Software Engineering Journal

Name: Rudra Mehta

Roll No: IT077

EXPERIMENT – 1

Aim: To study phases of SDLC and applicability of different SDLC Models

(1) Open an appropriate software engineering guide and study the software development life cycle and related topics.

The Software Development Life Cycle (SDLC) is a structured process that outlines the key stages or phases for building high-quality software. It provides a systematic framework for planning, creating, testing, and deploying a software system, ensuring that the final product meets customer expectations.

The typical phases of the SDLC are:

1. **Requirement Analysis & Planning:** This is the foundational phase. Project goals are determined, and all requirements from the client and stakeholders are gathered. This stage involves feasibility studies and creating a project plan.
2. **Design:** The system and software design are prepared based on the requirements gathered in the first phase. This phase produces a "blueprint" of the software, detailing the system architecture, database schema, user interfaces, and component specifications.
3. **Implementation / Coding:** This is the phase where the actual coding happens. Developers use the design documents to write the source code for the software components using a suitable programming language.
4. **Testing:** The developed software is thoroughly tested against the requirements to identify and fix any defects or bugs. This phase involves various types of testing, such as unit testing,

integration testing, system testing, and acceptance testing, to ensure the product is high-quality and works correctly.
5. **Deployment:** Once the software is tested and approved, it is released and deployed to the production environment where customers can start using it.
6. **Maintenance:** After deployment, the software enters the maintenance phase. This involves fixing any issues that arise in the live environment, making enhancements, and updating the software to adapt to new needs or technologies.

---

(2) Study the need of the software engineering.

Software engineering is necessary because it transforms the chaotic and unpredictable nature of programming into a manageable and predictable process. Without a structured engineering approach, software projects often fail due to being over budget, behind schedule, or not meeting user needs.

The key reasons we need software engineering are:

- **To Manage Complexity:** Modern software can be incredibly complex. Software engineering provides principles and techniques to break down large, complex problems into smaller, manageable parts.
- **To Ensure Quality:** It establishes processes for quality assurance, verification, and validation, leading to more reliable, secure, and efficient software.
- **To Improve Predictability:** By using established models and planning techniques, software engineering helps in accurately estimating project costs, timelines, and required resources.
- **To Facilitate Teamwork:** It provides a common framework and documentation standards that allow large teams of

developers, testers, and project managers to collaborate effectively.

- **To Simplify Maintenance:** A well-engineered software system, with clear design and documentation, is significantly easier and less costly to update, fix, and enhance over its lifespan.

(3) Study the coverage of topics such as life cycle models, their applicability and their Comparisons.

Several SDLC models offer different approaches to software development. Choosing the right one is crucial for a project's success and depends on factors like project size, complexity, and how well the requirements are understood.

The Waterfall Model

The Waterfall model is the most traditional and straightforward SDLC model. It follows a linear, sequential approach where each phase must be fully completed before the next one can begin, just like a waterfall flowing downwards.

- **Applicability:** It's best suited for small, simple projects where the requirements are crystal clear, well-documented, and not expected to change at all.
- **Comparison:**
  - **Pros:** It's simple to understand and manage. The rigid structure makes planning and tracking progress easy.
  - **Cons:** It's very inflexible. A change in requirements can be disastrous. The working software is only delivered at the very end, and testing happens late in the cycle, making bug fixes costly.

The V-Model (Verification & Validation Model)

The V-Model is an extension of the Waterfall model. Its key feature is that a corresponding testing phase is planned for *every* stage of development. For example, unit tests are planned during the coding phase, and system tests are planned during the architectural design phase. It follows a V-shape, where the development process goes down one side and the testing process goes up the other.

- **Applicability:** It's ideal for projects where failure is not an option, such as medical software, aviation control systems, and automotive safety software. The emphasis on early test planning ensures high reliability.
- **Comparison:**
  - **Pros:** It's highly disciplined, and the proactive test planning helps find bugs early in the development cycle.
  - **Cons:** Like the Waterfall model, it is very rigid and not flexible enough to handle changes in requirements.

The Iterative Model

The Iterative model develops a system through repeated cycles (iterations). In the first iteration, a simple, basic version of the software is created. In each subsequent iteration, new features are added and refined until the complete system is built.

- **Applicability:** This model is excellent for large and complex projects where the complete requirements aren't known at the start. It allows for a functional version of the product to be available early on.
- **Comparison:**
  - **Pros:** It's more flexible than the Waterfall model. It generates working software quickly, and user feedback can be incorporated in later iterations.

    - **Cons:** It requires more robust management. The full system architecture must be well-planned from the beginning to avoid issues later.

The Spiral Model

The Spiral model is a sophisticated model that focuses heavily on risk analysis. The project moves through four phases in a spiral manner: Planning, Risk Analysis, Engineering (development), and Evaluation. This cycle is repeated with increasing complexity until the project is finished.

- **Applicability:** It's designed for very large, expensive, and high-risk projects. It's also great for projects involving new or unclear technology where significant research and development are needed.
- **Comparison:**
    - **Pros:** Its key strength is advanced risk management. It's flexible and accommodates changes well.
    - **Cons:** The model is very complex and can be expensive due to the expertise needed for risk analysis. It's overkill for smaller, low-risk projects.

The Agile Model

Agile is not a single model but a philosophy that prioritizes flexibility and collaboration. It involves developing software in short, incremental cycles called "sprints." The focus is on delivering working software quickly, adapting to changing requirements, and getting continuous feedback from the customer.

- **Applicability:** It's perfect for dynamic projects with evolving requirements, such as mobile apps and startups. It thrives in

environments where the market changes quickly and a fast turnaround is needed.

- **Comparison:**
  - **Pros:** It offers maximum flexibility to handle changes. It ensures a high degree of customer satisfaction through continuous involvement.
  - **Cons:** It can be less predictable in terms of final cost and timeline. It requires a highly collaborative team and a committed customer.

(4) List out various type of software for each SDLC Model.

Here are examples of software types that are well-suited for each specific SDLC model:

- **Waterfall Model:**
  - **Customer Relationship Management (CRM) System:** A simple, well-defined CRM for a small business with fixed requirements.
  - **Human Resource Management System (HRMS):** A system for a small company where all modules (payroll, attendance) are known in advance.
- **V-Model:**
  - **Patient Monitoring Software:** Medical software for hospitals where accuracy, reliability, and safety are non-negotiable.
  - **Automotive ABS (Anti-lock Braking System) Software:** A safety-critical system in a car that must be rigorously tested against its design.
- **Iterative Model:**
  - **E-commerce Website (like Amazon):** The core shopping cart functionality can be developed first, and

features like user reviews, recommendation engines, and wish lists can be added in later iterations.
    - **Large Content Management System (CMS):** Build basic article publishing first, then add user roles, analytics, and multimedia support later.
- **Spiral Model:**
    - **New Operating System Development:** A highly complex project with many unknown risks that need to be addressed in phases.
    - **Advanced Game Engine:** Developing a new, cutting-edge game engine involves significant R&D and risk.
- **Agile Model:**
    - **Mobile App Development (like Instagram or Uber):** The market changes quickly, and user feedback is vital. New features are developed and released in short cycles (sprints).
    - **Web-based Startup Projects:** Startups often pivot and change features based on early user adoption and market response, which requires maximum flexibility.

EXPERIMENT – 2

Aim: To perform requirement engineering tasks.

(1) Identify the different requirement engineering tasks.

Requirement Engineering is the process of defining, documenting, and maintaining the requirements for a software system. The primary tasks involved are:

- **Feasibility Study:** Determining if the proposed project is technically, financially, and operationally viable. It answers the question, "Is this project worth doing?"
- **Requirement Elicitation:** This is the process of gathering requirements from stakeholders (like customers, users, and experts). Common techniques include interviews, surveys, workshops, and observation.
- **Requirement Analysis & Negotiation:** Raw requirements are analyzed to resolve conflicts, remove ambiguities, and understand them in detail. If stakeholders have conflicting needs, negotiation is used to find a compromise.
- **Requirement Specification:** The analyzed requirements are formally documented in a **Software Requirement Specification (SRS)** document. This document serves as a contract between the development team and the client.
- **Requirement Validation:** This task involves checking the SRS document to ensure that it's correct, complete, and consistent. It confirms that the documented requirements truly reflect the stakeholders' needs.
- **Requirement Management:** This is an ongoing process of managing changes to the requirements throughout the project's lifecycle. It ensures that any changes are properly tracked, evaluated, and implemented.

