THE NORTHCAP UNIVERSITY

POWERED BY Arizona State University

Software Engineering and Project Management

Lab Manual

Department of Computer Science and Engineering
The NorthCap University, Gurugram

Software Engineering and Project Management

CSL229

Dr. Sujata

Department of Computer Science and Engineering

NorthCap University, Gurugram- 122001, India Session

2021-22

Published by:

School of Engineering and Technology Department of

Computer Science & Engineering The NorthCap

University Gurugram

• **Laboratory Manual is for Internal Circulation only**

Copying or facilitating copying of lab work comes under cheating and is considered as use of unfair means. Students indulging in copying or facilitating copying shall be awarded zero marks for that particular experiment. Frequent cases of copying may lead to disciplinary

action. Attendance in lab classes is mandatory.

Labs are open up to 7 PM upon request. Students are encouraged to make full use of labs beyond normal lab hours.

PREFACE

Software Engineering and Project Management Lab Manual is designed to meet the course and program requirements of NCU curriculum for B.Tech III year students of CSE branch. The concept of the lab work is to give brief practical experience for basic lab skills to students. It provides the space and scope for self-study so that students can come up with new and creative ideas.

The Lab manual is written on the basis of “teach yourself pattern” and expects that students who come with proper preparation should be able to perform the experiments without any difficulty. Brief introduction to each experiment with information about self- study material is provided. The laboratory exercises include understanding Unified Modeling Language and its implementation using open source tool. Building basic workflows for all the system’s views like Requirements’ view, structural and behavioral view and finally deployment view are discussed as part of the curriculum. Finally, the students would require doing guided and unguided projects. Students are expected to come thoroughly prepared for the lab. General disciplines, safety guidelines and report writing are also discussed.

The lab manual is a part of curriculum for TheNorthCap University, Gurugram. Teacher’s copy of the experimental results and answers for the questions are available as sample guidelines.

We hope that the lab manual would be useful to students of CSE, ECE branches and the author requests the readers to kindly forward their suggestions / constructive criticism for further improvement of the work book.

Author expresses deep gratitude to Members, Governing Body-NCU for encouragement and motivation.

Authors
The NorthCap University
Gurugram, India

CONTENTS

1. INTRODUCTION

That 'learning is a continuous process' cannot be over emphasized. The theoretical knowledge gained during lecture sessions need to be strengthened through practical experimentation. Thus practical makes an integral part of a learning process.

The purpose of conducting experiments can be stated as follows:

1. To familiarize the students with the basic concepts of Unified Modelling Language. The lab sessions will be based on exploring the concepts discussed in class.
2. Learn the object-oriented analysis phase by understanding the methods of class elicitation and finding the classes in an object- oriented Systems
3. Design the Interaction diagrams, sequence and collaboration diagrams with the help of software engineering tool.
4. Learn to design the test cases.
5. Hands-on experience

2. LAB REQUIREMENTS

S.No.	Requirements	Details
1	**Software Requirements**	**Open Source Tools** https://app.diagrams.net/
2	**Operating System**	**Modern Operating System**
3	**Hardware Requirements**	• **Modern Operating System:** • **x86 64-bit CPU (Intel / AMD architecture)** • **4 GB RAM.** • **5 GB free disk space**
4	**Required Bandwidth**	NA

3. GENERAL INSTRUCTIONS

a. General discipline in the lab

- Students must turn up in time and contact concerned faculty for the experiment they are supposed to perform.
- Students will not be allowed to enter late in the lab.
- Students will not leave the class till the period is over.
- Students should come prepared for their experiment.
- Experimental results should be entered in the lab report format and certified/signed by concerned faculty/ lab Instructor.
- Students must get the connection of the hardware setup verified before switching on the power supply.
- Students should maintain silence while performing the experiments. If any necessity arises for discussion amongst them, they should discuss with a very low pitch without disturbing the adjacent groups.
- Violating the above code of conduct may attract disciplinary action.
- Damaging lab equipment or removing any component from the lab may invite penalties and strict disciplinary action.

b. Attendance

- Attendance in the lab class is compulsory.
- Students should not attend a different lab group/section other than the one assigned at the beginning of the session.
- On account of illness or some family problems, if a student misses his/her lab classes, he/she may be assigned a different group to make up the losses in consultation with the concerned faculty / lab instructor. Or he/she may work in the lab during spare/extra hours to complete the experiment. No attendance will be granted for such case**.**

c. Preparation and Performance

- Students should come to the lab thoroughly prepared on the experiments they are assigned to perform on that day. Brief introduction to each experiment with information about self study reference is provided on LMS.
- Students must bring the lab report during each practical class with written records of the last experiments performed complete in all respect.
- Each student is required to write a complete report of the experiment he has

performed and bring to lab class for evaluation in the next working lab. Sufficient space in work book is provided for independent writing of theory, observation, calculation and conclusion.
- Students should follow the Zero tolerance policy for copying / plagiarism. Zero marks will be awarded if found copied. If caught further, it will lead to disciplinary action.
- Refer **Annexure 1** for Lab Report Format

4. LIST OF EXPERIMENTS

Sr. No.	Title of the Experiment	Software based	CO covered	Time Required
1	Understanding basics of Software Engineering and Project Management by case studies[(a) wants to build a house in his plot, (b) to introduce a new dish at a restaurant, (c) to make a movie, (d) open a stationery shop, (e) to create a show so as to take interviews of famous personalities**] and design Feasibility Document**	Manually	CO1	2 hrs
2	Identify suitable Process model/s for 21st Century Health Care (Waterfall and iterative, Evolutionary, Spiral, RAD, V and V model) or your own project	Manually	CO1	2 hrs
3	Design the software requirements specification (SRS) by using a CASE tool.	Manually	CO2	2 hrs
4	Understanding the concept of UML and design the use case diagram for specific project statement	CASE Tool	CO3	2 hrs

5	Learn the object-oriented analysis phase by understanding the methods of class elicitation and finding the classes in an object- oriented system.	CASE Tool	CO3	2 hrs
6	Design of activity diagram by using a CASE tool.	CASE Tool	CO3	2 hrs
7	Design the Interaction diagrams, sequence and collaboration diagrams with the help of software engineering tool.	CASE Tool	CO3	2hrs
8.	Design Test Cases and Test Scenario using Excel.	CASE Tool	CO4	2hrs
9.	Learn to find the critical path using cyclomatic complexity	Manually	CO4	2hrs
10	To design the Gantt Chart using open source tool.	CASE Tool	CO5	2 hrs
11	Total hours including evaluation			30 hrs
12	Value Added Experiments • Design user stories considering the epics of your system • Design Data Flow diagram for your system.			

5. LIST OF PROJECTS

Sr No.	Project Title	Mapped CO
1.	Consider the development of E commerce website and design all the software artifacts for the same.	CO1, CO2, CO3, CO4, CO5
2.	Identify all the requirements of ingenious hospital management system and design the complete SRS for the same.	CO1, CO2
3.	Consider the scenario of how to increase the number of active users and revenue for Google pay and draw use case diagram and class diagram for the same.	CO1, CO2, CO3
4.	Creating and Managing the various projects on open source project management tool.	CO5

6. RUBRICS

Marks Distribution	
Continuous Evaluation(50 Marks)	**Project Evaluations/End term Viva (20 Marks)**
Each experiment shall be evaluated for 10 marks and at the end of the semester proportional marks shall be awarded out of total 50.	**Both the projects shall be evaluated and at the end of the semester viva will be conducted related to the projects as well as concepts learned in labs and this component carries 20 marks.**
Following is the breakup of 10 marks for each **4 Marks: Observation & conduct of experiment. Teachers may ask questions about experiments.** **3 Marks: For report writing** **3 Marks: For the 15 minutes quiz/viva to be conducted in every lab.**	

Annexure 1

<Software Engineering and Project Management > (CSL229)

Lab Practical Report

Faculty Name: **Dr. Sujata**

Student Name: **Piyush Gambhir**
Roll No.: **21CSU349**
Semester: **VI**
Group: AIML **(A3)**

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2021-22

Project Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - End Semester Project at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)

EXPERIMENT NO. 1

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:

Objective:

- Introduce the lab environment and tools used in the software engineering lab.
- Discuss the Project & learn how to write project definition.

Outcome:

Students will understand the importance of SEPM and about various software artifacts
Students will learn to perform feasibility analysis by designing Feasibility Study Document

Problem Statement:

- Introduction to the lab plan and objectives.
- Software Engineering and Project definition and, artifacts discussion.

Background Study:

The software engineer is a key person analyzing the business, identifying opportunities for improvement, and designing information systems to implement these ideas. It is important to understand and develop through practice the skills needed to successfully design and implement new software systems.

1. In this lab we will practice the software development life cycle (project management, requirements engineering, systems modeling, software design, prototyping, and testing) using CASE tools within a teamwork environment.
2. UML notation is covered in this lab as the modeling language for analysis and design.

Question Bank:

1. “Software engineers should not use their technical skills to *misuse* other people’s computers.” Here

the term *misuse* refers to:
a) Unauthorized access to computer material
b) Unauthorized modification of computer material
c) Dissemination of viruses or other malware
d) All of the mentioned

2. Identify the correct statement: "Software engineers shall
a) act in a manner that is in the best interests of his expertise and favour."
b) act consistently with the public interest."
c) ensure that their products only meet the SRS."
d) none

3. Select the incorrect statement: "Software engineers should
a) not knowingly accept work that is outside your competence."
b) not use your technical skills to misuse other people's computers."
c) be dependent on their colleagues."
d) Maintain integrity and independence in their professional judgment."

4. Efficiency in a software product does not include _________
a) responsiveness
b) licensing
c) memory utilization
d) processing time

5. As per an IBM report, "31%of the project get cancelled before they are completed, 53% overrun their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts". What is the reason for these statistics?
a) Lack of adequate training in software engineering
b) Lack of software ethics and understanding
c) Management issues in the company

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Problem Definition

The primary objective of the project is to develop a predictive model using Long Short-Term Memory (LSTM) neural networks that forecasts future Bitcoin prices based on historical price data. The goal is to accurately predict the price movements of Bitcoin to aid in investment decisions and risk management.

Feasibility Analysis

Financial Feasibility

1. **Initial Costs**:
   - **Data Acquisition**: While historical Bitcoin price data is generally available for free or at a low cost, acquiring high-quality, granular data may involve expenses.
   - **Computational Resources**: Investment in GPUs or cloud computing services to handle the training of LSTM models, which can be significant depending on the complexity and scale of the model.
   - **Development Team**: Costs associated with hiring data scientists, machine learning engineers, and possibly financial analysts.
2. **Operational Costs**:
   - **Model Maintenance and Updates**: Regular updates and maintenance to adapt to new market conditions and data, which might incur recurring costs.
   - **Cloud Services Fees**: Ongoing costs for cloud computing services if not managed in-house.
3. **Return on Investment (ROI)**:
   - **Increased Accuracy in Predictions**: Can potentially lead to higher profits from trading, improved risk management, and better strategic decisions.
   - **Market Edge**: Competitive advantage in trading strategies if the model outperforms other market predictions.

