

EXPERIMENT NO: 01

Aim: Apply suitable software development model for the given scenario.

Theory:

- **Software Development Process:**

A software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle. Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development and rapid application development.

- **Types of Development Models:**

1. Waterfall Model.
2. Incremental Model.
3. RAD Model.
4. Prototyping.
5. Spiral Model.

- **Waterfall Model:**

1. The Waterfall Model is earliest SDLC approach.
2. It is also known as Classic Life Cycle Model.
3. In this Model, each phase must be completed before the next phase can begin.

- **Features:**

1. It is good to use when technology is well understood.
2. The project is short and cost is low.
3. Risk is zero or simple and easy.

1. **Advantages:**

1. It is simple and easy.
2. Easy to manage.

3. It is good for low budget small project.

2. Disadvantages:

1. It is not good for complex and object-oriented programming.
2. It is a poor model for long and ongoing project.

3. Diagram:

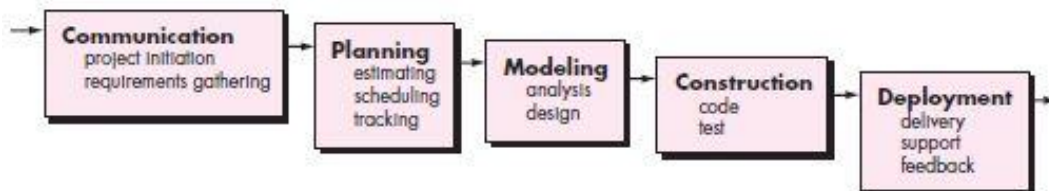


Fig.1.1 Waterfall Model

- **Incremental Model:**

Incremental model is a process of Software Development where requirements are broken down into multiple modules of Software Development Cycle. Increment means to add something. Incremental development is done in steps or in increments from communication, planning, modeling, constructions and deployment. Each iterations passes through all 5 phases . And each subsequent release of the system adds function to the previous relese until all designed fuctionality has been implemeted.

4. Features :

1. Incremental is the combination of Linear and Prototype.
2. Incremental Delivery.
3. Priority to user's highly recommended requirements.
4. Involvement of Users.
5. Lower risk.
6. Highest Priority System means in each increment they can easily find errors.
7. At small requirements we can start our model.
8. After every cycle the product is given to the customer.

5. Advantages:

1. Work with small size team.

2. Initial product delivery is faster.
 3. Customer response or feedback is considered.
 4. It is easier to test and Debug during a smaller iteration.
 5. The model is more flexible.
 6. Easier to manage risk.
6. Disadvantages:
1. Actual cost will be more than Estimated cost.
 2. Needs good planning and design.
7. Diagram:

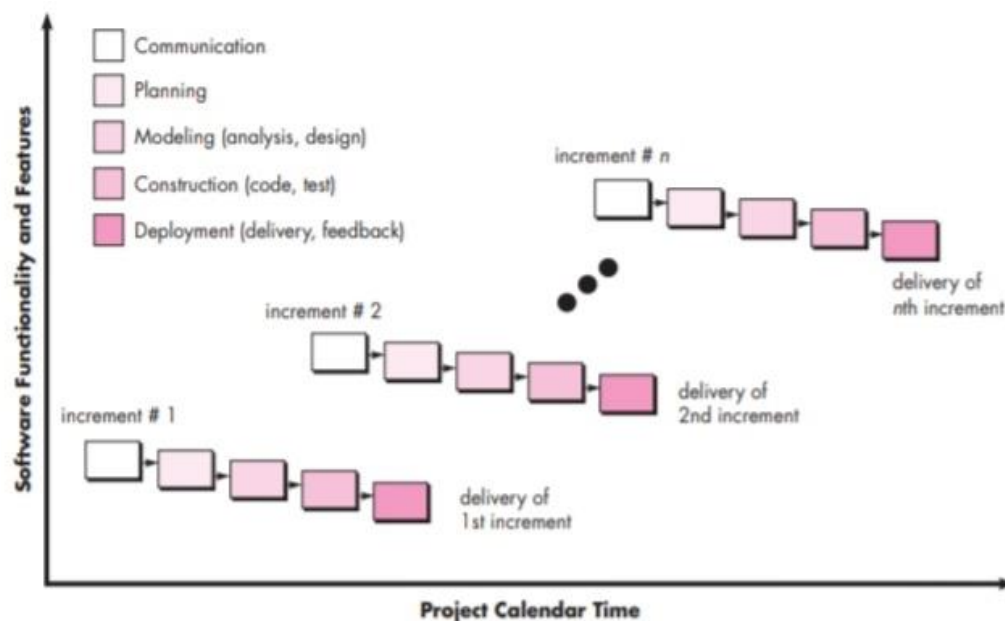


Fig.1.2 Incremental Model.

- **RAD Model:**

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets at developing software in a short span of time. RAD follow the iterative.

- SDLC RAD model has following phases
 1. Business Modelling.
 2. Data Modelling.
 3. Process Modelling.
 4. Application Generation.
 5. Testing and Turnover.

It focuses on input-output source and destination of the information. It emphasizes on delivering projects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD model are that it focuses on the reuse of templates, tools, processes, and code.

8. Different phases of RAD model include:

- Business Modelling:

On basis of the flow of information and distribution between various business channels, the product is designed.

- Data Modelling:

The information collected from business modelling is refined into a set of data objects that are significant for the business.

- Process Modelling

The data object that is declared in the data modelling phase is transformed to achieve the information flow necessary to implement a business function.

- Application Generation

Automated tools are used for the construction of the software, to convert process and data models into prototypes.

- Testing and Turnover

As prototypes are individually tested during every iteration, the overall testing time is reduced in RAD.

9. When to use RAD Methodology?

1. When a system needs to be produced in a short span of time (2-3 months).
2. When the requirements are known.
3. When the user will be involved all through the life cycle.
4. When technical risk is less.
5. When there is a necessity to create a system that can be modularized in 2-3 months of time.
6. When a budget is high enough to afford designers for modelling along with the cost of automated tools for code generation.

10. Advantages:

1. Flexible and adaptable to changes.
2. It is useful when you have to reduce the overall project risk.
3. It is adaptable and flexible to changes.

4. It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used.
 5. Due to code generators and code reuse, there is a reduction of manual coding.
 6. Each phase in RAD delivers highest priority functionality to client.
 7. With less people, productivity can be increased in short time.
11. Disadvantages:
1. It can't be used for smaller projects.
 2. Not all application is compatible with RAD.
 3. When technical risk is high, it is not suitable.
 4. If developers are not committed to delivering software on time, RAD projects can fail.
 5. Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period.
 6. Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application.
 7. Requires highly skilled designers or developers.

12. Diagram:

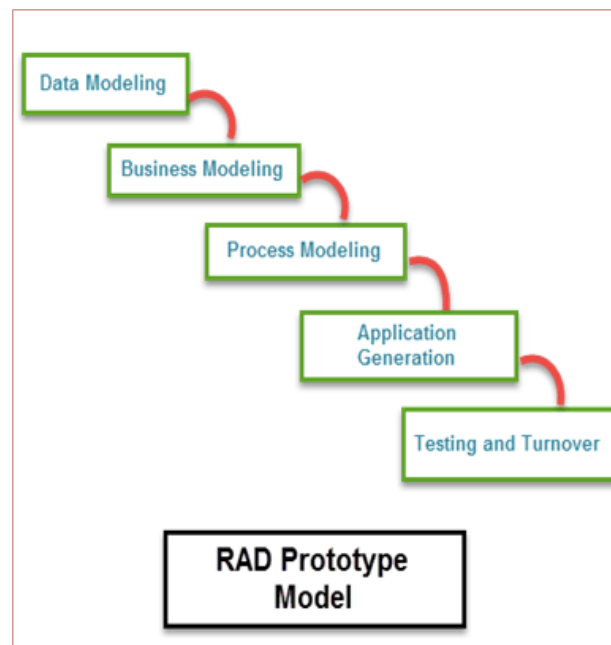


Fig.1.3 RAD Model.

- **Prototyping Model:**

In this model, a prototype of an end product is first developed, tested and defined as per customer's feedback repeatedly till the final expectable prototype is achieved. The intension behind creating this model is to get actual requirements more deeply from the user.

13. Process of Prototyping:

1. Initial requirement identification.
2. Prototype development.
3. Review.
4. Revise.

14. Advantages:

1. The customer gets to see the partial product early in the life cycle.
2. Missing functionality can be easily figured out.
3. Flexibility in design.
4. New requirements can be easily accommodated.

15. Disadvantages:

1. Costly with respect to time as well as money.
2. Poor documentation due to continuously change in requirements.
3. After seeing an early prototype, the customer demands the actual product will be delivered soon.

16. Diagram:

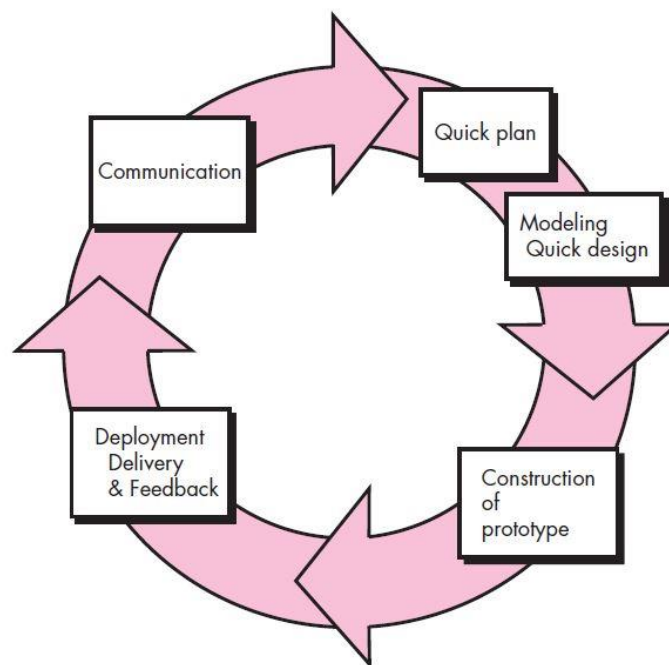


Fig.1.4 Prototype Model.

- **Spiral Model:**

The Spiral model was proposed by Barry Boehm. The Spiral model is the combination of Waterfall model and Incremental model. The Spiral model is divided into 4 task regions. Each task region begins with the design goal and ends with the client reviewing. Software is developed in the series of incremental release. The task region of Spiral model is:

1. Concept Development.
2. System Development.
3. System Enhancement.
4. System Maintenance.

17. Features:

1. When the release is frequent this model is used.
2. Used for large projects.
3. Compatible even if requirements are complex and unclear.
4. Changes can be done at any time.
5. When risk evaluation is important Spiral model is used.

18. Advantages:

1. Additional functions or changes can be done at later stage.
2. Cost estimation becomes easy.
3. There is always space for customer's feedback.
4. Development is fast and features are added in systematic way.

19. Disadvantages:

1. Documentation is more as it has intermediate phases.
2. It is not advisable for smaller projects, as it may cost them a lot.

20. Diagram:

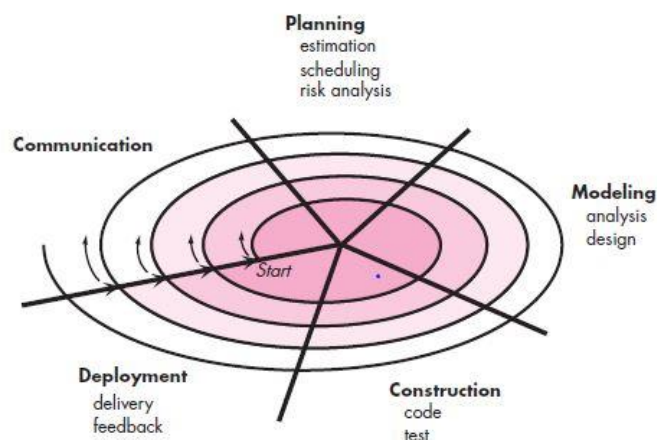


Fig. 1.5 Spiral Model.

- **Scenario:** “Online communication/ chatting web application”

- **Questions:**

- 1. Which model will you use?**

I will use the Incremental Model.

- 2. Why this model? Elaborate.**

- The Incremental Model's main focus is on breaking down the software into multiple modules of software development life cycles
- Incremental development is done in steps from communication, planning, modeling, construction to deployment. Each iteration passes through all the five phases and after each iteration, an updated version of application with added functionalities is provided to the client. This process continues till all the designed functionalities have been implemented.
- We will choose this model for software development because the client wants the operational/working software faster and is ready to accept a working initial product too.
- Also by choosing this model we get the advantage of being able to consider the customer response and feedback after each increment is made to the product and then make the adequate changes.
- The testing and debugging process of this model will be simpler as the previous versions would be already tested and thus we won't have to test the whole project again and again.
- By using Incremental model , we will also be able to make this project more flexible and adaptable to future changes and the process of risk management will also become easier.

- Hence, I chose the Incremental model for the development of the ‘Online communication / chatting web application’.

- **Conclusion:**

Hence, by performing this experiment I learnt about the concept of software development process and the various types of software development models, also I learnt the various pros and cons of the software models and which model to use in the given scenario based on the client’s needs.

(10)	(20)	(10)	(10)	Total (50)

EXPERIMENT NO: 02

Aim: Identify the objectives and summarize outcomes for given scenario, for each SDLC phase.

Theory:

- **Software Development Life Cycle:**

A software development process life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. SDLC is a structure followed by a development team within the software organization. It consists of a detailed plan describing how to develop, maintain and replace specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

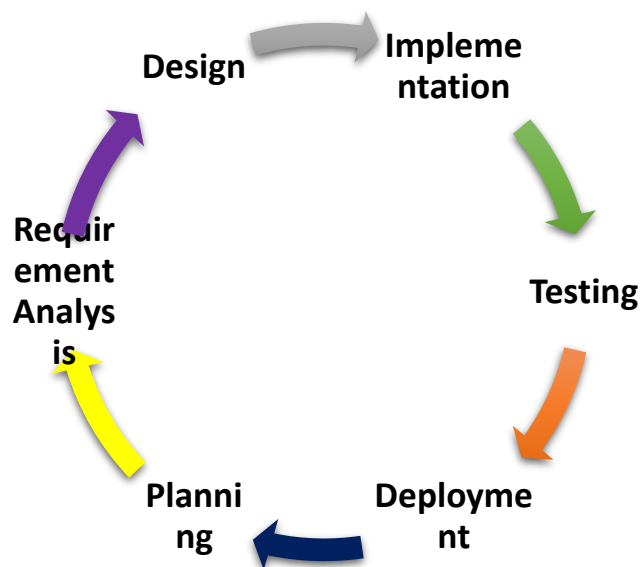


Fig: SDLC phases

- **SDLC Phases**

1. Planning
2. Requirement Analysis
3. Design
4. Implementation

5. Testing

6. Deployment and Maintenance

- **Planning**

The first step in the SDLC defines the scope of the project. For example, your sales reps might be stuck with a hodgepodge of apps on Androids and iPhones to track sales leads. You, the owner, want them on a unified system that works better with your company's internal software. This may mean changing to a single type of hardware and choosing sales tracking software managed by the home office. System planning is the process of deciding what your new information system should look like and then identifying the resources needed to develop it.

- **Requirement Analysis:**

1. Communication

Another Part of Requirement Analysis is Communication where the stakeholders discuss the requirements of the software that needs to be developed to achieve a goal. The aim of the requirement analysis phase is to capture the detail of each requirement and to make sure everyone understands the scope of the work and how, spiral development and each requirement is going to be fulfilled

2. Requirement Gathering

Requirements Gathering (also known as Requirements elicitation or Capture) is the process of generating a list of requirements (functional, system, technical, etc.) from the various stakeholders (customers, users, vendors, IT staff, etc.) that will be used as the basis for the formal Requirement Definition. The process is not as straightforward as just asking the stakeholders what they want the system to do, as in many cases, they are not aware of all the possibilities that exist, and may be limited by their immersion in the current.