EXPERIMENT - 3

Aim: To perform the system analysis: Requirement analysis, SRS

1. Introduction

- Blinkit is Quick commerce platform , Blinkit is an on-demand grocery and essentials delivery application that promises delivery within 10-20 minutes.
- This document serves as a guide for developers, testers, project managers, and stakeholders to understand the system's functionality, constraints, and specifications.
- Blinkit operates as a hyperlocal marketplace connecting consumers with local vendors through a network of dark stores and delivery partners.
- The platform covers 50 cities across india and manage more than 7000 product across multiple categories including groceries , electronics , personal care .
- The system consists of three main applications: Customer mobile application, Delivery partner application, and Admin dashboard. The platform utilize advanced technologies including real time inventory sync ,route optimization ,micro-services architecture.

2. Description

2.1 Product Perspective :

Blinkit operates on an inventory-based model utilizing a network of dark stores for ultra-fast delivery. The platform integrates with multiple systems including payment gateways, mapping services, notification systems, and third-party logistics providers. The architecture follows microservices design patterns enabling independent scaling and deployment.

2.2 Product Functions:

- User registration and authentication with OTP verification
- Real-time product browsing with category-based organization
- Shopping cart management with real-time pricing updates• Order placement and processing with 10-minute delivery promise
- Real-time order tracking with GPS integration
- Multiple payment options including UPI, cards, and cash on delivery
- Inventory management across multiple dark stores
- Delivery partner assignment and route optimization
- Customer support and feedback system

2.3 General Constraints:

- Delivery radius limited to 2-3 km from dark stores
- Service availability restricted to operational hours and locations
- Minimum order values and delivery charges based on location
- GST compliance requirements for sellers
- Real-time inventory synchronization dependencies
- Network connectivity requirements for real-time features

3. Specific Requirements

3.1 Functional Requirements (Input - Output Format)

3.1.1 User Management System

- User Registration
    - Input: Phone number and OTP
    - Output: User account created after OTP verification
    - This allows users to register using their mobile number with a one-time password for quick and secure sign-up.

- Social Media Login
  - Input: Google or Facebook credentials
  - Output: User account logged in or created
  - Enables users to sign in instantly using their social media accounts without creating a new password.
- Profile Creation
  - Input: Name, email, phone number, delivery addresses
  - Output: User profile saved
  - Stores personal and delivery information to customize the shopping experience.
- Email Verification
  - Input: Email address
  - Output: Email verified successfully
  - Ensures user identity and improves account security by confirming the email address.
- Password Reset
  - Input: Email or phone number
  - Output: OTP or reset link sent, new password set
  - Helps users reset forgotten passwords through a secure OTP or email link.
- Manage Addresses
  - Input: New or updated address
  - Output: Address saved or updated in profile
  - Users can add, update, or delete multiple delivery addresses as per their preference.
- Update Personal Info
  - Input: New personal details
  - Output: Profile updated
  - Allows users to modify personal details like name, phone, or email anytime
- View Order History
  - Input: User request
  - Output: Display of previous orders
  - Displays a list of previously placed orders for easy reordering or tracking.

- Wishlist / Favorites
    - Input: Product added to wishlist
    - Output: Item saved in wishlist
    - Lets users bookmark favorite products for quicker access in the future.
- Delete Account
    - Input: User request
    - Output: Account and data deleted○ Users can delete their account and associated data to ensure privacy and control.

3.1.2 Product Management System

- Search Products by Category
    - Input: Search keyword or selected category
    - Output: List of matching products
    - Users can easily search for products using filters or keywords based on categories.
- Show Real-Time Inventory
    - Input: User location
    - Output: Products available in nearest store shown
    - Shows only the items available at the user’s nearest dark store for faster delivery.
- View Product Details
    - Input: Product ID
    - Output: Image, description, and price shown
    - Displays full product details including images, specifications, and pricing.
- Barcode Scanning
    - Input: Product barcode via camera
    - Output: Product details displayed
    - Allows users or sellers to quickly find or add products by scanning barcodes.
- Voice Search
    - Input: Voice command

    - Output: Related products shown
    - Users can search products hands-free using natural voice input.
- Product Recommendations
    - Input: User browsing and purchase history
    - Output: Suggested products
    - Provides personalized product suggestions based on user behavior and trends.

Inventory Management

- Update Stock Levels
    - Input: Inventory changes from store
    - Output: Real-time stock updates
    - Ensures the product availability reflects actual stock at all times.
- Auto Remove Out-of-Stock Items
    - Input: Product out of stock
    - Output: Product hidden from catalog
    - Automatically hides unavailable products from customer view to avoid order issues.
- Location-Based Availability
    - Input: Customer location
    - Output: Only available items shown
    - Shows products only if they are in stock at the customer’s local fulfillment center.
- Suggest Alternatives
    - Input: Unavailable product
    - Output: Substitute products displayed
    - Suggests similar or related products when the desired item is not in stock.

3.1.3 Order Management System

Order Processing

- Add to Cart & Modify Quantity
    - Input: Product and quantity
    - Output: Updated cart
    - Allows customers to add products and adjust quantities before checkout.
- Price Calculation
    - Input: Cart items, taxes, and delivery fee
    - Output: Final price shown
    - Calculates the total amount including tax and delivery in real-time.
- Schedule Order
    - Input: Desired delivery time
    - Output: Order scheduled
    - Customers can choose a preferred delivery slot as per their convenience.
- Modify Order
    - Input: Change request before packing
    - Output: Order updated
    - Lets customers make changes to the order before it is packed.
- Bulk Order
    - Input: Multiple items
    - Output: Single bulk order created
    - Allows placing large orders easily with grouped product processing.

Order Fulfillment

- Assign Nearest Store
    - Input: Customer location, inventory status
    - Output: Store selected for order
    - Automatically selects the closest dark store that has the ordered items.
- Pick and Pack Order

    - Input: Order details
    - Output: Packed order within 2.5 minutes
    - Ensures ultra-fast picking and packing for express delivery.
- Assign Delivery Partner
    - Input: Location, partner availability
    - Output: Partner assigned
    - Uses AI to assign the most suitable delivery partner based on distance and time.
- Track Order Status
    - Input: Order updates
    - Output: Real-time status shown to user
    - Shows order progress (e.g., packed, dispatched) in real-time to the user.

3.1.4 Payment System

- Choose Payment Method
    - Input: Payment option (UPI, card, wallet, COD)
    - Output: Transaction completed
    - Supports multiple payment options for user convenience.
- UPI Payment
    - Input: UPI ID
    - Output: Instant payment confirmation
    - Processes real-time UPI transactions securely and instantly.
- Card Payment
    - Input: Card details
    - Output: Payment processed securely
    - Accepts debit/credit cards with tokenization for safety.
- Wallet Payment
    - Input: Digital wallet selection
    - Output: Payment deducted from wallet
    - Supports wallet-based payments like Paytm or MobiKwik.

- Cash on Delivery
    - Input: Selected as payment mode
    - Output: COD confirmed with change info
    - Offers cash payment with proper handling of change by delivery agents.
- Subscribe for Premium
    - Input: Subscription plan selection and payment
    - Output: Premium service activated
    - Enables users to subscribe to faster delivery or other benefits.

3.1.5 Delivery Management System

Logistics Optimization

- Optimize Route
    - Input: Delivery location, traffic data
    - Output: Best route planned
    - Ensures timely deliveries through smart GPS-based route planning.
- Estimate Delivery Time
    - Input: Current location and traffic
    - Output: Estimated delivery time○ Provides users with accurate delivery time predictions.
- Multi-Drop Planning
    - Input: Multiple delivery points
    - Output: Efficient route created
    - Combines nearby orders to reduce travel time and increase efficiency.
- Monitor Partner Performance
    - Input: Delivery data
    - Output: Performance report
    - Tracks delivery partner speed, success rate, and user feedback.