Technical Feasibility

1. **Data Handling and Processing**:
   - **Data Quality and Integrity**: Ensuring the data is clean, accurate, and free from biases is crucial for reliable outputs.
   - **Storage and Security**: Managing large datasets with secure and efficient storage solutions.
2. **Model Development**:
   - **LSTM Network Suitability**: LSTM's are proven in handling time-series data, but require careful tuning and optimization to capture the complex patterns in Bitcoin prices.
   - **Hyperparameter Tuning**: Extensive testing and validation to find the optimal settings for the model.
3. **Infrastructure and Tools**:
   - **Computational Infrastructure**: Requires robust infrastructure to support intensive computations.
   - **Software and Tools**: Availability of advanced machine learning frameworks and libraries (e.g., TensorFlow, PyTorch) to build and deploy LSTM models.

Social Feasibility

1. **Market Impact**:
   - **Ethical Considerations**: Concerns about the potential for model-based manipulations or exacerbating market volatility.
   - **Transparency and Trust**: Balancing model secrecy for competitive edge while maintaining transparency to foster trust among users and regulators.
2. **Regulatory Compliance**:

- **Adherence to Financial Regulations**: Ensuring the model and its usage comply with local and international financial regulations, including data privacy laws.

3. **Stakeholder Acceptance**:
    - **Investor Confidence**: Building confidence among stakeholders through demonstrable accuracy and reliability of predictions.
    - **Collaboration with Financial Analysts**: Integrating insights from financial experts with technical predictions to enhance model reliability and acceptance.

EXPERIMENT NO. 2

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:
Objective: To identify best suited process model for the given statement.
Outcome: Student will understand the importance of process models as per business use case and shall define the suitable process model for given project statement.
Problem Statement: Identify the suitable process model/s for the given project statement.
Background Study: A Software Process Model o Forms a common understanding of activities among the software developers. o Helps in identifying inconsistencies, redundancies, and omissions in the development process. o Helps in tailoring a process model for specific projects. o The development team must identify a suitable life cycle model and then adhere to it. o Helps monitor the progress of the project otherwise the project manager would have to depend on the guesses of the team members. This usually leads to a problem known as the 99% complete syndrome.
Question Bank: 6. Which of the following does data requirements allow for data? a. Entering data b. Leaving data c. Storing data in product d. All of the mentioned

7. Technical level abstraction includes?
   a. User level requirement
   b. physical level requirement
   c. operational level requirement
   d. All of the mentioned

8. Which of these is true?
   a. A physical-level requirement is a statement about how a product must support stakeholders in achieving their goals or tasks
   b. A operational-level requirement is a statement about the details of the physical form of a product, its physical interface to its environment, or its data formats
   c. All of the mentioned
   d. None of the mentioned

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Why the Agile Model is Ideal for Bitcoin Price Prediction

The Agile development model aligns exceptionally well with the task of predicting Bitcoin prices due to its:

- **Adapts to Changes:** Agile methodology is designed to accommodate changes in project requirements and market conditions, which is crucial for financial models affected by fluctuating markets like Bitcoin.
- **Iterative Development:** Allows for frequent adjustments based on feedback and new insights, which can be critical when dealing with complex and unpredictable data such as Bitcoin prices.
- **Regular Feedback Cycles:** Agile promotes regular reviews and iterations, enabling continuous improvement of the model based on the latest data and technological advancements.
- **Incremental Delivery:** Ensures that the model is developed in manageable increments, allowing for ongoing testing and refinement which is vital for tuning high-performance LSTM networks.
- **Collaborative Approach:** Agile facilitates closer collaboration between developers, data scientists, financial analysts, and other stakeholders, enhancing the model's relevance and effectiveness.
- **Transparency:** Regular updates and iterations provide transparency, helping build trust and manage expectations among stakeholders, including potential investors and regulatory bodies.
- **Early Problem Detection:** Frequent iterations enable early detection of issues and potential risks, reducing the likelihood of major setbacks and ensuring smoother project progress.
- **Feedback Loops:** Regular feedback and testing loops help in identifying and mitigating risks associated with data quality, model accuracy, and compliance with financial regulations.

EXPERIMENT NO. 3

Student Name and Roll Number: Piyush Gambhir - 21CSU349	
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)	
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)	
Date:	
Faculty Signature:	
Marks:	

Objective:

- Gain a deeper understanding of the Software Requirement Specification phase and the Software Requirement Specification (SRS).
- Learn to write requirements and specifications.

Outcome:
Students will understand the importance of SRS and design the SRS as per their system's requirement

Problem Statement:

- Introduction to the lab plan and objectives.
- Software Engineering and Project definition and, artifacts discussion.

Background Study:
Procedure:

Step 1:

Introduction:

Purpose

Identify the product whose software requirements are specified in this document. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.Describe the different types of user that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.

Project Scope

Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.
Step 2:

Overall Description

Product Perspective

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

Product Features

Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or a class diagram, is often effective.

User Classes and Characteristics

Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the favored user classes from those who are less important to satisfy.

Operating Environment

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

Design and Implementation Constraints

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

Step 3:

System Features

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

System Feature 1

Don't really say "System Feature 1." State the feature name in just a few words.

1 Description and Priority

Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).

2 Stimulus/Response Sequences

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

3 Functional Requirements

Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary.

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

REQ-1:
REQ-2:

Step 4:

External Interface Requirements

User Interfaces

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

Hardware Interfaces

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

Software Interfaces

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

Communications Interfaces

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

Nonfunctional Requirements

Performance Requirements

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

Safety Requirements

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

Security Requirements

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

Software Quality Attributes

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

Other Requirements

Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.

Question Bank:

1. Which is true among these?
   a. The job of creating, modifying, and managing requirements over a product's lifetime is called requirement development
   b. The portion of requirements engineering concerned with initially establishing requirements is termed requirements engineering
   c. The portion of requirements engineering concerned with controlling requirements changes is called requirement management
   d. All of the mentioned

2. Which is true for SRS?
   a. SRS is the main input of the software product design process
   b. SRS is the main output to the engineering design process
   c. SRS is also the main output of the requirements specification activity
   d. a and b
   c. b and c

3. SRS consists of?
   a. Problem statement
   b. Product design
   c. a,c
   d. None of the mentioned

4. Which of these are non-technical requirements?
   a. Functional Requirements
   b. Non-Functional Requirements
   c. Developer's Requirements
   d. Data Requirements

5. Which is true about functional requirement?
   a. A functional requirement is also called behavioural requirement
   b. A functional requirement includes development and operational requirements
   c. A functional requirement is a statement of how a software product must map program inputs to program outputs
   d. None of the mentioned.

6. Which of these are true for non functional requirements?
   a. A non-functional requirement is also called behavioral requirements
   b. A non-functional requirement is a statement that a software product must have certain properties
   c. It consists of Development and operational requirements
   d. b, c
   e. a, b, c

7. Which of these does not belong to the qualities of development requirements?
   a. Performance
   b. Response time

c. Maintainability
d. a, b
e. b, c

8. Which of these does not belong to the qualities of operational requirements?
   a. Memory usage
   b. Portability
   c. Reusability
   d. b, c
   e. a, c

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

SRS

1. Introduction

1.1 Purpose

This document specifies the software requirements for a Bitcoin price prediction system using an LSTM (Long Short-Term Memory) model. The system is designed to provide accurate predictions of Bitcoin prices to help users make informed trading decisions. This SRS aims to clearly outline functionality, constraints, and the interaction of the system with users to ensure alignment between stakeholders and the development team.

1.2 Document Conventions

This document follows the IEEE standard for SRS documents. Key terms and required software tools are listed and explained in the Glossary and References sections.

1.3 Intended Audience and Reading Suggestions

This document is intended for the project team, including developers, project managers, and testers, as well as external stakeholders such as investors and end-users (cryptocurrency traders). It is suggested that readers familiarize themselves with the project background before proceeding to the specific requirements.

1.4 Project Scope

The project will develop a predictive software system that utilizes an LSTM model to forecast Bitcoin prices based on historical data. The system will provide predictive insights through a user-friendly interface that allows traders to view predicted prices. Features include data ingestion, model training, result prediction, and a graphical display of historical and predicted prices.

1.5 References

- https://github.com/uttej2001/Bitcoin-Price-Prediction/tree/main

---

2. Overall Description

2.1 Product Perspective

The system is a standalone application but could potentially integrate with existing trading platforms through APIs in future versions. It is reliant on continuous data feeds from cryptocurrency exchanges and uses historical pricing data to train the LSTM model.

2.2 Product Features

- Data collection and preprocessing
- LSTM model training
- Price prediction generation
- User interface for data visualization
- Option to adjust model parameters (for advanced users)

2.3 User Classes and Characteristics

- **End-Users (Traders)**: Require accurate and timely predictions with an easy-to-use interface.
- **Data Scientists**: Need the ability to tweak algorithms and access detailed model performance data.
- **System Administrators**: Manage system operations and ensure data integrity and security.

2.4 Operating Environment

The system will be developed in Python, utilizing libraries such as TensorFlow, Keras, and Pandas. It will run on Linux and Windows operating systems and will require a GPU for efficient model training.

2.5 Design and Implementation Constraints

- The system requires a continuous and reliable data source.

- High-performance computing resources are needed for real-time data processing and model training.
- The system must comply with financial data protection regulations.

2.6 Assumptions and Dependencies

- Accurate predictions depend on the continuous availability of recent and historical Bitcoin price data.
- System performance is dependent on the advancements in LSTM and other machine learning technologies.

---

3. System Features

3.1 Data Collection and Management

Description: The system will automatically collect data from predefined cryptocurrency exchange APIs. The data will be cleaned, preprocessed, and stored in a structured format suitable for LSTM processing.

Requirements:

- R1.1: The system shall update the dataset at least every hour.
- R1.2: The system shall handle anomalies and outliers in data before processing.

3.2 LSTM Model Training and Prediction

Description: The LSTM model will be trained on historical data and used to make price predictions.

Requirements:

- R2.1: The system shall retrain the model on a daily basis or upon significant market events.
- R2.2: The system shall provide predictive outputs with an accuracy metric.

3.3 User Interface

Description: A graphical user interface that displays both historical data and predictions clearly.

Requirements:

- R3.1: The interface shall display predictive results graphically.
- R3.2: The interface shall allow users to view details on demand.

---

4. External Interface Requirements

4.1 User Interfaces

- GUI developed using a NextJS React Framework.

4.2 Hardware Interfaces

- Requires GPU support for model training.
- Standard PC hardware for running the software.

4.3 Software Interfaces

- Python 3.8 or higher.
- TensorFlow 2.x, Keras for LSTM model development.

4.4 Communications Interfaces

- HTTPS for secure data transfer from APIs.

---

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The system shall process new data and update predictions within 1 minute of data receipt.

5.2 Security Requirements

- The system shall implement standard cybersecurity measures to protect data integrity and privacy.

5.3 Software Quality Attributes

- **Reliability**: The system should perform consistently under varying conditions.
- **Usability**: The interface should be intuitive and require minimal training for users.
- **Maintainability**: Code should be well-documented and maintainable.

EXPERIMENT NO. 4

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:

Objective:

- Study the benefits of visual modeling.
- Learn use case diagrams: discovering actors and discovering use cases.
- Practice use cases diagrams using CASE tool.