3. Defining Requirement

Once the requirement analysis is done the next step is to define the documents the product requirement and get them approved from the customer or the market analysts. This is done

through an SRS document which is consist of all the product requirement to be designed and deployed during the project life cycle. It Also Consist: 1. Cost Estimation

- **Design and Functionality:**

Functionality

Step three in the SDLC is reserved for listing features that support the system's proper functioning. For example, an inventory control system may need to handle at least 15 users or that it should interface with the U.S. Customs database as a compliance check on imports.

1.Tracking

2.Scheduling

Design

1. Algorithm

2. Flowchart

- During the design phase, developers and technical architects start the high-level design of the software and system to be able to deliver each requirement.
- The technical details of the design are discussed with the stakeholders and various parameters such as risks, technologies to be used, capability of the team, project constraints, time and budget are reviewed and then the best design approach is selected for the product.
- The selected architectural design, defines all the components that needs to be developed, communications with third party services, user flows and database communications as well as front-end representations and behavior of each components. The design is usually kept in the Design Specification Document (DSD)
- **Implementation/Coding**

After the requirements and design activity is completed, the next phase of the Software Development Life Cycle is the implementation or development of the software. In this phase, developers start coding according to the requirements and the design discussed in previous phases. Database admits

create the necessary data in the database, front-end developers create the necessary interfaces and GUI to interact with the back-end all based on guidelines and procedures defined by the company.

- **Testing**

After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase all types of functional testing like unit testing, integration testing, system testing, acceptance testing are done as well as non-functional testing are also done.

This cycle is repeated until all requirements have been tested and all the defects have been fixed and the software is ready to be shipped.

- **Deployment and Maintenance**

Once the software has been fully tested and no high priority issues remain in the software, it is time to deploy to production where customers can use the system. Once a version of the software is released to production, there is usually a maintenance team that look after any post-production issues .If an issue is encountered in the production the development team is informed and depending on how severe the issue is, it might either require a hot-fix which is created and shipped in a short period of time or if not very severe, it can wait until the next version of the software.

- **Scenario: “Development of employment salary attendance and biodata management system for a huge government company”**

I. Planning

- Firstly, we discussed the various features and functions of the software to be developed such as attendance report generation per employee/users, editing user details and other functions.
- We also planned which framework to use for the software application while planning as well as which parts can be reused or open source codes.
- Along with that e also planned about the hardware to be used in the development of the application, we planned to have a user log in through the company’s website using his credentials due to the work from home situation, and otherwise we planned to install a fingerprint / ID-card scanner in the office to check-in and check-out incase for the office jobs.
- All the work to be done was planned and organized based on the capabilities of the team members.

II. Requirement Analysis

- The foremost part of requirement analysis is communication. This is where the stakeholders discuss the requirements of the software that needs to be developed to achieve a goal.
- In this phase we communicated with the stakeholders to get an overall understanding of the requirements of the projects and to capture its details and also making sure that everyone understands the scope of the work was also looked after during this process.
- Once the requirement analysis was done, the SRS document of the project was designed after the approval from the stakeholders and was deployed. It also consists of the cost estimation.

III. Modelling

- In the modeling phase of the Software Development Life Cycle, models/prototypes of the software to be created are made in order to better understand the requirements of the software and the design that will achieve these requirements.
- In our software project the modelling activity consisted of two phases namely, analysis and designing.
-

i) Analysis:

In analysis phase, we analyze the need and working of every model. There are three types of models in our project, namely, the GUI (Graphic User Interface) of the application, the flow model of the software application and the database relationship model of the application's database. The analysis of these models to be made will be done in this phase. Further in the design phase the models will be made.

ii) Designing:

In designing phase, the models/prototypes for the software application will be made from the details confirmed from the analysis phase in the modelling.

IV. Construction

- The construction phase is mainly divided into two parts, code generation and software testing.
- In the code generation phase, the software application is coded by the developers.
- For our software project, 'Employment salary attendance and biodata management system', we used flutter and dart for making the android applications and firebase with the use of NOSQL as the database for our application.
- For the web version of the software application, we used HTML, CSS, JavaScript and PHP as backend language along with firebase as database.
- In the testing phase, the software is given to the testers to find the vulnerabilities of the software and to catch any bugs missed by the developers and the priority is finding the bugs fast and getting them fixed.

- After the testing is done and no bugs that can cause an adverse effect on the software application are found then the software product is set for the deployment stage.

V. Deployment

- The software application is given to the customer who evaluate the delivered product and provides feedback based on the evaluation. The feedback that the customer will gives will be used to determine the future additions/updates to be made to the software.

• Questions:

1. Which process according to you works simultaneously and Why?

According to me, the processes communication and modelling are the phases that can work simultaneously. As while communicating and forming ideas in our mind we can create a rough model of the software to be made by getting the specific requirements and details from the customers, and as we get these details we can simultaneously make diagrams and other documents for our modeling phase.

2. Which of the following is the most important phase in SDLC? why?

- I think that all the phases of the SDLC have their own importance but the communication phase is the most important phase of the SDLC according to me.
- The communication phase is the phase where all the stakeholders of the project communicate regarding the software project's details, specific requirements required by the customers as well as the team members.
- Also, the majority of the bugs in the software applications (around 55%) happen due to lack of communication between the stakeholders. As if there are any miscommunication between the stake holders or the team members.
- There is also a risk of failure if there is lack of communication between the stakeholders as it might result into unclear specification and creation of a wrong software product.
- For this reason I think that the communication phase is very crucial and according to me is the most important phase of the Software Developmental Life Cycle.

3. From where do defects and failures in software testing arise?

- The defects and the failures in the process of software testing arises from the errors in the communication, modeling and construction phases.

- The lack of communication or miscommunication in the communication phase can result in errors in the software specifications ultimately resulting errors in the software application.
- The errors in modelling phase can include UI related errors. For example the screen not being responsive, hard to understand the UI. It can also include the database related errors and software model related errors.
- Whereas the errors in the construction phase include run time errors and logical errors which might be caused by the developer's mistake.
- The other types of errors that occur are errors while using the system, intentionally caused damage, environmental conditions (specifications of the device), etc.

4. Which phase of the SDLC is known as the “ongoing phase” and why?

- The maintenance phase of the SDLC is known as the “Ongoing phase”.
- Maintenance is the only phase which is in action after the product has been delivered to the customer.
- As our product is a software application, it will need changes/updates according to the customer's needs.
- The maintenance phase involves making changes to the software according to the changes requested by the customer, enhancing security, the bug fixes missed by the testing team, etc.
- Thus, Maintenance phase is the phase which is also known as ‘Ongoing phase’ in the Software Development Life Cycle.

• Conclusion:

Hence, by performing this practical I learnt about the Software Development Life Cycle (SDLC), the various phases of SDLC, and I also identified the various objectives and summarized the outcomes for the ‘Employment salary attendance and biodata management system for a huge government company’.

(10)	(20)	(10)	(10)	Total (50)

EXPERIMENT NO: 03

Aim: Design Software Requirement Specification (SRS) document for the project. Consider any project to be developed in any technology as a software Architect or project Manager.

Theory:

- **What is a Software Requirements Specification?**

A software requirements specification is a document which is used as a communication medium between the customer and the supplier. When the software requirement specification is completed and is accepted by all parties, the end of the requirements engineering phase has been reached. This is not to say, that after the acceptance phase, any of the requirements cannot be changed, but the changes must be tightly controlled. The software requirement specification should be edited by both the customer and the supplier, as initially neither has both the knowledge of what is required (the supplier) and what is feasible (the customer).

- **Why is a Software Requirement Specification Required?**

A software requirements specification has a number of purposes and contexts in which it is used. This can range from a company publishing a software requirement specification to companies for competitive tendering, or a company writing their own software requirement specification in response to a user requirement document. In the first case, the author of the document has to write the document in such a way that it is general enough as to allow a number of different suppliers to propose solutions, but at the same time containing any constraints which must be applied. In the second instance, the software requirement specification is used to capture the user's requirements and if any, highlight any inconsistencies and conflicting requirements and define system and acceptance testing activities.

A software requirement specification in its most basic form is a formal document used in communicating the software requirements between the customer and the developer. With this in mind then the minimum amount of information that the software requirement specification should contain is a list of requirements which has been agreed by both parties. The requirements, to fully satisfy the user should have the However the requirements will only give a narrow view of the system, so more information is required to place the system into a context which defines the purpose of the system, an overview of the systems functions and the type of user that the system will have. This additional information will aid the developer in creating a software system which will be aimed at the user's ability and the client's function.

- **Types of Requirements**

Whilst requirements are being collated and analysed, they are segregated into type categories. The European Space Agency defined possible categories as

- ☐ Functional requirements,
- ☐ Performance requirements,
- ☐ Interface requirements,
- ☐ Operational requirements,
- ☐ Resource requirements,
- ☐ Verification requirements,
- ☐ Acceptance testing requirements,
- ☐ Documentation requirements,
- ☐ Quality requirements,
- ☐ Safety requirements,
- ☐ Reliability requirements and
- ☐ Maintainability requirements

- **Functional Requirements**

Functional or behavioural requirements are a sub-set of the overall system requirements. These requirements are used to consider trade-offs, system behaviour, redundancy and human aspects. Trade-offs may be between hardware and software issues, weighing up the benefits of each. Behavioural requirements, as well as describing how the system will operate under normal operation should also consider the consequences and response due to software failure or invalid inputs to the system.

- **Performance Requirements**

All performance requirements must have a value which is measurable and quantitative, not a value which is perceptive. Performance requirements are stated in measurable values, such as rate, frequency, speeds and levels. The values specified must also be in some recognised unit, for example metres, centimetre square, BAR, kilometres per hour, etc. The performance values are based either on values extracted from the system specification, or on an estimated value.

- **Interface Requirements**

Interface requirements, at this stage are handled separately, with hardware requirements being derived separately from the software requirements. Software interfaces include dealing with an existing software system, or any interface standard that has been requested. Hardware requirements, unlike software give room for trade-offs if they are not fully defined, however all assumptions should be defined and carefully documented.

- **Operational Requirements**

Operational requirements give an "in the field" view to the specification, detailing such things as:

- ☐ how the system will operate,
- ☐ what is the operator syntax?
- ☐ how the system will communicate with the operators,

- ☐ how many operators are required and their qualification?
- ☐ what tasks will each operator be required to perform?
- ☐ what assistance/help is provided by the system,
- ☐ any error messages and how they are displayed, and
- ☐ what the screen layout looks like.

▪ **Resource Requirements**

Resource requirements divulge the design constraints relating to the utilisation of the system hardware. Software restrictions may be placed on only using specific, certified, standard compilers and databases. Hardware restrictions include amount, percentage or mean use of the available memory and the amount of memory available. The definition of available hardware is especially important when the extension of the hardware, late in the development life cycle is impossible or expensive.

▪ **Verification Requirements**

Verification requirements take into account how customer acceptance will be conducted at the completion of the project. Here a reference should be made to the verification plan document.

Verification requirements specify how the functional and the performance requirements are to be measured and verified. The measurements taken may include simulation, emulation and live tests with real or simulated inputs. The requirements should also state whether the measurement tests are to be staged or completed on conclusion of the project, and whether a representative from the client's company should be present.

▪ **Acceptance Testing Requirements**

Acceptance test requirements detail the types of tests which are to be performed prior to customer acceptance. These tests should be formalised in an acceptance test document.

▪ **Documentation Requirements**

Documentation requirements specify what documentation is to be supplied to the client, either through or at the end of the project.

The documentation supplied to the client may include project specific documentation as well as user guides and any other relevant documentation.

▪ **Quality Requirements**

Quality requirements will specify any international as well as local standards which should be adhered to. The quality requirements should be addressed in the quality assurance plan, which is a core part of the quality assurance document. Typical quality requirements include following ISO9000-3 procedures. The National Aeronautics and Space Administration's software requirement specification - SFW-DID-08 goes to the extent of having subsections detailing relevant quality criteria and how they will be met. These sections are Quality Factors:

- ☐ Correctness
- ☐ Reliability
- ☐ Efficiency
- ☐ Integrity
- ☐ Usability
- ☐ Maintainability
- ☐ Testability
- ☐ Flexibility
- ☐ Portability
- ☐ Reusability
- ☐ Interoperability
- ☐ Additional Factors

Some of these factors can be addressed directly by requirements, for example, reliability can be stated as an average period of operation before failure. However most of the factors detailed above are subjective and may only be realised during operation or post-delivery maintenance. For example, the system may be vigorously tested, but it is not always possible to test all permutations of possible inputs and operating conditions. For this reason, errors may be found in the delivered system. With correctness the subjectifies of how correct the system is, is still open to interpretation and needs to be put into

context with the overall system and its intended usage. An example of this can be taken from the recently publicised 15th point rounding error found in Pentium (processors). In the whole most users of the processor will not be interested in values of that order, so as far as they are concerned, the processor meets their correctness quality criteria, however a laboratory assistant performing minute calculations for an experiment this level of error may mean that the processor does not have the required quality of correctness.

▪ **Safety Requirements**

Safety requirements cover not only human safety, but also equipment and data safety. Human safety considerations include protecting the operator from moving parts, electrical circuitry and other physical dangers. There may be special operating procedures, which if ignored may lead to a hazardous or dangerous condition occurring. Equipment safety includes safeguarding the software system from unauthorised access either electronically or physically. An example of a safety requirement may be that a monitor used in the system will conform to certain screen emission standards or that the system will be installed in a Faraday Cage with a combination door lock.

▪ **Reliability Requirements**

Reliability requirements are those which the software must meet in order to perform a specific function under certain stated conditions, for a given period of time. The level of reliability requirement can be dependent on the type of system, i.e. the more critical or life threatening the system, the higher the level of reliability required. Reliability can be measured in a number of ways including number of bugs per x lines of code, mean time to failure and as a percentage of the time the system will be operational before crashing or an error occurring. Davis states however that the mean time to failure and percent reliability should not be an issue as if the software is fully tested, the error will either show itself during the initial period of use, if the system is asked to perform a function it was not designed to do or the hardware/software configuration of the software host has been changed. Davis suggests the following hierarchy when considering the detail of reliability in a software requirement specification.

- ☐ Destroy all humankind
- ☐ Destroy large numbers of human beings
- ☐ Kill a few people
- ☐ Injure people
- ☐ Cause major financial loss
- ☐ Cause major embarrassment
- ☐ Cause minor financial loss
- ☐ Cause mild inconvenience
- ☐ Naming conventions
- ☐ Component headers
- ☐ In-line document style
- ☐ Control constructs
- ☐ Use of global/common variables

- **Maintainability Requirements**

Maintainability requirements look at the long-term life of the proposed system. Requirements should take into consideration any expected changes in the software system; any changes of the computer hardware configuration and special consideration should be given to software operating at sites where software support is not available. Davis suggests defining or setting a minimum standard for requirements which will aid maintainability i.e.

- **Characteristics of a Good Software Requirements Specification**

A software requirements specification should be clear, concise, consistent and unambiguous. It must correctly specify all of the software requirements, but no more. However, the software requirement specification should not describe any of the design or verification aspects, except where constrained by any of the stakeholder's requirements.

- **Complete**

For a software requirements specification to be complete, it must have the following properties:

Description of all major requirements relating to functionality, performance, design constraints and external interfaces.

Definition of the response of the software system to all reasonable situations.

Conformity to any software standards, detailing any sections which are not appropriate

Have full labelling and references of all tables and references, definitions of all terms and units of measure.

Be fully defined, if there are sections in the software requirements specification still to be defined, the software requirements specification is not complete.

