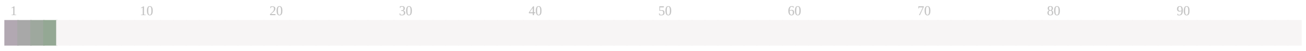
The Report is Generated by DrillBit Plagiarism Detection Software

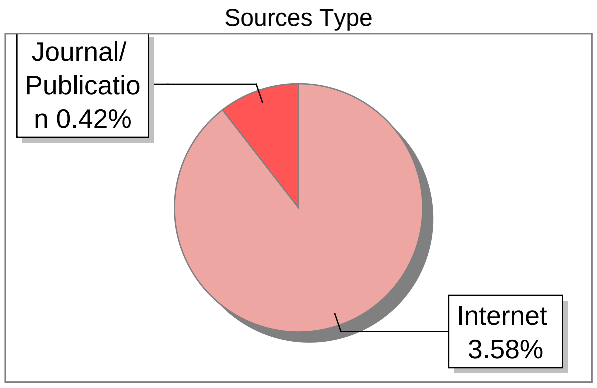
Submission Information

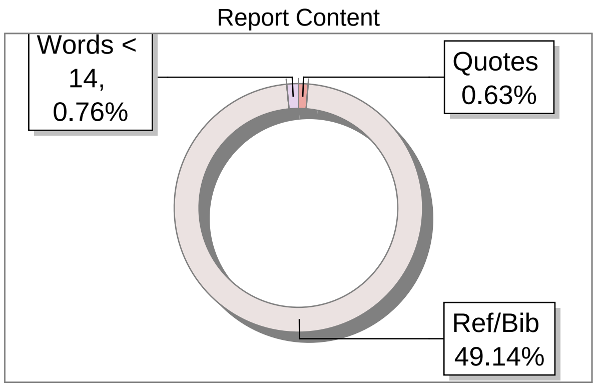
Author Name	ArhathRaj Shetty M
Title	Task
Paper/Submission ID	3335288
Submitted by	nnm23is030@nmamit.in
Submission Date	2025-02-16 08:15:25
Total Pages, Total Words	11, 2375
Document type	Project Work

Result Information

Similarity **4 %**







Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Source: Excluded < 14 Words	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

4	**5**	**A**
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	www.linkedin.com	2	Internet Data
2	dovepress.com	1	Internet Data
3	bmcbioinformatics.biomedcentral.com	<1	Internet Data
4	uir.unisa.ac.za	<1	Publication
5	www.synopsys.com	<1	Internet Data

PROJECT REPORT

ON

Analysis of OLA Cab's On-Demand Ride-Hailing Service: A Comparative Study of Different SDLC Models

"A Comparative Study of Different Models in Relation to OLA Cab's Ride-Hailing and Mobility Services"

Submitted to

NMAM INSTITUTE OF TECHNOLOGY, NITTE

(Off-Campus Centre, Nitte Deemed to be University, Nitte - 574 110, Karnataka, India)

In partial fulfilment of the requirements for the award of the

Degree of Bachelor of Technology

In

INFORMATION SCIENCE AND ENGINEERING

By

ArhathRaj Shetty M NNM23IS022

Under the guidance of

Dr. Jason Elroy Martis, Associate Professor,

Department of Information Science and Technology,

NMAM Institute of Technology. Nitte Karnataka, India

Comparative Analysis of SDLC Models and Requirements Engineering for OLA Cab System