Outcome:
Students will understand the importance of SEPM and about various software artifacts
Students will learn to perform feasibility analysis by designing Feasibility Study Document

Problem Statement:
Draw the use case diagram of your project by identifying primary, secondary actors and use cases.

Background Study:

Basic elements used in use case diagram:

Actor

The **use case diagram** allows a designer to graphically show these use cases and the actors that use them. An **actor** is a role that a user plays in the system. It is important to distinguish between a user and an actor (better thought of as a role). A user of the system may play several different roles through the course of his, her or its job (since an actor may be another system). Examples of actors are salesperson, manager, support person, and web store system. It is possible that the same person may be a salesperson and also provide support. When creating a use case model, we are not concerned with the individuals, only the roles that they play.

Actors can be of two types:

- Primary actors, using system in daily activities
- Secondary actors, enabling primary actors to use a system.

How do you know who the actors are in a UCD?

When working from an Action/Response table, identifying the actors is easy: entities whose behavior appears in the "Actor's Actions" column are the actors, and entities whose behavior appears in the "System's Response" column are components in the system.
the actors are typically those entities whose behavior cannot control or change (i.e., agents that are not part of the system that you are building or describing.)

Use cases

A use case

- Constitutes complete course of events initiated by actor
- Defines interaction between actor and the system
- Is a member of set of all use cases which together define all existing ways of using the system

How to select the use cases:

Consider the simple scenario in which a photographer takes the picture. So the use cases will be Take picture and Change film not that first he/she opens the shutter, then take picture and then close shutter. These do not focus on the functionality of the photographer.

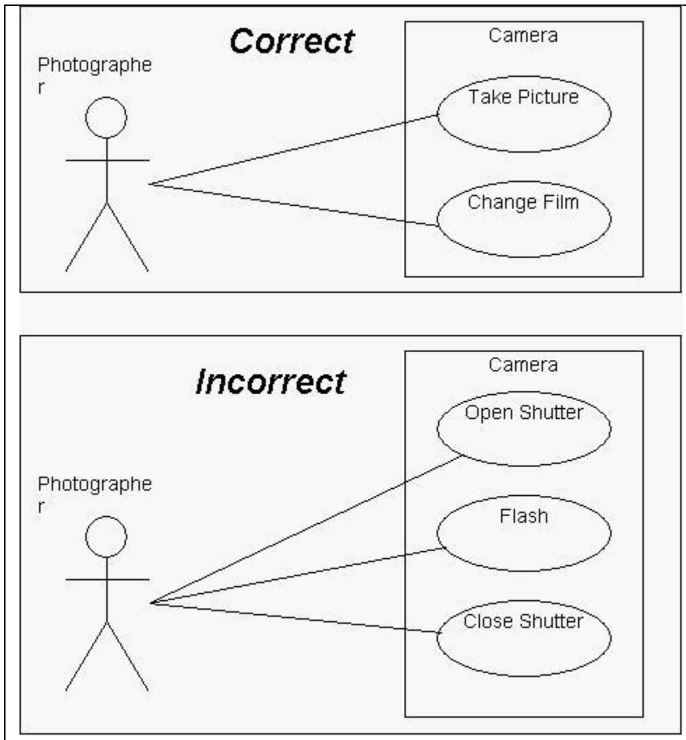
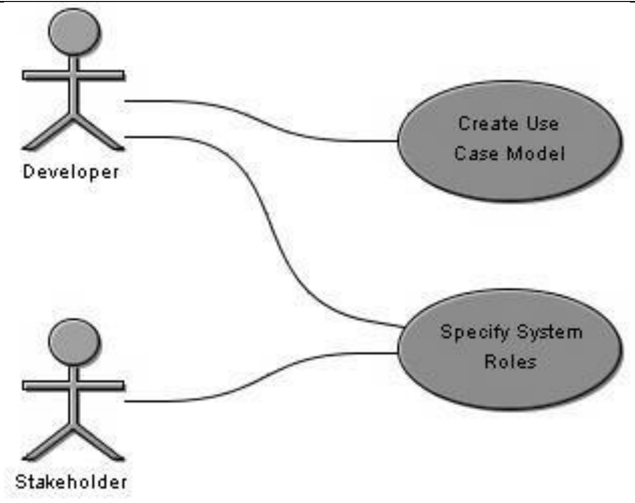


Figure 1: Use case diagram (example)

Associations

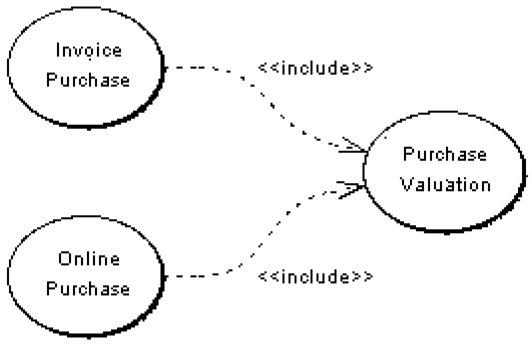
On a use case diagram, associations are drawn between actors and use cases to show that an actor carries out a use case. A use case can be carried out by many actors and an actor may carry out many use cases.



In this diagram associations between actors and use cases are represented by the connecting lines. The developer and the stakeholder both are responsible for specifying the system roles, but only the developer creates the model.

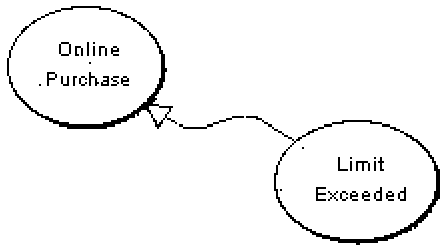
Includes

Use cases can also be related to each other with different links. The diagram below shows the use of the includes link. The best way to think of an include dependency is that it is the invocation of a use case by another one. In this figure both *invoice purchase* and *online purchase* include the scenarios defined by *purchase valuation*. In general, the includes link is to avoid repetition of scenarios in multiple use cases.



Generalization

When a use case describes a variation on another use case, use a generalization link. In the example below, the use case *limit exceeded* describes a situation in which the usual scenario of *online purchase* is not performed. Use cases that generalize another use case should only specify an alternative, even exceptional, scenario to the use case being generalized. The overall goal of the use cases should be the same.

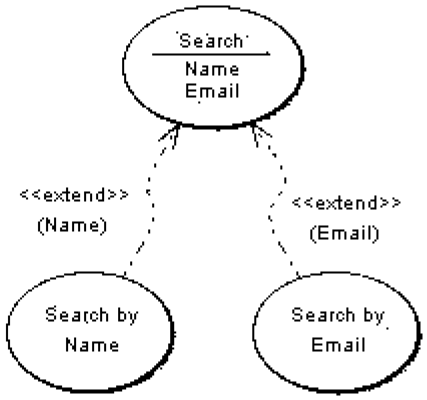


Extends

In some instances you want to describe a variation on behaviour in a more controlled form. In such instances you can define **extension points** in the extended use case. An extending **use case** continues the behavior of a base use case. The extending use case accomplishes this by conceptually inserting additional action sequences into the base use-case sequence. This allows an extending use case to continue the activity sequence of a base use case when the appropriate extension point is reached in the base use case and the extension condition is fulfilled. When the extending use case activity sequence is completed, the base use case continues.

In the example below, *search by name* is said to extend *search* at the *name* extension point. The extended link is more controlled than the generalization link in that functionality can only be added at the extension points.

An extends relationship shows optional/exceptional behavior



Question Bank:

1. **Differentiate between primary and secondary actors.**
2. **Draw the use case diagram of courseware management system.**

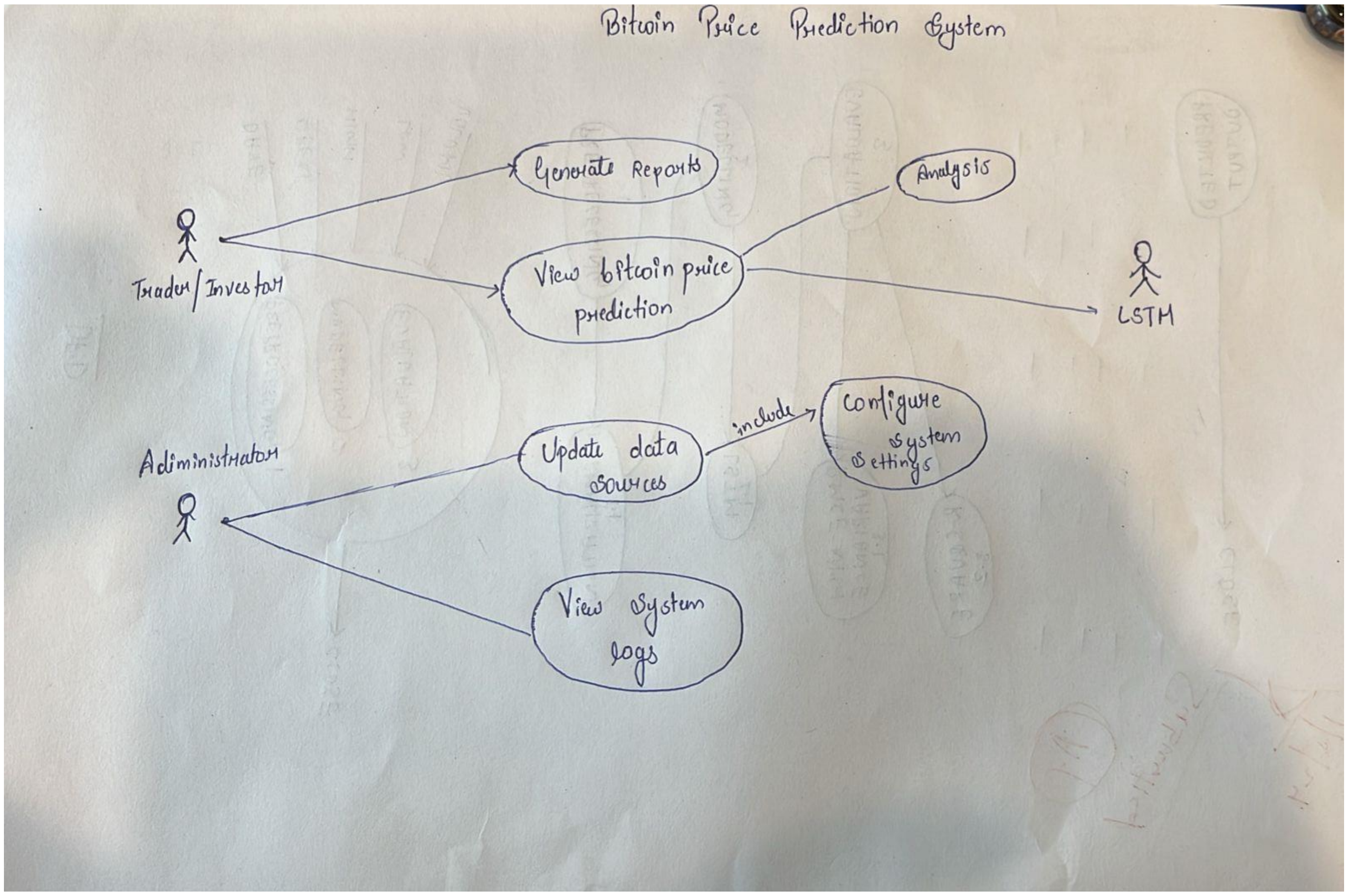
Student Work Area

Algorithm/Flowchart/Code/Sample Output

User Stories

1. **As a trader**, I want to input a starting date so that I can specify the date from which to start predicting Bitcoin prices.
2. **As a trader**, I want to input the number of days for which I want predictions so that I can specify the duration of the forecast.
3. **As a trader**, I want to receive predicted Bitcoin prices for the specified duration so that I can make informed decisions about buying, selling, or holding Bitcoin.
4. **As a trader**, I want the predictions to be displayed in dollars so that I can easily understand the predicted prices in my preferred currency.
5. **As a trader**, I want the predicted prices to be visually represented over time so that I can visualize the trend and fluctuations in Bitcoin prices.
6. **As a trader**, I want the predictions to be accurate and reliable so that I can trust the information provided by the system.
7. **As a trader**, I want the system to handle errors gracefully and provide helpful error messages in case of input mistakes or other issues.
8. **As a trader**, I want the system to be easy to use and navigate so that I can quickly input my preferences and obtain the predicted prices without any hassle.
9. **As a trader**, I want the system to be accessible from various devices and platforms so that I can use it conveniently from my desktop, laptop, or mobile device.
10. **As a trader**, I want the system to be regularly updated with the latest data and improvements so that I can continue to rely on it for accurate predictions over time.