- **Consistent**

A software requirement specification is consistent if none of the requirements conflict. There are a number of different

types of confliction:

Multiple descriptors - This is where two or more words are used to reference the same item, i.e. where the term cue and prompt are used interchangeably.

Opposing physical requirements - This is where the description of real-world objects clash, e.g. one requirement states that the warning indicator is orange, and another states that the indicator is red.

Opposing functional requirements - This is where functional characteristics conflict, e.g. perform function X after both A and B has occurred, or perform function X after A or B has occurred.

- **Traceable**

A software requirement specification is traceable if both the origins and the references of the requirements are available. Traceability of the origin or a requirement can help understand who asked for the requirement and also what modifications have been made to the requirement to bring the requirement to its current state. Traceability of references are used to aid the modification of future documents by

stating where a requirement has been referenced. By having foreword traceability, consistency can be more easily contained.

- **Unambiguous**

As the Oxford English dictionary states the word unambiguous means "not having two or more possible meanings". This means that each requirement can have one and only one interpretation. If it is unavoidable to use an ambiguous term in the requirements specification, then there should be clarification text describing the context of the term. One way of removing ambiguity is to use a formal requirements specification language. The advantage to using a formal language is the relative ease of detecting errors by using lexical syntactic analysers to detect ambiguity. The disadvantage of using a formal requirements specification language is the learning time and loss of understanding of the system by the client.

- **Verifiable**

A software requirement specification is verifiable if all of the requirements contained within the specification are verifiable. A requirement is verifiable if there exists a finite cost-effective method by which a person or machine can check that the software product meets the requirement. Non-verifiable requirements include "The system should have a good user interface" or "the software must work well under most conditions" because the performance words of good, well and most are subjective and open to interpretation. If a method cannot be devised to determine whether the software meets a requirement, then the requirement should be removed or revised.

- **Scenario: Industrial Employees Attendance Management System.**

□ INTRODUCTION:

An Industrial Employees Attendance Management System, allows the users (employees) to log in to their account and mark their respective attendance from anywhere through this software. This software helps the users by making the process of marking their attendance easily from their home due to work from home policies.

Purpose:

The software allows the user to access the attendance system and mark their attendance from anywhere by online accessing through the software.

Scope:

- An online attendance management system which provides attendance record, login-logout functions and other facilities.
- It has cut down the time-consuming and manual process of marking the attendance in the register by making online attendance marking facility available.

Definitions, Acronyms, and Abbreviations:

- It is a web application which provides attendance facilities to employees and admin access to the administrators.
- Few of the acronyms used in the software are
 - AB – Absent
 - P – Present
 - ID – Identity Card
 - Dt – Date

References:

- Software Engineering A practitioner's approach – Roger. S. Pressman
- Clean Code: A handbook of agile software craftsmanship. – Robert. C. Martin

□ OVERALL DESCRIPTION

Product Perspective:

- This software product, i.e. an Industrial Employee Attendance Management System is a standalone product/software system.
- As hardware this software can be accessed through any web-browser.
- The software also stores the user's data in a secure database which contains user information, login details, etc.

Product Function:

- Employee information editing system.
- Login and logout system.
- Employee Attendance checking (Admin)

User Characteristics:

- Little to minimal education about a few technical terms.
- Little knowledge about compute/android device.
- Basic knowledge about web-browsing.

Constraints:

- A mobile phone with android version 7.0 or above, or a computer system with windows XP or above.
- An active internet connection for accessing the web site.
- Compatible web browser, e.g. chrome, brave, etc.

Assumptions and Dependencies:

- Users can log in through their respective username or user ID no.
- Admins will be able to print the attendance of the employees in an excel sheet format.

□ SPECIFICATION REQUIREMENTS

External interfaces:

- It uses resources from hardware components which include RAM, Hard-disk, network card, etc.
- The web browser should also be the latest version for the software to run well.

Functions:

- Logging in and logging out by the employee at the time of starting, ending work respectively.
- Creating account for new employee in the attendance system
- Accessing employee attendance details
- Editing the employee details.

Performance Requirements:

For Computer Users:

- Intel Pentium 4 processor or above
- 2 GB or more than that of RAM
- A minimum screen resolution of the display/monitor should be 800*600, whereas a display of 1600*900 or 1920*1080 is recommended.
- A compatible browser

For Android users:

- 512MB or more than that of RAM
- A compatible browser
- Android version 7.0 or above

Logical Database Requirement:

- As the software will be storing all the data on cloud storage, it does not need any write permissions.
- The admins should have access to the data in the database and should be able to fetch it whenever required.

Design constraints:

- As this software is a web-application, there are no specific constraints apart from the hardware requirements stated beforehand.
- The website can be accessed directly through any web-browser and a stable internet connection.

Software System Quality:

- Maintainability :
 - The client will have some time after we deliver this app to them, once the software application is delivered.
 - If anything happens during the period, the software will be fixed and updated without any extra charges.
- Correctness:
 - The tests done by the testing team should be done manually by the testing team members, so that the accuracy of the checks can be maintained, and errors are debugged before the delivery of the product.
- Flexibility :
 - The software system should be flexible and any changes that the user needs after the software system is delivered shall be made to the software at the decided cost by the customer and firm in the contract.

Object Oriented Models:

- The online Industrial Employees Attendance Management System can be described by the use of class diagram, flow state diagram and other such models to show the relationship between the various classes that compose the system.

□ Appendices:

- RAM – Random Access Memory
- MB – Mega Byte
- GB – Giga Byte

□ Index:

- 1 Functions - 2.2, 2.3
- 2 Overview - 1.5, 2.4, 3.7
- 3 Performance Requirements – 3.1, 3.3, 3.4

• Conclusion:

Hence, by performing this practical I learnt about the Software Requirement Specification (SRS) document, its requirement and how to design a SRS document. I also designed a Software Requirement Specification (SRS) document for the ‘Industrial Employees Attendance Management System’ considering the software to be developed in any technology as a software Architect or a project Manager.

(10)	(20)	(10)	(10)	TOTAL (50)

EXPERIMENT NO: 04

Aim: Classify above identified requirement into functional and non-functional requirements.

Theory:

- **What is SRS?**

SRS stands for Software Requirement Specification. Software requirement is a functional or non-functional need to be implemented in the system. Functional means providing particular service to the user. For example, in context to banking application the functional requirement will be when customer select “View Balance” they must be able to look at their latest account balance. Software requirement can also be a non-functional, it can be a performance requirement. For example, a non-functional requirement is where every page of the system should be visible to the users within 5 seconds. So, basically Software requirement is a ☐ Functional or ☐ Non-functional need that has to be implemented into the system. Software requirement is usually expressed as a statement.

- **Functional Requirement:**

Functional requirements are the desired operations of program, or system as defined in software development and systems engineering. It describes the functions a software must perform. A function is nothing but inputs, its behaviour, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other functionality which defines what function a system is likely to perform. Functional Requirements are also called Functional Specification.

- **Non-Functional Requirement:**

Non-functional requirements describe how the system works. A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of system. A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Non-functional requirements are often called Quality attributes of a system. Example, how fast does the website load?

- **Scenario: “Students attendance management system”.**

- **Questions:**

1. What are the Functional Requirements for above scenario?

The various functional requirements for the scenario – ‘Students attendance management system’ are:

- Storing the required details of the students
- Updating the attendance of the students in the database
- Generation of Excel sheet of student attendance record
- Editing student attendance details
- Deleting student attendance details
- Notifying students with low attendance percentage

2. What are Model: n-functional Requirements for above scenario?

The non-functional requirements for the above scenario are:

- **Maintainability:**
The software application should be maintainable. That is, the software should be able to be updated according to the new changes requested by the client depending on the feedback of the client.
- **Security:**
The software application should have two factor authentication for login and sign up. Also the software database should be secure so that there is no chance for data breach of student information. For this the software system should have only administrator access.
- **Safety:**
The software application should a backup server with data backup so that if any mishaps were to happen the damage could be minimized.
- **Portability:**
The software application should be portable, i.e. if the user is logging in from a different device with the same id, the user should have full access to the data of the attendance and students.

3. What are Benefits of Non-functional Requirements?

The benefits of Non-functional Requirements are:

- Non-functional Requirements ensure the maintainability and reliability of the software.
- Non-functional Requirements ensure that the software is safe and secure by constructing the security policy of the software system and by creating backup of the data to keep it safe.
- Non-functional Requirements ensure that the performance of the software and User Experience is up to the mark.

4. Differentiate between Functional Requirements and Non-functional Requirements?

Functional Requirements:

1. Functional Requirement defines a component/part of a system.
2. Functional Requirement is specified by the user.
3. Functional Requirement is mandatory.
4. Functional Requirement is usually easier to define.
5. Functional Requirement is defined at the component level.
6. Functional Requirement helps in verifying the functionality of the software.
7. Functional Requirement specifies/describes what the product does.
8. Functional Requirement's end result can be said to be the features of the product.
9. Example,
 - a. Editing student attendance details.
 - b. Uploading data to database.

Non-functional Requirements:

1. Non-functional Requirement defines the quality attribute of a system.
2. Non-functional Requirement is specified by the technical team, i.e. the Software architect, Software developer, etc.
3. Non-functional Requirement is not mandatory.
4. Non-functional Requirement is comparatively harder to define.
5. Non-functional Requirement is applied to the system as a whole.
6. Non-functional Requirement helps in verifying the performance of the software.
7. Non-functional Requirement specifies/describes how the product works.
8. Non-functional Requirement's end result can be said to be the features of the product.
9. Example,
 - a. The attendance should be updated within 500ms of application starting.
 - b. Backup of data should be stored for safety.

- **Conclusion:**

Hence, in this practical I have classified the above identified requirements on 'Students attendance management system' into functional and non-functional requirements and also differentiated between functional requirements and non-functional requirements.

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO: 05

Aim: Design USE case diagram, state diagram for given scenario.

Theory:

- **What is a USE case diagram?**

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

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

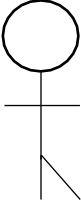
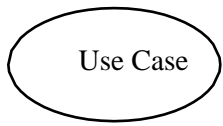

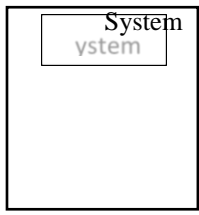
- **When to apply use case diagrams**

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known.

These diagrams are used at a very high level of design. This high-level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the pre-condition, post condition, and exceptions. These extra elements are used to make test cases when performing the testing.

Use case diagrams can be used for –

- Requirement analysis and high-level design.
- Model the context of a system.
- Reverse engineering.
- Forward engineering

Notation Description	Visual Representation
<p>Actor Someone interacts with use case (system function). Similar to the concept of user, but a user can play different roles Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).</p>	
<p>Use Case System function (process - automated or manual) i.e. Do something Each Actor must be linked to a use case, while some use cases may not be linked to actors.</p>	
<p>Communication Link Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.</p>	
<p>Boundary of system The system boundary is potentially the entire system as defined in the requirements document. For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc.</p>	

Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

- Association between an actor and a use case
- Generalization of an actor
- Extend relationship between two use cases
- Include relationship between two use cases
- Generalization of a use case

- **How to Create a Use Case Diagram**

- **Identifying Actors**

Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system, the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role you're trying to show in the use case.

- **Identifying Use Cases**

Now it's time to identify the use cases. A good way to do this is to identify what the actors need from the system. In a banking system, a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So, all of these can be considered as use cases.

Top level use cases should always provide a complete function required by an actor. You can extend or include use cases depending on the complexity of the system.

- **Look for Common Functionality to use Include**

Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that it's always called when the original use case is executed.

- **Is it Possible to Generalize Actors and Use Cases?**

There may be instances where actors are associated with similar use cases while triggering a few use cases unique only to them. In such instances, you can generalize the actor to show the inheritance of functions.

One of the best examples of this is "Make Payment" use case in a payment system. You can further generalize it to "Pay by Credit Card", "Pay by Cash", "Pay by Check" etc. All of them have the attributes and the functionality of payment with special scenarios unique to them.

- **Optional Functions or Additional Functions**

There are some functions that are triggered optionally. In such cases, you can use the extend relationship and attach an extension rule to it. In

the below banking system example “Calculate Bonus” is optional and only triggers when a certain condition is matched.