Delivery Tracking

- Track Order Live
  - Input: Order ID
  - Output: GPS-based tracking shown
  - Shows users the real-time location of their delivery on a map.
- Send Notifications
  - Input: Order status changes
  - Output: Push/email/SMS alerts
  - Notifies users at every stage of the order process.
- Delivery Confirmation
  - Input: Photo of delivery
  - Output: Proof of delivery stored
  - Confirms that the order was delivered with image verification.
- Collect Feedback
  - Input: Customer review or rating
  - Output: Feedback saved
  - Allows customers to share their delivery experience for quality improvement.

3.1.6 Seller Management System

Seller Onboarding

- Register Seller
  - Input: Documents (GST, PAN, etc.)
  - Output: Seller account created
  - Onboards sellers securely by verifying all required documents.
- Brand Authorization
  - Input: Authorization letter
  - Output: Brand linked to seller

    - Verifies permission to sell branded products under specific sellers.
- Upload Product Catalog
    - Input: Product list and details
    - Output: Products published
    - Lets sellers manage and update their product offerings.
- Set Commission
    - Input: Percentage rates
    - Output: Commission plan set
    - Configures how much commission is taken per transaction.

Seller Operations

- Update Inventory
    - Input: Stock changes
    - Output: Real-time updates to dark stores
    - Synchronizes stock levels between seller and store systems.
- Analytics Reporting
    - Input: Sales data
    - Output: Reports generated
    - Provides insights into seller performance, revenue, and trends.
- Payment Settlement
    - Input: Completed transactions
    - Output: Commission and payments calculated
    - Manages and tracks earnings, commission cuts, and payouts.

3.1 Non-Functional Requirements

3.1.1 Performance Requirements

- Response Time: API responses within 200ms for core functions
- Delivery Promise: 10-20 minute delivery SLA in operational areas
- Throughput: Support for 1 million+ orders per day across all locations Concurrent Users: Handle 100,000+ simultaneous active users
- Order Processing: Complete order picking within 2.5 minutes

3.2.2 Scalability Requirements

- Horizontal Scaling: System must support horizontal scaling to accommodate increased user load without degradation of performance.
- Microservices Architecture: Adopt a microservices approach to enable independent scaling of individual services based on distinct resource demands.
- Database Sharding: Implement sharding to ensure data is distributed geographically, improving both performance and data locality for global users.
- CDN Integration: Utilize a Content Delivery Network (CDN) to optimize and accelerate the delivery of static content to end-users worldwide.
- Auto-Scaling: Enable automated scaling of infrastructure resources in response to real-time demand patterns, maintaining efficiency and cost-effectiveness.

3.2.3 Reliability and Availability

- Uptime: 99.9% during operational hours
- Fault Tolerance: Graceful degradation on failure• Data Backup: Real-time replication and backups
- Disaster Recovery: RTO within 4 hours
- Error Handling: Detailed logging and monitoring

3.2.4 Security

- Data Encryption: End-to-end for sensitive data
- Payment Security: PCI DSS compliant
- User Authentication: Multi-factor authentication
- API Security: OAuth 2.0 and JWT tokens
- Data Privacy: GDPR compliant

3.2.5 Usability

- User Interface: Features within 3 taps
- Loading Time: Startup within 3 seconds
- Offline Capability: Basic functionality offline
- Accessibility: Support for disabilities
- Multi-language: Local language options

3.3 Interface Requirements

3.3.1 User Interfaces

- Mobile Application:
  Native iOS and Android apps with responsive, clean design for easy navigation. Real-time updates with push notifications keep users informed. Gesture controls enhance interaction, along with dark mode and accessibility features for comfort and inclusivity.
- Web Interface:
  Responsive for desktop and tablets, supporting all major browsers (Chrome, Firefox, Safari, Edge). Includes Progressive Web App capabilities for offline use and faster load times.

3.3.2 Hardware Interfaces

- GPS: Used for accurate delivery tracking and status updates.

- Camera: Enables barcode scanning and delivery confirmation photos.
- Push Notifications: Deliver real-time updates and promotions.
- Sensors: Use accelerometer and gyroscope to support gesture- based controls and improve user experience.

3.3.3 Software Interfaces

Third-party Integrations:

- Payment gateways: Razorpay, Stripe, Paytm integration for secure transactions
- Mapping services: Google Maps API for location tracking and optimized routing
- SMS and Email: Services to send timely notifications and alerts
- Analytics: Firebase and Google Analytics for user behavior tracking and app performance insights
- Social Media APIs: Enable login and sharing through platforms like Facebook, Google, etc. Internal System Interfaces:
- Inventory management APIs for stock tracking and updates
- Order management system to synchronize order processing
- Customer support system connectivity for issue tracking and resolution
- Business intelligence tools to generate reports and insights for decision-making

3.3.4 Communication Interfaces

- RESTful APIs: JSON-based communication enabling interaction between microservices and external components
- WebSocket Connections: Real-time channels for order tracking, notifications, and live updates
- Message Queues: Asynchronous processing of orders and tasks to improve performance and decouple systems•

Database Connections: Secure, encrypted links to transactional and analytical data stores

4. System Architecture and Design Constraints

4.1 Technology Stack

- Frontend: React Native (cross-platform mobile), React.js (web), with Flutter as an alternate for mobile
- Backend: Node.js with Express.js, Golang for performance-critical services, Python/Django for specific modules
- Databases: PostgreSQL (transactions), MongoDB (catalog/user data), Redis (caching/sessions), Elasticsearch (search)
- Cloud: AWS or Google Cloud for hosting, Docker containers for deployment, Kubernetes for orchestration
- CDN: Global delivery of static assets to optimize load times

4.2 Design Patterns and Principles

- Microservices for modularity and scalability
- API Gateway for centralized routing and management
- Event-driven communication for decoupling services
- CQRS pattern separating read/write operations
- Circuit Breaker for fault tolerance and resilience

4.3 Regulatory and Compliance Constraints

- GST compliance for sellers with complete documentation
- FSSAI licensing for food safety of sellers
- Adherence to local data protection laws
- RBI guidelines for digital payment operations
- Compliance with labour laws governing delivery partners

5. Validation and Verification

5.1 Testing Requirements

- Unit Testing: Minimum 80% coverage, automated API and database tests
- Integration Testing: End-to-end workflow validation, third- party integration checks, cross-platform compatibility• Performance Testing: Load and stress tests, database optimization under peak loads
- Security Testing: Penetration tests, payment gateway security, data encryption validation

5.2 User Acceptance Testing

- Beta testing with selected user groups
- Usability testing for UI and UX improvements
- Validation of delivery partner workflows
- Testing admin dashboard features and functionality

6. Project Timeline and Deliverables

6.1 Development Phases

- Phase 1 (Months 1-3): Core platform (user management, product catalog, basic order/payment)
- Phase 2 (Months 4-6): Real-time tracking, delivery partner app, search enhancements
- Phase 3 (Months 7-9): Seller platform, admin dashboard, analytics, performance scaling
- Phase 4 (Months 10-12): Advanced features (voice, AI), security hardening, launch preparation

6.2 Resource Requirements

- Development Team: 2 Frontend, 3 Backend, 1 DB architect, 2 DevOps, 1 UI/UX, 2 QA, 1 Product Manager
- Infrastructure: Cloud hosting with auto-scaling, dev/staging environments, monitoring and security tools