Use Case Diagram



EXPERIMENT NO. 5

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:
Objective: Learn the object-oriented analysis phase by understanding the methods of class elicitation and finding the classes in an object-oriented system.
Outcome: Students will learn to discover the classes and mapping from use case to class diagram.
Problem Statement: Draw the class diagram of mentioned project by discovering classes.
Background Study: A class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagram are static-they display what interacts but not what happens when they do interact. **Class diagram address the static design view of the system. Class diagram that include active classes address the static process view of a system**. Classes define the properties of the objects, which belongs to them. These include: **Attributes**-(second container)t he data properties of the classes including type, default value and constraints **Operations**-(third container) the signature of the functionality that can be applied to the objects of the classes including parameters, parameter types, parameter constraints, return types and the semantics.

Associations-(solid lines between classes) the references contained within the objects of the classes to other objects enabling interaction with those objects. So, the purpose of the class diagram can be summarized as:

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

Build the class diagram:

Till now we have use cases which show that what the system do?

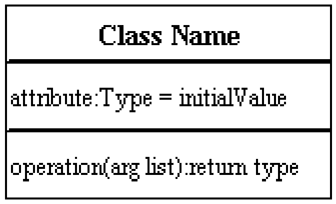
Now the next step is to design the system which means try to find the answer that:

- How the system does accomplish its goal?
    - Find the responsibilities from requirements, and distribute them over classes.
- Identifying the classes:
    - Actors are primary entities in the system and they will be mapped on classes.
    - Also we have passive entities, also they will be mapped to classes.
- Identify the relationships among classes

Basic Class Diagram Symbols and Notations

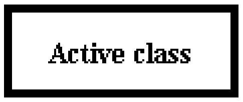
Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and write operations into the third.


Class Name
attribute:Type = initialValue
operation(arg list):return type

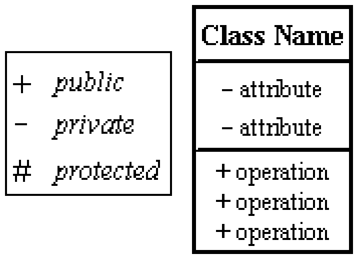
Active Class

Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.



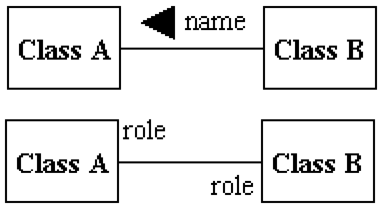
Visibility

Use visibility markers to signify who can access the information contained within a class. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class.



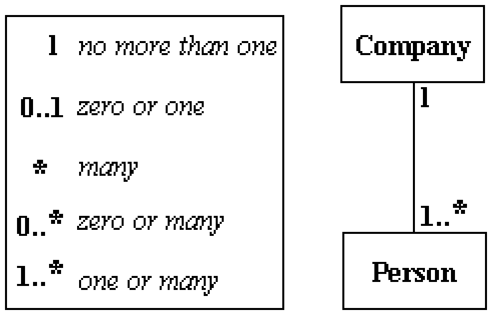
Associations

Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other. ***Note:*** It's uncommon to name both the association and the class roles.



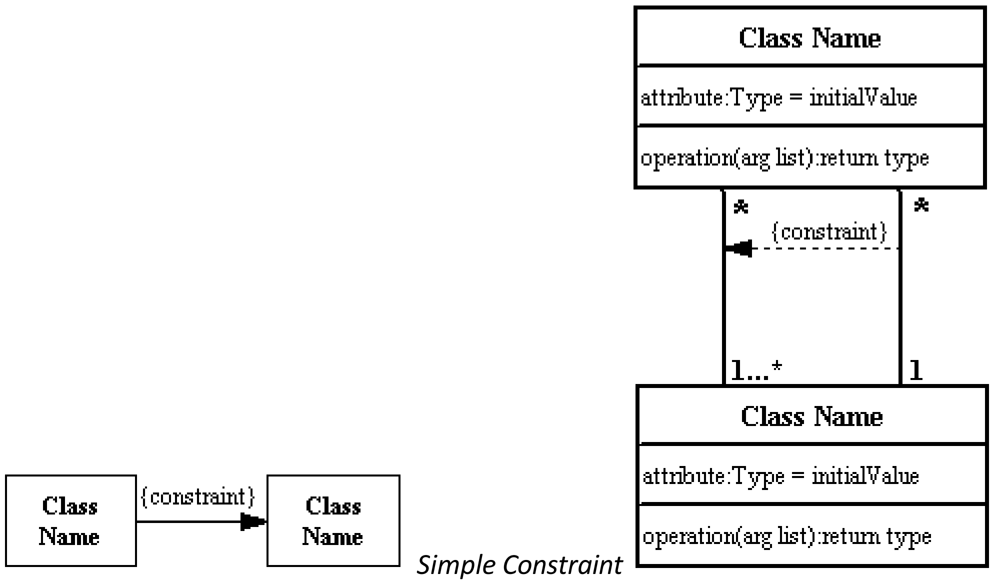
Multiplicity (Cardinality)

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for one company only.



Constraint

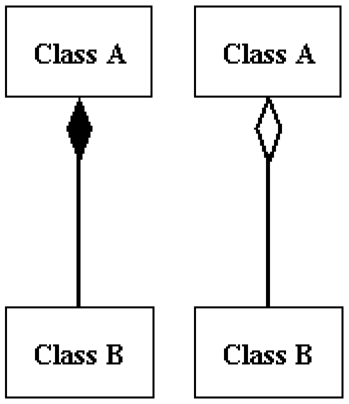
Place constraints inside curly braces {}.


Class Name
attribute:Type = initialValue
operation(arg list):return type
*
{constraint}
*
1...*
1
Class Name
attribute:Type = initialValue
operation(arg list):return type
Class Name
{constraint}
Class Name

Simple Constraint

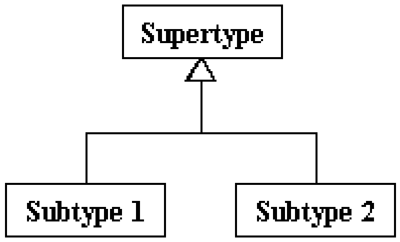
Composition and Aggregation

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate.



Generalization

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.

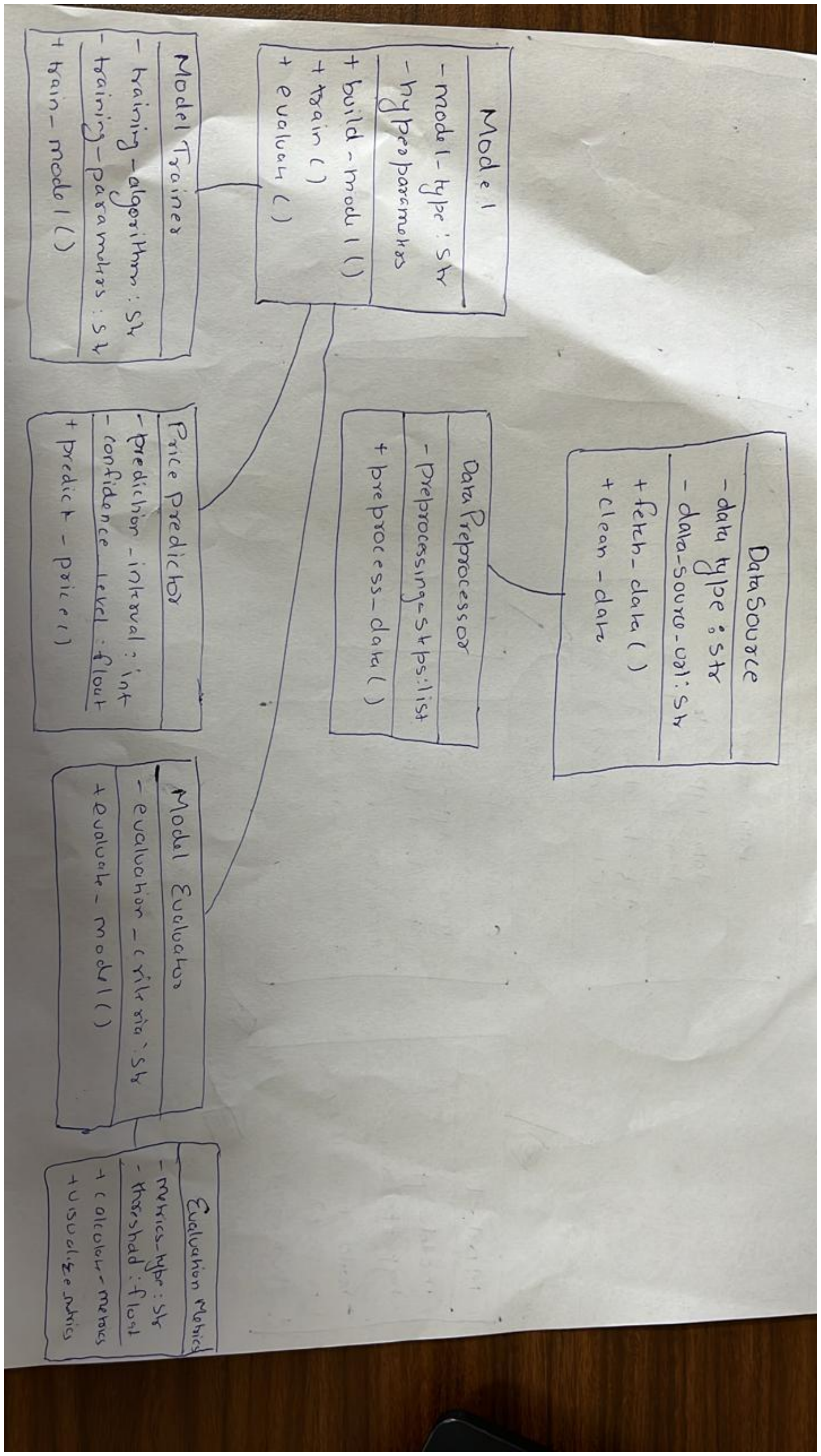


In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.

Question Bank:
Differentiate between active and passive classes
Draw the class and object diagram for courseware management system

Student Work Area

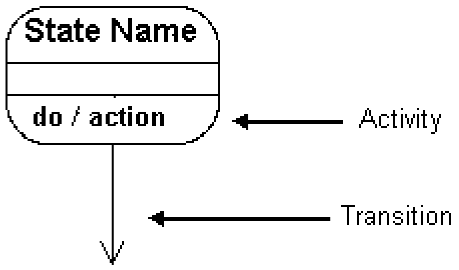
Algorithm/Flowchart/Code/Sample Outputs



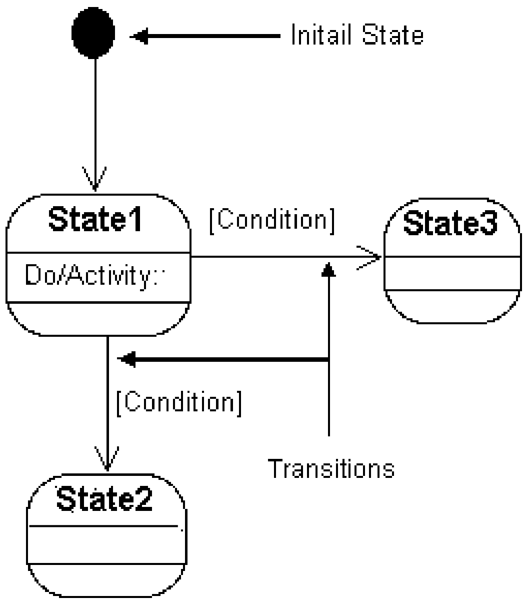
EXPERIMENT NO. 6

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:
Objective: • Deeper understanding of UML state transition diagrams (STD). • Practicing using CASE tool.
Outcome: Students will learn to design the state and activity diagrams.
Problem Statement: Draw the state and activity diagrams of mentioned project.
Background Study: While coding, it is necessary to understand the details of the modes an Object of a Class can go through and its transitions at time intervals with the occurrence of any event or action. State diagrams (also called State Chart diagrams) are used to help the developer better understand any complex/unusual functionalities or business flows of specialized areas of the system. In short, State diagrams depict the dynamic behavior of the entire system, or a sub-system, or even a single object in a system. This is done with the help of *Behavioral elements*. It is important to note that having a State diagram for your system is not a compulsion, but must be defined only on a need basis

State diagrams have very few elements. State diagrams have very few elements. The basic elements are rounded boxes representing the state of the object and arrows indicting the transition to the next state. The activity section of the state symbol depicts what activities the object will be doing while it is in that state.



All state diagrams being with an initial state of the object. This is the state of the object when it is created. After the initial state the object begins changing states. Conditions based on the activities can determine what the next state the object transitions to.



Below is an example of a state diagram might look like for an Order object. When the object enters the Checking state it performs the activity "check items." After the activity is completed the object transitions to the next state based on the conditions [all items available] or [an item is not available]. If an item is not available the order is canceled. If all items are available then the order is dispatched. When the object transitions to the Dispatching state the activity "initiate delivery" is performed. After this activity is complete the object transitions again to the delivered state.

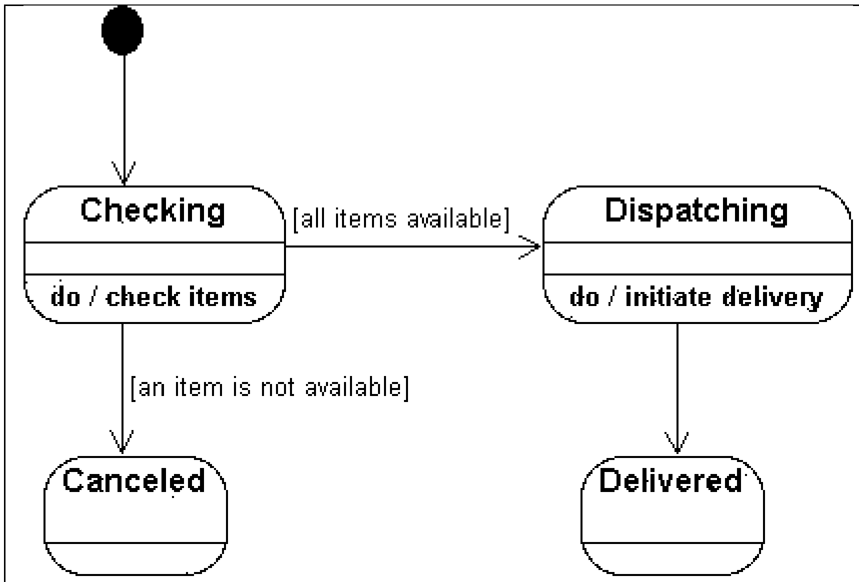


Figure- State diagram for an order object

State diagrams can also show a super-state for the object. A super-state is used when many transitions lead to the a certain state. Instead of showing all of the transitions from each state to the redundant state a super-state can be used to show that all of the states inside of the super- state can transition to the redundant state. This helps make the state diagram easier to read.

The diagram below shows a super-state. Both the Checking and Dispatching states can transition into the Canceled state, so a transition is shown from a super-state named Active to the state Cancel. By contrast, the state Dispatching can only transition to the Delivered state, so we show an arrow only from the Dispatching state to the Delivered state.

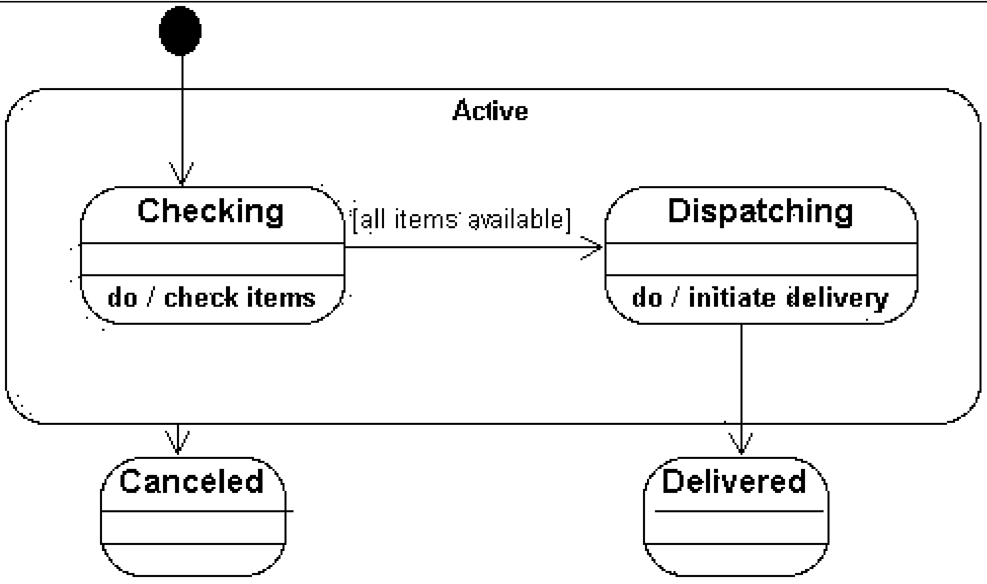


Figure: state diagram showing super state

Activity Diagram:

Activity diagram are fancy flowcharts. Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

Creating an Activity Diagram

Let us consider the example of attending a course lecture, at 8 am.

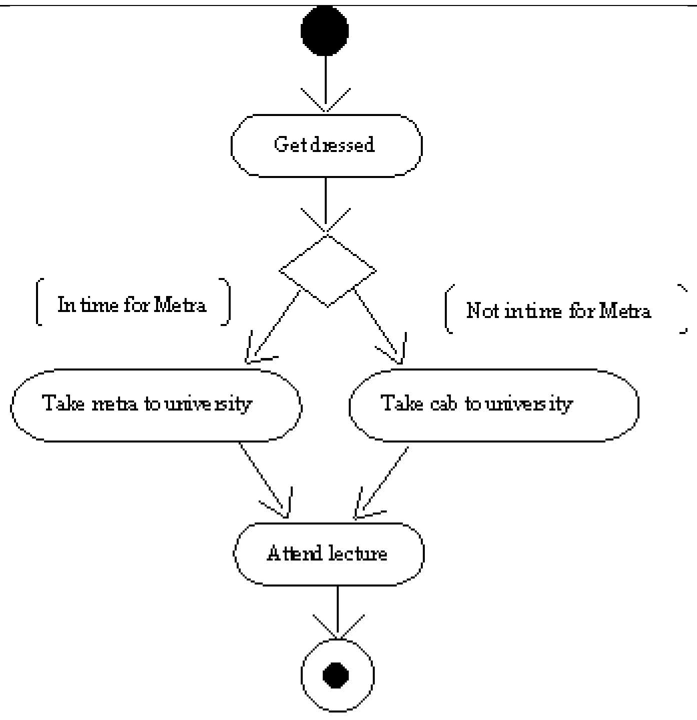


Figure —an example Activity diagram

As you can see in Figure 6 the first activity is to get dressed to leave for the lecture. A decision then has to be made, depending on the time available for the lecture to start, and the timings of the public trains (metra). If there is sufficient time to catch the train, then take the train; else, flag down a cab to the University. The final activity is to attend the lecture, after which the Activity diagram terminates.