1. Introduction

1.1 Background

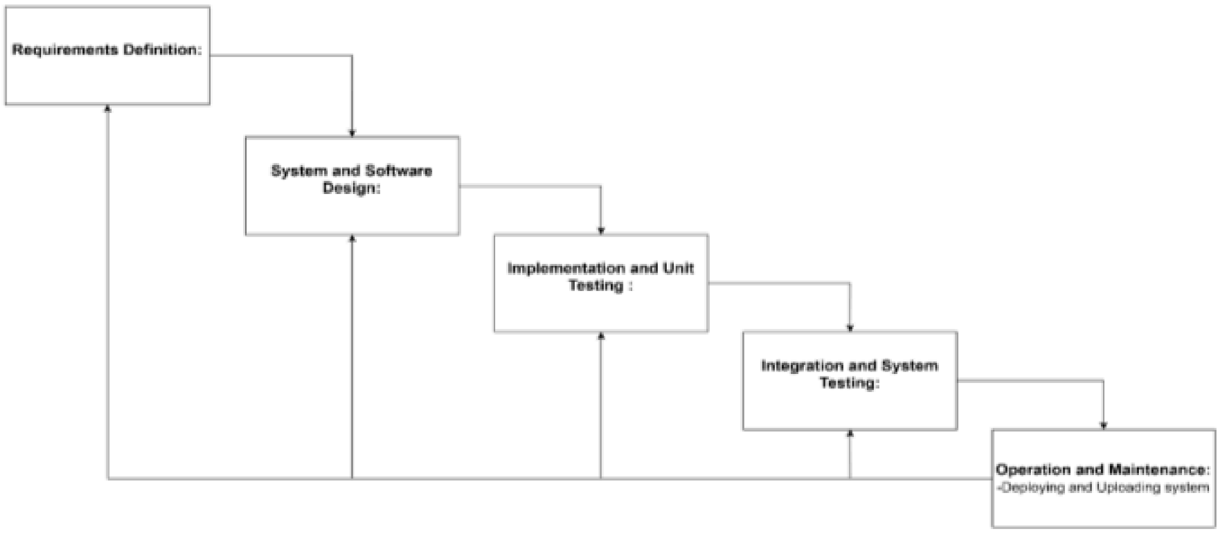
OLA Cab is an advanced ride-hailing platform that transforms urban transportation by effectively linking passengers with drivers via an intuitive mobile application. This innovative system offers real-time vehicle tracking while seamlessly incorporating sophisticated payment processing features, strong user authentication protocols, and adaptive pricing algorithms to improve user experience and operational effectiveness. Given the complex and evolving nature of OLA's needs, this report presents a thorough comparison of three leading Software Development Lifecycle (SDLC) models: the Waterfall, Spiral, and Incremental Development models. Each model is carefully assessed for its appropriateness in addressing OLA's functional and non-functional requirements, its capabilities in risk and change management, and its compliance with time and cost constraints. Additionally, the report explores the nuances of requirements engineering by creating a comprehensive requirements document and outlining a strategic approach for validating these requirements, identifying potential challenges, and proposing solutions to ensure the successful deployment of OLA's software system. This extensive analysis aims to furnish valuable insights into the selection of the most suitable SDLC model and requirements management strategy for OLA Cab, thereby ensuring the platform's ongoing success and scalability in the competitive ride-hailing sector..

1.2 Objectives

- To assess the appropriateness of various SDLC models for OLA Cab.
- To create a comprehensive requirements document.
- To establish a framework for requirements validation.

2. Comparative Analysis of SDLC Models

2.1 Waterfall Model


Requirements Definition:
System and Software Design:
Implementation and Unit Testing :
Integration and System Testing:
Operation and Maintenance:
-Deploying and Uploading system

2.1.1 Overview

The Waterfall Model represents a linear, sequential approach to the Software Development Life Cycle (SDLC), in which each phase is completed consecutively. This model is most effective for projects with clearly defined requirements and minimal anticipated changes.