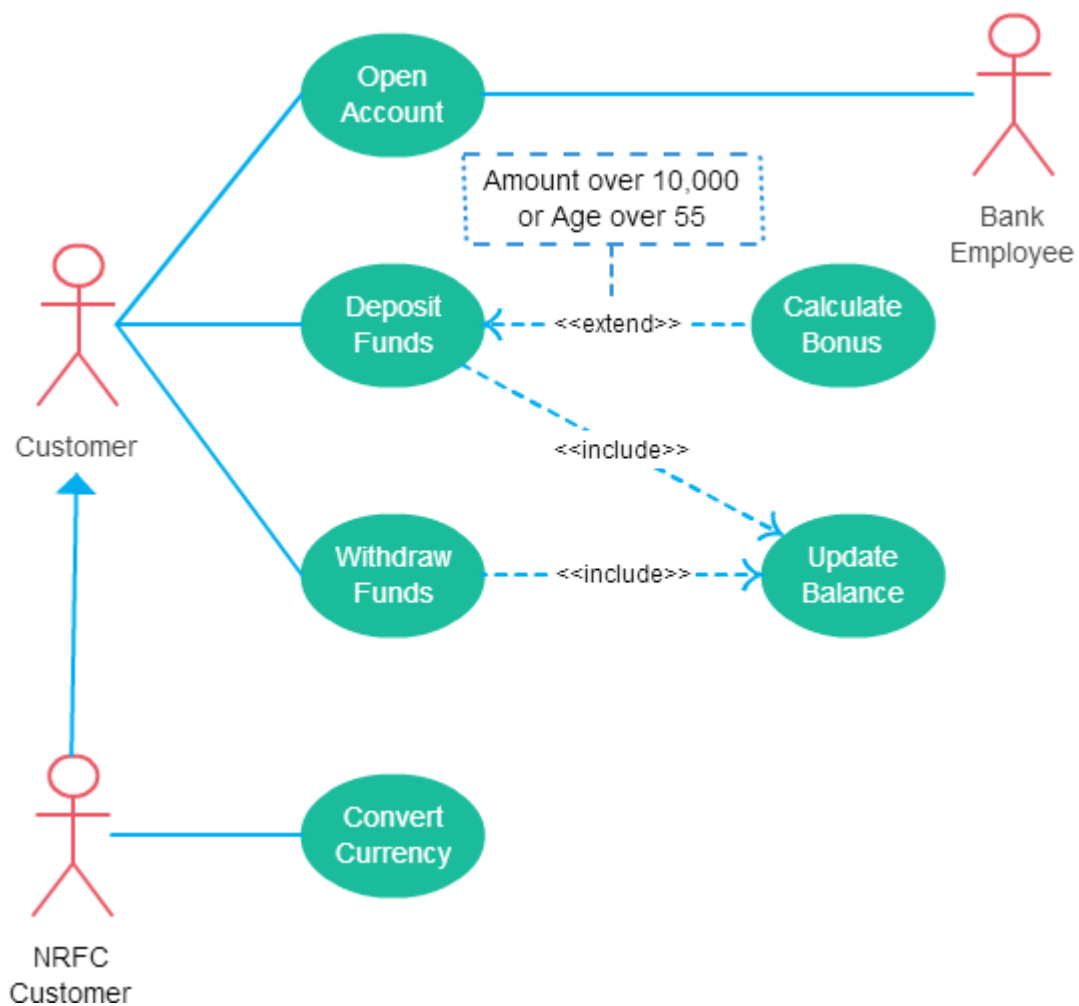


Fig. Example of use case diagrams

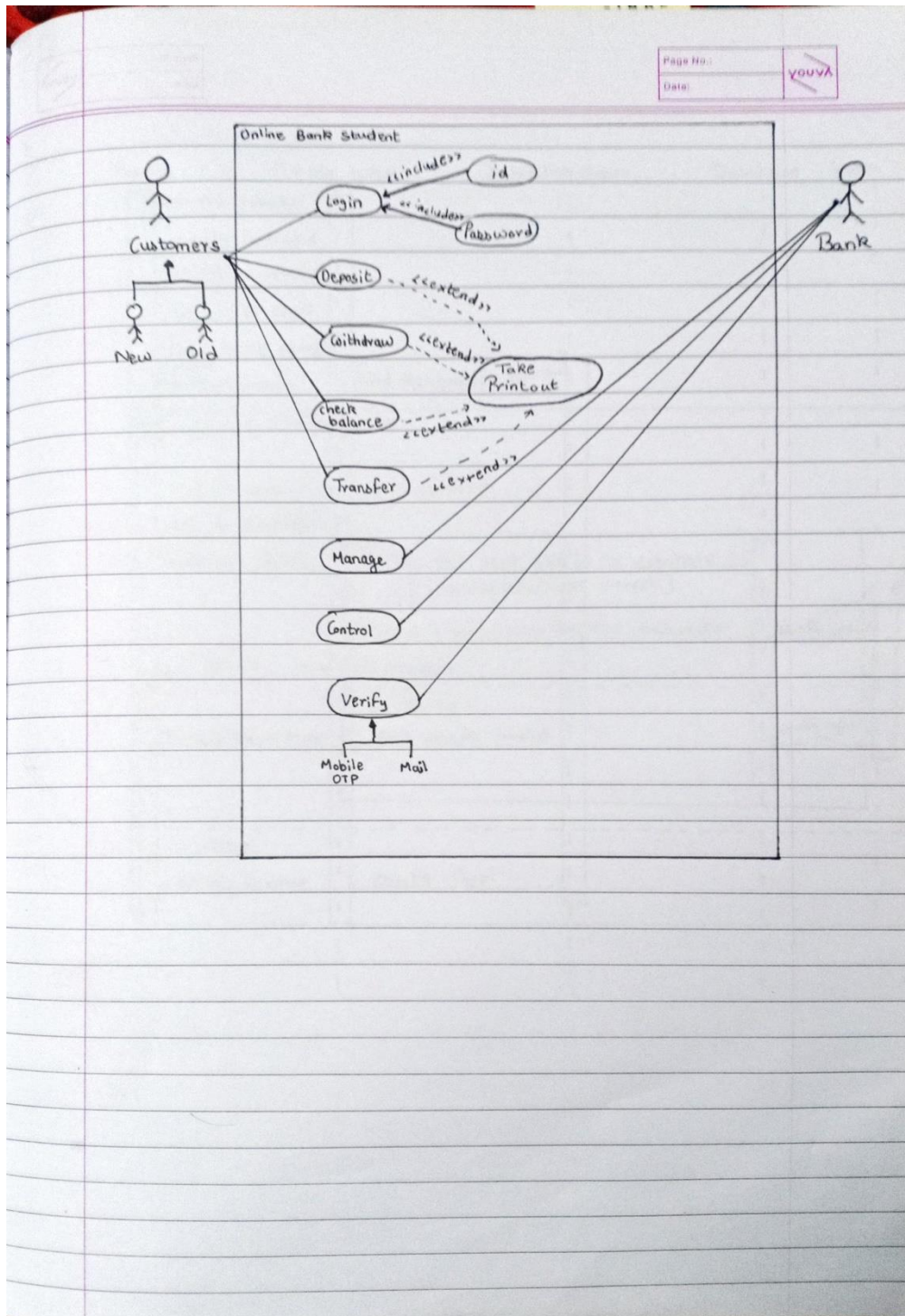
- **Scenario: Online Banking System**

- **Questions:**

- 1. What is the Importance of Use Case Diagrams?**

- The use case diagram can summarize the details of your system's users (also called as actors) and their interactions with the system.
- Use case diagrams specify the events of a system and their flows, An effective use case diagram can help the team in discussing and representing factors such as:
 - Scenarios in which your system or application interacts with people, organisations, or external systems.
 - Goals that your system or application helps those entities (known as actors) achieve.
 - The scope of your system.
- A well-structured use case also describes the pre-condition, post condition and exceptions of the software system.

2. Draw a USE case diagram for given scenario?



3. Are use cases the same as functional requirements are different from use cases?

- No, use case diagrams are different from functional requirements.
- Functional requirements are a set of requirements that define the system functionality being developed.
- Functional requirements is mostly in text form.
- Whereas, use case diagrams can be said to be a pictorial / diagrammatic representation of a software system.
- The major difference is that use case diagrams are a graphical representation of the systems requirements, whereas functional requirements are in text form.
- Use case diagrams can also have text in them, but the major focus is on the diagram itself, whereas in functional requirements the major focus is on the written text.

4. Which part of a use case description can also be modelled by using an activity diagram? and why?

- It is possible to model the events and activities of a use case diagram in an activity diagram.
- It is because in the use case diagram the data flow in the system shows with the help of flow activity part which is also used in activity diagram to show the flow of data in the system.
- By using that part, both the diagrams show the flow of data or sequence.

- **Conclusion:**

Hence, by performing this experiment on the Use Case Diagrams. I learnt about the use case diagram, the various notations, descriptions, visual representation and how to create it. I also designed Use Case Diagram for the given scenario.

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO: 06

Aim: Create Sequence diagram, state diagram for given scenario.

Theory:

- **What is a Sequence diagram?**

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are sometimes called event diagrams or event scenarios. Sequence diagrams are preferred by both developers and readers alike for their simplicity.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

- **High-Level Sequence Diagrams:**

High-level sequence diagrams give a good overview of the interactions between customers, partners, and the business system. They serve as the basis for the electronic data transfer between the business system and customers, business partners, and suppliers (see Modelling for System Integration).

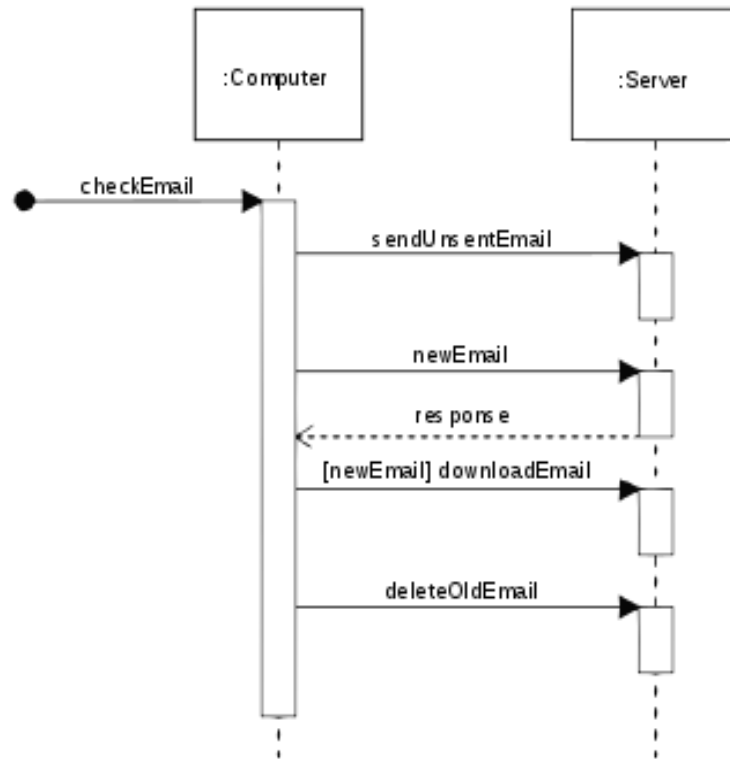


Fig: Example of sequence diagram

- **What is State diagram?**

A state diagram is a type of diagram used in computer science and related fields to describe the behaviour of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

State diagrams are used to give an abstract description of the behaviour of a system. This behaviour is analysed and represented as a series of events that can occur in one or more possible states. Hereby "each diagram usually represents objects of a single class and track the different states of its objects through the system".

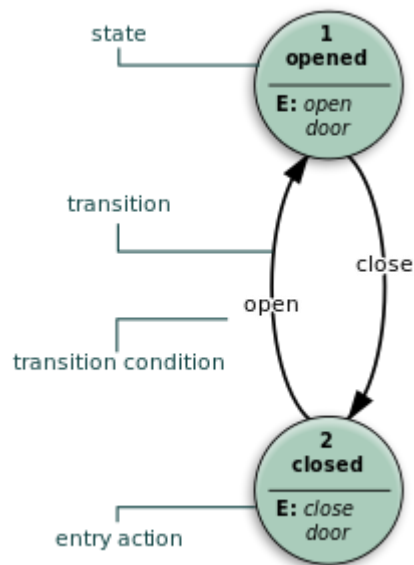


Fig: Example of State diagram

- **Purpose of state diagram:**

Its specific purpose is to define the state changes triggered by events. Events are internal or external factors influencing the system. State chart diagrams are used to model the states and also the events operating on the system.

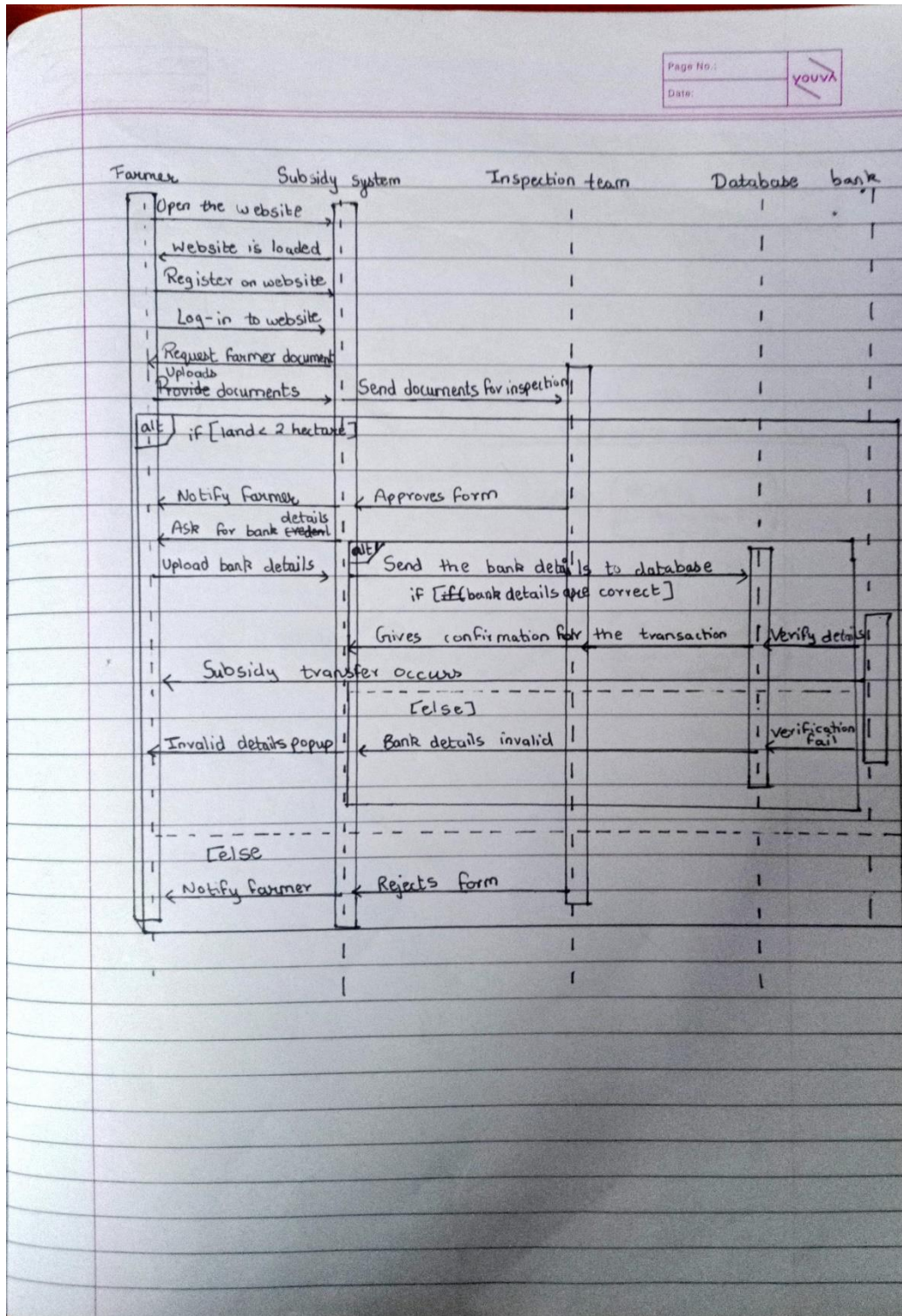
- **Scenario:**

1. Government wish to transfer subsidy to those farmer's bank account having land less than 2 hectares.

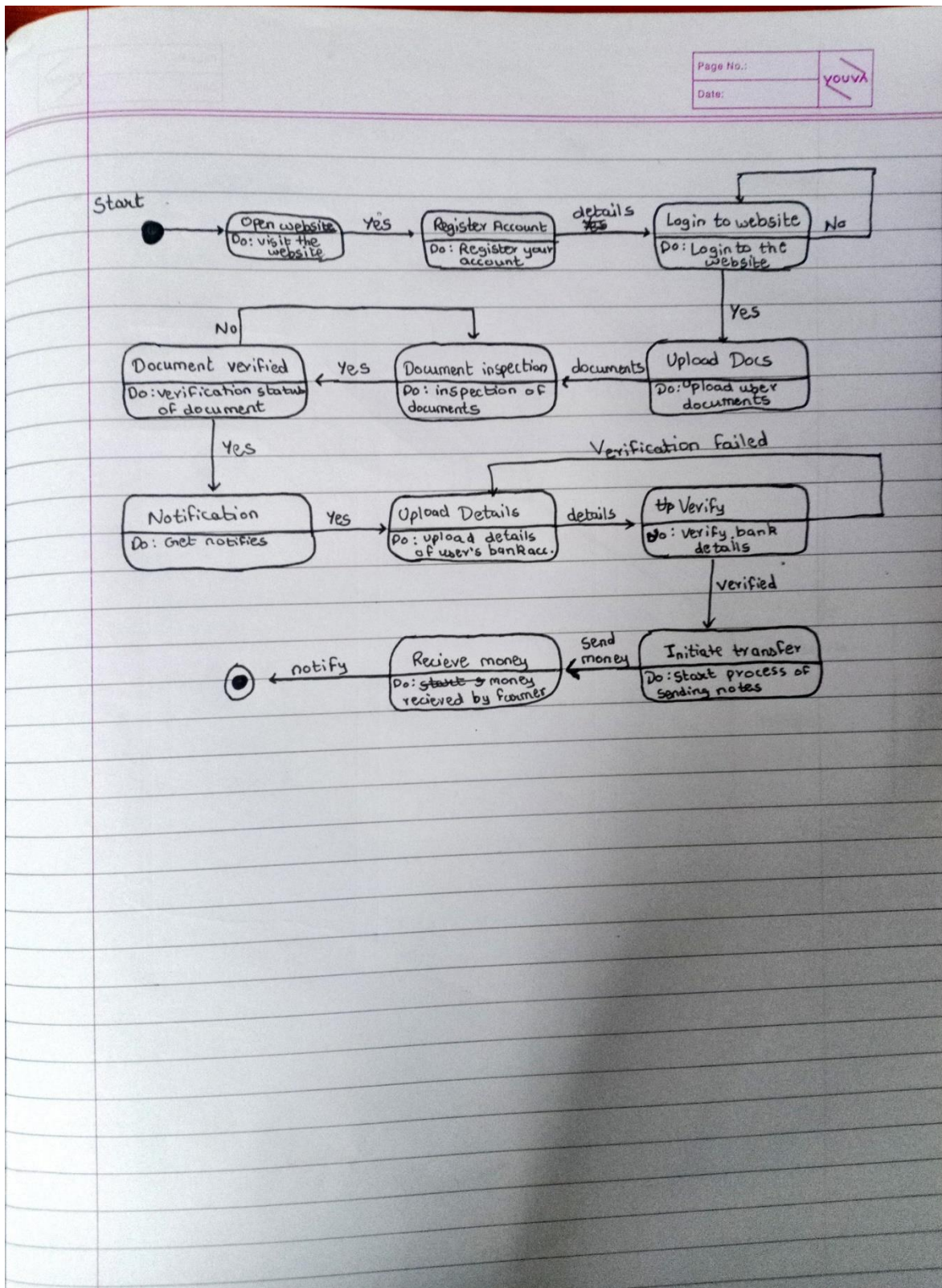
- a. Farmer needs to register themselves on a provided website.
- b. Farmer needs to upload required documents.
- c. Inspection team inspects farmer document and land.
- d. Government successfully transfers the subsidy in farmer's bank account.