Swim Lanes:

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

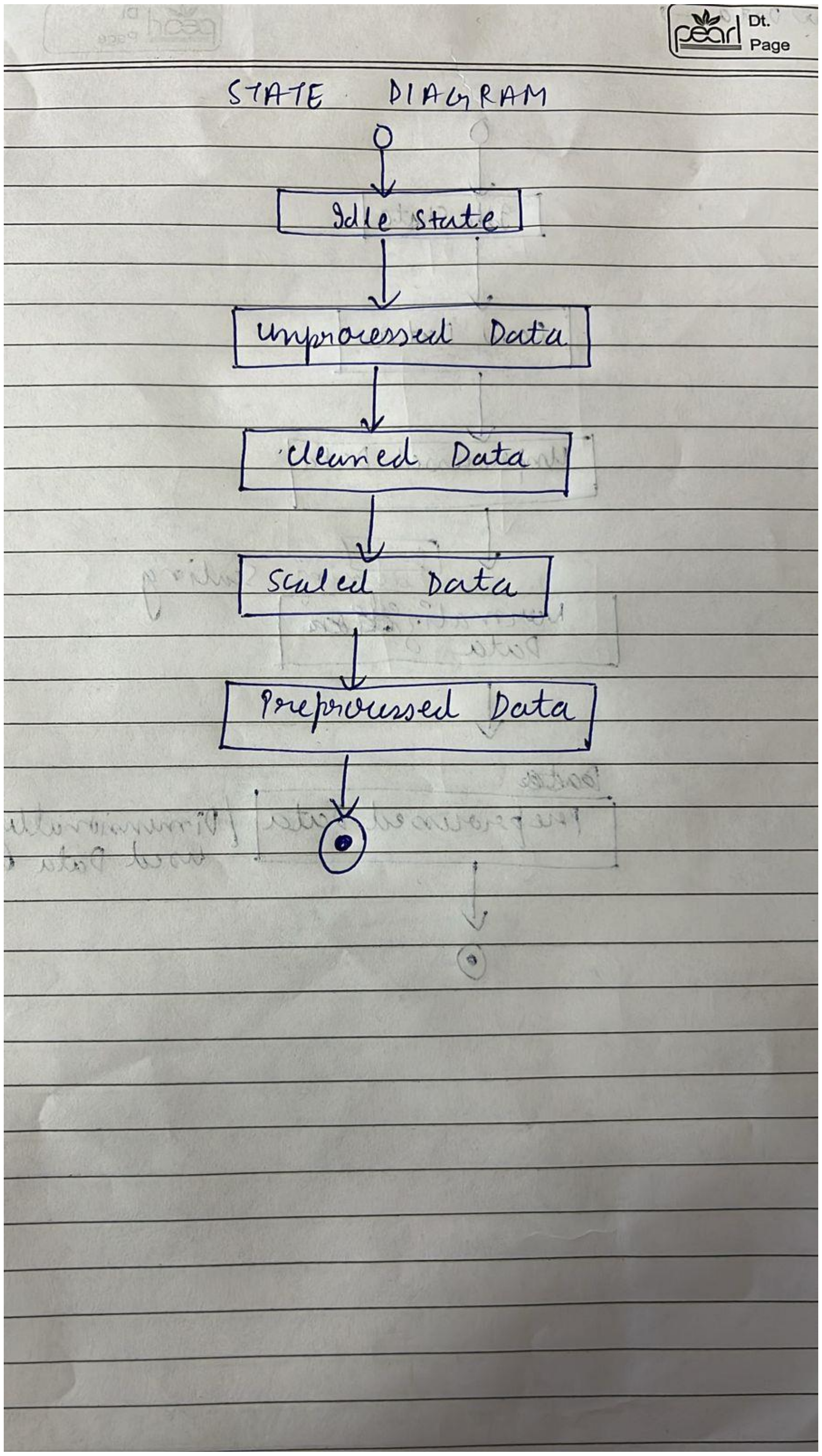
A state chart diagram shows the possible states of the objects and the transition that cause a change in the state.

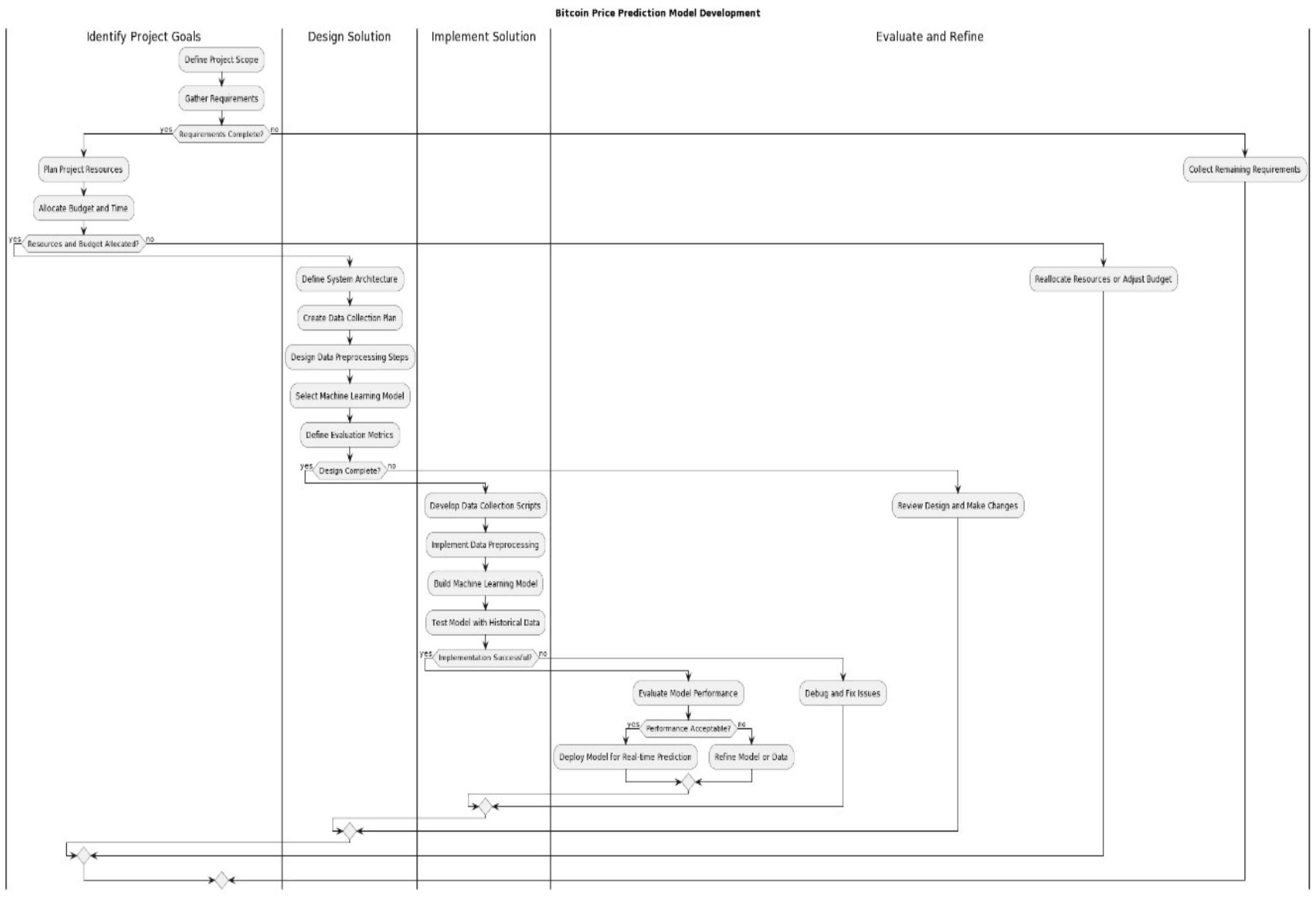
An activity diagram are related to state chart diagram except that state chart diagram focuses attention on an object undergoing a process(or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one

another.
Question Bank: 1. Draw the use case diagram on ATM Withdrawal involving: Actors, Preconditions, Post conditions, Use Cases. 2. Differentiate between activity diagram and state chart diagram 3. Draw the state chart diagram and Activity diagram of ATM system.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs


STATE DIAGRAM
Idle State
Unprocessed Data
Cleaned Data
Scaled Data
Preprocessed Data


Bitcoin Price Prediction Model Development
Identify Project Goals
Design Solution
Implement Solution
Evaluate and Refine
Define Project Scope
Gather Requirements
Requirements Complete?
yes
no
Collect Remaining Requirements
Plan Project Resources
Allocate Budget and Time
Resources and Budget Allocated?
yes
no
Reallocate Resources or Adjust Budget
Define System Architecture
Create Data Collection Plan
Design Data Preprocessing Steps
Select Machine Learning Model
Define Evaluation Metrics
Design Complete?
yes
no
Review Design and Make Changes
Develop Data Collection Scripts
Implement Data Preprocessing
Build Machine Learning Model
Test Model with Historical Data
Implementation Successful?
yes
no
Debug and Fix Issues
Evaluate Model Performance
Performance Acceptable?
yes
no
Deploy Model for Real-time Prediction
Refine Model or Data

EXPERIMENT NO. 7

Student Name and Roll Number: Piyush Gambhir - 21CSU349	
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)	
Link to Code: [ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)]()	
Date:	
Faculty Signature:	
Marks:	

Objective:
Better understanding of the interaction diagrams.
Get familiar with sequence & collaboration diagrams.
Practice drawing the interaction diagrams using CASE tool.

Outcome:
Students will learn to design sequence and collaboration diagrams.

Problem Statement:
Draw the sequence and collaboration diagrams of mentioned project.

Background Study:

*A use case diagram presents an **outside** view of the system. Then, how about the **inside** view of the system?*

Interaction diagrams describe how use cases are ***realize***d in terms of interacting objects.

Two types of interaction diagrams

1. Sequence diagrams
2. Collaboration *(Communication)* diagrams

Once the use cases are specified, and some of the core objects in the system are prototyped on class diagrams, we can start designing the dynamic behavior of the system.

Let's start with the simple example above: a user logging onto the system. The Logon use case can be specified by the following step:

1. Logon dialog is shown
2. User enters user name and password
3. User clicks on OK or presses the enter key
4. The user name and password are checked and approved
5. The user is allowed into the system

   Alternative: Logon Failed - if at step 4 the user name and password are not approved, allow the user to try again

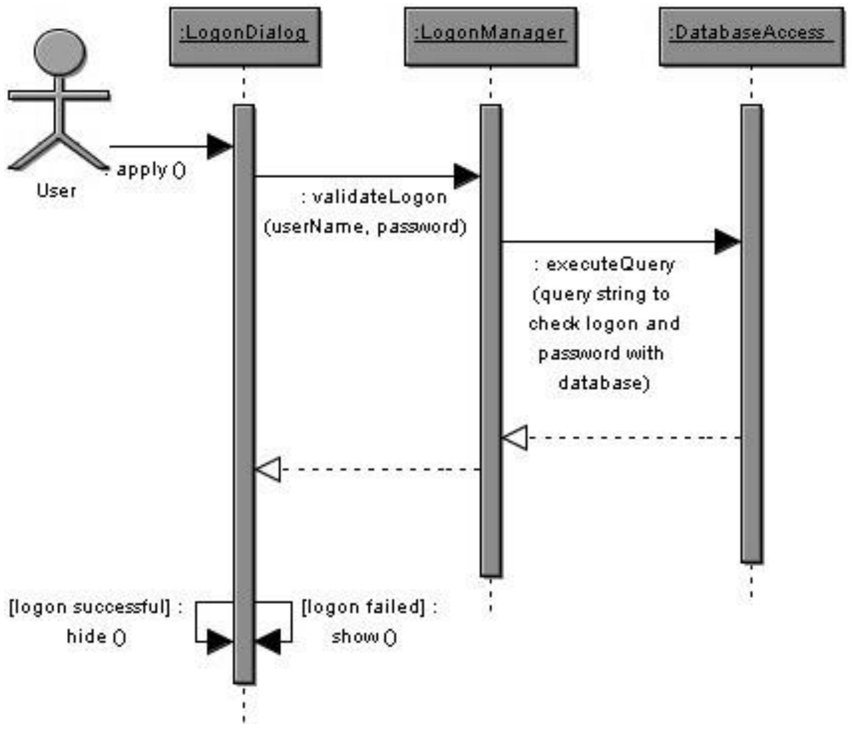

:LogonDialog
:LogonManager
:DatabaseAccess
User
: apply ()
: validateLogon (userName, password)
: executeQuery (query string to check logon and password with database)
[logon successful] : hide ()
[logon failed] : show ()

Figure- An example sequence diagram for our logon collaboration

Collaboration Diagrams

Collaborations are more complex to follow than sequence diagrams, but they do provide the added benefit of more flexibility in terms of spatial layout.