7. Risk Analysis and Mitigation

7.1 Technical Risks

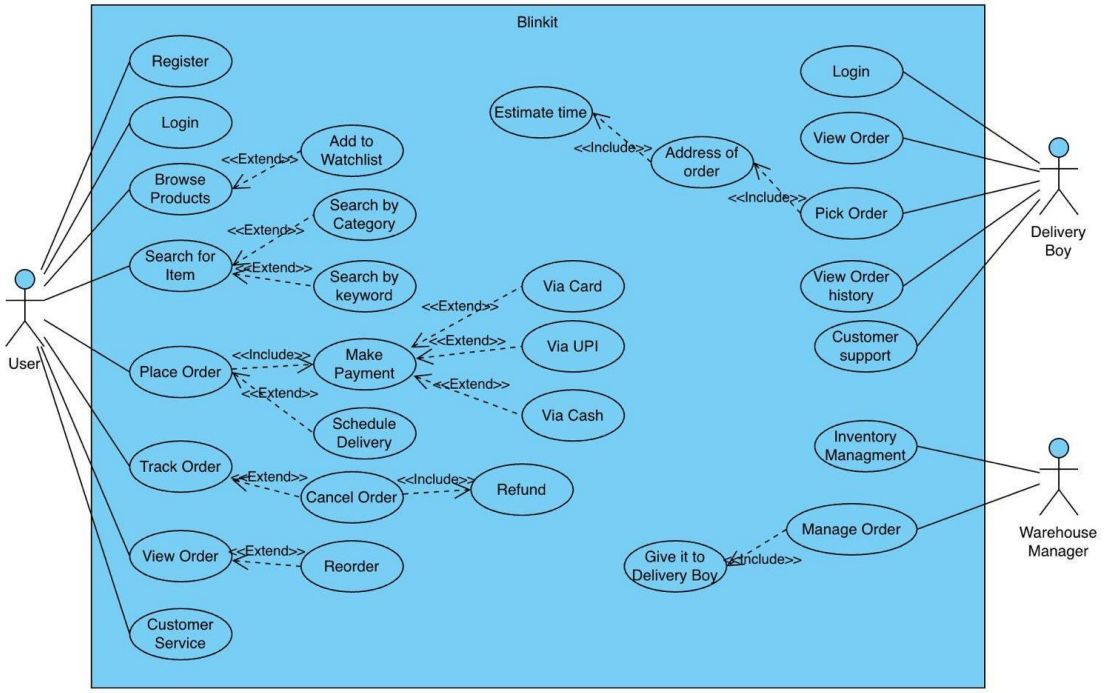
- Scalability issues: Mitigated with microservices, load testing, auto-scaling
- Third-party dependencies failure: Multiple providers and fallback plans7.2 Business Risks
- Delivery SLA compliance risk: Optimized dark stores and logistics algorithms
- Regulatory non-compliance: Legal consultancy and monitoring

8. Conclusion

This document defines a scalable, secure, and user-friendly foundation for the Blinkit platform enabling ultra-fast grocery delivery. Success depends on strict adherence to these specifications, iterative testing, and ongoing updates aligned with user feedback and evolving technology. Regular review ensures the system remains aligned with business goals and compliance requirements.