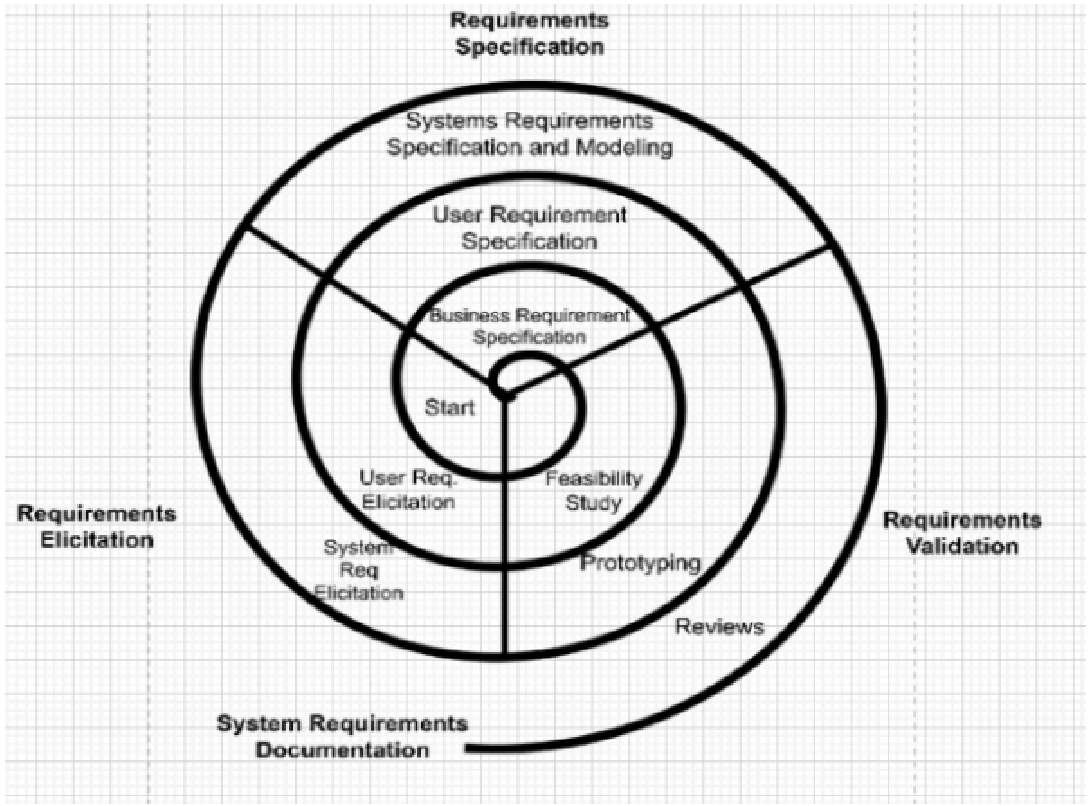
2.1.2 Suitability for OLA

- **Functional Requirements:** Unsuitable. OLA's requirements change frequently (e.g., surge pricing, new safety features). The inflexible phases of Waterfall limit its adaptability.
- **Non-Functional Requirements:** Restricted flexibility. Updates for scalability and security would necessitate expensive rework in later stages.
- **Risk/Change Management:** Low adaptability. Changes in regulations (e.g., data privacy legislation) or user expectations cannot be seamlessly integrated.
- **Time/Cost Constraints:** Predictable schedules but a significant risk of delays due to changes in requirements. Initial expenditures are lower; however, the costs of rework are substantial.
-

2.1.3 Example

A delay in the integration of a new payment gateway would necessitate revisiting the design and testing phases, resulting in increased costs.

2.2 Spiral Model


Requirements Specification
Systems Requirements Specification and Modeling
User Requirement Specification
Business Requirement Specification
Start
User Req. Elicitation
Feasibility Study
Requirements Elicitation
Requirements Validation
System Req Elicitation
Prototyping
Reviews
System Requirements Documentation

2.2.1 Overview

The Spiral Model is an iterative Software Development Life Cycle (SDLC) methodology that merges the phases of the Waterfall Model with a focus on risk assessment. This model is particularly advantageous for extensive and intricate projects that involve considerable risks.