- Questions:

1. Draw Sequence diagram for given scenario.



2. Draw State diagram for given scenario.



3. Comparison between sequence diagram and state diagram.

- A sequence diagram generally shows the execution of a particular use case for the application of and the objects that are involved in carrying out that use case.
- A state diagram shows the various states that are valid for an object. That could be a particular functionality or the system as a whole.
- A sequence diagram is aimed at one specific function, example, fetching a document from database and printing it. A state diagram.
- A state diagram on the other hand can show a model for the whole system.

- **Conclusion:**

Hence, by performing this experiment I learnt about the state diagram and the sequence diagram, and their application. I also created sequence diagram and state diagram for the scenario – ‘Government wish to transfer subsidy to farmer’s bank with less than 2 hectares of land.’

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO:07

Aim: Draw E-R diagram, DFD and create data dictionary for above system.

Theory:

- **What is Entity?**

An entity is any object in the system that we want to model and store information about. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database.

Some specific examples of entities are Employee, Student, Lecturer.

- **What is E-R diagram?**

An entity relationship model, also called an entity - relationship (ER) diagram, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems.

For example: In the following ER diagram we have - two entities Student and College and these two entities have many to one relationship as many student's study in a single college.

- **ER Diagram Uses:**

- ☐ When documenting a system or process, looking at the system in multiple ways increases the understanding of that system.

ERD diagrams are commonly used in conjunction with a data flow diagram to display the contents of a data store.

- **Common Entity Relationship Diagram Symbols:**

- ☐ **Entities**, which are represented by rectangles. An entity is an object or concept about which you want to store information.



- **weak entity** is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



- **Relationship**, which are represented by diamond shapes, show how two entities share information in the database.



- **Attributes**, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity.



- **multivalued attribute** can have more than one value. For example, an employee entity can have multiple skill values.

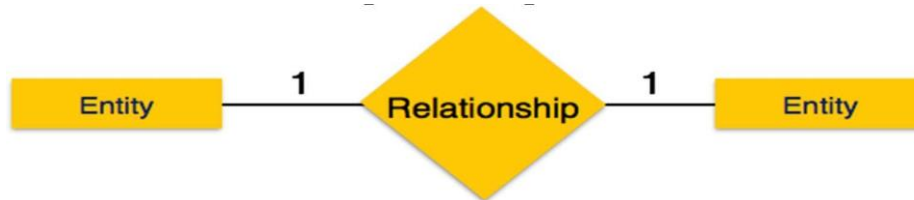


- **Connecting lines**, solid lines that connect attributes to show the relationships of entities in the diagram.

- **Relationship**

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

- **One-to-one** – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



- **One-to-many** – When more than one instance of an entity is associated with a relationship, it is marked as '1: N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many

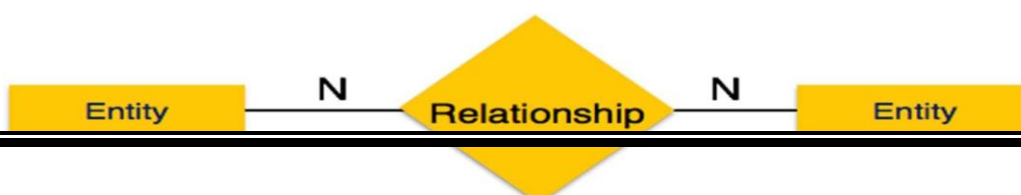


relationship.

- **Many-to-one** – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship



- **Many-to-many** – The following image reflects that more than one instance of an entity on the left and more than one

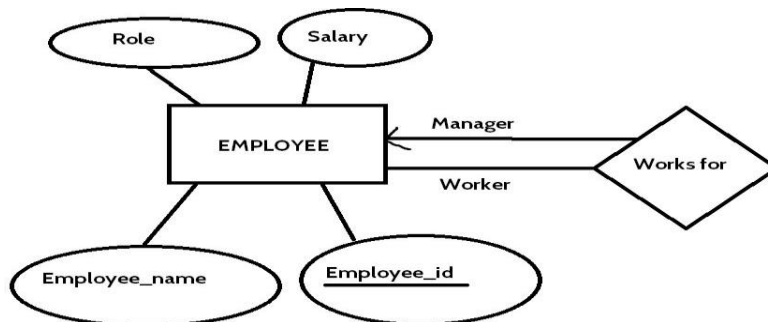


instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.

- **How to Draw ER Diagrams?**

Below points show how to go about creating an ER diagram.

1. Identify all the entities in the system. An entity should appear only once in a particular diagram. Create rectangles for all entities and name them properly.
2. Identify relationships between entities. Connect them using a line and add a diamond in the middle describing the relationship.
3. Add attributes for entities. Give meaningful attribute names so they can be understood easily



Advantages of ER Diagram:

- ❑ Conceptually it is very simple: ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.
- ❑ Better visual representation: ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.

- Effective communication tool: It is an effective communication tool for database designer.
- Highly integrated with relational model: ER model can be easily converted into relational model by simply converting ER model into tables.
- Easy conversion to any data model: ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

Disadvantages of ER Diagram:

- Limited constraints and specification.
- Loss of information content: Some information be lost or hidden in ER model.
- Limited relationship representation: ER model represents limited relationship as compared to another data models like relational model etc.
- No representation of data manipulation: It is difficult to show data manipulation in ER model.
- Popular for high level design: ER model is very popular for designing high level design

• What is Data Flow Diagram?

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

- **DFD Symbols:**

There are four basic symbols that are used to represent a data-flow diagram.

- **Process**

A process receives input data and produces output with a different content or form. Processes can be as simple as collecting input data and saving in the database, or it can be complex as producing a report containing monthly sales of all retail stores in the northwest region.

Every process has a name that identifies the function it performs. The name consists of a verb, followed by a singular noun.

Example:

- ☐ Apply Payment
- ☐ Calculate Commission
- ☐ Verify Order

- **Data Flow**

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or it can represent a set of data element (or a data structure).

Example:

- ☐ Customer info (Last Name, FirstName, SS#, Tel #, etc.)
- ☐ Order info (Ordered, Item#, Order Date, Customer, etc.).

- **Data Store**

A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data because one or more processes need to use the stored data in a later time.

- **External Entity**

It Is also known as actors, sources or sinks, and terminators, external entities produce and consume data that flows between the entity and

the system being diagrammed. These data flows are the inputs and outputs of the DFD.

- **How to draw a data flow diagram?**

Lucid chart makes it easy to create a customized data flow diagram starting with a simple template. Choose the symbols you need from our library—processes, data stores, data flow, and external entities—and drag-and-drop them into place. Since Lucid chart is an online tool, it facilitates collaboration and bypasses the hassles of desktop DFD software.

- **Levels in Data Flow Diagrams (DFD)**

In Software engineering DFD (data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in data flow diagram, which are: 0- level DFD, 1-level DFD, and 2-level DFD.

0- level DFD:

It is also known as context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as single bubble with input and output data indicated by incoming/outgoing arrows.

1- level DFD:

In 1-level DFD, context diagram is decomposed into multiple bubbles/processes.in this level we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.

2- level DFD:

2- level DFD goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system's functioning.

▪ Advantages of data flow diagram:

- ☐ A simple graphical technique which is easy to understand.
- ☐ It helps in defining the boundaries of the system.
- ☐ It is useful for communicating current system knowledge to the users.
- ☐ It is used as the part of system documentation file.
- ☐ It explains the logic behind the data flow within the system.

▪ Disadvantages of data flow diagram:

- ☐ Data flow diagram undergoes lot of alteration before going to users, so makes the process little slow.
- ☐ Physical consideration is left out. It makes the programmers little confusing towards the system.

• What is Data Dictionary?

A data dictionary contains metadata i.e. data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators.

The different types of data dictionary are:

- **Active Data Dictionary**

If the structure of the database or its specifications change at any point of time, it should be reflected in the data dictionary. This is the responsibility of the database management system in which the data dictionary resides.

- **Passive Data Dictionary**

This is not as useful or easy to handle as an active data dictionary. A passive data dictionary is maintained separately to the database whose contents are stored in the dictionary. That means that if the database is modified the database dictionary is not automatically updated as in the case of Active Data Dictionary.

- **Creating the Data Dictionary**

When you use the Database Configuration Assistant to create a database, Oracle automatically creates the data dictionary. Thereafter, whenever the database is in operation, Oracle updates the data dictionary in response to every DDL statement.

The data dictionary base tables are the first objects created in any Oracle database. They are created in the system tablespace and must remain there. The data dictionary base tables store information about all user-defined objects in the database.

- **Advantages of data Dictionary:**

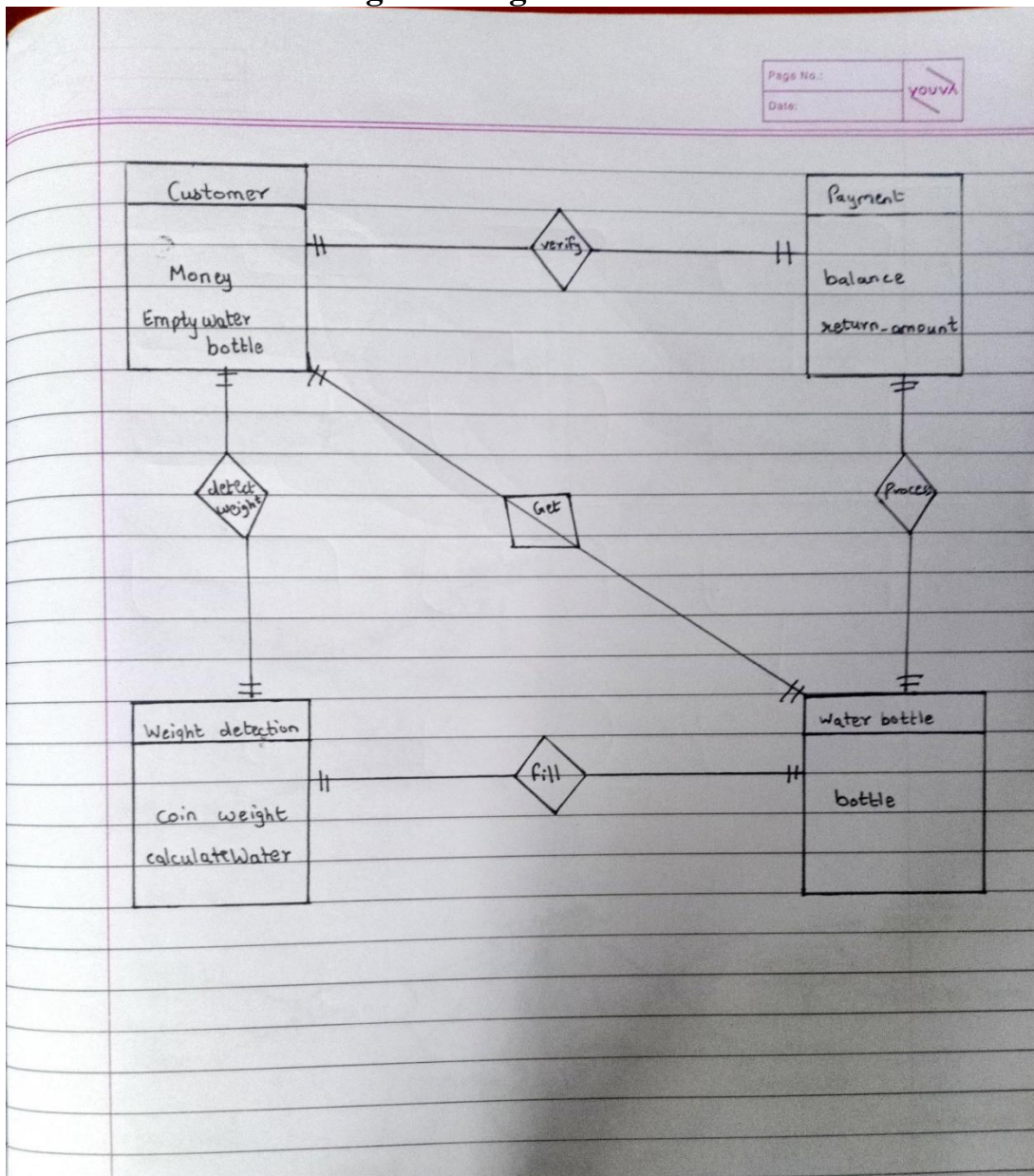
There are a number of advantages of using Data Dictionary in computer system analysis and design. The advantages are: consistency, clarity; reusability; completeness; increase in sharing and integration; and ease of use for the developer.