A Sequence diagram is dynamic, and, more importantly, is time ordered. A Collaboration diagram is very similar to a Sequence diagram in the purpose it achieves; in other words, it shows the dynamic interaction of the objects in a system. A distinguishing feature of a Collaboration diagram is that it shows the objects and their association with other objects in the system apart from how they interact with each other. The association between objects is not represented in a Sequence diagram.

A Collaboration diagram is easily represented by modeling objects in a system and representing the associations between the objects as links. The interaction between the objects is denoted by arrows. To identify the sequence of invocation of these objects, a number is placed next to each of these arrows.

Defining a Collaboration diagram

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

Let us see in detail what the elements of Collaboration diagram are.

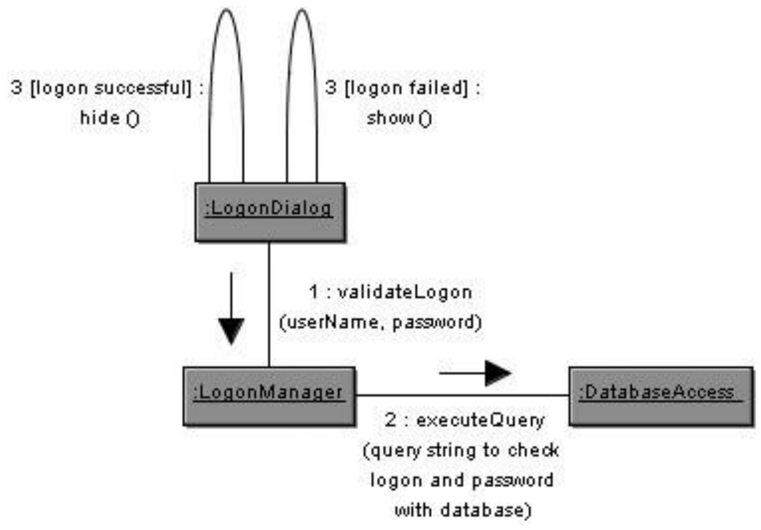

3 [logon successful] : hide ()
3 [logon failed] : show ()
:LogonDialog
1 : validateLogon (userName, password)
:LogonManager
:DatabaseAccess
2 : executeQuery (query string to check logon and password with database)

Figure-:A Collaboration diagram for Logon interaction .

Notice that each message is numbered in sequence, because it is not obvious from the diagram, the order of the messages.

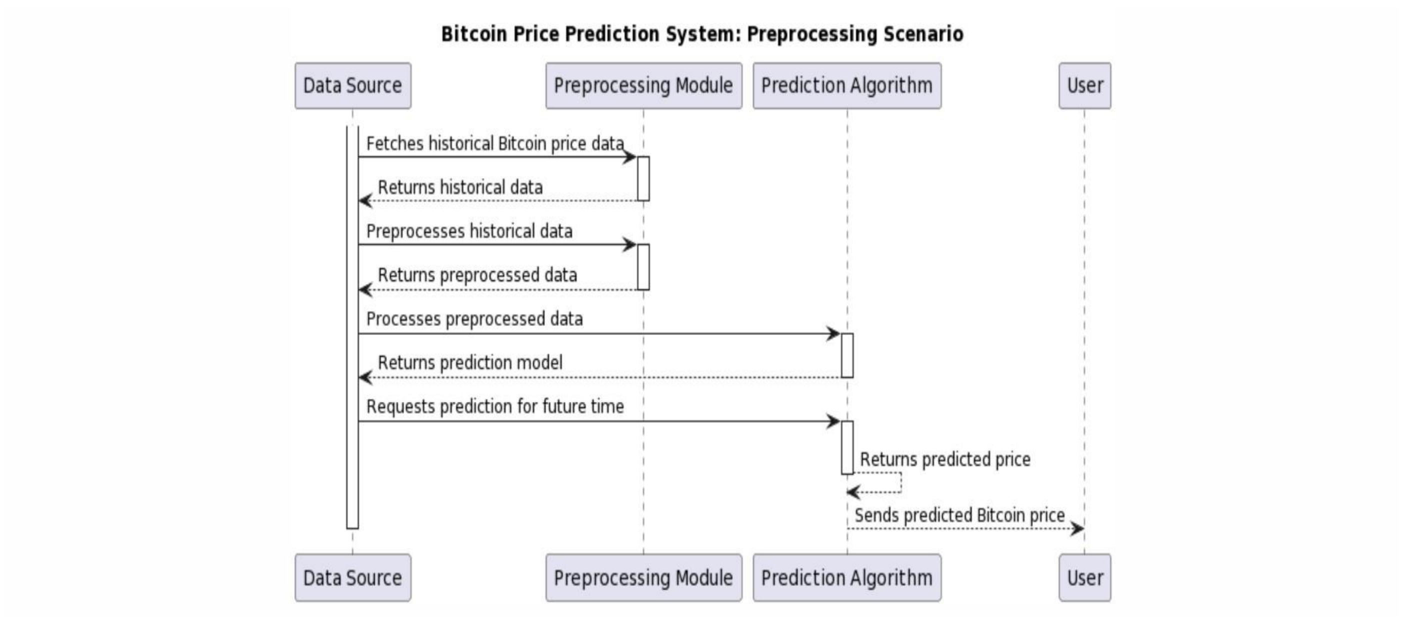
Sequence diagrams emphasize the order in which things happen, while collaboration diagrams give more flexibility in their layout. You can use whichever you prefer when drawing interactions, as both show the same information.

Question Bank:

1. Draw the interaction diagrams for courseware management system
2. Differentiate between activity diagram and state chart diagram

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs


Bitcoin Price Prediction System: Preprocessing Scenario
Data Source
Preprocessing Module
Prediction Algorithm
User
Fetches historical Bitcoin price data
Returns historical data
Preprocesses historical data
Returns preprocessed data
Processes preprocessed data
Returns prediction model
Requests prediction for future time
Returns predicted price
Sends predicted Bitcoin price
Data Source
Preprocessing Module
Prediction Algorithm
User

EXPERIMENT NO. 8

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:
Objective: • Gain a deeper understanding of software testing and the software testing document. • Become familiar with a Test Plan Document
Outcome: Test Plan document using excel.
Problem Statement: Write the test cases (minimum 10) for Traffic Light System /your defined project

Background Study:

Testing is the process of executing a program with the intent of finding errors. A good test case is one with a high probability of finding an as-yet undiscovered error. A successful test is one that discovers an as-yet-undiscovered error.

The causes of the software defects are: specification may be wrong; specification may be a physical impossibility; faulty program design; or the program may be incorrect.

Basic Definitions

- **A failure** is an unacceptable behavior exhibited by a system.
- **A defect** is a flaw in any aspect of the system that contributes, or may potentially contribute, to the occurrence of one or more failures. It might take several defects to cause a particular failure.
- **An error** is a slip-up or inappropriate decision by a software developer that leads to the introduction of a defect.

Good Test Attributes

A good test has a high probability of finding an error, not redundant, and should not be too simple or too complex.

The characteristics of a Banking application are as follows:

- Multi tier functionality to support thousands of concurrent user sessions
- Large scale Integration , typically a banking application integrates with numerous other applications such as Bill Pay utility and Trading accounts
- Complex Business workflows
- Real Time and Batch processing
- High rate of Transactions per seconds
- Secure Transactions
- Robust Reporting section to keep track of day to day transactions
- Strong Auditing to troubleshoot customer issues
- Massive storage system
- Disaster Management.

Test Cases

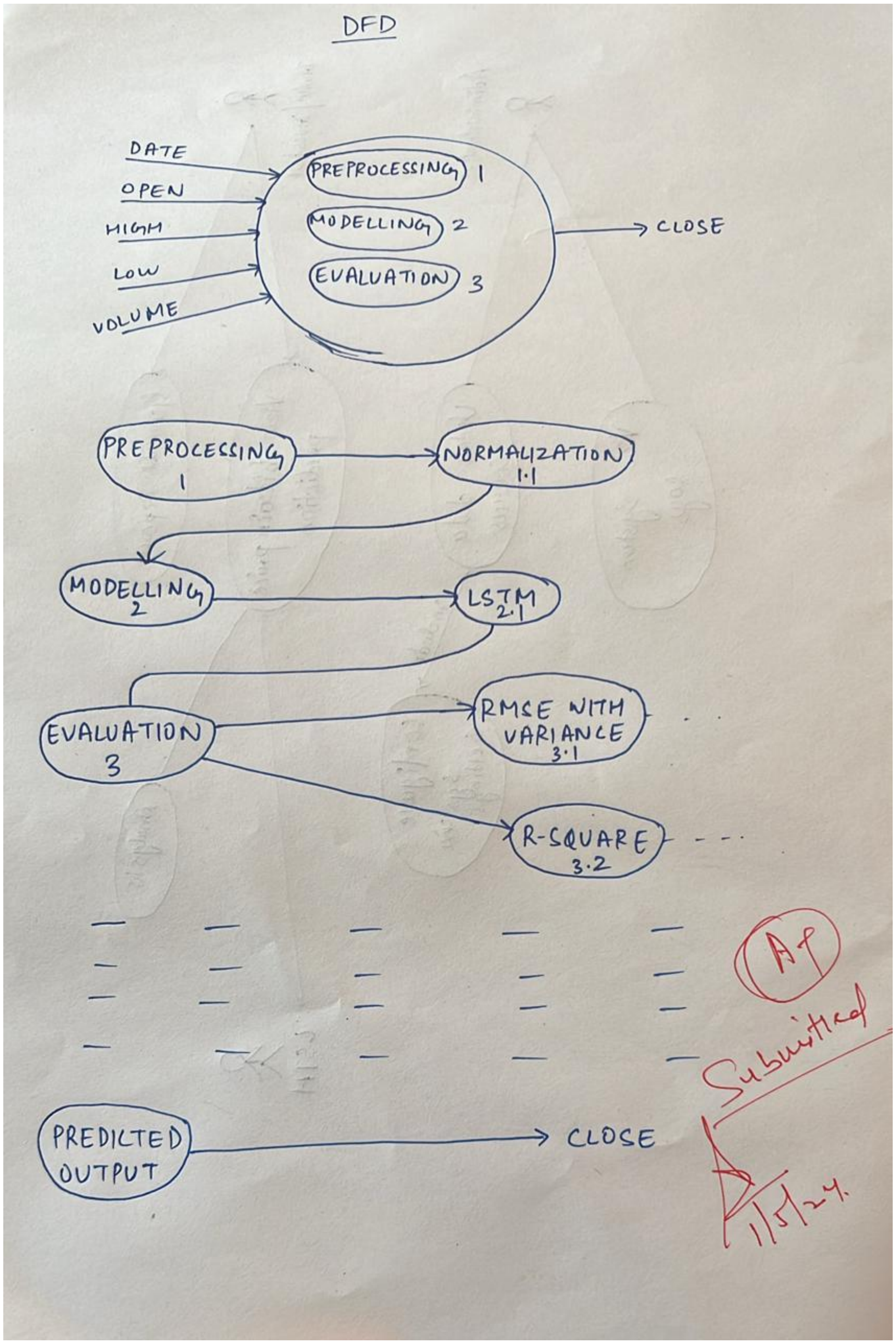
File Open - Test case
Steps to reproduce:
1. Launch Application
2. Select "File" menu
File menu pulls down
3. Choose "Open"
"Open" dialog box appears
4. Select a file to open
5. Click OK
Result: File should open
Test case
Test case ID: B 001
Test Description: verify B - bold formatting to the text
Function to be tested: B - bold formatting to the text
Environment: Win 98
Actual Result: pass

Question Bank:

Explain the complete test bug life cycle and discuss the importance of test document for that

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs


DFD
DATE
OPEN
HIGH
LOW
VOLUME
PREPROCESSING 1
MODELLING 2
EVALUATION 3
CLOSE
PREPROCESSING 1
NORMALIZATION 1.1
MODELLING 2
LSTM 2.1
EVALUATION 3
RMSE WITH VARIANCE 3.1
R-SQUARE 3.2
PREDICTED OUTPUT
CLOSE
A+
Submitted
1/5/24.

EXPERIMENT NO. 9

Student Name and Roll Number: Piyush Gambhir - 21CSU349
Semester /Section: AIML-B: 6th Semester – AIML-B (A3)
Link to Code: ncu-lab-manual-and-end-semester-projects/NCU-CSL229 - SEPM - Lab Manual at main · piyush-gambhir/ncu-lab-manual-and-end-semester-projects (github.com)
Date:
Faculty Signature:
Marks:
Objective: To make the students aware about white box testing techniques including finding of independent paths.
Outcome: Students will learn to find the independent paths for defined system/module.
Problem Statement: Consider the triangle problem. Write the program and find all the existing independent paths.

Background Study:

Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module. Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:

Cyclomatic complexity = E - N + 2*P

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

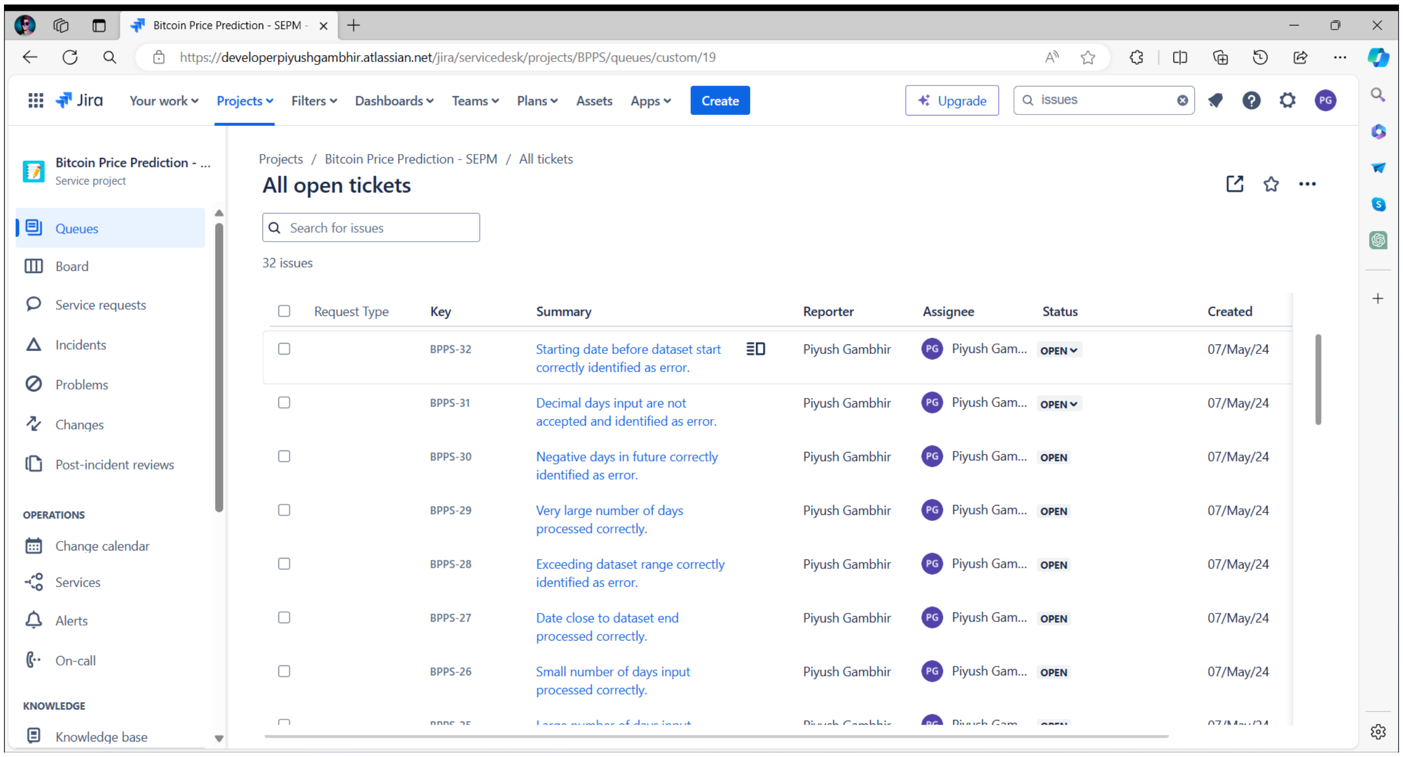
P = number of nodes that have exit points

Question Bank:
Write a program to find the roots of quadratic equation and find all the existing independent paths in it.

Student Work Area

Algorithm/Flowchart/Code/Sample Outputs

Tets Plan.xlsx   Test Cases.xlsx

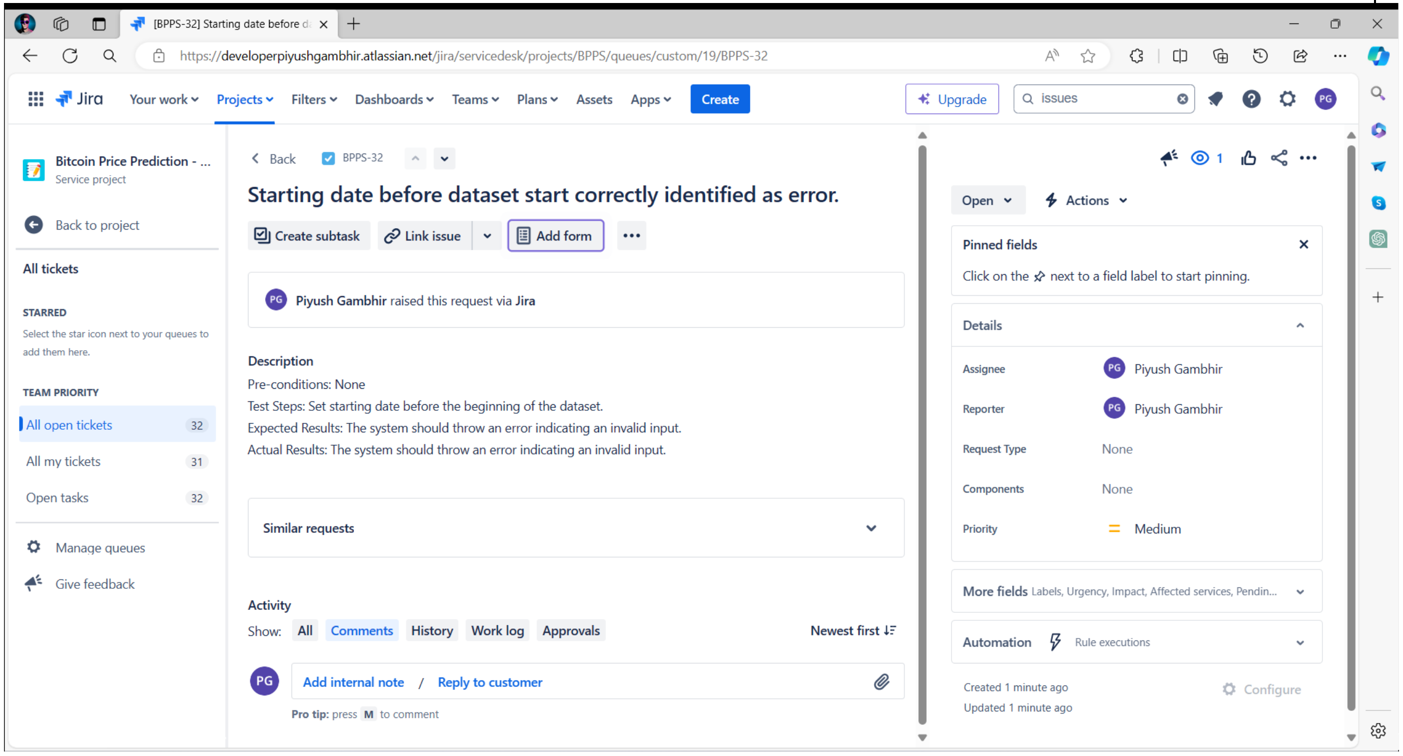


Projects / Bitcoin Price Prediction - SEPM / All tickets

All open tickets

32 issues

Request Type	Key	Summary	Reporter	Assignee	Status	Created
	BPPS-32	Starting date before dataset start correctly identified as error.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-31	Decimal days input are not accepted and identified as error.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-30	Negative days in future correctly identified as error.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-29	Very large number of days processed correctly.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-28	Exceeding dataset range correctly identified as error.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-27	Date close to dataset end processed correctly.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-26	Small number of days input processed correctly.	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24
	BPPS-25	Large number of days input	Piyush Gambhir	Piyush Gam...	OPEN	07/May/24



[BPPS-32] Starting date before d

https://developerpiyushgambhir.atlassian.net/jira/servicedesk/projects/BPPS/queues/custom/19/BPPS-32

Bitcoin Price Prediction - ...
Service project

BPPS-32

Starting date before dataset start correctly identified as error.

Piyush Gambhir raised this request via Jira

Description
Pre-conditions: None
Test Steps: Set starting date before the beginning of the dataset.
Expected Results: The system should throw an error indicating an invalid input.
Actual Results: The system should throw an error indicating an invalid input.

Similar requests

Activity

Pinned fields
Click on the next to a field label to start pinning.

Details

Assignee	Piyush Gambhir
Reporter	Piyush Gambhir
Request Type	None
Components	None
Priority	Medium

More fields Labels, Urgency, Impact, Affected services, Pendin...

Created 1 minute ago
Updated 1 minute ago

JIRA Board Link:
https://developerpiyushgambhir.atlassian.net/jira/servicedesk/projects/BPPS/boards/3?assignee=unassigned&assignee=712020%3A4dd22410-6521-4470-9375-56f4ef2c7461&atlOrigin=eyJpIjoiOTY0OTJlYzE1YTY5NDA1YWJjODNhMWJiNzI1OTdmZTAiLCJwIjoiaiJ9

Annexure 2

<Software Engineering and Project Management > (CSL229)

Project Report

Faculty name

Student name

Roll No.:

Semester:

Group:

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2020-21

Table of Contents