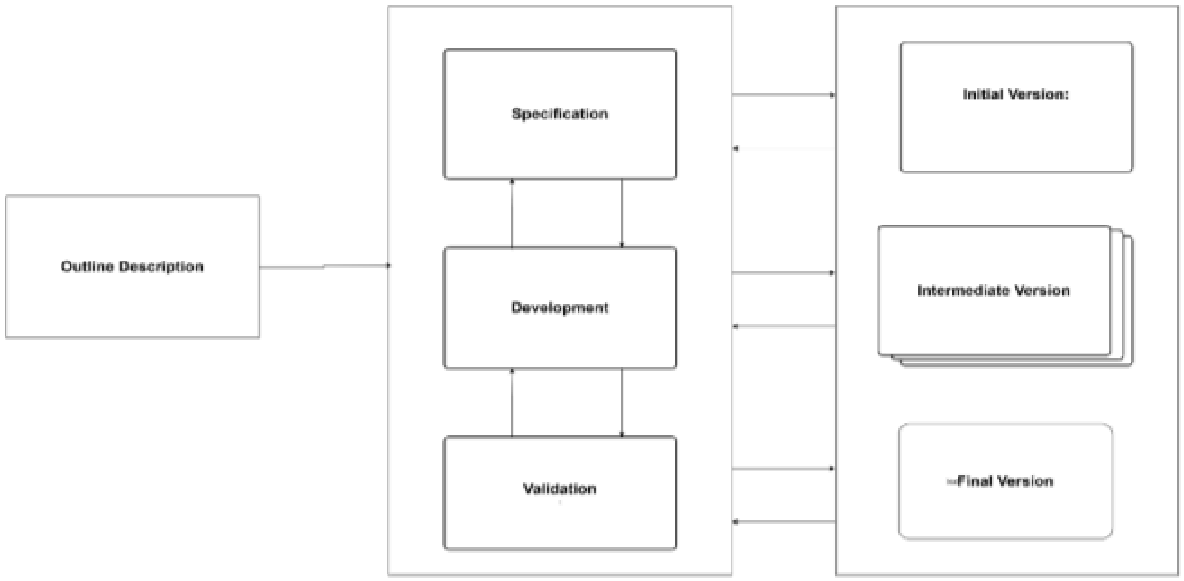
2.2.2 Suitability for OLA

- **Functional Requirements:** Highly adaptable. The iterative nature of risk analysis enables the prioritization of essential features (e.g., emergency response tools) at an early stage.
- **Non-Functional Requirements:** Excellent. Risk-driven iterations effectively address scalability during peak demands and strengthen security measures (e.g., data encryption).
- **Risk/Change Management:** Robust. Regular risk evaluations help in mitigating potential issues such as driver fraud or system outages.
- **Time/Cost Constraints:** Higher initial expenses can lead to long-term cost savings. Prototyping minimizes rework, while phased deliveries are aligned with investor expectations.
-

2.2.3 Example

A prototype for real-time tracking could serve to validate performance requirements prior to full-scale development.

2.3 Incremental Development



2.3.1 Overview

Incremental Development is an iterative SDLC approach wherein the system is built in increments. Each increment enhances the functionality of the preceding version.

2.3.2 Suitability for OLA

- **Functional Requirements:** Ideal. Features such as ride-sharing or loyalty programs can be developed incrementally, with user feedback steering refinements.
- **Non-Functional Requirements:** Moderate. Scalability can be managed incrementally; however, security updates may necessitate the integration of multiple components.

- **Risk/Change Management:** Flexible. Smaller increments lessen the impact of requirement changes (e.g., adjusting surge pricing algorithms).
- **Time/Cost Constraints:** Accelerated delivery of core features (e.g., booking system) with lower initial investment. Long-term costs can be kept manageable through iterative enhancements.
-

2.3.3 Example

Initial release of a basic application, followed by the addition of premium features such as luxury car bookings.

3. Requirements Document (Simplified)

3.1 Functional Requirements

3.1.1 User Registration

- Secure sign-up via email or phone, including one-time password (OTP) verification.
- Integration of social media login options (Facebook, Google).

3.1.2 Ride Booking

- Real-time display of availability.
- Fare estimation and customization options for rides (e.g., AC/non-AC).
- Multiple payment methods supported (UPI, credit/debit cards).

3.1.3 Driver Assignment

- Algorithm-driven matching based on location and ratings.
- Real-time tracking capabilities for the driver's location.

3.1.4 Customer Support

- In-app chat functionality for addressing complaints and inquiries.
- 24/7 emergency contact availability.

3.2 Non-Functional Requirements

3.2.1 Scalability

- Capability to support over 100,000 concurrent users without compromising performance.
- Provision for horizontal scaling.

3.2.2 Security

- Implementation of end-to-end encryption for safeguarding user data.
- Compliance with GDPR and relevant data protection regulations.

3.2.3 Availability

- Infrastructure to maintain 99.9% uptime with failover strategies.

- Regular health assessments of the system.

3.2.4 Performance

- Response time for ride requests to be under 2 seconds.
- Optimized data processing and storage capabilities.

4. Requirements Validation Strategy

4.1 Strategy

4.1.1 Prototyping

- Create a minimum viable product (MVP) for user evaluation (e.g., fundamental ride-booking feature).
- Utilize tools such as Figma or Adobe XD for UI/UX design prototyping.

4.1.2 Reviews

- Conduct review sessions based on sprints with stakeholders to validate project increments.
- Employ Jira or Trello for progress tracking and feedback collection.

4.1.3 Testing

- **Functional:** Automated regression testing focused on payment processing.
- **Non-Functional:** Conduct load testing for scalability and penetration testing for security assurance.

4.1.4 Feedback Loops

- Implement in-app surveys and rating systems for drivers/passengers to enhance requirement gathering.
- Conduct regular meetings with stakeholders to review progress and solicit feedback.

4.2 Challenges

4.2.1 Scope Creep

- Finding equilibrium between user requests and essential functionalities (e.g., preventing unnecessary complexity in the booking interface).
- Implementing change control protocols to address scope creep.

4.2.2 Non-Functional Validation

- Simulating peak demand for scalability tests may necessitate expensive infrastructure investments.
- Utilizing cloud-based testing solutions to mitigate costs.

4.2.3 Regulatory Changes

- Adapting to emerging legal requirements (e.g., driver background checks) during the development phase.
- Establishing a legal compliance team to oversee and integrate regulatory updates.

5. Conclusion

For OLA Cab, the **Incremental Development** model is the most appropriate choice due to its adaptability in accommodating changing requirements and its capacity to deliver core functionalities rapidly. The **Spiral Model** is advisable for components with significant risk factors (e.g., payment security) in order to address potential issues proactively. While the Waterfall Model is less favorable, it may still be suitable for stable, low-risk modules (e.g., static content delivery). A hybrid approach that integrates elements of Incremental and Spiral methodologies would enhance both agility and risk management.
Validation should prioritize ongoing user feedback and automated testing to ensure conformity with both functional and non-functional specifications.

6. References

- Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.
- Royce, W. W. (1970). Managing the Development of Large Software Systems. In *Proceedings of IEEE WESCON*.
- Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Software*, 5(2), 61–72.
- Ambler, S. W. (2004). *The Object Primer: Agile Model-Driven Development with UML 2.0* (3rd ed.). Cambridge University Press.
- Krutchen, P. (1995). The 4+1 View Model of Architecture. *IEEE Software*, 12(6), 42–50.

7. Appendices

7.1 Appendix A: Comprehensive Comparison Table

Criteria	**Waterfall Model**	**Spiral Model**	**Incremental Development**
Functional Requirements	Poor fit for evolving requirements	High adaptability	Ideal for incremental feature addition
Non-Functional Requirements	Limited flexibility	Excellent for addressing scalability and security	Moderate, can be addressed incrementally
Risk Management	Low tolerance for changes	Strong risk assessment and mitigation	Flexible, reduces impact of changes
Time/Cost Constraints	Predictable timelines, high rework costs	Higher upfront costs, long-term savings	Faster delivery, manageable long-term costs

7.2 Appendix B: Additional References

- IEEE Software Engineering Standards
- OMG Model-Driven Architecture
- Agile Modeling

8. In-Depth Analysis of Each Model

8.1 In-Depth Analysis of the Waterfall Model

8.1.1 Advantages

- **Predictability:** The Waterfall Model presents a transparent and predictable project timeline.
- **Documentation:** Comprehensive documentation is produced, beneficial for future reference and upkeep.
- **Simplicity:** The model is clear-cut and easy to comprehend.

8.1.2 Disadvantages

- **Rigidity:** Once a phase concludes, modifying it becomes challenging.
- **Risk of Late Modifications:** Changing requirements late in the project can incur substantial costs.
- **Lack of Early User Feedback:** Users do not review the system until project completion, potentially resulting in dissatisfaction.

8.1.3 Case Study: OLA Cab

- **Scenario:** OLA aims to introduce a new feature for real-time ride tracking.
- **Waterfall Approach:** A complete redesign and retesting of the entire system would be necessary, resulting in considerable delays and expenses.
- **Outcome:** The Waterfall Model proves unsuitable for this scenario due to its lack of flexibility.

8.2 In-Depth Analysis of the Spiral Model

8.2.1 Advantages

- **Risk Management:** The Spiral Model facilitates early detection and mitigation of risks.
- **Flexibility:** The model can adapt to shifting requirements.
- **User Engagement:** Users can contribute feedback early in the process, resulting in an improved final product.

8.2.2 Disadvantages

- **Complexity:** The model is more intricate than the Waterfall Model, necessitating more experienced project managers.
- **Costs:** The iterative nature may lead to increased expenditures.
- **Time Consumption:** The process may take longer due to multiple cycles.

8.2.3 Case Study: OLA Cab

- **Scenario:** OLA requires the implementation of a new payment gateway.
- **Spiral Approach:** A prototype is developed and assessed with user input. Feedback is integrated, and the feature is refined prior to full deployment.
- **Outcome:** The Spiral Model is appropriate for this scenario, enabling early risk evaluation and user feedback.

8.3 In-Depth Analysis of Incremental Development

8.3.1 Advantages

- **Flexibility:** The model is remarkably adaptable to evolving requirements.
- **User Feedback:** Users can provide insights on each increment, enhancing the final output.
- **Quicker Delivery:** Core functionalities can be launched rapidly, with additional features introduced progressively.

8.3.2 Disadvantages

- **Complexity:** Managing the model can become complicated, particularly with numerous increments.
- **Integration Challenges:** Incorporating new increments can pose difficulties and potentially introduce bugs.
- **Scope Creep:** There is a danger of scope creep if not properly overseen.