- Scenario:

1. Library Management System
2. Water Vending Machine

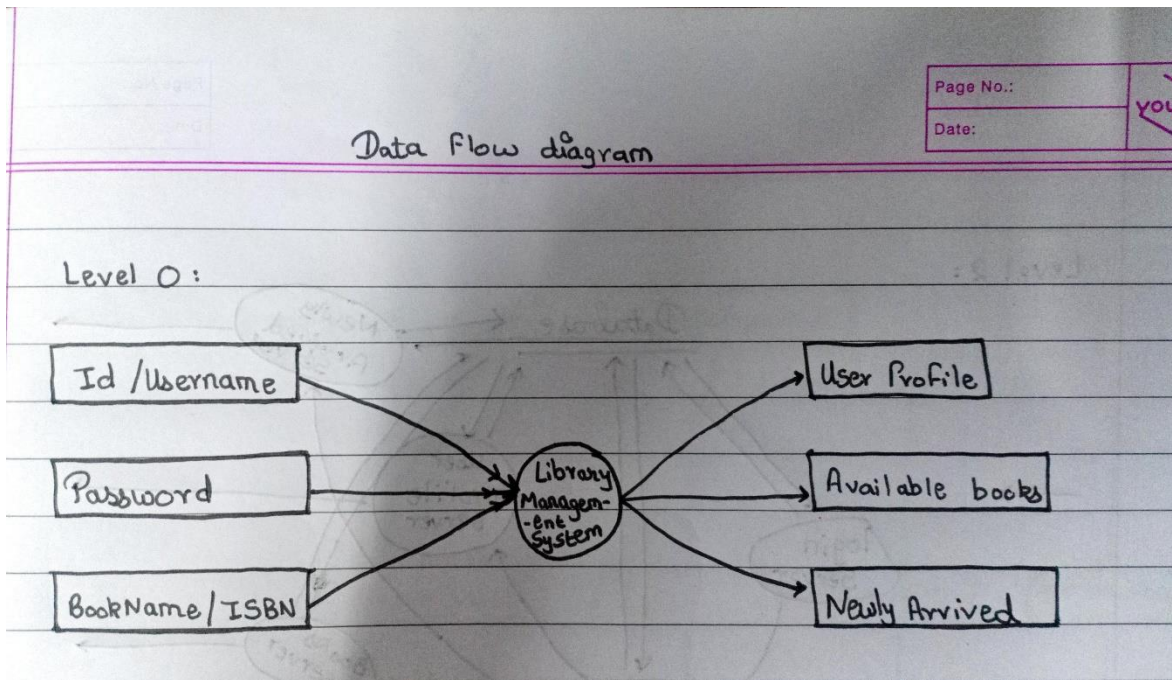
- Questions:

1. Draw the ER diagram for given scenario 2?

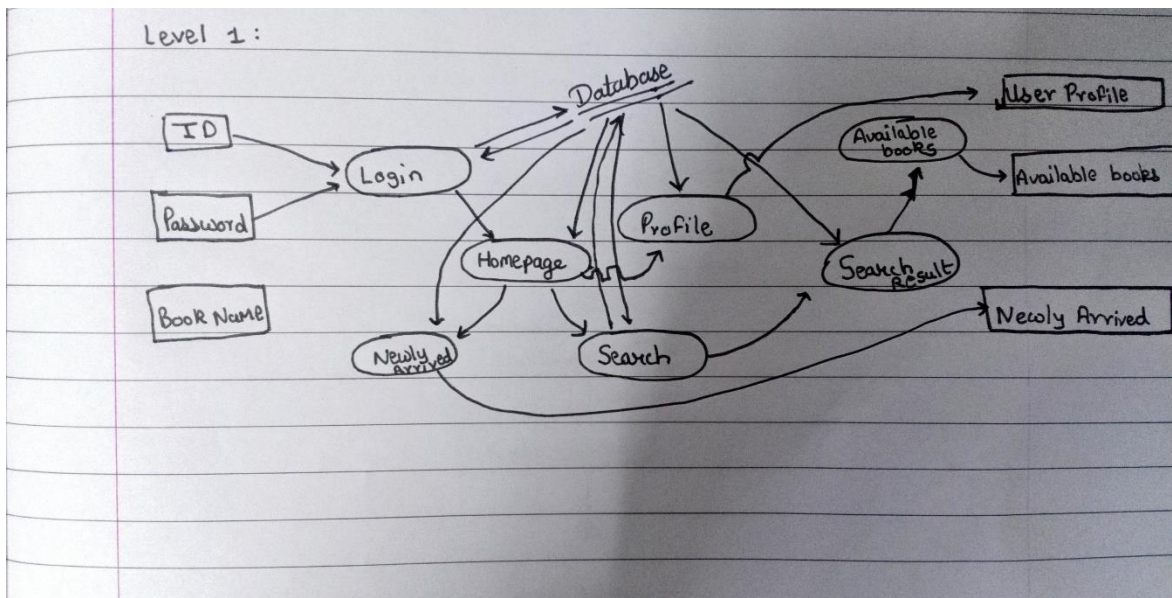


2. Draw DFD diagram for given scenario 1?

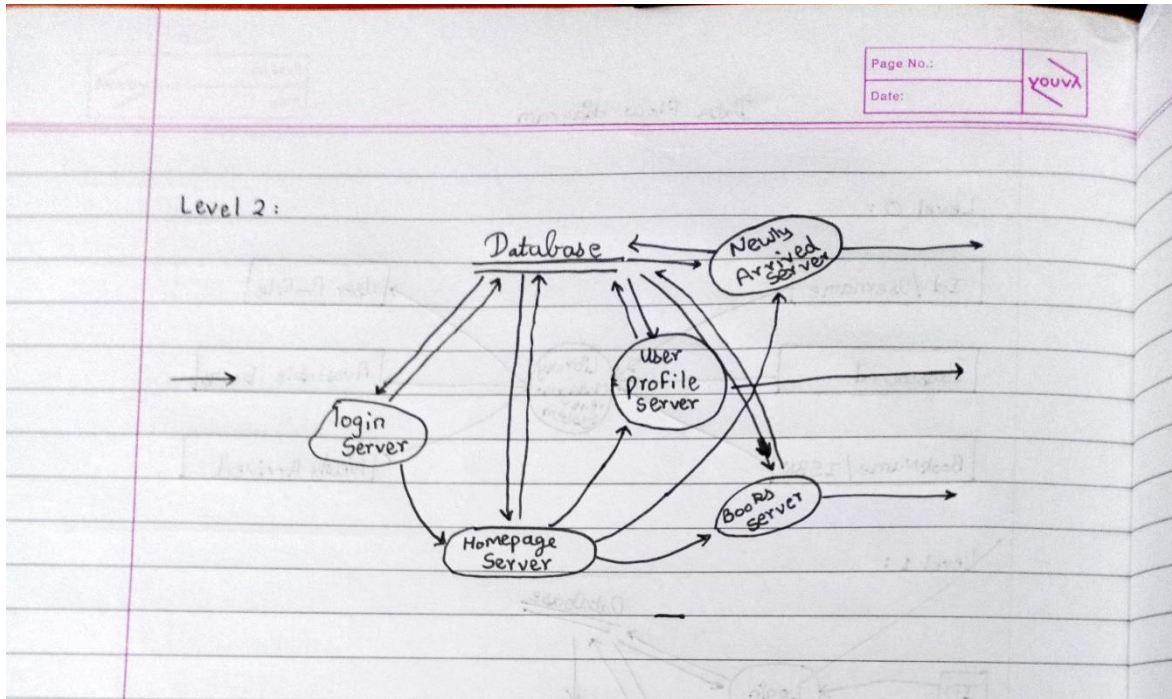
Level 0:



Level 1:



Level 1:



3. Create data dictionary for given scenario 1?

Data Dictionary

Name	Data type	Max Fields	Required
Username	String	32	Optional
ID	String	16	Optional
Password	String	32	Yes
Book Name	String	128	Yes
Books	Array[Strings]	128	No
NewlyArrived	Array[Strings]	128	Yes
Timestamp	TIMESTAMP	32	No
BookCover	BLOB	16MB	No
bookDescription	String	1024	No

4. What is the objective of maintaining data dictionary?

Data dictionaries are used to provide detailed information about the contents of a dataset or database, such as the names of measured variables, their data types or formats, and text descriptions.

This will provide a document that is consistently formatted and contains what is needed for others to understand your data

Conclusion:

Hence, by performing this practical, I learnt about the concepts of data dictionary, E-R diagram, and DFD, their uses and application. I also drew ER diagram, data flow diagram, and data dictionary for Library management system and water vending machine

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO. 8

Aim: Draw activity diagram for above system.

Theory:

- **Activity diagram:**

- Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.
- Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.
- The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc
- The basic purposes of activity diagrams are similar to other four diagrams. It captures the dynamic behaviour of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity Diagram

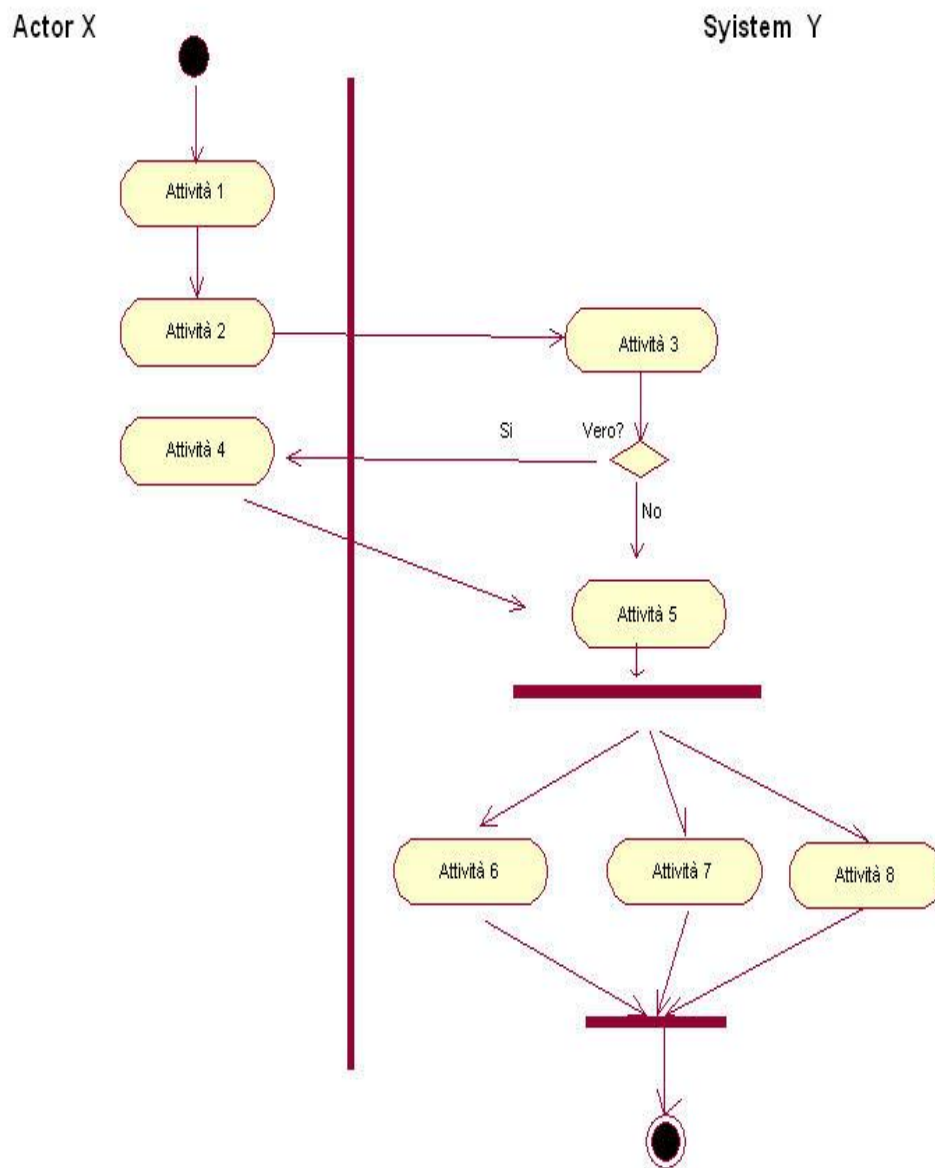


Fig: Activity Diagram

The purpose of an activity diagram can be described as –

- The activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

Activity diagram can be used for –

- Modelling work flow by using activities.
- Modelling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

- **Scenario: New online bank account opening and management system**

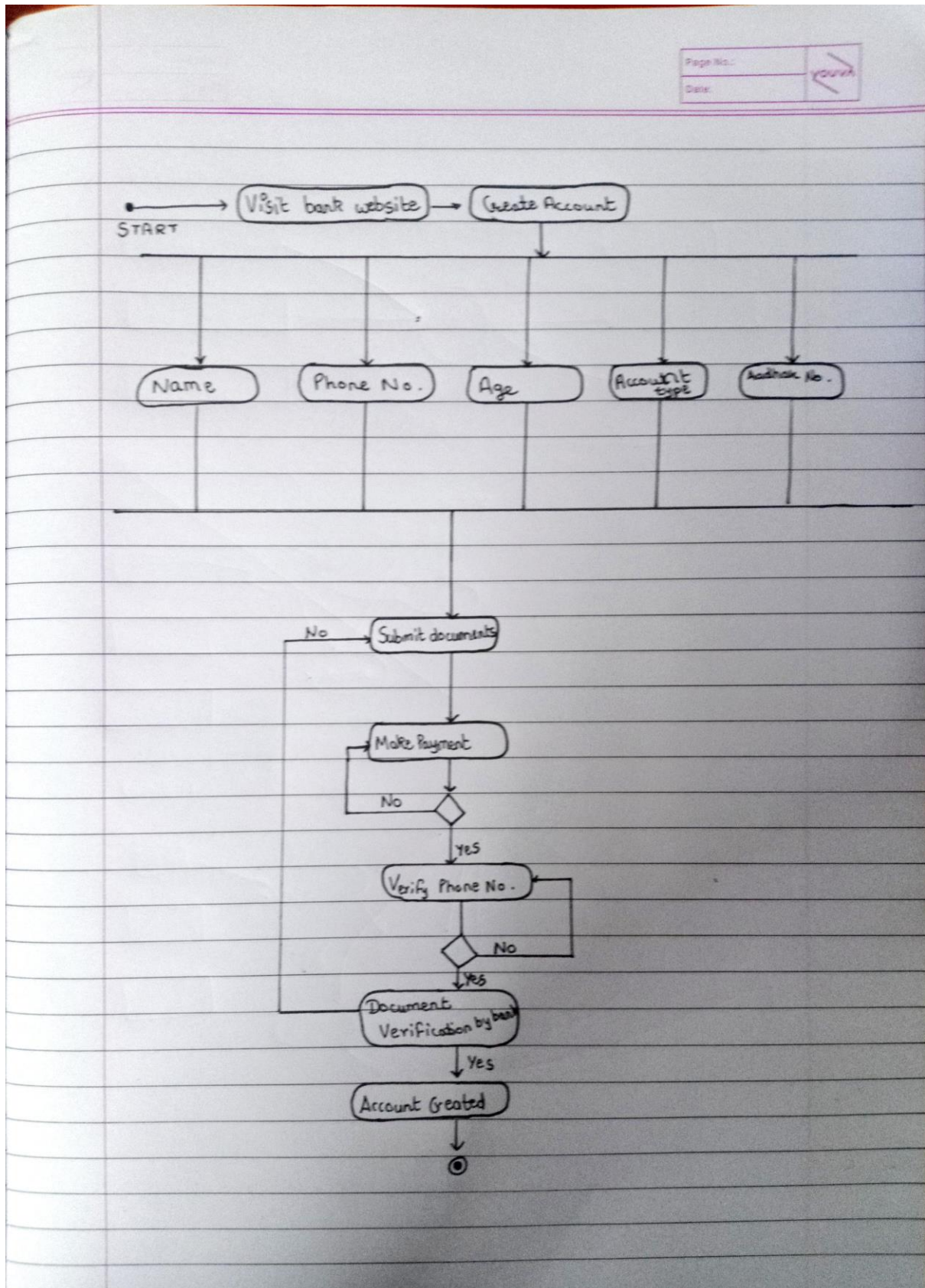
- **Questions:**

1) When to Use Activity Diagram?

Activity diagram can be used in the following scenarios:

- Modelling work flow by using activities.
- Modelling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

2) Draw activity diagram for given scenario.



3) Comparison between activity diagram and sequence diagram.

Sequence Diagram	Activity Diagram
The sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality. The sequence diagram shows the message flow from one object to another object.	The activity diagram represents the UML, which is used to model the workflow of a system. The Activity diagram shows the message flow from one activity to another.
Main focus is the interaction between different objects over a specific period of time.	Main focus is the flow of activities.
Helps to visualize the sequence of calls in a system to perform a specific functionality.	Helps to model the workflow a system.
Sequence diagram issued for the purpose of dynamic modelling. Sequence diagram is mainly used to represent the time order of a process.	Activity diagram is used for the purpose of functional modelling. Activity diagram is used to represent the execution of the process.

- **Conclusion:**

Hence, by performing this practical I got to know about the activity diagrams, its various components, and its use. I also drew an activity diagram for the scenario – ‘New online bank account opening and management system.’

10	20	10	10	Total

EXPERIMENT NO: 09

Aim: Identify the design principle that is being violated in relation to the given scenario. (Give any Scenario)

Theory:

- **Design Principles**

Software design is both a process and a model. The design process is a sequence of steps that enable the designer to describe all aspects of the software to be built. It is important to note, however, that the design process is not simply a cookbook. Creative skill, past experience, a sense of what makes “good” software, and an overall commitment to quality are critical success factors for a competent design.

There are 10 Design Principles: -

1. The design process should not suffer from “tunnel vision.” :-

A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job.

2. The design should be traceable to the analysis model: -

Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.

3. The design should not reinvent the wheel: -

Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited! Design time should be invested in representing truly new ideas and integrating those patterns that already exist.

4. The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world: -

That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.