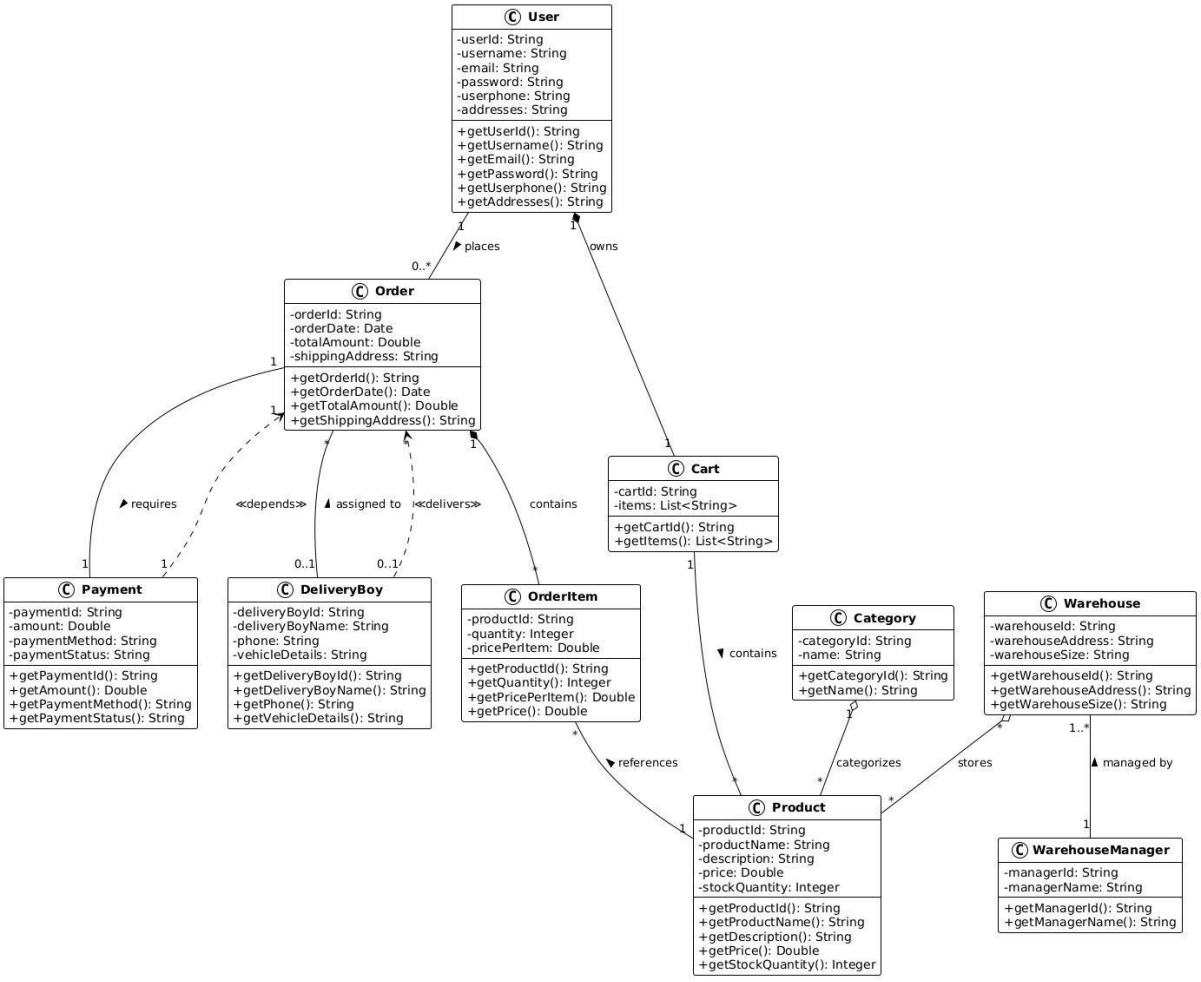
EXPERIMENT - 4

Aim: To perform the user's view analysis: Use case diagram



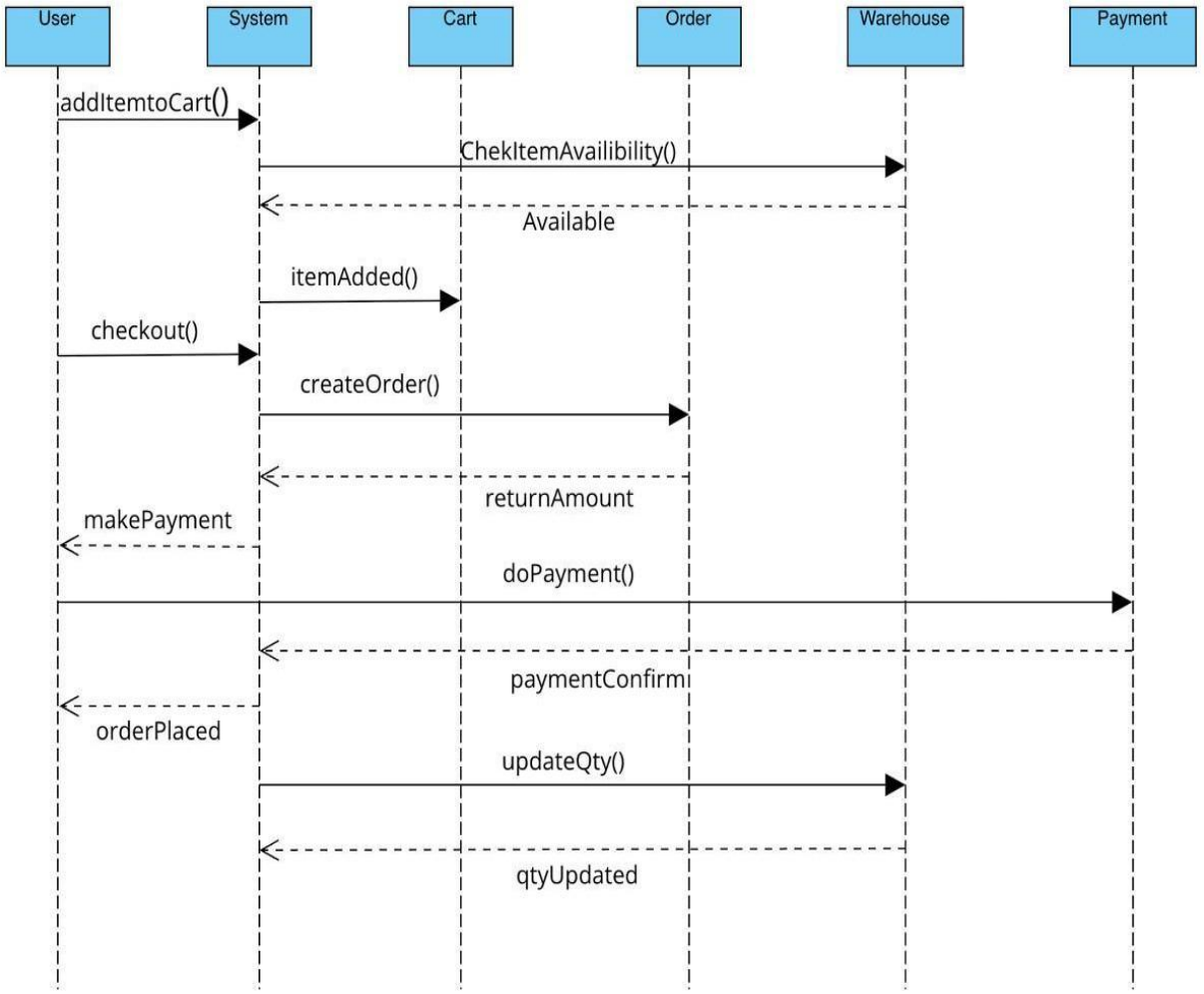
EXPERIMENT - 5

Aim: To draw the structural view diagram: Class diagram, Object diagram



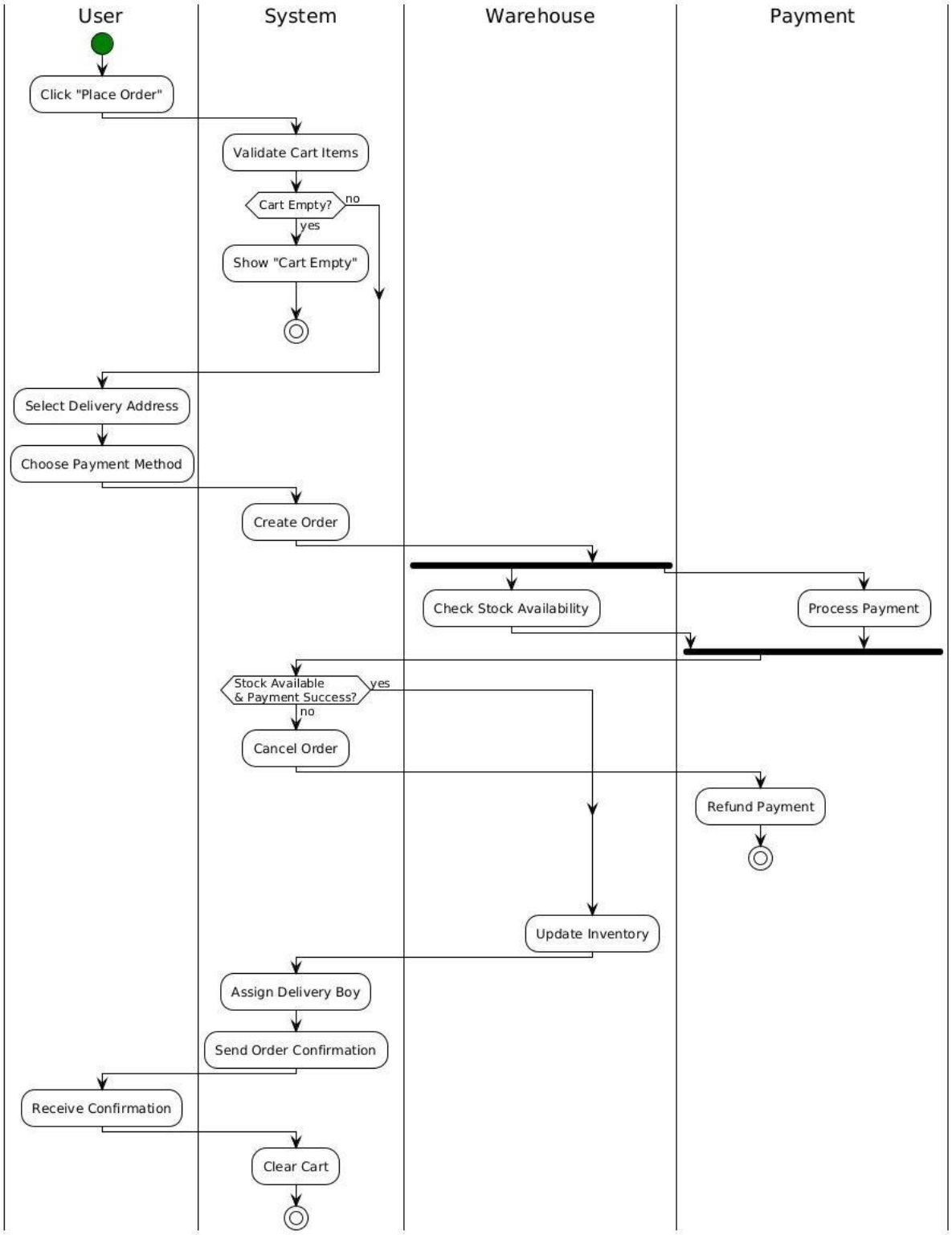
Experiment – 6

Aim: To draw the behavioral view diagram: Sequence diagram, Collaboration diagram