8.3.3 Case Study: OLA Cab

- **Scenario:** OLA intends to implement a new ride-sharing feature.
- **Incremental Approach:** The fundamental ride-booking feature is developed initially. Later increments will introduce ride-sharing functionality.
- **Outcome:** The Incremental Development model fits this scenario well, allowing for iterative advancement and user input.

9. Comprehensive Requirements Engineering Process

9.1 Requirements Elicitation

9.1.1 Techniques

- **Interviews:** Engage stakeholders through interviews to collect requirements.
- **Surveys:** Deploy surveys to gather feedback from a wider audience.
- **Observation:** Observe users in their operational environment to gain insights into their needs.

9.1.2 Tools

- **Jira:** Utilized for requirement tracking and feedback management.
- **Trello:** Employed for overseeing the requirements backlog.
- **Figma:** Applied for the development of wireframes and prototypes.

9.2 Requirements Specification

9.2.1 User Requirements

- **High-Level Requirements:** Articulate the general goals and objectives of the system.
- **Detailed Requirements:** Decompose high-level requirements into specific, actionable tasks.

9.2.2 System Requirements

- **Functional Requirements:** Specify the precise functions the system must execute.
- **Non-Functional Requirements:** Outline the system's performance, security, and other non-functional characteristics.

9.3 Requirements Validation

9.3.1 Techniques

- **Prototyping:** Construct prototypes to verify requirements with users.
- **Reviews:** Facilitate reviews with stakeholders to validate compliance with requirements.
- **Testing:** Execute functional and non-functional testing to confirm requirements.

9.3.2 Tools

- **Automated Testing Tools:** For validating functional and non-functional aspects.
- **Feedback Tools:** For obtaining user feedback on prototypes.

9.4 Requirements Management

9.4.1 Techniques

- **Change Control:** Establish a change control process to oversee requirement modifications.
- **Version Control:** Implement version control systems to document changes in requirements.
- **Documentation:** Preserve exhaustive documentation of requirements and modifications.

9.4.2 Tools

- **Version Control Systems:** Such as Git for monitoring changes.
- **Documentation Tools:** Such as Confluence for upholding requirements documentation.

10. Case Studies and Real-World Examples

10.1 Case Study: OLA Cab

10.1.1 Scenario: Real-Time Ride Tracking

- **Objective:** Implement real-time ride tracking for passengers.
- **Approach:** Employ the Incremental Development model to develop the feature in phases.
- **Outcome:** Core ride-booking functionality is delivered initially, followed by real-time tracking in a subsequent increment.

10.1.2 Scenario: New Payment Gateway

- **Objective:** Integrate a new payment gateway.
- **Approach:** Utilize the Spiral Model to create a prototype, gather feedback, and enhance the feature.
- **Outcome:** The new payment gateway is successfully integrated with minimized risk.

10.2 Case Study: Uber

10.2.1 Scenario: Surge Pricing

- **Objective:** Introduce surge pricing to manage demand.
- **Approach:** Apply the Incremental Development model to implement surge pricing incrementally.
- **Outcome:** Surge pricing is effectively implemented and refined based on user feedback.

10.2.2 Scenario: Driver Background Checks

- **Objective:** Implement driver background checks to guarantee safety.
- **Approach:** Utilize the Spiral Model to develop a prototype, gather feedback, and enhance the feature.
- **Outcome:** Driver background checks are successfully executed with minimal risk.

11. Conclusion and Recommendations

11.1 Conclusion

For OLA Cab, the **Incremental Development** model is most appropriate due to its adaptability in accommodating evolving requirements and timely delivery of core features. The **Spiral Model** is advised for high-risk components (e.g., payment security) to mitigate risks at an early stage. Although the Waterfall Model may not be optimal, it could be applied to stable, low-risk modules (e.g., static content delivery). A hybrid methodology combining Incremental and Spiral elements would enhance agility and risk management.

11.2 Recommendations

- **Adopt Incremental Development:** For essential features and swift delivery.
- **Utilize Spiral Model for High-Risk Components:** For features carrying considerable risks, such as payment security.
- **Implement Change Control Processes:** To manage scope creep and guarantee requirements are fulfilled.
- **Conduct Regular Stakeholder Meetings:** To solicit feedback and ensure concordance with user needs.

12. Future Work

- **Continuous Improvement:** Regularly assess and refine the development process based on feedback and emerging requirements.
- **Adapt to Emerging Technologies:** Remain aware of new technologies and integrate them into the development framework.
- **Enhance User Experience:** Consistently improve the user interface and experience informed by user feedback.
