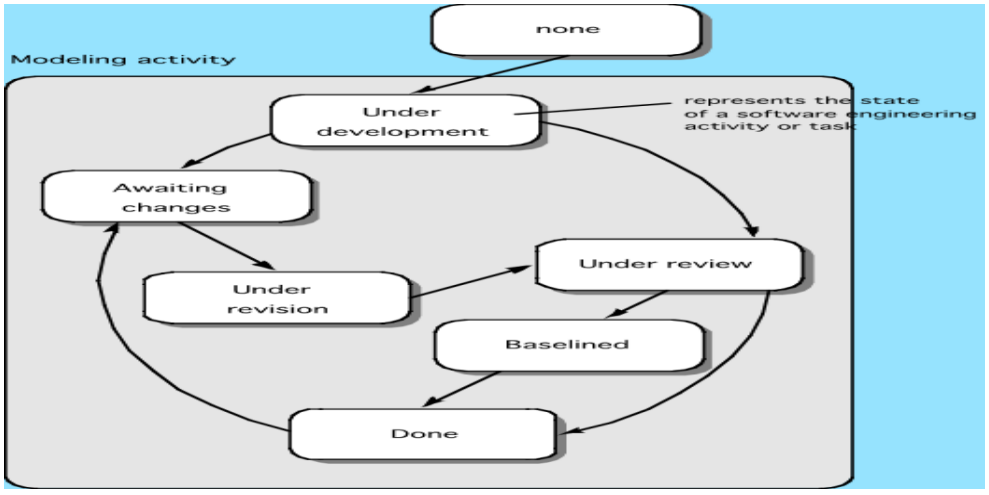
5. **Spiral Model**

The spiral model of the software process represents the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral. Each loop in the spiral represents a phase of the software process. Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on.



6. Concurrent Development Model

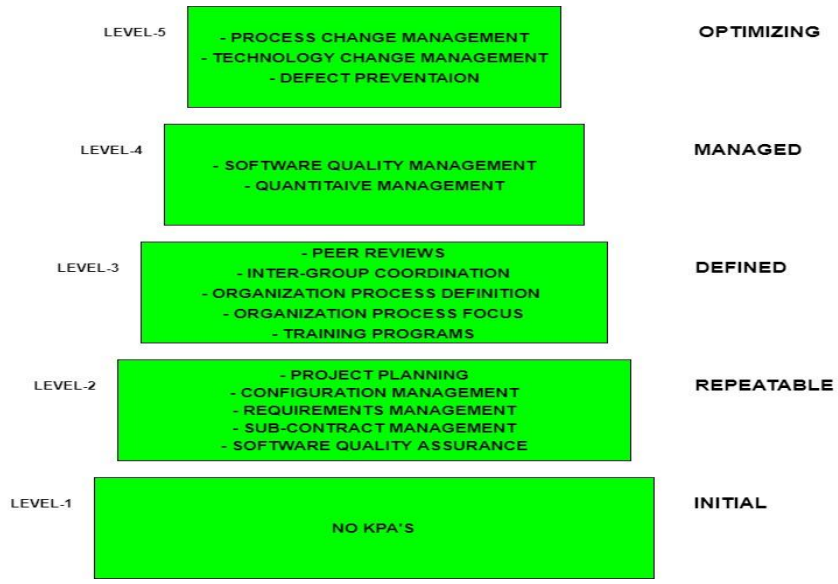
The concurrent development model, sometimes called concurrent engineering, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states. The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.



7. Capability Maturity Model (CMM)

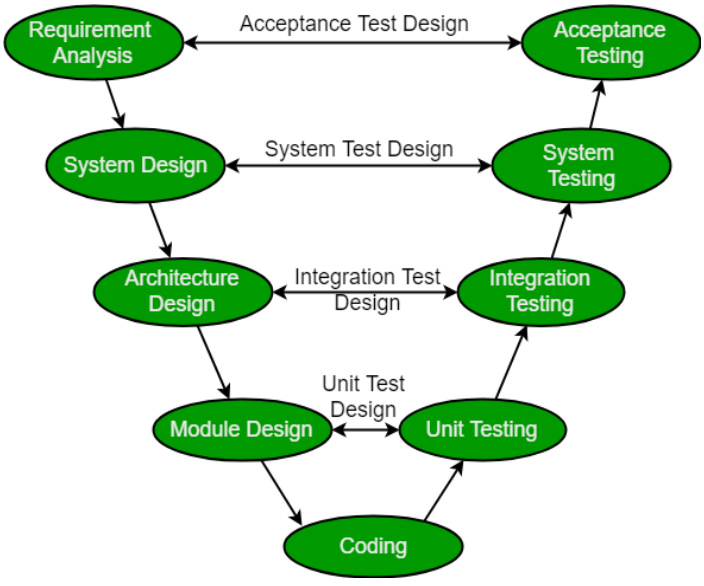
It is not a software process model. It is a framework which is used to analyse the approach and techniques followed by any organization to develop a software product. This model describes a strategy that should be followed by moving through 5 different

levels. Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).



8. V-model

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e. for each development activity, there is a testing activity corresponding to it.



6. Procedure:

Students should decide their case study topics and software development process model that they are applying on their case study.

7. Conclusion:

Software Process Models plays an important role in development of any software process or project, as software developers has to manage large-scale software projects. Different types of process models are available. As per the requirements establish, choose one of the process model.

8. Viva Questions:

1. Differentiate between software process models.
2. State how evolutionary process approach is better than linear process approach?

References:

1. **http://www.thomasalspaugh.org/pub/fnd/softwareProcess.html**

2. **Roger Pressman, —Software Engineering: A Practitioner‘s approach ”, McGraw-Hill Publications**

Experiment No. : 2

Develop Software Requirement Specification (SRS) document in IEEE format for the project.

Experiment No. 2

1. **Aim:** Develop Software Requirement Specification (SRS) document in IEEE format for the project.
2. **Objectives:** From this experiment, the student will be able to,
    - To gain a deeper understanding of the Software Requirement Specification phase and the Software Requirement Specification (SRS).
    - To learn how to write requirements and specifications.
    - To learn how documentation is prepared according to functional & non-functional requirements

3. **Outcomes:** From this experiment, the student will be able to,
    - Review of the requirements engineering process.
    - Write requirements and specifications.

4. **Hardware / Software Required:**

5. **Theory:** Any text editor, Open source tool

    - A software requirements specification (SRS) fully describes what the software will do and how it will be expected to perform.
    - It is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of *requirements* engineering phase.
    - It lays out functional and non-functional *requirements*, and may include a set of use cases that describe user interactions that the *software* must provide.

    - **What should the SRS address?**

      As per the IEEE standard:

      The basic issues that the SRS writer(s) shall address are the following:

      a) Functionality: What is the software supposed to do?

      b) External interfaces: How does the software interact with people, the system hardware, other hardware, and other software?

      c) Performance: What is the speed, availability, response time, recovery time of various software functions, etc.?
      d) Attributes: What is the portability, correctness, maintainability, security, etc. considerations?

e) 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.?

- **Template for RFP**

Software Requirements Specification for

<Project>

Version 1.0 approved
Prepared by <author>
<Organization>
<Date created>

Table of Contents

History

Name	Date	Reason For Changes	Version

- **Introduction**

  - **Purpose**

Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.

  - **Document Conventions**

Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.

  - **Intended Audience and Reading Suggestions**

Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and

documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.

- **Product Scope**

Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.

- **References**

List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.

- **Overall Description**

  - **Product Perspective**

Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.

  - **Product Functions**

Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.

  - **User Classes and Characteristics**

Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product

functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.

- **Operating Environment**

Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.

- **Design and Implementation Constraints**

Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).

- **User Documentation**

List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.

- **Assumptions and Dependencies**

List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).

- **External Interface Requirements**

  - **User Interfaces**

Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details

of the user interface design should be documented in a separate user interface specification.

- **Hardware Interfaces**

Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.

- **Software Interfaces**

Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.

- **Communications Interfaces**

Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.

- **System Features**

This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.

- **System Feature 1**

(Just not "System Feature 1." State the feature name in just a few words.)

4.1.1 Description and Priority

Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component

ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9)

1.1.2 Stimulus/Response Sequences

List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.

1.1.3 Functional Requirements

Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.

Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.

- **System Feature 2... (And so on)**

- **Other Nonfunctional Requirements**
  - **Performance Requirements**

If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.

- **Safety Requirements**

Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.

- **Security Requirements**

Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.

- **Software Quality Attributes**

Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.

- **Business Rules**

List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.

6. Documentation:

Students should prepare Software Requirement Specification (SRS) Document as per the template given above (for their case study)

7. Conclusion:

SRS establish the basis for agreement between the customers and the suppliers also reduce the development efforts, provide a basis for estimating costs and schedules, and provide a baseline for validation and verification. The SRS makes it easier to transfer the software product to new users or new machines &Serve as a basis for enhancement.

8. Viva Questions:

1. What are the contents of SRS?
2. State functional & nonfunctional requirements in SRS?

9. Reference books & Links:

1. https://www.ibm.com/.../community/.../srs%20format.pdf
2. https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc
3. www.tricity.wsu.edu/~mckinnon/cpts322/cpts322-srs-v1.doc
4. www.jaysonjc.com/.../how-to-write-a-software-requirements-specification.

Experiment No. : 3

Use project management tool to prepare schedule for the project.

Experiment No. 3

1. **Aim:** Use Project management tool to prepare schedule for the project.

2. **Objectives:** From this experiment, the student will be able,
   - To introduce the concept of advance software methodology
   - To understand project management software for scheduling, cost control and budget management, resource allocation, collaboration software, communication, quality management and documentation or administration systems which are used to deal with the complexity of large projects.

3. **Outcomes:** Students will be able,
   - To learn and apply different project scheduling techniques.
   - To plan and determine schedule of the project.
   - To draw timeline chart for the project.

4. **Hardware / Software Required:** PERT & CPM tools, MS Project..etc

5. **Theory:**

   Introduction

   Basically, PERT (Programed Evaluation Review Technique) and CPM (Critical Path Method) are project management techniques, which have been created out of the need of Western industrial and military establishments to plan, schedule and control complex projects.

   Planning, Scheduling & Control

   Planning, Scheduling (or organizing) and Control are considered to be basic Managerial functions, and CPM/PERT has been rightfully accorded due importance in the literature on Operations Research and Quantitative Analysis. PERT/CPM provided a focus around which managers could brain-storm and put their ideas together. It proved to be a great communication medium by which thinkers and planners at one level could communicate their ideas, their doubts and fears to another level. Most important, it became a useful tool for evaluating the performance of individuals and teams.

   There are many variations of CPM/PERT which have been useful in planning costs, scheduling manpower and machine time. CPM/PERT can answer the following important questions:

   - How long will the entire project take to be completed? What are the risks involved?

- Which are the critical activities or tasks in the project which could delay the entire project if they were not completed on time?
- Is the project on schedule, behind schedule or ahead of schedule?
- If the project has to be finished earlier than planned, what is the best way to do this at the least cost?

The Framework for PERT and CPM

Essentially, there are six steps which are common to both the techniques. The procedure is listed below:

1. Define the Project and all of its significant activities or tasks. The Project (made up of several tasks) should have only a single start activity and a single finish activity.

2. Develop the relationships among the activities. Decide which activities must precede and which must follow others.

3. Draw the "Network" connecting all the activities. Each Activity should have unique event numbers. Dummy arrows are used where required to avoid giving the same numbering to two activities.

4. Assign time and/or cost estimates to each activity

5. Compute the longest time path through the network. This is called the critical path.

6. Use the Network to help plan, schedule, and monitor and control the project.

The Key Concept used by CPM/PERT is that a small set of activities, which make up the longest path through the activity network control the entire project. If these "critical" activities could be identified and assigned to responsible persons, management resources could be optimally used by concentrating on the few activities which determine the fate of the entire project. Non-critical activities can be re-planned, rescheduled and resources for them can be reallocated flexibly, without affecting the whole project.

Five useful questions to ask when preparing an activity network are:

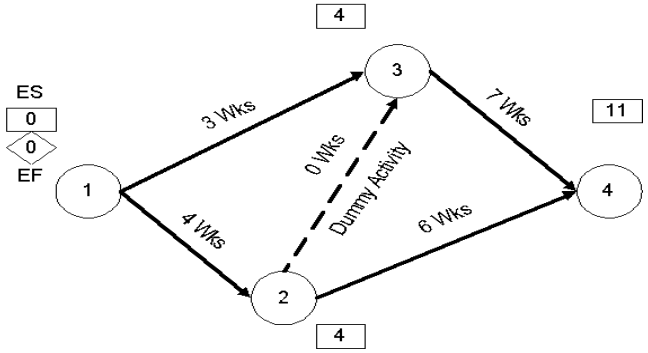
- Is this a Start Activity?
- Is this a Finish Activity?
- What Activity Precedes this?
- What Activity Follows this?
- What Activity is Concurrent with this?

Some activities are serially linked. The second activity can begin only after the first activity is completed. In certain cases, the activities are concurrent, because they are independent of each other and can start simultaneously. This is especially the case in organizations which have supervisory resources so that work can be delegated to various departments which will be responsible for the activities and their completion

as planned. When work is delegated like this, the need for constant feedback and co-ordination becomes an important senior management pre-occupation.

- Drawing the CPM/PERT Network

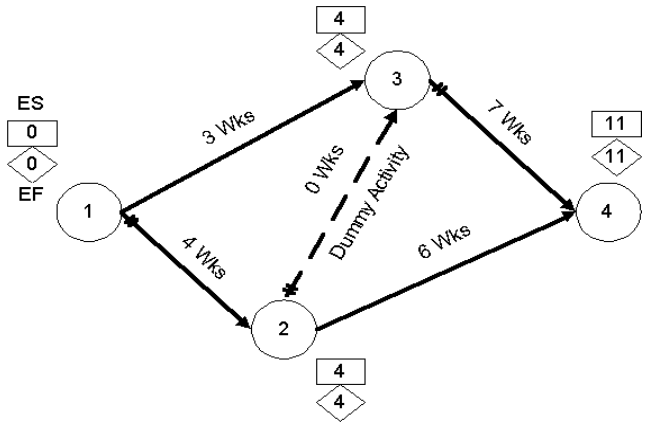
Each activity (or sub-project) in a PERT/CPM Network is represented by an arrow symbol. Each activity is preceded and succeeded by an event, represented as a circle and numbered.



At Event 3, we have to evaluate two predecessor activities - Activity 1-3 and Activity 2-3, both of which are predecessor activities. Activity 1-3 gives us an Earliest Start of 3 weeks at Event 3. However, Activity 2-3 also has to be completed before Event 3 can begin. Along this route, the Earliest Start would be 4+0=4. The rule is to take the longer (bigger) of the two Earliest Starts. So the earliest Start at event 3 is 4. Similarly, at Event 4, we find we have to evaluate two predecessor activities - Activity 2-4 and Activity 3-4. Along Activity 2-4, the Earliest Start at Event 4 would be 10 wks, but along Activity 3-4, the Earliest Start at Event 4 would be 11 wks. Since 11 wks is larger than 10 wks, we select it as the Earliest Start at Event 4.We has now found the longest path through the network. It will take 11 weeks along activities 1-2, 2-3 and 3-4. This is the Critical Path.

- The Backward Pass - Latest Finish Time Rule

To make the Backward Pass, we begin at the sink or the final event and work backwards to the first event.



At Event 3 there is only one activity, Activity 3-4 in the backward pass, and we find that the value is 11-7 = 4 weeks. However at Event 2 we have to evaluate 2 activities, 2-3 and 2-4. We find that the backward pass through 2-4 gives us a value of 11-6 = 5 while 2-3 gives us 4-0 = 4. We take the smaller value of 4 on the backward pass.

- Tabulation & Analysis of Activities

We are now ready to tabulate the various events and calculate the Earliest and Latest Start and Finish times. We are also now ready to compute the SLACK or TOTAL FLOAT, which is defined as the difference between the Latest Start and Earliest Start.

Event	Duration(Weeks)	Earliest Start	Earliest Finish	Latest Start	Latest Finish	Total Float
1-2	4	0	4	0	4	0
2-3	0	4	4	4	4	0
3-4	7	4	11	4	11	0
1-3	3	0	3	1	4	1
2-4	6	4	10	5	11	1

- The Earliest Start is the value in the rectangle near the tail of each activity
- The Earliest Finish is = Earliest Start + Duration
- The Latest Finish is the value in the diamond at the head of each activity
- The Latest Start is = Latest Finish - Duration

There are two important types of Float or Slack. These are Total Float and Free Float.

- Total Float is the spare time available when all preceding activities occur at the earliest possible times and all succeeding activities occur at the latest possible times.

Total Float = Latest Start - Earliest Start

- Activities with zero Total float are on the Critical Path. Free Float is the spare time available when all preceding activities occur at the earliest possible times and all succeeding activities occur at the earliest possible times.When an activity has zero Total float, free float will also be zero.
- There are various other types of float (Independent, Early Free, Early Interfering, Late Free, Late Interfering), and float can also be negative. We shall not go into these situations at present for the sake of simplicity and be concerned only with Total Float for the time being.
- Having computed the various parameters of each activity, we are now ready to go into the scheduling phase, using a type of bar chart known as the Gantt chart.
- There are various other types of float (Independent, Early Free, Early Interfering, Late Free, Late Interfering), and float can also be negative. We shall not go into

these situations at present for the sake of simplicity and be concerned only with Total Float for the time being. Having computed the various parameters of each activity, we are now ready to go into the scheduling phase, using a type of bar chart known as the Gantt chart.

6. Procedure:

Students should define activities planned for their case study & use PERT & CPM method to draw Timeline chart & project table

7. Conclusion:

A Timeline chart & project table helps in planning, how long that project might take and that can really save time in the end and more so if you are on a deadline.

8. Viva Questions:

1. What are the components of timeline chart?
2. Explain critical path method in detail.

9. References:

1. www2.kimep.kz/bcb/omis/our_courses/is4201/Chap14.pdf
2. https://www.youtube.com/watch?v=zC2ji72DxXg
3. Roger Pressman, ―Software Engineering: A Practitioner‘s approach ”, McGraw-Hill Publications

Experiment No. : 4

Identify Scenarios & Develop UML Use Case & Class Diagram for the project

Experiment No. 4

1. **Aim:** Identify Scenarios & Develop UML Use Case & Class Diagram for the project

2. **Objectives:** From this experiment, the student will be able to

   - Become conversant with the Unified Language (UML) notation and symbols.
   - Learn the efficient tools to implement the method.
   - Understand the importance of this experiment from application point of view.
   - Learn the efficient tools to design use case & class diagram

3. **Outcomes:** Students will be able to apply UML notation to construct and graphically present various diagrams for Object- Oriented systems analysis.

4. **Hardware / Software Required:** Rational Software Architect tool.

5. **Theory:**

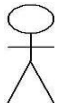
   Use Case Diagram:

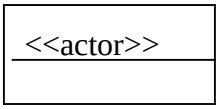
   Use Case describes the behaviour of a system from a user's standpoint, Provides functional description of a system and its major processes, Provides graphic description of the users of a system and what kinds of inheritance to expect within that system, Displays the details of the processes that occur within the application area, Used to design the test cases for testing the functionality of the system.

   Elements of Use Case:

   A Use Case diagram is quite simple in nature and depicts two types of elements: one representing the business roles and the other representing the business processes. Elements of Use Case diagrams are:

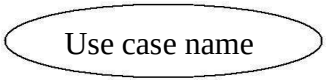
   Actors: An actor portrays any entity (or entities) that perform that perform certain roles in a given system. An actor in a use case diagram interacts with a use case and is depicted "outside" the system boundary.

   Actor Name

   Use case: A use case in a Use Case diagram is a visual representation of distinct business functionality in a system. Each use case is a sequence of transactions performed by the system that produces a major suit for the actor.

System Boundary: A system boundary defines the scope of what a system will be. A system cannot have infinite functionality.

Relationships in Use Case:

1. Include: Include is used when two or more use cases share common portion in the flow of events. The stereotype <<include>> identifies the relationship include.

2. Extend: In an extend relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case.

3. Generalization: A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning, but is an enhancement of the parent use case.

DOCUMENTING USE CASE DIAGRAM:

USE CASE 1:

Use case name	Name of use case
Use case Id	Unique identifier of the use case
Super use case	If the use case inherits a parent use case
Actor	The actor which are participating in the execution of use case.
Brief Description	Description of scope of use case and
Preconditions	Constraints that must be satisfied before
Post conditions	Condition will be established after the use case.
Priority	Development priority from the view of development team
Flow of events	Ste by step description of the interaction
Alternative flows and exceptions	Alternatives or exceptions that may occur.
Non-behavioral requirements	Non-functional requirements like h/w and s/w requirements.
Assumptions	All the assumptions made about the use
Issues	All outstanding issues related to the use case need to be resolved.

Source	Includes references and materials used in developing use case

Class Diagram:

An object model captures a static structure of a system by showing the objects in the system, relationship between the objects, and the attributes and operation that characterize each class of objects. Object models provide an intuitive graphic representation of a system and are valuable for communicating with customers and documenting the structure of the system.

Steps for Constructing Object Model (Class Diagram):

The first step in analyzing the requirements is to construct an object model. The object model shows the static data structure of the real world system and organizes it into workable pieces. The object model describes real world object classes and their relationships to each other.

The following steps are performed in constructing an object model:

(1) Identify objects and classes
(2) Prepare a data dictionary
(3) Identify associations (including aggregations) between objects.
(4) Identify attributes of objects and links.
(5) Organize and simplify object classes using inheritance.
(6) Identify operations to be included in a class.
(7) Verify that access paths exist for likely queries.
(8) Iterate and refine the model
(9) Group classes into modules

Elements of Class Diagram:

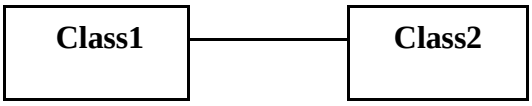
Class: Classes are composed of three things: a name, attributes, and operations. Below is an example of a class:

Class-Name
Attribute Name - 1 : datatype 1 = default value 1 Attribute Name - 2 : datatype 2 = default value 2
Operation Name 1 (argument List 1) : result Type 1 Operation Name 2 (argument List 2) : result Type 2

Attributes: An attribute is data value held by the objects in a class. An attribute should be a pure data value, not an object. Unlike objects, pure data values do not have identity.

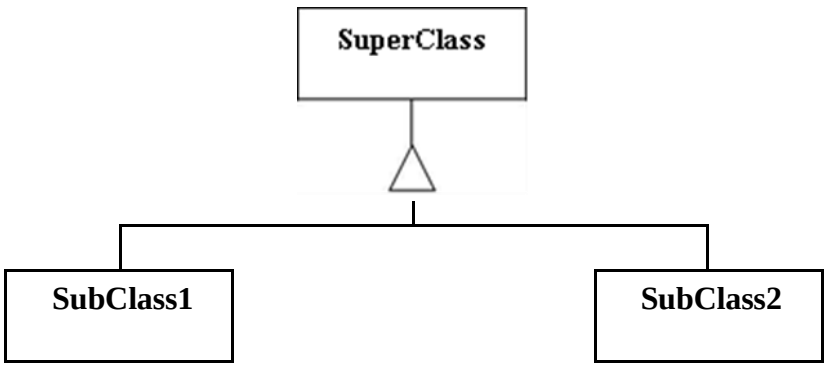
Operations and Methods: An operation is a function or transformation that may be applied to or by objects in a class. Operations are listed in the lower third of the class box.

Links and Association: Links and association are the means for establishing relationships among objects and classes. A link is a physical or conceptual connection between object instances. An association describes a group of links with common structure and common semantics.
An association describes a set of potential links in the same way that a class describes a set of potential objects. Associations are inherently bi-directional.



Association

Multiplicity: Multiplicity specifies how many instances of one class may relate to a single instance of an associated class. Multiplicity constrains the number of related objects. Multiplicity is often described as being " one" or "many" but more generally it is subset of non negative integers.
Object diagram indicate multiplicity with special symbols at the ends of association lines. Multiplicity can be specified with a number or set of intervals, such as "1", "1+"(1 or more), "3-5"(3 to 5, inclusive), and "2, 4,18" (2,4 or 18)
Generalization and Inheritance: Generalization and Inheritance are powerful abstractions for sharing similarities among classes while preserving their differences.
Generalization is the relationship between a class and one or more refined versions of it. The class being refined is called the superclass and each refined version is called subclass. Attributes and operations common to a group of subclasses are attached to the superclass and shared by each subclass. Each subclass is said to inherit the features of its superclass. Generalization is sometimes called the "is-a" relationship because each instance of a subclass is an instance of the superclass as well.



Generalization

Aggregation: Aggregation is the "part- whole" or "a-part-of" relationship is which objects representing the components of something are associated with an object representing the entire assembly. Aggregation is a tightly coupled form of association with some extra semantics. The most significant property of aggregation is transitivity that is if A is part of B, and B is part of C then A is part of C. Aggregation is also antisymmetric, that is if A is part of B then B is not part of A. Finally, some properties of the assembly propagate to the components as well possibly with some local modifications.

Documenting Class Diagram:
CLASS 1

Class Id	Unique Id of Class
Class Name	Name of Class
Attributes	List of attributes for each class
Methods	Functions carried out by class
Associations	Relationship between different classes
Inheritance	Classes sharing similarities
Multiplicity	Identify how many instances of one class may relate to single instance of an associated class
Description	Description of Diagram

CLASS 2
….
CLASS N

6. Conclusion:

Class diagram is useful for abstract modelling and for designing actual programs. Class Diagram matches the industry requirements in the domains of Database management, Programming and Networking which gives the complete knowledge and understanding of the system. It also helps in analyzing the local and global impact of computing on system and organizations.

Use Case diagram helps to understand the different processes and entities involved in a system and help the analyst to understand and document the system

7. Viva Questions:

- Write different notations for class diagram?
- Draw class diagram for hospital management system?
- What is the difference between Aggregation and Generalization?
- Are use cases the same as functional requirements or functional requirements are different from use cases?
- What is use case diagram?
- What is actor in use case diagrams?
- What is the difference between use case diagram and use case?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."
- https://www.youtube.com/watch?v=TL4ABTx_RtE
- home.iitk.ac.in/~blohani/Limulator/publication/Rakesh_Indore_SE.pdf

Experiment No. : 5

Data Flow Diagram and model (Level 0, level 1 DFD) of selected / allotted project

Experiment No. 5

1. **Aim:** To develop DFD and model (level 0, level 1 DFD) of Selected / allotted project. (Write the name of your project)

2. **Objectives:** From this experiment, the student will be able to

    - Get balanced exposure to both traditional and object oriented approaches to system analysis & design.
    - To understand the importance of this experiment from application point of view.

3. **Outcomes:** The learner will be able to

    - To construct the candidate system following design methodology.
    - Have leadership and management skills to accomplish a common goal.
    - Communicate effectively in both graphical and written form.

4. **Hardware / Software Required:** Rational Software Architecture tool.

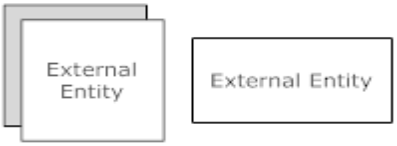
5. **Theory:**

Data Flow Diagram

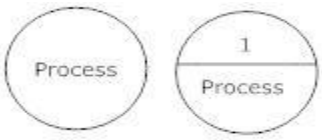
A data flow data diagram is one means of representing the functional model of a software product. DFDs do not represent program logic like flow transitions do. The DFD can be created for different levels. The context level DFD (also considered as level 0) shows the entire system as a single process, and gives no clues as to its internal organization. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole.

Data Flow Diagram Notations:

External Entity: External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.



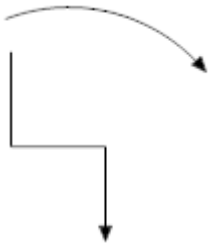
Process: A process transforms incoming data flow into outgoing data flow.



Datastore Notations

DataStore: Datastores are repositories of data in the system. They are sometimes also referred to as files.

Dataflow: Dataflow are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it



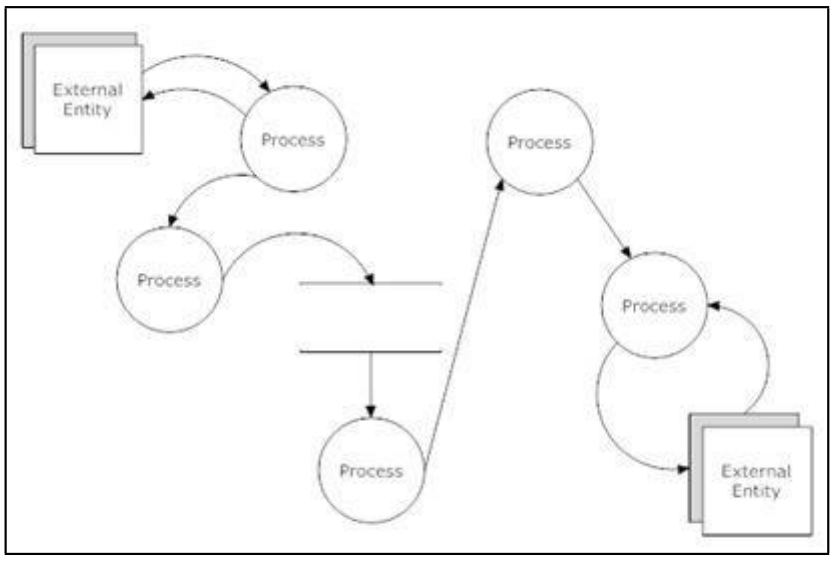


Fig: Data Flow Diagram

Documenting Data Flow Diagram:

Level 0 DFD Documentation: Process :

Process Name	Name of the process
Entities	List of entities related with this process
Data Flow	Name of the flow
Data store	List all the related datastores to this process
Description (optional)	Description of process if needed

LEVEL 1 DFD

Process Name	Name of the process
Entities	List of entities related with this process
Data Flow	Name of the flow
Data store	List all the related datastores to this process
Description (optional)	Description of process if needed

Process 2:

Process Name	Name of the process
Entities	List of entities related with this process
Data Flow	Name of the flow
Data store	List all the related datastores to this process
Description (optional)	Description of process if needed

|

|

Process n

6. Conclusion:

DFD is a Graphical representation of functional representation of an information system.

7. Viva Questions:

- What is the need of DFD?
- What do you mean by functional representation?

8. References:

- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- "Modern Systems Analysis and Design by Jeffrey A. Hoffer, Joey F. George, Joseph S.Valacich, Prabin K. Panigrahi, Pearson Education Publication, 4th Edition."

Experiment No. : 6

Activity diagram for selected / allotted project

Experiment No. 6

1. **Aim:** To study and draw Activity diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Apply key modelling concepts to both the traditional structured approach and the Object Oriented approach.
   - Captures the dynamic behavior of the system.
   - Learn the efficient tools to design Activity diagram.

3. **Outcomes:** The learner will be able to

   - Apply various tools and techniques for key modelling to both the traditional structured approach and the Object Oriented approach.
   - Model the activities which are nothing but business requirements.

- Use current techniques, skills, and tools necessary to design activity diagram.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

Activity diagram is used for business process modelling, for modelling the logic captured by a single use case or usage scenario, or for modelling the detailed logic of a business rule. Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state. The easiest way to visualize an activity diagram is to think of a flow transition and data flow diagrams (DFDs).

Need of an Activity Diagram:

The general purpose of activity diagrams is to focus on flows driven by internal processing vs. external events. Activity diagrams are also useful for: analysing a use case by describing what actions needs to take place and when they should occur; describing a complicated sequential algorithm; and modelling applications with parallel processes.

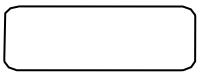
Elements of an Activity Diagram

An Activity diagram consists of the following behavioural elements:

Initial Activity: This shows the starting point or first activity of the flow and denoted by a solid circle. There can only be one initial state on a diagram.

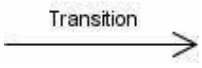
Initial Activity

Activity: Represented by a rectangle with rounded (almost oval) edges.



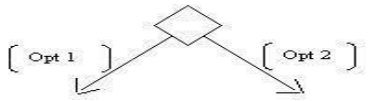
Activity

Transition: When an activity state is completed, processing moves to another activity state. Transitions are used to mark this movement. Transitions are modelled using arrows.



Transition

Decisions: Similar to flowcharts, a logic where a decision is to be made is depicted by a diamond, with the options written on either side of the arrows emerging from the diamond, within box brackets.



Decisions

Synchronization Bar: Activities often can be done in parallel. To split processing ("fork"), or to resume processing when multiple activities have been completed ("join"), Synchronization Bars are used. These are modelled as solid rectangles, with multiple transitions going in and/or out. Fork denotes the beginning of parallel activity. Join denotes the end of parallel processing.

Synchronization Bar

Final Activity: The end of the Activity diagram is shown by a bull's eye symbol, also called as a final activity. An activity diagram can have zero or more activity final nodes.

Final Activity

Swim Lanes

Activity diagrams provide another ability, to clarify which actor performs which activity. If you wish to distinguish in an activity diagram the activities carried out by individual actors, vertical columns are first made, separated by thick vertical black lines, termed swim lanes and name each of these columns with the name of the actor involved. You place each of the activities below the actor performing these activities and then show how these activities are connected.

Documenting Activity Diagram:

Activity 1:

Activity name	Name of the activity
Description	Description about activity

Events	Events occurred during activity
Actor	Actor performing activity
Join synchronization bar	To show multiple activities occurring simultaneously
Fork synchronization bar	To resume processing when multiple activities have been completed
Activity Diagram Id	ID that identifies diagram uniquely
Name	Name o the diagram
Preconditions	Conditions before starting the activities
Post Conditions	Conditions after the activities are over

6. Conclusion:

It can be concluded saying activity diagrams focus on flows driven by internal processing vs. external events and used to model a logic captured by single scenario.

7. Viva Questions:

- What is an Activity Diagram and what is its purpose?
- What is swim lanes?

8. References:

- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- Object-Oriented and Design with UML by Michael Blaha, James Rumbaugh, Pearson Education Publication, 2nd Edition.

http://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_paradigm.htm

Experiment No. : 7

Sequence and Collaboration diagram for selected / allotted project

Experiment No. 7

1. **Aim:** To study and draw Sequence diagram for selected / allotted project.

2. **Objectives:** From this experiment, the student will be able to

   - Apply UML notation to construct and graphically present communication diagram for Object- Oriented systems analysis.
   - Show objects interactions arranged in time sequence and messages exchanged.
   - Learn the efficient tools to design sequence diagram.

3. **Outcomes:** The learner will be able to

- Identify the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.
- Show object interactions arranged in time sequence.

4. **Hardware / Software Required:** Rational software architect tool.

5. **Theory:**

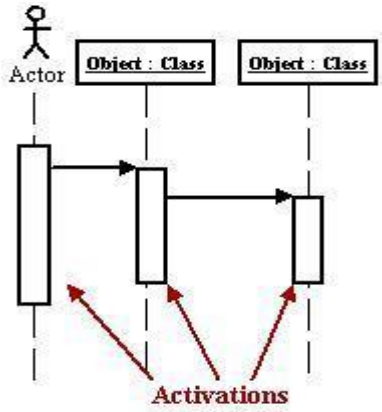
A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence. Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction. There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.
A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

Need of Sequence Diagram:
Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. This type of diagram is best used during early analysis phases in design because they are simple and easy to comprehend. Sequence diagrams are normally associated with use cases.

Elements of Sequence Diagram:
The following tools located on the sequence diagram toolbox enable to model sequence diagrams:

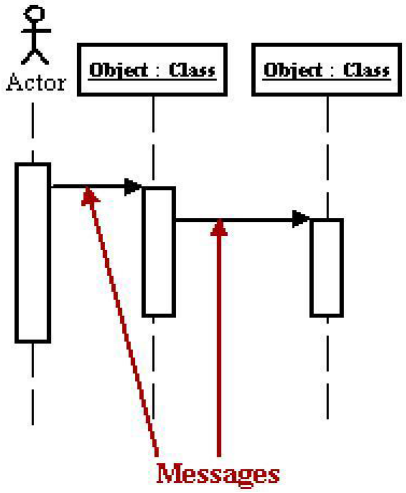
Class roles
Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Activation
Activation boxes represent the time an object needs to complete task.



Messages

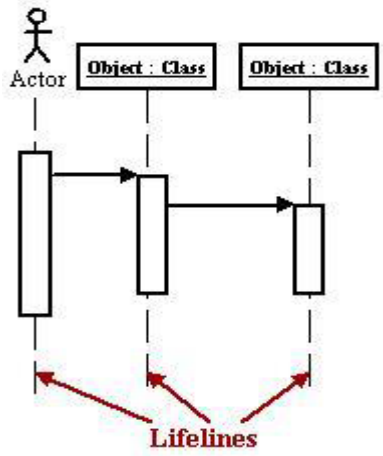
Messages are arrows that represent communication between objects. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.



Arrow	Message type
→	Simple
→	Synchronous
→	Asynchronous
↩	Balking
⏱→	Time out

Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



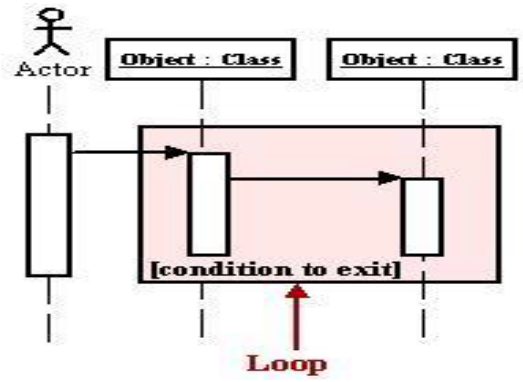
Destroying Objects:

Objects can be terminated early using an arrow labeled "< < destroy > >" that points to an X.

< < destroy > >

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



Documenting Sequence Diagram:

Scenario Id

Scenario Name

Objects Participating in Sequence

Sequence of Operations in Scenario

Collaboration Diagram

A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.

We form a collaboration diagram by first placing the objects that participate in the interaction as the vertices in a graph, and then add the links that connect these objects as the arcs of this graph. Finally, adorn these links with the messages that objects sends and receive with sequence numbers.

Need of Collaboration Diagram:

We use collaboration diagram to describe a specific scenario. Numbered arrows show the movement of messages during the course of a scenario. A distinguishing feature of a Collaboration diagram is that it shows the objects and their association with other

objects in the system apart from how they interact with each other. The association between objects is not represented in a Sequence diagram.

Elements of Collaboration Diagram:

A sophisticated tool can easily convert a collaboration diagram into a sequence diagram and the vice versa. Hence, the elements of a Collaboration diagram are essentially the same as that of a Sequence diagram.

Object : The objects interacting with each other in the system. Depicted by a rectangle with the name of the object in it, preceded by a colon and underlined.

Object

Relation/Association: A link connecting the associated objects. Qualifiers can be placed on either end of the association to depict cardinality.

* 1..*

Association

Messages: An arrow pointing from the commencing object to the destination object shows the interaction between the objects. The number represents the order/sequence of this interaction.

Message

Documenting Collaboration Diagram:

Scenario Id Scenario Name
Objects Participating in Collaboration
Association between Objects
Sequence of Operations in Scenario

6. Conclusion:

Sequence diagram helps to model different objects in a system and to represent the sequence of operations in scenario. Collaboration diagram helps to model different objects in a system and to represent the associations between the objects as links.

7. Viva Questions:

- How to model exception handling in sequence diagram?
- Do we show private or protected methods (messages) on sequence diagrams?
- Can a sequence diagram be replaced by communication diagram?
- What is difference between collaboration and sequence diagram?
- What are the diagram notations for collaboration and sequence diagram?

8. References:

- The Unified Language - User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson, Pearson Education Publication.
- Analysis and Design of Information Systems by James a. Senn, 2nd Edition, McGrawHill.
- https://www.youtube.com/watch?v=4WDbte6cPa8
- home.iitk.ac.in/~blohani/Limulator/publication/Rakesh_Indore_SE.pdf

Experiment No. : 8

Compute the Cohesion and Coupling for given case study.

Experiment No. 8

1. **Aim:** To compare cohesion & coupling for given case study

2. **Objectives:** From this experiment, the student will be able,
    - Analyzing software design by using factors such as Cohesion & Coupling
    - To bring out the creativity in each student by building innovative user friendly applications.

3. **Outcomes:** The learner will be able to
    - To apply the core concepts and implementation guidelines of Software Engineering to improve them
    - To understand, identify, analyze and design the problem, implement using current techniques and skills.

4. **Hardware / Software Required:** Any Text editor, Open source tools

5. **Theory:**

    - **Cohesion vs. Coupling**

    Coupling refers to the interdependencies between modules, while cohesion describes how related are the functions within a single module. Low cohesion implies that a given module performs tasks which are not very related to each other and hence can create problems as the module becomes large

    Cohesion refers to what the class (or module) will do. Low cohesion would mean that the class does a great variety of actions and is not focused on what it should do. High cohesion would then mean that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.

    - Example of Low Cohesion:

```
-------------------
| Staff           |
-------------------
| checkEmail()    |
| sendEmail()     |
| emailValidate() |
| PrintLetter()   |
```

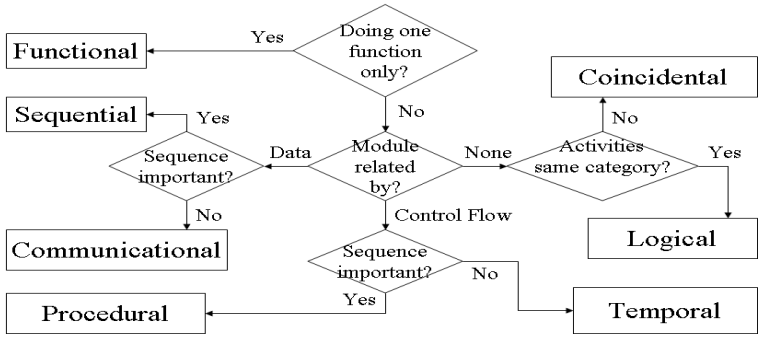
    - Example of High Cohesion:

```
-----------------------------
| Staff                     |
-----------------------------
| -salary                   |
| -emailAddr                |
-----------------------------
| setSalary(newSalary)      |
| getSalary()               |
| setEmailAddr(newEmail)    |
| getEmailAddr()            |
---------------------------------
```

As for coupling, it refers to how related are two classes / modules and how dependent they are on each other. Being low coupling would mean that changing something major in one class should not affect the other. High coupling would make your code difficult to make changes as well as to maintain it, as classes are coupled closely together, making a change could mean an entire system revamp. One should aim for strongly cohesive modules. Everything in module should be related to one another - focus on the task. Strong cohesion will reduce relations between modules - minimize coupling

- Types of Cohesion
  - Functional cohesion (Most Required)
  - Sequential cohesion
  - Communicational cohesion
  - Procedural cohesion
  - Temporal cohesion
  - Logical cohesion
  - Coincidental cohesion (Least Required)

- Determining Module Cohesion



- Determining Module coupling

For data and control flow coupling:

- di: number of input data parameters
- ci: number of input control parameters
- do: number of output data parameters
- co: number of output control parameters

For global coupling:

- gd: number of global variables used as data
- gc: number of global variables used as control

For environmental coupling:

- w: number of modules called (fan-out)
- r: number of modules calling the module under consideration (fan-in)

Coupling(C) makes the value larger the more coupled the module is. This number ranges from approximately 0.67 (low coupling) to 1.0 (highly coupled). For example, if a module has only a single input and output data parameter. If a module has 5 input and output data parameters, an equal number of control parameters, and accesses 10 items of global data, with a fan-in of 3 and a fan-out of 4

6. Procedure:

List down modules for respective case study & compute cohesion & coupling based on that.

7. Conclusion:

Software design is a creative process. To see a wrong design, we can check with the requirements in the analysis model. To see a bad design, we need to assess the design model and analyze the components, whether the performance can be improved by changing the modules or the interfaces by analyzing Cohesion & Coupling. All good software design will go for high cohesion and low coupling.

8. Viva Questions:

- Why software design is important?
- Differentiate between coupling & cohesion.
- What are different types of Cohesion?

9. References:

1. www.math-cs.gordon.edu/courses/.../Cohesion%20and%20Coupling.pdf
2. www.tutorialspoint.com/software_engineering/software_design_basics.htm
3. www.letu.edu/.../Software-Engineering/.../Pressman/pressman-ch-9-desig.
4. ourses.cs.washington.edu/courses/cse403/96sp/coupling-cohesion.htm

Experiment No. 9

To Prepare RMMM plan for the project.

Experiment No. 9

1. **Aim:** Prepare RMMM plan for the project.

2. **What you will learn by performing this experiment?**

   i) To identify risk & calculate risk probabilities along with qualitative risk impacts for software under development.

3. **S/w's/Tools Required:**

4. **Theory:**

The risk denotes the uncertainty that may occur in the choices due to past actions and risk is something which causes heavy losses.

Risk management refers to the process of making decisions based on an evaluation of the factors that threats to the business.

Various activities that are carried out for risk management are-

- Risk identification
- Risk projection
- Risk refinement
- Risk mitigation, monitoring and management

➢ **Different types of risks:**

1. Project risk:

Project risk arises in the software development process then they basically affect budget, schedule, staffing, resources and requirements. When project risks become severe then the total cost of project gets increased.

2. Technical risk:

These risks become reality then potential design implementation, interface, verification, and maintenance problems get created. Technical risk occurs when problem becomes harder to solve.

3. Business risk:

When feasibility of software product is in suspect then business risks occur. Business risks can be further categorized as:

i) Market risk- When a quality software product is built but if there is no customer for this product then it is called market risk.

ii) Strategic risk-When a product is built and if it is not following the company's business policies then such product brings strategic risk.

iii) Sales risk- When a product is built but how to sell is not clear then such a situation brings sales risk.

iv) Management risk- When senior management or the responsible staff leaves the organization then management risk occurs.

v) Budget risk-Losing the overall budget of the project is called budget risk.

5. Documentation:

Students should prepare Risk Table & RMMM plan Template for Risk table

- **Template for Risk table:**

No.	Risk	Category	Probability	Impact	RMMM
1.					
2.					
3.					

- **Risk Table Construction**

- List all risks in the first column of the table
- Classify each risk and enter the category label in column two
- Determine a probability for each risk and enter it into column three
- Enter the severity of each risk (negligible, marginal, critical, and catastrophic) in column four.

- **Template for RMMM Plan**

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/02			
Current status: 5/12/02: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

6. Conclusion:

- From Risk table it will be helpful to define & assess for each project risk that can threat to project
- Once RMMM plan has been documented & project has begun & risk mitigation monitoring steps begin.

Experiment No. 10

Change Specification and different versions using SCM tool

Experiment No. 11

To develop test cases using white box testing

Experiment No. 11

1. **Aim:** To Develop test cases using White box testing

2. **Objectives:** From this experiment, the student will be able,

   - To stress the importance of a testing in the development of a software
   - Learn how traceability of tests from the source, allowing future changes to the software to be easily captured in changes to the tests.

3. **Outcomes:** The learner will be able to
   - To validate the software project
   - To engage in continuing professional development and higher studies.

4. **Hardware / Software Required:** Any open source tool or technology can be used for implementation.

5. **Theory:**

   - White-box testing is also known as clear box testing, glass box testing, transparent box testing, and structural testing. It is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).
   - White-box testing is testing that takes into account the internal mechanism of a system or component

   - White-Box Testing Techniques
     - Statement coverage
     - Loop testing
     - Path testing
     - Branch testing

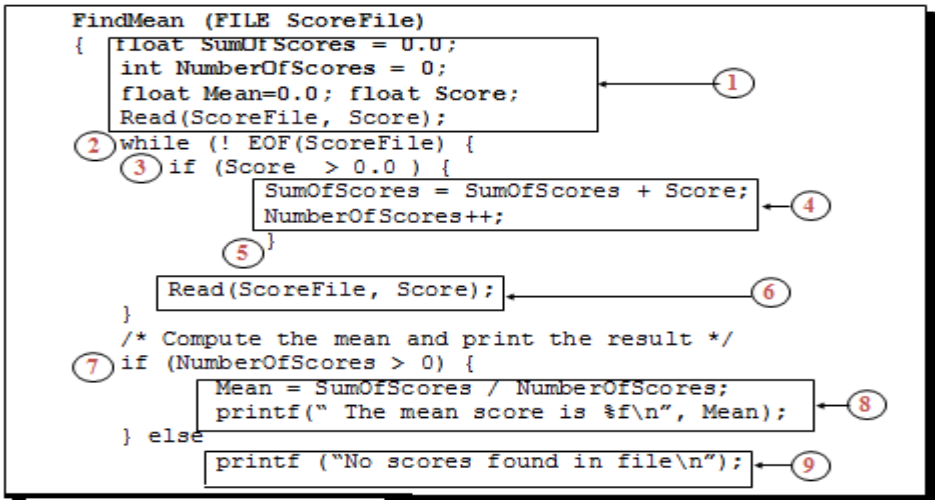
   - 100% method coverage: All methods in all classes are called.
   - 100% statement coverage: All statements in a method are executed.

```
void foo (int a, b, c, d, e)
{
        if (a == 0)
         {
                return;
          }
int x = 0;
```

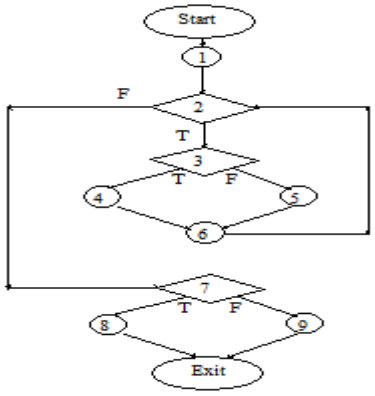
```
if ((a==b) OR ((c==d) AND bug(a) ))
        {
                x =1;
         }
                e = 1/x;
}
```

- Make sure all the paths are covered.
- Determine the paths
- Construct a logic flow chart

- **Path testing: Determine Paths**



- **Logic flow diagram**



- Cyclomatic complexity measures the amount of decision logic in the program module.Cyclomatic complexity gives the minimum number of paths that can generate all possible paths through the module. It is used to define minimum number of test cases required for a module and it is used during software development lifecycle to quantify maintainability and testability. Cyclomatic complexity is defined as

$$CC = E - N + P$$

- Where

  E = the number of edges of the graph

  N = the number of nodes of the graph

  P = the number of connected component

  In case of connected graph

  CC = E - N + 2

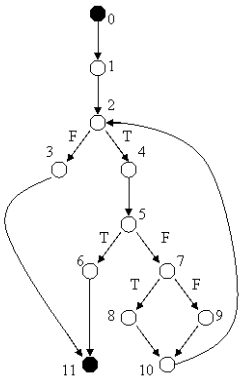
  In simplified way it can be defined as

  $$CC = D + 1$$

  Where

  D = the number of decision points in the graph

- In following example the CC is 4, thus this module has 4 linearly independent paths and this is the minimum number of test paths to achieve maximum coverage



- TC0: 0->1->2->3->11
- TC1: 0->1->2->4->5->6->11
- TC2: 0->1->2->4->5->7->8->10->2->3->11
- TC3: 0->1->2->4->5->7->9->10->2->4->5->6->11

6. Procedure:

- Develop at least 2 test cases from case study & calculate cyclomatic complexity with the help of above description.
- Results are kept in following format:

Sr No.	Test cases	Expected output	Actual output

7. Conclusion:

- Test cases play important role in basis path testing. Using flow graph, we can compute the number of independent paths through the code.

8. Viva Questions:

- What are different testing techniques?
- What is difference between white box testing & Black box testing?

9. References Books & Links:

1. agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf

2. airccse.org/journal/ijesa/papers/2212ijesa04.pdf

3. http://www.whiteboxtest.com/cyclomatic-complexity.php

4. http://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm

Experiment No. :12

Study & Installation of JIRA Agile Project Management Tool

Experiment No. 12

Aim: Study & Installation of JIRA Agile Project Management Tool.

Objectives: From this experiment, the student will be able to

- Learn the efficient tool to understand agile model for Project Development.
- Understand the importance of this experiment from application point of view.
- Learn the efficient tool to manage the Project

Outcomes: Students will be able to understand application of agile process model.

Hardware / Software Required:

Theory:

What is JIRA?

JIRA is a tool developed by Australian Company Atlassian. It is used for **bug tracking, issue tracking,** and **project management**. The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla".

The basic use of this tool is to track issue and bugs related to your software and Mobile apps. It is also used for project management. The JIRA dashboard consists of many useful functions and features which make handling of issues easy. Some of the key features are listed below.

JIRA Defect and Project tracking

Jira Software is an agile project management tool that supports any agile methodology, be it scrum, kanban, or your own unique flavor. From agile boards to reports, you can plan, track, and manage all your agile software development projects from a single tool. Pick a framework to see how Jira Software can help your team release higher quality software, faster.

Agile tool for scrum

Scrum is an agile methodology where products are built in a series of fixed-length iterations. There are four pillars that bring structure to this framework: sprint planning, stand ups (also called daily scrums), sprints, and retrospectives. Out-of-the-box, Jira Software comes with a comprehensive set of agile tools that help your scrum team perform these events with ease.

JIRA Components

Components are sub-sections of a project; they are used to group issues within a project into smaller parts. Components add some structures to the projects, breaking it up into features, teams, modules, subprojects and more. Using components you can generate reports, collect statistics, and display it on dashboards and so on.

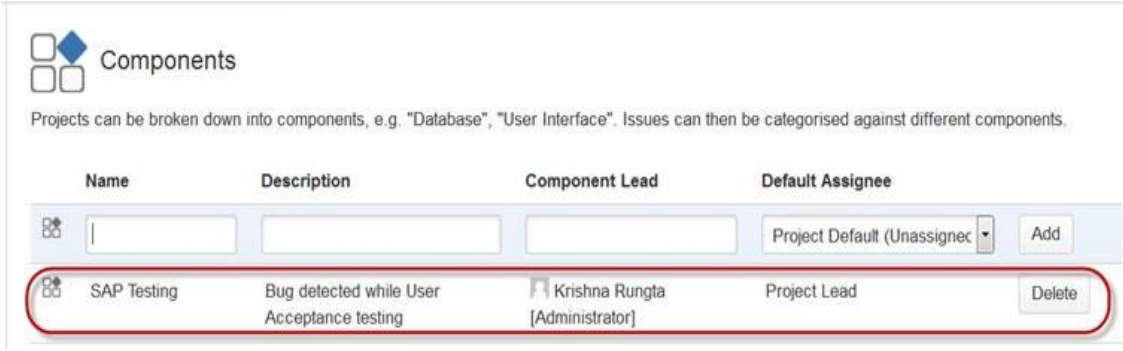


Fig 1: JIRA Components

To add new components, as shown in the above screen you can add **name, description, component lead and default assignee.**

JIRA screen

When issue is created in JIRA, it will be arranged and represented into different fields, this display of field in JIRA is known as a screen. This field can be transitioned and edited through workflow. For each issue, you can assign the screen type as shown in the screen-shot. To add or associate an issue operation with a screen you have to go in main menu and click on **Issues** then click on Screen **Schemes** and then click on **"Associate an issue operation with a screen"** and add the screen according to the requirement.

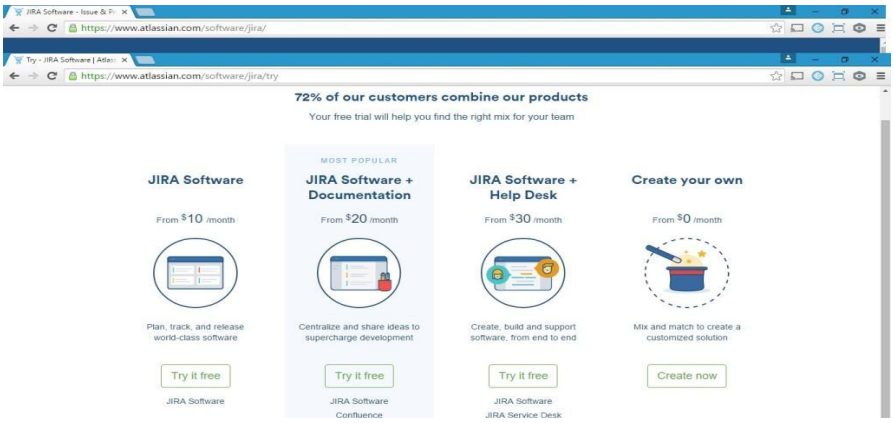
Fig 2: JIRA Screen

Steps to install JIRA Tool

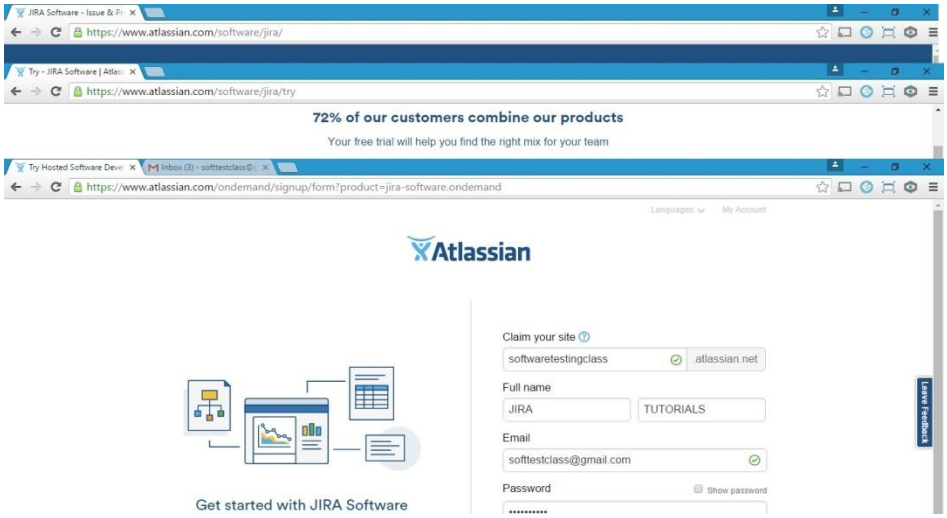
Step 1: Open a web browser and type or click on the below link. It will open the JIRA software website as shown. You can buy the product immediately or try the 7 days free trial version of JIRA.

Download link of JIRA tool

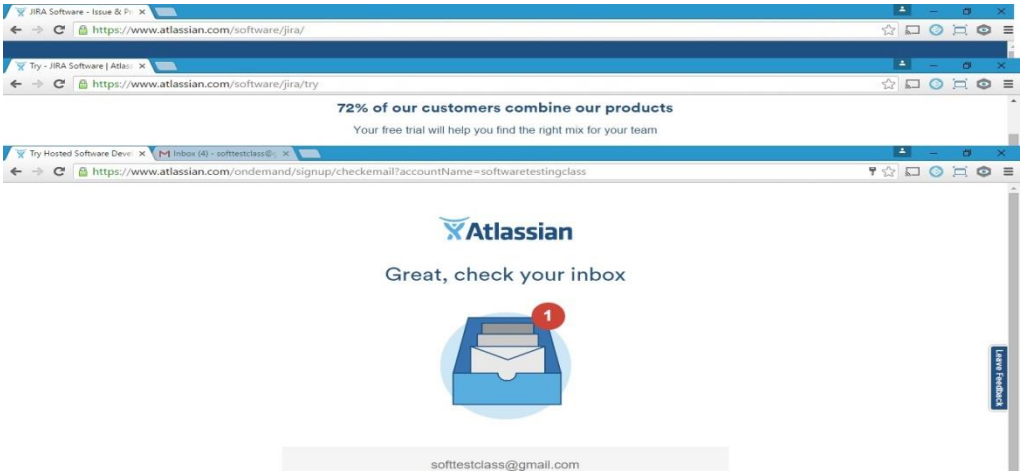
Step 2: Click on the green button with label as "Try it free" which will open up the screen as shown below. We can choose the product among the given 4 options. First option provides the JIRA software which we are going to use in this installation tutorial. Other options provide flexibility of documentation, help desk and customized JIRA software where you can create your own solution by mix and match various features.



Step 3: Click on the JIRA Software option (From $10) and the button with label as "Try it free". It will fetch you a screen as given below. It is a 7 days free trial version.