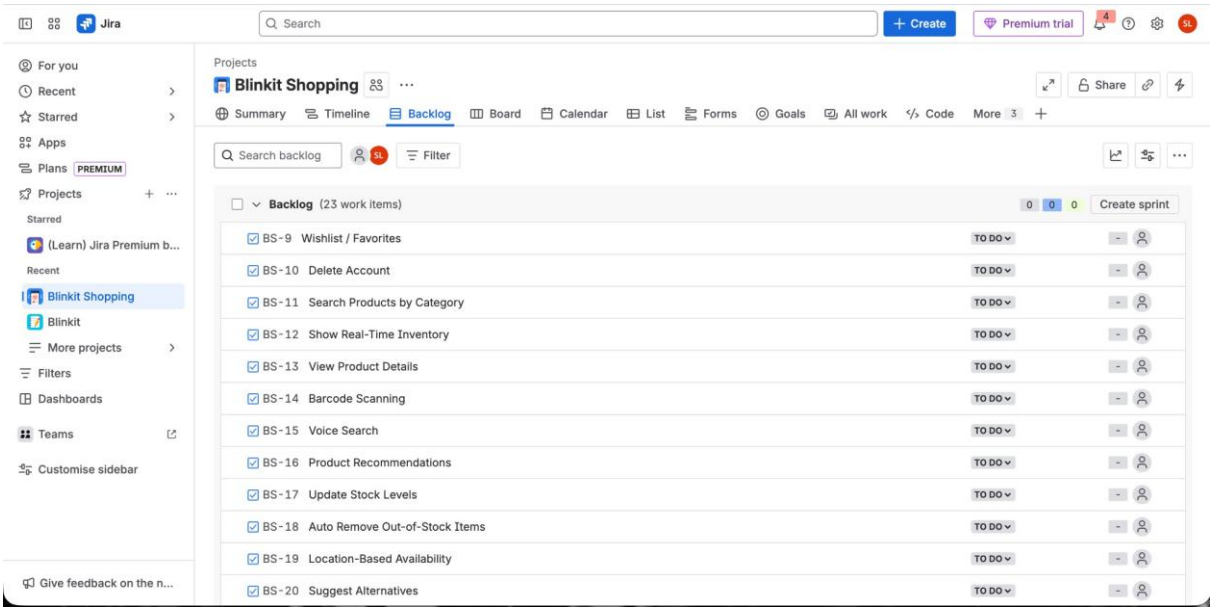
Experiment – 7

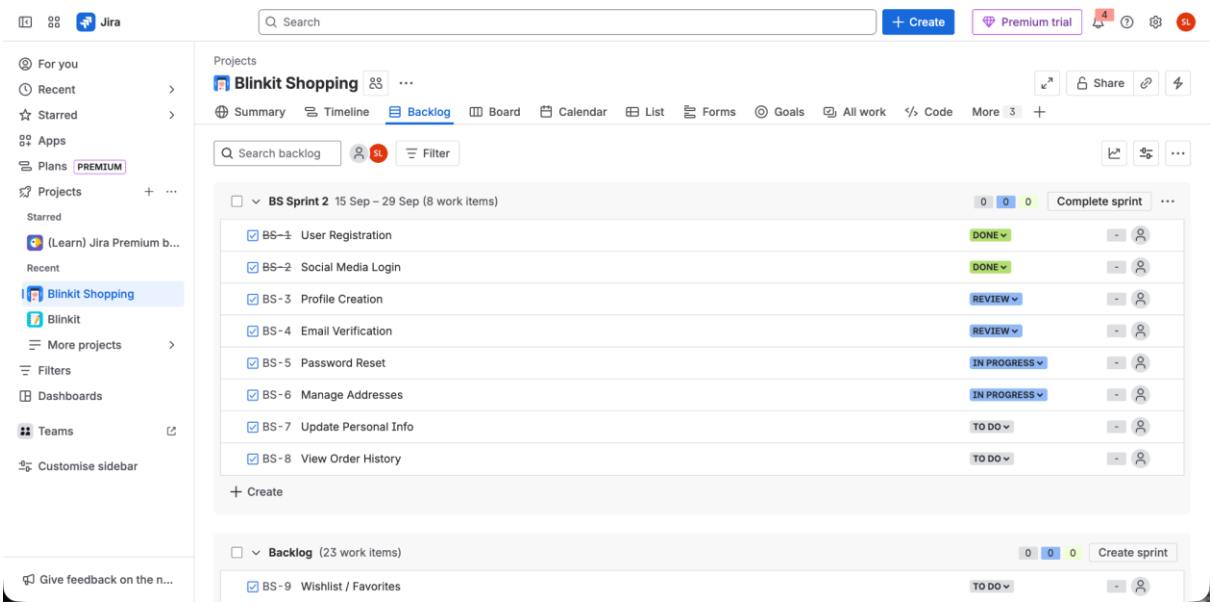
Aim: To draw the behavioral view diagram : State-chart diagram, Activity diagram