5. The design should exhibit uniformity and integration: -

A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.

6. The design should be structured to accommodate change: -

The design concepts discussed in the next section enable a design to achieve this principle.

7. The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered: -

Well-designed software should never “bomb.” It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.

8. Design is not coding, coding is not design: -

Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.

9. The design should be assessed for quality as it is being created not after the fact: -

A variety of design concepts and design measures are available to assist the designer in assessing quality.

10. The design should be reviewed to minimize conceptual (semantic) errors: -

There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest for the trees. A design team should

ensure that major conceptual elements of the design (omissions, ambiguity, inconsistency) have been addressed before worrying about the syntax of the design model.

When these design principles are properly applied, the software engineer creates a design that exhibits both external and internal quality factors. External quality factors are those properties of the software that can be readily observed by users (e.g., speed, reliability, correctness, usability). Internal quality factors are of importance to software engineers. They lead to a high-quality design from the technical perspective. To achieve internal quality factors, the designer must understand basic design concepts.

- **Scenario: Home security system with camera and sensors**

- **Questions:**

1. Which Principles are being invalid with the given scenario.

In the given scenario, 'Home Security system with camera and sensors' the principles being violated are:

1. The Design should be structured to accommodate change.
2. The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.

2. How it is violated with the given scenario.

The principle – 'The Design should be structured to accommodate change' is violated in the scenario. As the scenario states – 'Home security system with camera and sensors', it has been specified that the security system must have camera and sensors only nothing less, nothing more. This scenario is too specific and thus it negates the ability to accommodate change, thus violating the principle.

The principle – 'The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered' is violated in the Scenario, as the scenario of 'home security system with camera and sensors' uses hardware equipment such as camera and sensors. These equipments can wear out and as a result the system might not work as intended. These equipments might get affected by the climate, voltage fluctuations, etc. Due to this the system might get corrupted and even has a risk of failing. But the principle states that 'the software must not be a bomb' that is it should be designed to accommodate unusual circumstances and must terminate the processing in refined manner if a problem arises, which is not applicable for the provided scenario. The failures in camera and sensors would not be able to cope to these unusual circumstances and may act like a bomb and just get corrupted or stop working, thus violating the principle.

3. Which Principles are valid with the given scenario.

In the given Scenario – ‘Home security system with camera and systems’ the principles which are valid to the scenario are:

1. The design should be traceable to the analysis model.
2. The design should exhibit uniformity and integration.
3. The design process should not suffer from “tunnel vision”.
4. Design is not coding, and coding is not design.

The principle – ‘The design should be traceable to the analysis model’ is valid as the design statement is concrete and clear which makes tracking the other required elements easier.

The principle - ‘The design should exhibit uniformity and integration.’ Is valid as the design statement clearly states that it should be a security system for the home consisting of camera and sensors, thus it makes sure that the design stays uniform.

The principle – ‘The design process should not suffer from “Tunnel vision”.’ Here the approach has been confirmed which can be done by judging the requirements of the problem, and the resources available to do the job.

The principle – ‘Design is not coding, and coding is not design’, is valid in this scenario as the design clearly states only the design components of the system and not the code for the components.

● Conclusion:

Hence, by performing this practical I learnt about the about the concept of Design Principles, the 10 Design Principles and also got to understand how to apply them in a scenario by checking if they are valid or invalid with the principles or not. I also identified the principles which were being violated in relation to the given scenario – ‘Home security system with camera and sensors.’

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO: 10

Aim: Using all the concepts of software engineering make a mini-project on e-commerce store – Book Store.

Theory:

- **Introduction:**

The Book store application is an application software made to help students, teachers, and a majority of book readers for buying books and magazines and get them delivered to you through our android app. This application's major goal is to provide students, teachers and book readers books at their homes with zero-contact delivery due to the Covid-19 Pandemic situation.

- **Software Development Processes:**

1. **Communication:**

The project was given by Prof. L L Bhadekar sir to the team.

The topic of the project was - 'Book Store app'.

The basic features for the app were to have option to buy books using credit card, get books suggestion and to review the books by giving a rating (1-5 stars).

2. **Planning:**

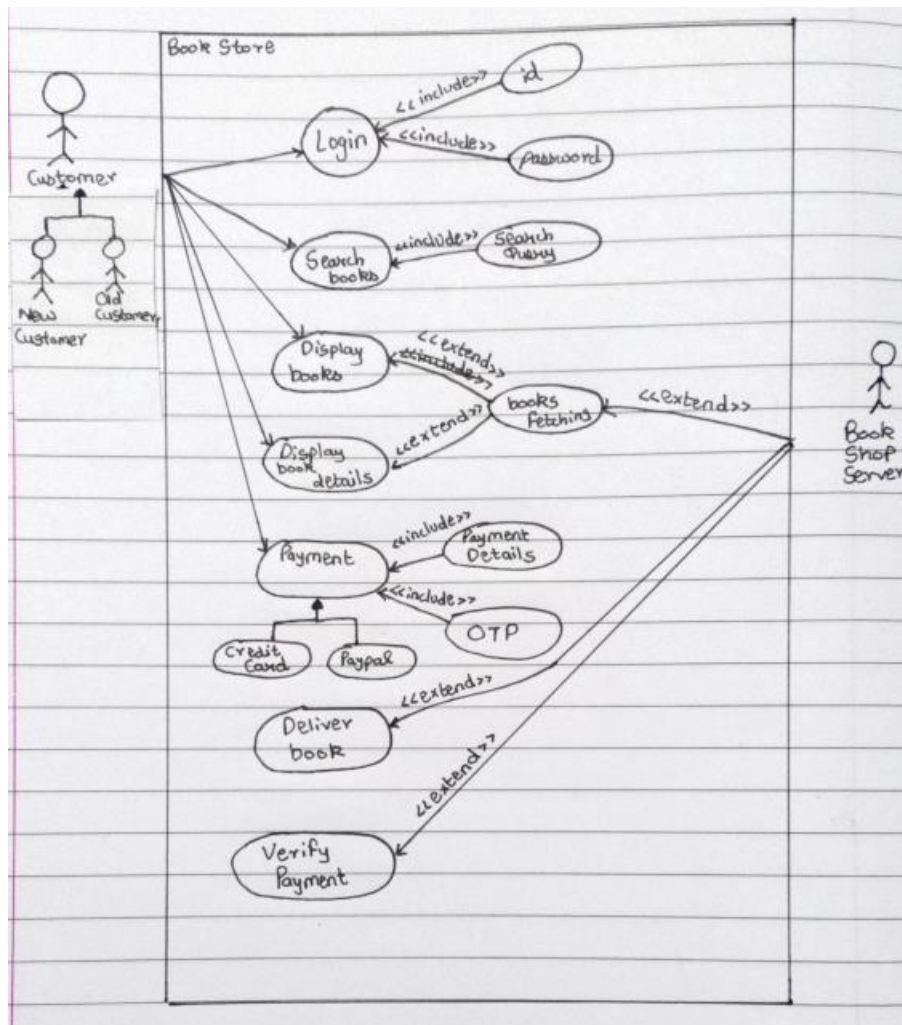
We discussed the plans to be made for the features of the app.

The software and tools to be used in the process of development of the software are decided and a plan is made for the software.

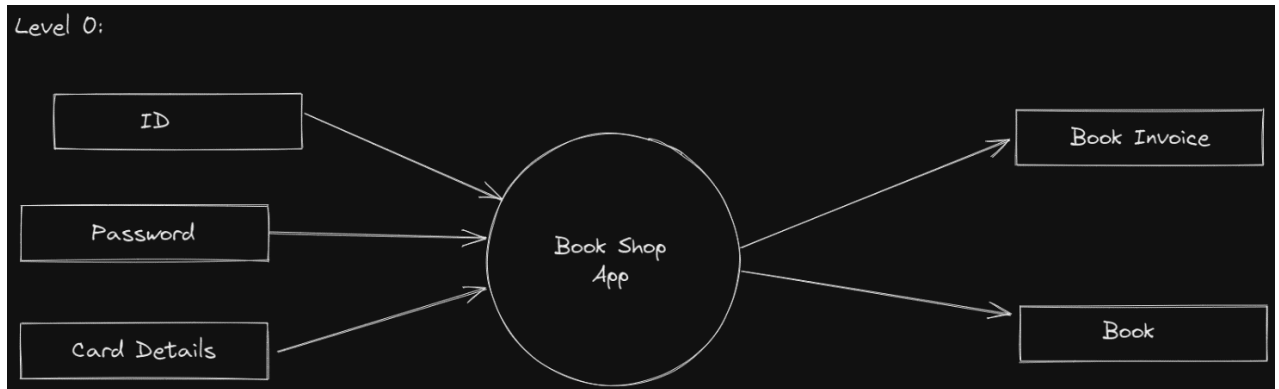
The working for each member is decided and distributed.

3. Modelling:

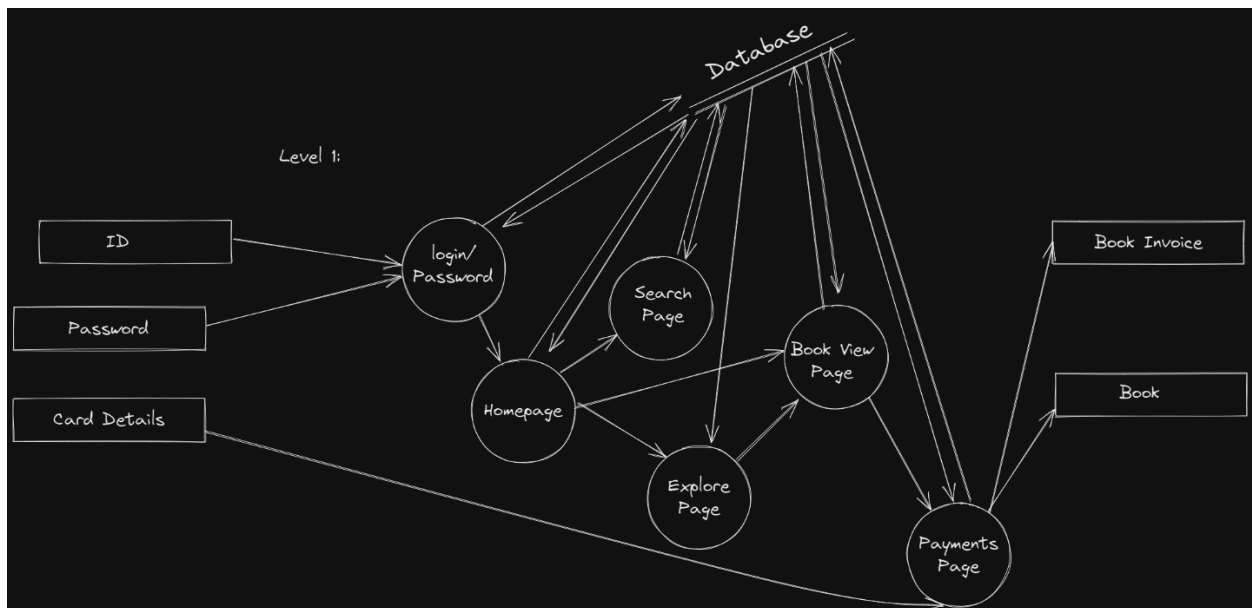
Use Case Diagram:



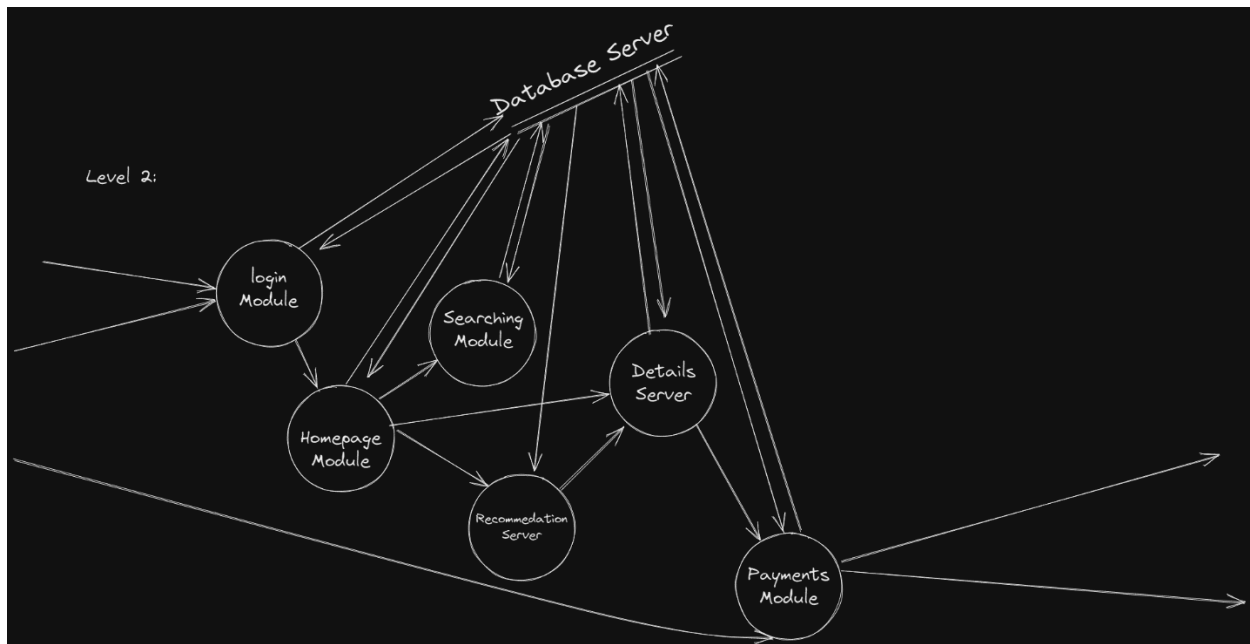
Data Flow Diagram:



Level 0

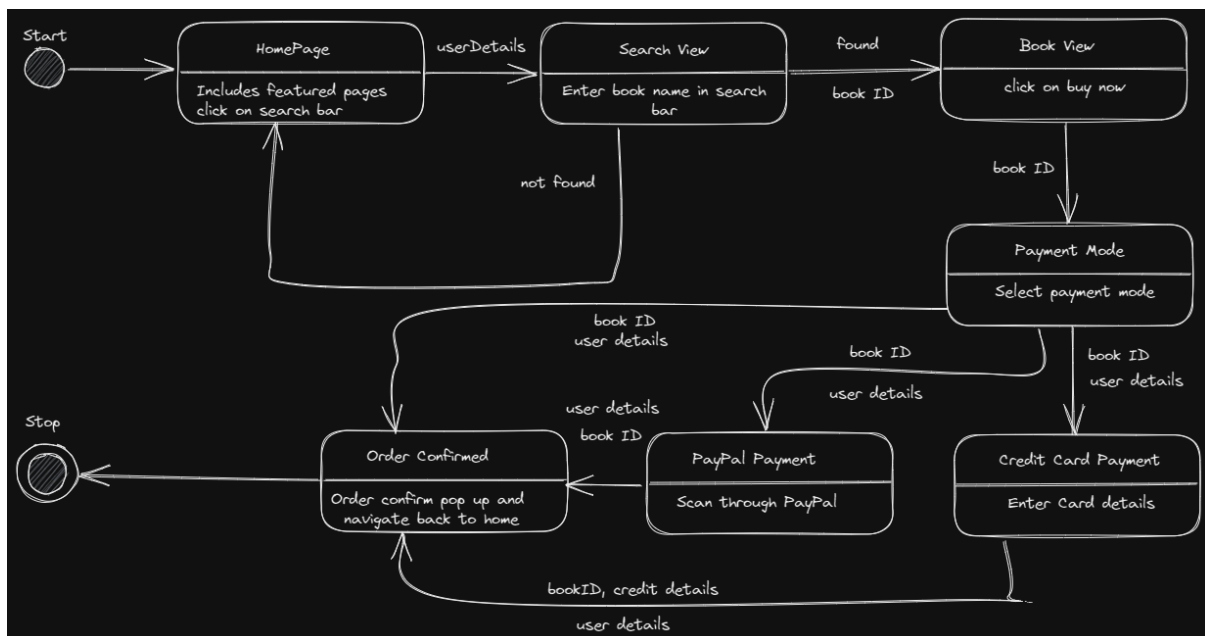


Level 1



Level 2

State transition Diagram:



Critical Path Diagram:

Task	Preced. Task	Description	Time (hrs).
A	none	Communication	3
B	A	Decision for the topic	1
C	A	Choosing the platform	1
D	B	making UI on figma	6
E	D	Start disklenting the models to team members.	3
F	C, D	start coding	9, 9
G	E	merging the coding through github.	3
H	E	Documentation	1

Task	Preced. Task	Description	Time (hrs).
A	none	Communication	3
B	A	Decision for the topic	1
C	A	Choosing the platform	1
D	B	making UI on figma	6
E	D	Start diskingenting the models to team members.	3
F	C, D	start coding	9, 9
G	E	merging the coding through github.	3
H	E	Documentation	1

4. Construction:

For making the application, dart programming language and Flutter framework was used.