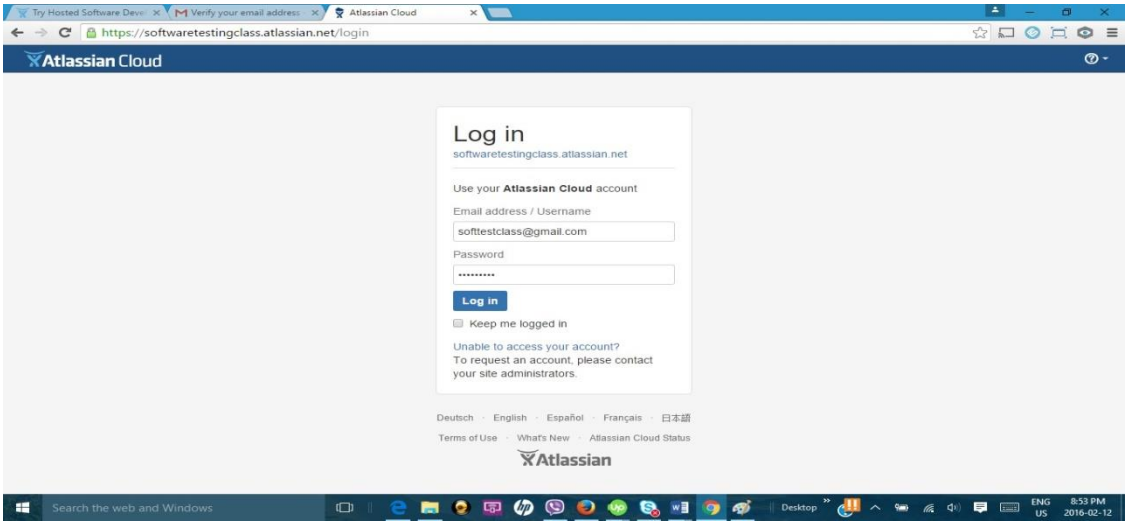


Step 4: On the above screen, you need to choose your own site name (here, I have used softwaretestingclass) and it should be unique for Atlassian.net domain. This URL will create your own secured issue tracking website which can be accessed at any time and at any place. To complete your details enter the full name, email address and password.Email address is required for activation andit should be unique. After entering all of these details click on the button with label as “Strat trial” to proceed further.

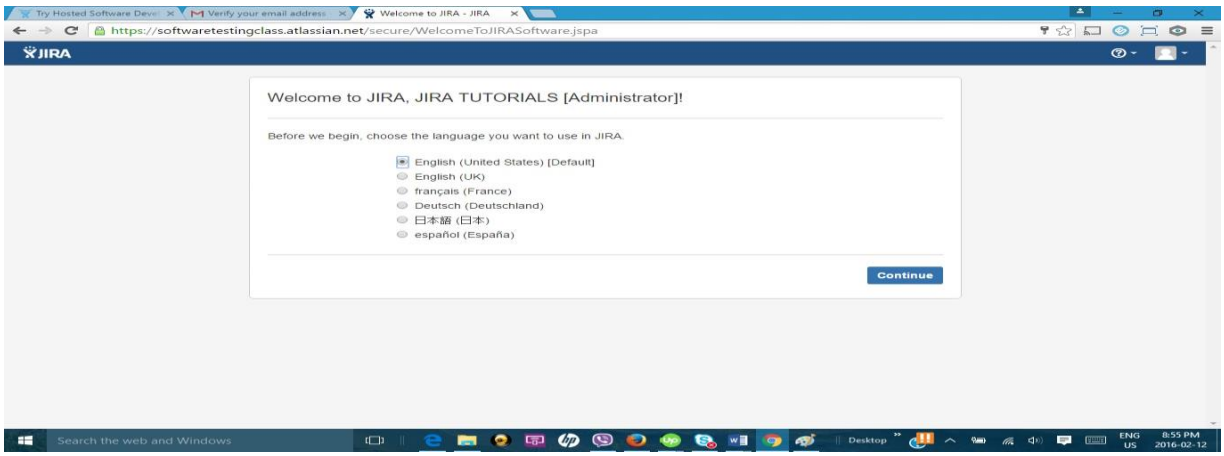


Step 5: After entering details in step 4, it will create your account at Atlassian JIRA cloud and ask to verify your email id after sending an email in your inbox which will look like as given in screenshot below.

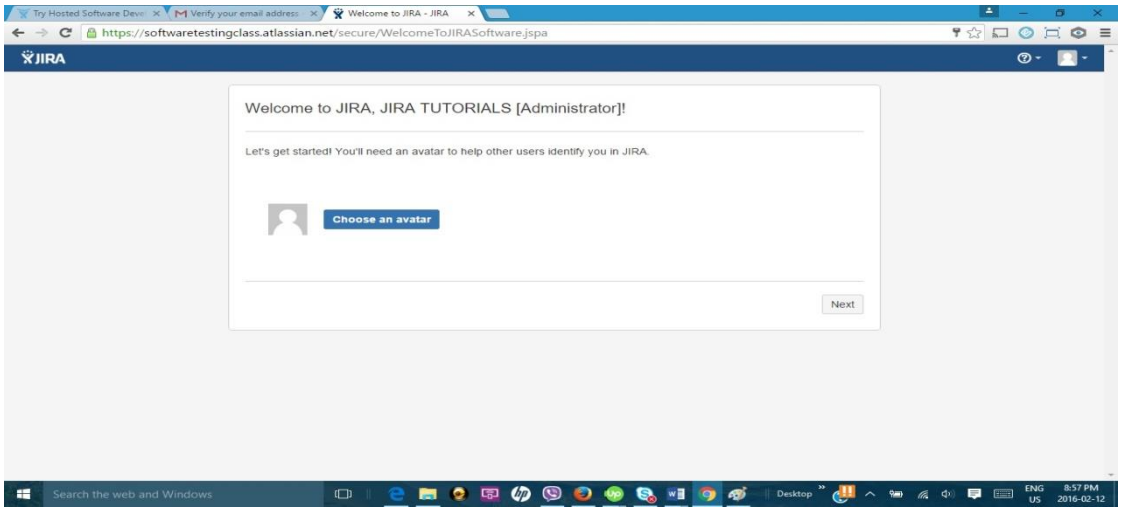
Step 6: Click to verify your email address after opening you email inbox. After verification, it will open up your own website with URL as chosen in the step 4 as shown in the screenshot below.



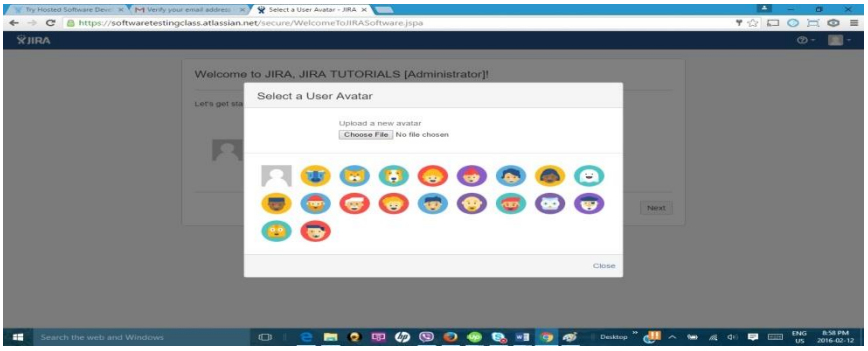
Step 7: Enter your credentials to login into JIRA issue tracking tool. All you need to do is to enter the email address as username and password which was chosen in step 4. Click on the Log in button to log into the JIRA tool.



Step 8: It will open up a screen as shown above which will ask you to choose a language. Default language will be English (United States), other language options available are English (UK), French, Deutsch, Chinese and Spanish. In this tutorial, I have chosen English (United States). Click on the continue button to proceed further.



Step 9: Above Welcome screen will be visible asking you to choose an avatar which is nothing but just a display picture or Profile picture. This picture can be selected from available options of avatars as shown below or can be uploaded from your local PC. It is an optional Step. Choose an avatar and Proceed further.



Step 10: JIRA tool has been successfully set up Atlassian JIRA Cloud. It can be accessed through your URL, username and Password.

6. Conclusion:

JIRA is a single tool which helps to plan, track, and manage all agile software development projects. JIRA Software helps team to release higher quality software, faster. Latest update on the progress of projects is possible with the help of JIRA. Understanding the use we tried installation of JIRA tool where we learn to plan, schedule & track the progress of the project.

Viva Questions:

- What can be referred as an issue in JIRA?
- What is the reason behind using JIRA ?
- Explain what is a workflow?
- What can be referred as an issue in JIRA?

References:

- www.atlassian.com
- https://career.guru99.com/top-18-jira-interview-questions/