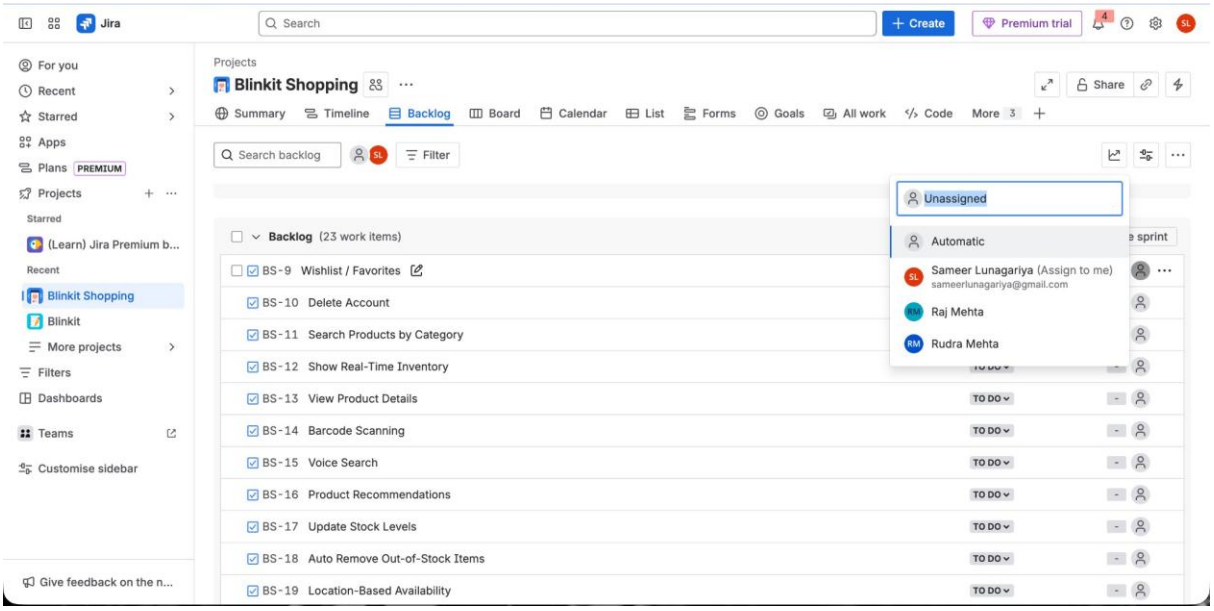


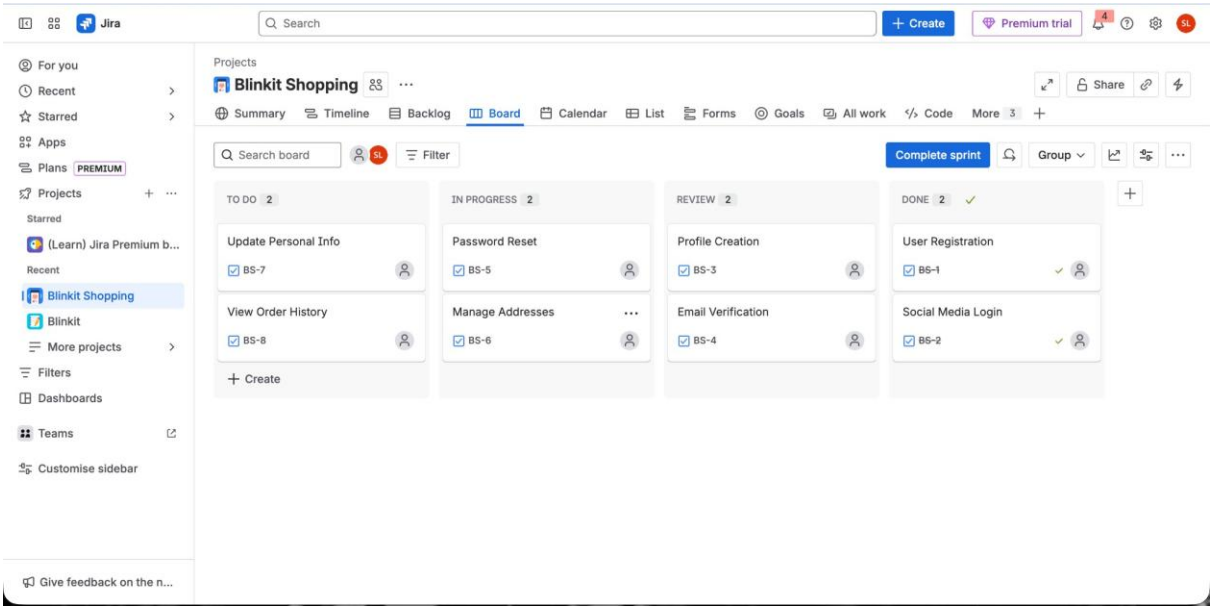
Experiment – 8

Aim: To perform Project Management with an Agile Model using Jira Tool






Jira
Search
Create
Premium trial
For you
Recent
Starred
Apps
Plans PREMIUM
Projects
Starred
(Learn) Jira Premium b...
Recent
Blinkit Shopping
Blinkit
More projects
Filters
Dashboards
Teams
Customise sidebar
Give feedback on the n...
Projects
Blinkit Shopping
Share
Summary
Timeline
Backlog
Board
Calendar
List
Forms
Goals
All work
Code
More 3
Search backlog
Filter
Unassigned
Automatic
Sameer Lunagariya (Assign to me)
sameerlunagariya@gmail.com
Raj Mehta
Rudra Mehta
Backlog (23 work items)
BS-9 Wishlist / Favorites
BS-10 Delete Account
BS-11 Search Products by Category
BS-12 Show Real-Time Inventory
BS-13 View Product Details
BS-14 Barcode Scanning
BS-15 Voice Search
BS-16 Product Recommendations
BS-17 Update Stock Levels
BS-18 Auto Remove Out-of-Stock Items
BS-19 Location-Based Availability
TO DO


Jira
Search
Create
Premium trial
For you
Recent
Starred
Apps
Plans PREMIUM
Projects
Starred
(Learn) Jira Premium b...
Recent
Blinkit Shopping
Blinkit
More projects
Filters
Dashboards
Teams
Customise sidebar
Give feedback on the n...
Projects
Blinkit Shopping
Share
Summary
Timeline
Backlog
Board
Calendar
List
Forms
Goals
All work
Code
More 3
Search board
Filter
Complete sprint
Group
TO DO 2
IN PROGRESS 2
REVIEW 2
DONE 2
Update Personal Info
BS-7
View Order History
BS-8
Create
Password Reset
BS-5
Manage Addresses
BS-6
Profile Creation
BS-3
Email Verification
BS-4
User Registration
BS-1
Social Media Login
BS-2

Experiment - 9

Aim: To perform software testing using the testing tool : Unit testing and Integration testing

BlackBoxTesting.java

Source | History

```java
// BlackBoxTesting.java
package blackboxtesting;

import java.util.Scanner;

public class BlackBoxTesting {

    public static String check(String input) {
        if (input == null || input.length() > 5) {
            return "Invalid Input";
        }

        String reversed = new StringBuilder(input).reverse().toString();

        if (input.equals(reversed)) {
            return "Valid Input";
        } else {
            return "Invalid Input";
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter String: ");
        String input = sc.nextLine();

        String result = check(input);
        System.out.println(result);

        sc.close();
    }
}
```

Output - BlackBoxTesting (run)

```
run:
Enter String: aba
Valid Input
BUILD SUCCESSFUL (total time: 8 seconds)
```

StringCheckPal.java

Source History

```java
// StringCheckPal.java

import blackboxtesting.BlackBoxTesting;
import static junit.framework.Assert.assertEquals;
import org.junit.Test;

public class StringCheckPal {

    @Test
    public void testIsPalindrome() {
        String result = BlackBoxTesting.check("abcba");
        assertEquals("Valid Input", result);
    }

    @Test
    public void testIsNotPalindrome() {
        String result = BlackBoxTesting.check("abcab");
        assertEquals("Invalid Input", result);
    }

    @Test
    public void testWithEmptyString() {
        String result = BlackBoxTesting.check("");
        assertEquals("Valid Input", result);
    }

    @Test
    public void testWithNullInput() {
        String result = BlackBoxTesting.check(null);
        assertEquals("Invalid Input", result);
    }

    @Test
    public void testWithLongInput() {
        String result = BlackBoxTesting.check("ababcbaba");
        assertEquals("Invalid Input", result);
    }
}
```

Test Results

StringCheckPal

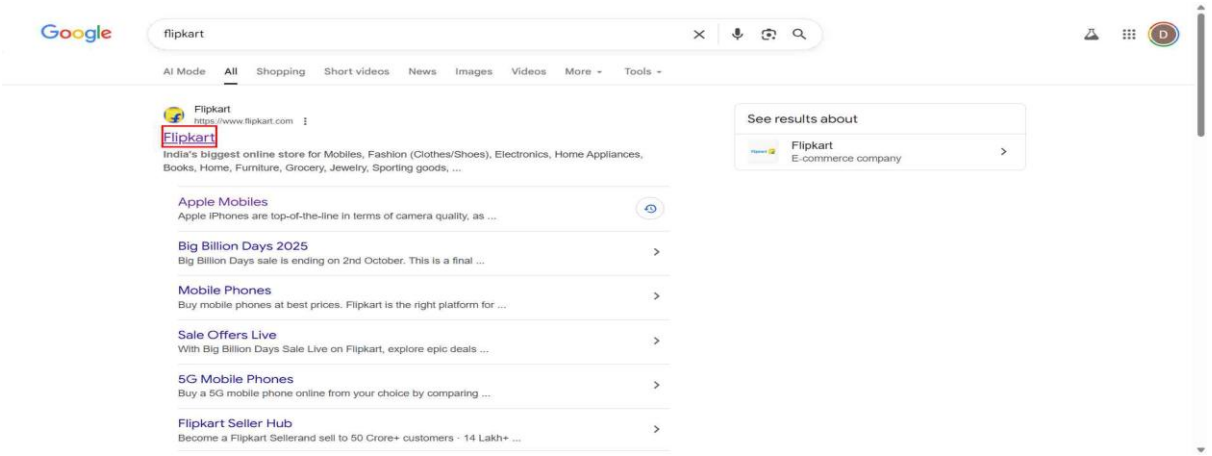
Tests passed: 100.00 %

All 5 tests passed. (0.179 s)