We used android studio and Visual Studio Code for coding the project.

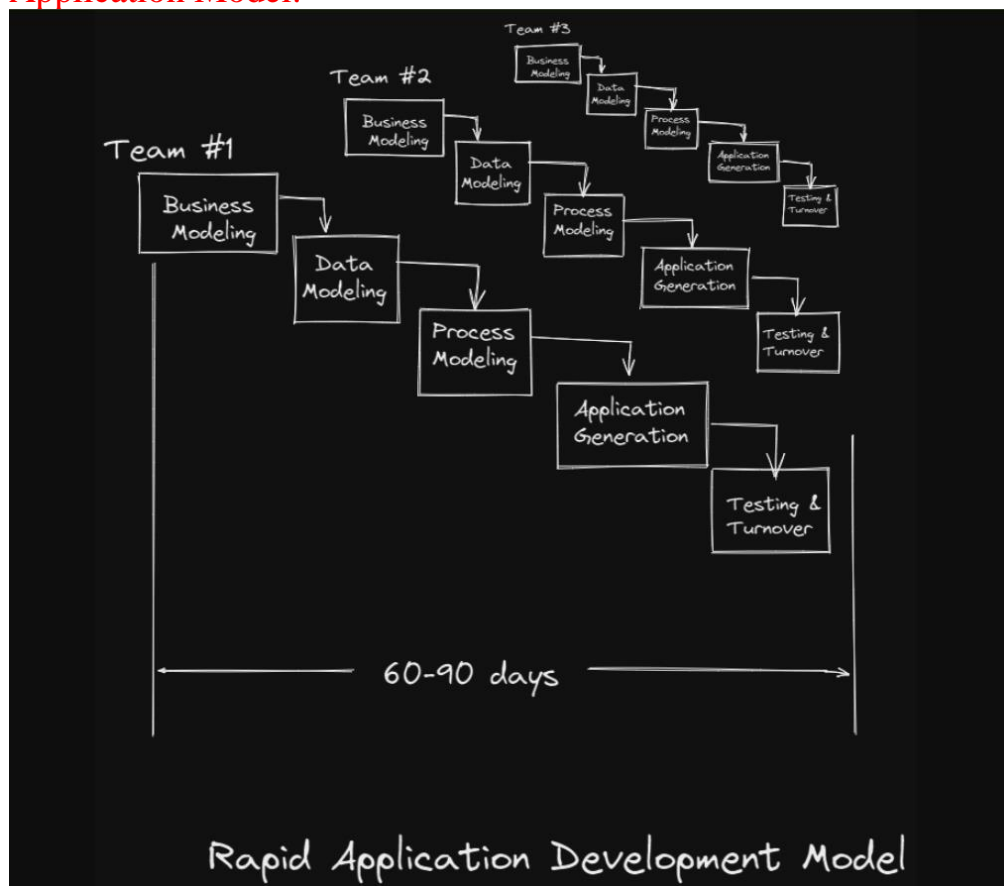
The testing was done by the team members itself for finding the errors and bugs.

5. Deployment:

The application is to be submitted on 18th December 2021 on the Schoology platform in pdf format.

Process Models

Rapid Application Model:



Our project has to be submitted in a short amount of time, so for this purpose we decided to use Rapid Application Model which focuses on dividing the functions into modules and carrying out the processes on the modules and then integrating the modules together.

The rapid application model is a high-speed adaptation of the linear model. In this model the incremental software

development process is emphasising on extremely short development cycle. In this model rapid development is achieved in very short-interval of time by using component based construction.

The RAD is further divided into 5 steps:

1. Business Modelling:

The information flow among business functions is modelled in a way that answers the following questions:

- What information drives the information process?
- What information is generated?
- Who generates it?
- Where does it go?
- Who processes it?

2. Data Modelling:

Set of data objects that are needed to support the business. The attributes of each object are identified and the relationships between these objects are defined.

3. Process Modelling:

The processing descriptions are created for adding, modifying, deleting or retrieving a data object.

4. Application generation:

Application development uses the fourth generation, techniques of RAD process. IT works to reuse existing program components. Automated tools are used to facilitate construction of the software.

5. Testing and turnover:

Since, RAD process emphasises on reuse, many program components have already been tested. This reduces overall testing time.

People:

The team leader Pratyay Dhond managed and defined the tasks for all the members of the team.

The modelling and constructing the various models and all other documents was done by the whole team, i.e. Pratyay Dhond, Priyanshu Lapkale, Aditi Khare, Harsh Naidu, Manthan Ghonge, Varad Nimbarte.

The designing of the UI was done by Priyanshu Lapkale.

The coding work for the application was done by Pratyay Dhond, Aditi Khare, Priyanshu Lapkale and Varad Nimbarte.

The testing of the software was done by Harsh Naidu and Manthan Ghonge.

The customer for our product were our faculty, i.e. Prof. L L Bhadekar sir.

The end users of the software product will be the students, teachers and book readers who want to get their books delivered to them.

Product:

Software Scope:

Book store app is an application software with the motive of providing access to books by getting them delivered to the user's homes once they have purchased the books from us.

Functions:

- Through our search for book functionality, the users can search and check the availability, summary of the book and price of the book easily.
- Through the rating functionality the customers can rate the book from a range of 1-5 stars.
- Through the payment functionality the users can pay for the books using credit cards.

Process:

- Rapid application model was used for the process.
- For process decomposition, the customer had given us very few details for the software, so we had to break down the components and needed to clarify certain points such as the filter by option and other details.

Project:

- Good decisions were made by defining proper jobs to the team members.
- Our major focus was on keeping the application minimalistic, easy to understand, and simple to use.
- We used Git as our version control system, and github for managing the software remotely.

W5HH Principle:

i. Why is the system being developed?

This system is being developed in order to make the books available online and for making the purchase procedure simpler. The system will also be providing samples of the book in epub format for the end users to view the book's sample. This app will also have details of the customer so that the delivery can be done to their house/address.

ii. What will be done, by when?

The time for each and every task was defined in the Gantt chart, for the key tasks: communication, planning, modeling(Use case diagram,

data flow diagram, state transition diagram, pert chart), construction, deployment. The time to be taken was defined as follows:

Tasks	Duration (hours)
Communication	1.5
Planning	2
Modelling	9
Construction	50
Deployment	1

- iii. Who is responsible for a function?
 - The tasks for the project were divided into team members according to their capabilities.
 - The team leader, Pratyay Dhond defined the tasks for the team members. The coding work was done by Pratyay Dhond, Priyanshu Lapkale and Aditi Khare.
 - The documentation work, i.e. the SRS was made by Varad Nimbarte and Harsh Naidu.
 - The testing of the software was done by Manthan Ghonge and Harsh Naidu.
- iv. Where are they organizationally located?
 - The client/customer for our project is Prof. L L Bhadekar sir.
- v. How will the job be done technically and managerially?
 - On the technical level, software development tools, such as android studio and Visual Studio code IDEs were used for the development of the software.
 - For the programming Flutter framework and dart language was used
 - Managerially the team was managed by the team leader, i.e. Pratyay Dhond and tasks were distributed by him to all the team members
- vi. How much of each resource is needed?
 - Human Resources i.e. the team members and customer will be needed for making the software and for making successful communication and deployment.
 - Hardware resources include an android device, with minimum 512MB RAM and an android version of Android 8.0 or higher.

Software Requirement Specification:

Title: Online Book Store

Table of Contents

1)Introduction

1.1 Purpose

1.2 Scope

1.3 Definition Acronyms and Abbreviations

1.4 Reference

1.5 Overview

2)Overall Description

2.1 Product Perspective

2.2 Product Functions

2.3 User Characteristics

2.4 Constraints

2.5 Assumptions and Dependencies

3)Specific Requirement

3.1 External Interfaces

3.2 Functions

3.3 Performance Requirements

4) Appendices

5) Index

1) Introduction

1.1 Purpose

This Software Requirement Specification is meant for an Online Bookstore App. The Online Bookstore App is meant as a way for customers to browse books on the App and buy them from home without the need to travel to a book shop. Defining the functions and specifications of the Online Bookstore App is the primary purpose of this SRS. The SRS illustrates in clear terms, the system's primary uses.

1.2 Scope

- **Search the book**
- **Check the availability of the book**
- **Addition of book to Wishlist**
- **Removal of book from Wishlist**
- **Updating of book price and availability**
- **Book order management**
- **Ordering the book by customer**
- **Getting more information about the author**
- **The customer can register himself/herself on the app**
- **Taking the feedback of the customer**

1. 3)Definitions and Abbreviations

Definitions:

Term	Description
Book	A single book belonging to an author.
Wishlist	Consists of items the user wishes to buy at a point of time
Cart	Consists of items the user wishes to buy
Customer	Store patron that orders/pays for ordering the book
Server	Operators of the Database that stores and manages the Book list
Genre	Filter to sort the booklist according to Genre of the book.
Author	Filter to sort the booklist according to Author of the book.
Year	Filter to sort the booklist according to year of book released.

Acronyms:

Short form	Description
SRS	Software Requirement Specification
BD	Books Data
DBMS	Database Management System
LAN	Local Area Network
IP	Internet Protocol
TCP	Transmission Control Protocol
RAM	Random Access Management
IEEE 802.11	Wireless Local Area Network Standard
WPA2-PSK	Wi-Fi Protected Access 2 with Pre-Shared Key

1.4 References:

- **Google play books:**

https://play.google.com/store/books?utm_source=apac_med&utm_medium=hasem&utm_content=Oct0121&utm_campaign=Evergreen&pcampaignid=MKT-EDRapac-in-1003227-med-hasem-bk-Evergreen-Oct0121-Text_Search_BKWSBKWS%7cONSEM_kwid_43700065205026403_creativeid_535350509885_device_c&gclid=Cj0KCQiAkZKNBhDiARIsAPsk0Wg3LnsGk98J4WHVn1HFq2jo2RyVg11N5OGDFyR_8L2H4MY8U5tfdwAIZNEALw_wcB&gclsrc=aw.ds

- **Good reads:**

<https://www.goodreads.com/>

1.5 Overview:

By using this application, the customer of the app will be able to purchase books online via paying either by card or cash. The customer will be able to check the book details and give the rating in 1-5 star range. The customer can search for the books available on the application. The books can be updated according to the availability of a particular book. The application will generate a receipt on the server end of every book order. The application will reduce the redundancy in the data. By using this application, the book store can increase their efficiency by many folds. The customer can register himself/herself on the application.

2) Overall Description

The following section presents an overall description of the subject Book Application. In particular, the product has been put into perspective through a detailed assessment of the system, user, hardware, software, and communication interfaces, memory considerations, operational modes, and app adaptation requirements. Further, the characteristics of the system's end-users are discussed along with the identified system constraints and assumptions.

2.1 Product Perspective

Business case:

- The books will change the data of availability on the server side
- The books purchased will generate a report and get all the receipts from the server-side

Customer case:

- The user will open the applications
- The user will register himself/herself on the application
- the registering user has to enter their details on the registration of the app
- The user will search the book which they need
- Then the user will fill the delivery form and will click on order now.

- The user can fill the feedback form for the book shop

Business Interface:

- Server Admin Login
- Generate report
- Get all the receipts
- Real-time orders

Customer Interface:

- User Register
- Feedback module
- More information module
- Books Module

External Interfaces required for running Book Application:

- A android system that can run a software
- Minimum 512MB of ram
- Internet connection with minimum 2mbps

2.2 Product Functions

Functions of Online Book Store are as follow:

User Register: By User register, the user can register themselves on the application.

- **Searching book:** The user can search the various book on the application.
- **Ordering an item:** The user can order the book by filling the delivery form.
- **Giving Feedback:** By this users can give feedback on the book items, on delivery services.

2.3 User Characteristics

Customer:

- The user should have a base knowledge of how to use an android application
- The user should know how to connect to the internet

2.4 Constraints

- This app will not run under the android version 7.0 or below platforms of android.
- The user using app should have a minimum memory of 512MB in the device.
- The user using login should have a minimum network speed of 2MBPS

2.5 Assumptions and Dependencies

Assumptions

- Users who use the app should use a valid username and password for registration.
- The application should have access to the internet.
- The user has a compatible android device with android version 8.0 or greater.

Dependencies

- Hardware and software which we used will be run properly.
- End-users should have a proper understanding of the application.

3) Specific Requirement

3.1 Externals Interfaces

User Interface

- Android
- iOS

Hardware Interfaces • Server-side

- Operating system: Android 8.0 or above
- RAM: 512MB

SOFTWARE INTERFACES

- Database
 - MYSQL

• *Programming Languages*

- Flutter
- Dart

• *COMMUNICATION INTERFACES*

- Dial-Up Modem
- Broadband Internet
- Cellular Data

3.2 Functions

Functions accessible by Customers

- **Create Account/Register**
Customers can make their account and access various features like booking, etc Users have to mention details like Username, password, first name, last name, address, phone no, and email id
- **Customer Login**
Let the user access their account and see the details stored in their account. To let the user login the user has to fill in a valid email address and correct password and the user should have created an account.
- **View Book Details**
By using this the user checks the details of the available books.
- **Search book Items**
By using this function users can search for books and in searching, users can apply filters for increasing their convenience and accuracy. Users need to enter the book genre to search for the required book.
- **Order**
Customers can order the selected book items as per his/her choice .The order will be delivered by the delivery person to the customers location.

Functions accessible for Book Shop

- **Login/Register**
- **Homepage Module**
- **Feedback Module**
- **More-Info Module**

- **Pages Module**
- **Delivery Module**
- **Order Module**

3.3 Performances Requirements

- The application can accommodate 50 users during the peak usage time window of 8:00 am to 10:00 am local time, with an estimated average session duration of 45 minutes.
- All Page states generated by the application shall be fully loaded in no more than 10 seconds over a 40KBps modem connection.
- Responses to filters shall take no longer than 10 seconds to load onto the screen after the user submits the filters.
- The system shall display Order confirmation messages to users within 4 seconds after the user confirms the order to the application.

4) Appendices

- Flutter : A software development framework for cross platform application development
- Dart : A programming language for developing flutter apps.
- MYSQL : a Database programming language.
- RAM : Random Access Memory

5) Index

1. Functions	2.2,3.2
2. Overview	1.5,2.4
3. Performance Requirements	3.1,3.3

- **Conclusion:**

Hence, in this experiment, we have made an application – ‘Online book store app’ using all the concepts of the software engineering. We have applied various concepts such as the process model, W5HH principles, the 4P’s of project management, prepared a SRS document, managed risks, and made the product along with all the required diagrams of the product.

(10)	(20)	(10)	(10)	TOTAL