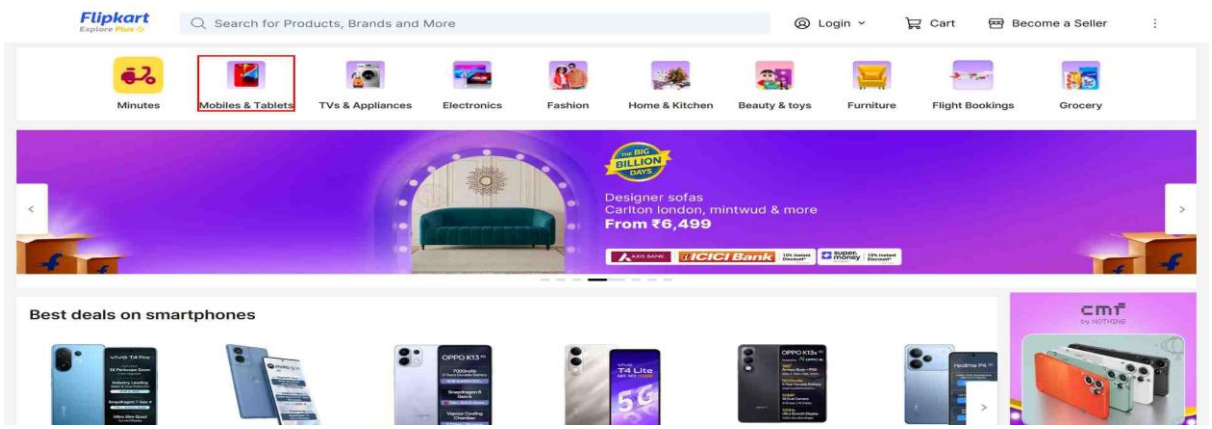
Experiment-10

Aim: To perform software testing using the testing tool : System testing

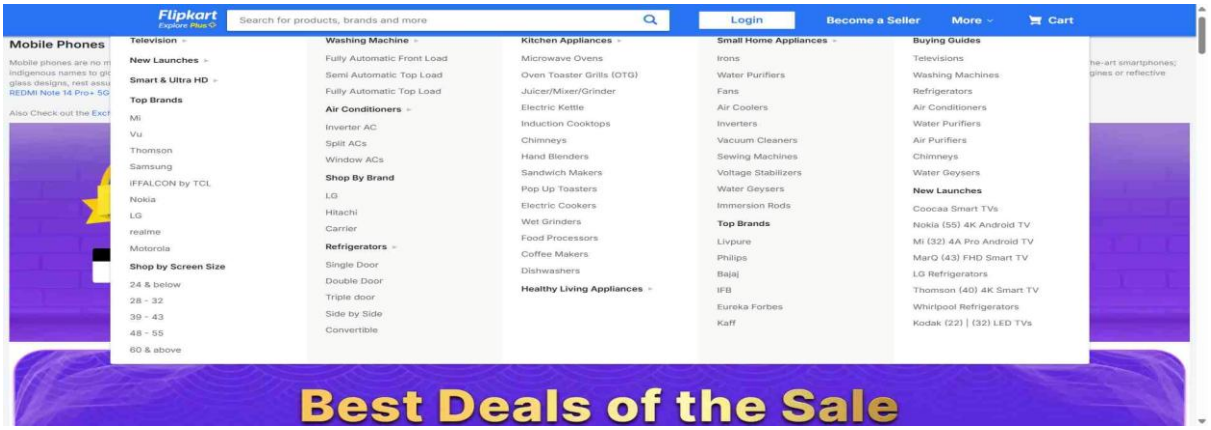
Select Flipkart:



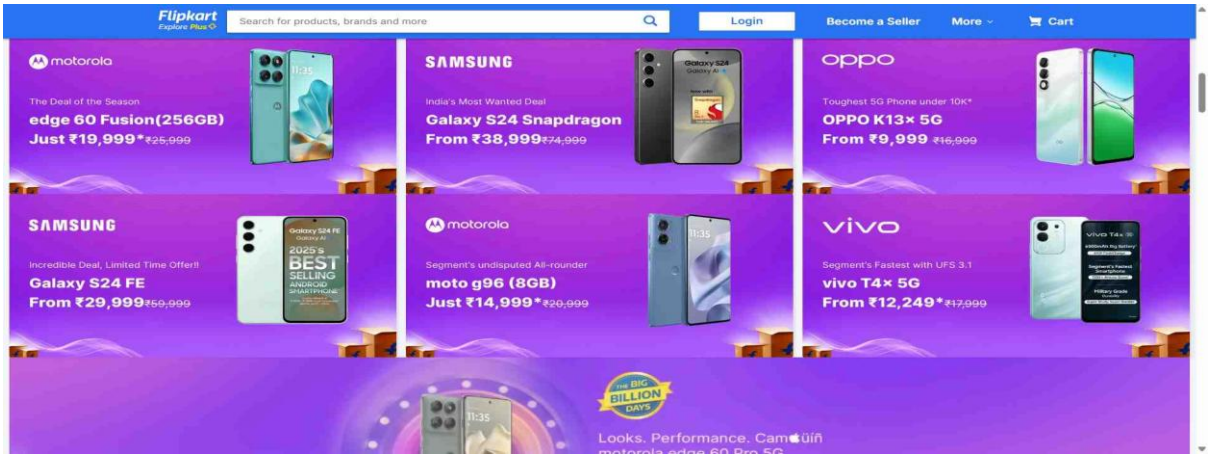
Select Mobiles And Tablets:



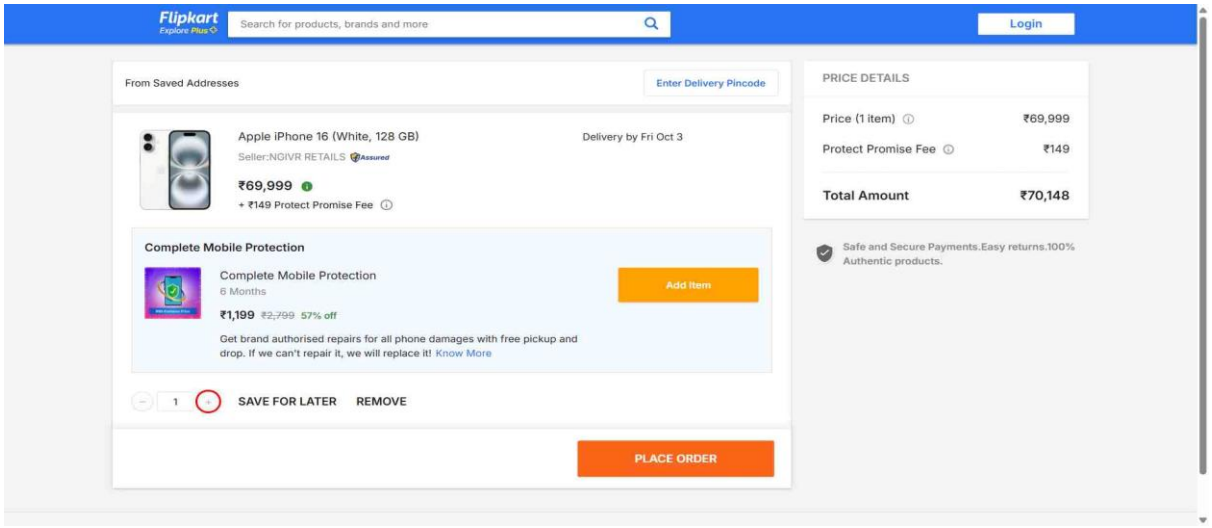
Scroll Down:



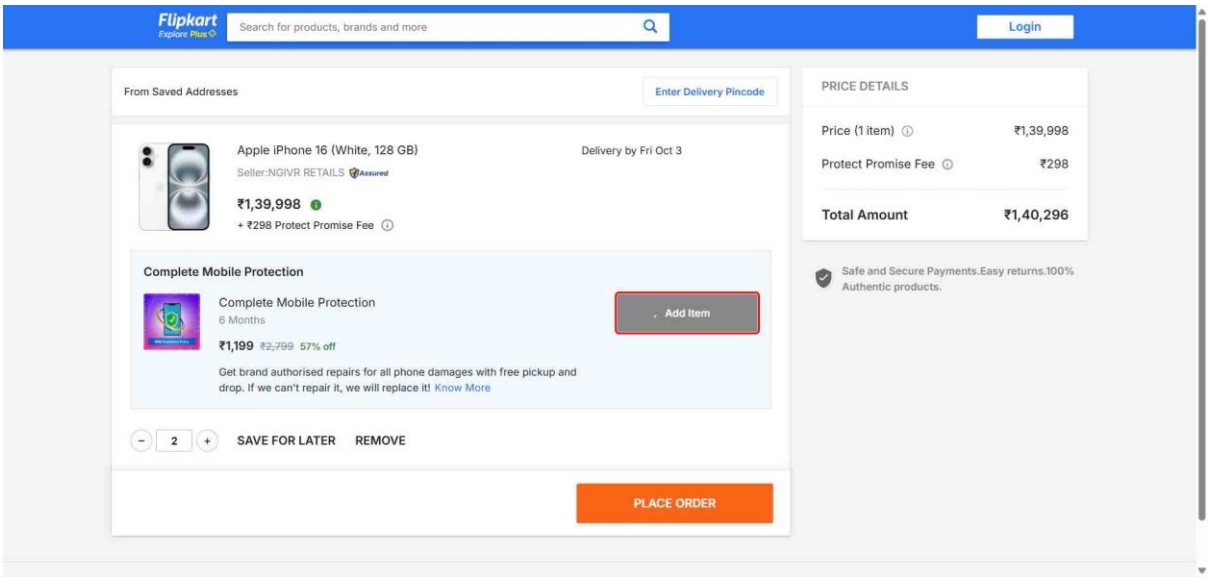
Select Device:



Increment Counter:



Click on Add Item:



Click On Remove: