**EXPERIMENT – 1**

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

**Tools / Apparatus:** None

**Aim**:- To study phases of SDLC and applicability of different SDLC models.

**Tools**:- None

**Q-1 Study the software development life cycle and related topics.**

The Software Development Life Cycle (SDLC) is a structured process followed by software development teams to design, develop, test, and deploy high-quality software efficiently. It defines phases and tasks to manage the lifecycle of a software product.

Phases of the Software Development Life Cycle (SDLC):-

**1. Requirement Gathering and Analysis**

This is the initial phase to understand what the software should do. Developers and analysts gather requirements from the client, end users, and experts to define the functionalities of the system. The collected requirements are then analyzed for clarity, completeness, and feasibility, and a Software Requirements Specification (SRS) document is prepared.

**2. Design**

In this phase, the system's architecture and design are planned based on the requirements. High-level design documents are created that describe how the software will be structured, including details about system modules, database design, and user interactions. This phase serves as a blueprint for developers to follow during implementation.

**3. Coding**

Once the design is finalized, the actual development of the software begins. Developers write code using suitable programming languages and tools based on the project requirements. This is typically the longest phase, and proper coding standards are followed to ensure maintainability.

**4. Testing**

After coding is complete, the software is tested to identify and fix any bugs. Testing ensures that the system works according to need and meets the quality standards. Different types of testing such as unit testing, integration testing, system testing, and user acceptance testing are carried out to validate both functionality and performance.

**5. Deployment**

Once the software has been tested successfully, it is deployed to the production environment where users can start using it. Depending on the project, this deployment can be done in stages or all at once. It may also include user training so that users can use it efficiently.

**6. Maintenance**

Even after deployment, software needs continuous monitoring and updates. The maintenance phase involves fixing bugs that were not previously detected, updating the system based on user feedback or changes in technology or requirements. This ensures the software remains relevant over time.

**Q-2. Study the need of software engineering.**

Software Engineering is needed because modern software systems are large, complex, and must meet high standards of quality, reliability, and performance.

Without a proper engineering approach, software development becomes difficult to manage. Software Engineering provides structured methods and scientific principles to plan, design, develop, test, and maintain software efficiently. It ensures that the final product meets the client’s requirements and can be useful over time, even as it needs change.

software engineering helps teams to manage deadlines, reduce development costs, minimize risks, and ensure team collaboration. It also makes it easier to handle large projects by breaking them into smaller parts.

**Q-3. Study the coverage of topics such as life cycle models, their applicability and their comparison.**

Coverage of Life Cycle Models:-

**1.Waterfall Model** – A linear and sequential approach where cannot able to go back at any phase once that phase will get ended. Requirements are well-known, fixed, and understood, generally real life projects never uses this model for development.

**2.Iterative Model** – Repeated cycles (iterations) to improve the software, we can go back to any phase if problem occurs.it is used when You want feedback after each cycle and gradual improvement

**3.V-Model** – An extension of Waterfall with validation and verification at each stage, when testing is very important part of software then it is used so chances of errors are very less.

**4.Prototyping Model** – Builds a quick prototype for user feedback before actual development. when client does not have clear idea about software requirements that time it is used.

**5.Incremental Model** – Software is developed and delivered in parts, Requirements will not changed after once it is fixed.

**6.Evolutionary Model** – Gradually develops software as user needs evolve. The system needs to evolve with time or data it means it have short term plans and changes requirements, estimates according to changes given by clients.

**7.RAD (Rapid Application Development)** – Focuses on fast delivery using prototyping and iterative development. You need rapid development with active user involvement.

**8.Spiral Model** – Combines iterative and risk-driven approaches, it resolves the risks which were not detected at starting of the project.it is used for large and high-risk projects that need risk management.

**9.Agile Model** –This model is currently applicable in real life projects and it is Iterative and flexible and emphasizes continuous feedback, it also include customer representative as a team member.

**Q-4. List out various type of software for each SDLC Model.**

1. Waterfall Model – Library Management System

2. Iterative Model – Game Development Software

3. V-Model – Aerospace Control System

4. Evolutionary Model – AI Chatbot (e.g., Siri)

5. Incremental Model – Microsoft Outlook

6. Prototype Model – E-learning Platform

7. RAD Model – HR Management System

8. Agile Model – WhatsApp

9. Spiral Model – SAP ERP System

**EXPERIMENT – 2**

**Aim**: To perform requirement engineering tasks.

**Tools**:- None.

**Q-1. Identify the different requirement engineering tasks.**

**1. Requirements Gathering**

Gathering and identifying what stakeholders need from Blinkit.

Sources:

· End-users (grocery shoppers)

· Delivery executives

· Store partners and vendors

· Business analysts

· Competitors (Zepto, Instamart, BigBasket)

· Legal compliance (FSSAI, delivery regulations)

Techniques:

· Interviews with customers and delivery partners

· Surveys on grocery buying behavior

· Analysis of competitor features

· Brainstorming sessions with developers and product owners

**2. Requirements Analysis**

Understanding, refining, and organizing the collected data into structured requirements.

Tasks:

· Classify into functional vs non-functional requirements

· Detect conflicts or contradictions (e.g., delivery speed vs product availability)

· Prioritize features (e.g., real-time tracking, quick reorder)

Examples:

· Functional: User should be able to search and order groceries by category

· Non-functional: Orders must be delivered within 10 minutes in urban areas

**3. Requirements Specification**

Clearly documenting what the Blinkit system should do.

Types of Specifications:

· Software Requirements Specification (SRS)

· Use Case Diagrams and Activity Flowcharts

· User Stories with Acceptance Criteria

Sample Entries:

· "The system shall allow users to search for groceries by name or category."

· "The system shall notify the user when the delivery partner picks up the order."

4**. Requirements Validation**

Ensuring that the documented requirements are correct, testable, and aligned with business goals.

**Activities:**

· Review SRS with internal and external stakeholders

· Validate user flows using wireframes or mock apps

· Compare requirements with business use cases

**Questions Asked:**

· Are users' expectations met?

· Is the requirement measurable and testable?

· Are we considering legal and operational constraints?

**5. Requirements Management**

Managing changes and updates to requirements over time.

Tasks:

· Track evolving user expectations (e.g., more payment options)

· Handle updates from business or tech side (e.g., adding dark mode)

Example: If a new express delivery zone is added, update the delivery logic and notify stakeholders through revised requirements.

**EXPERIMENT – 3**

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

**Tools:-** None.

**1. Title**

Blinkit – The Quick Grocery Delivery App  
An ultra-fast delivery app aimed to provide grocery and household essentials to users within minutes.

**2. Project Scope**

Blinkit is an online platform that allows users to order groceries, food items, and daily essentials with rapid delivery service. The system aims to offer a seamless and efficient experience for users—from account creation to real-time order tracking. The scope includes user account management, product search, personalized cart, secure payments, and delivery tracking. Blinkit also supports additional features such as wishlist, gift options, and notifications. It is built for scalability, security, and performance to serve thousands of concurrent users.

**3. Description**

Blinkit is designed as a mobile application and web platform to deliver daily essentials in under 10–12 minutes in supported areas. It uses real-time tracking, location-based services, and intelligent sorting to ensure fast and relevant service. It includes features such as account management, GPS-enabled address verification, a powerful search and filtering system, secure payment options, and personalized delivery preferences.

**4. Users (Actors)**

* Customer – End user who places orders and receives delivery.
* Delivery Partner – Responsible for picking up and delivering the order.
* Admin – Manages product listings, pricing, user reports, and system monitoring.
* Customer Support Executive – Handles user inquiries, complaints, and refund processes.

**5. Functional Requirements**

R-1. User Registration and Login

Type: Functional  
Input: User details – name, address, phone number, email.  
Output: Account creation or login success/failure.

* Users must provide valid personal details.
* OTP verification is used for authentication.
* Users can create new accounts or log in using existing credentials.
* Users can also edit the details in the accounts.
* Users can also use forgot password feature, if he/she forget password.

R-2. Address Verification

Type: Functional  
Input: GPS data or manually entered address.  
Output: Address accepted or error message.

* System validates address through location services.
* If invalid or out of service area, the user cannot proceed to order.

R-3. Product Search and Display

Type: Functional  
Input: Product name or keyword.  
Output: List of matched products with filtering and sorting options.

* Users can search by product names, categories, or keywords.
* Results can be filtered by: Veg/Non-Veg/Vegan, price range, ratings, and availability.
* Sorting options include popularity, price, and delivery time.

R-4. Cart Management

Type: Functional  
Input: Selected product(s), quantity.  
Output: Updated cart with pricing.

* Users can add/remove products using “Add” or “-” buttons.
* Users can select combo packs or offers.
* Cart dynamically updates total cost, applicable charges, and discounts.

R-4.1. Delivery Charges Calculation

Type: Functional  
Input: Cart total.  
Output: Delivery charges based on cart amount.

* If the cart value exceeds a minimum threshold, delivery is free with minimal handling charge.
* Otherwise, delivery and small-cart charges are applied.

R-4.2. Gift Option

Type: Functional  
Input: Gift option checkbox.  
Output: Extra charge in cart if applicable.

* Some products allow gift packaging. If selected, the system adds extra charges accordingly.

R-5. Stock Notifications

Type: Functional  
Input: “Notify me” selection on out-of-stock items.  
Output: Notification when product is restocked.

* Users are alerted via push notification or email when the item becomes available.

R-5.1. Other Notifications

Type: Functional  
Input: None (system-generated).  
Output: Notifications related to discounts, offers, and order updates.

* The app regularly sends notifications about deals, delivery status, and festivals.

R-6. Payment Modes and Tips

Type: Functional  
Input: Payment method (UPI, Card, COD, Wallet), optional tip.  
Output: Payment success or failure, tip recorded.

* Secure payment gateways are integrated.
* Users can add a tip for the delivery partner during checkout.

R-7. Order Tracking

Type: Functional  
Input: Order ID.  
Output: Real-time location and ETA of delivery partner.

* Users can track their delivery on the map in real-time.

R-8. Delivery Instructions

Type: Functional  
Input: Custom note or predefined instruction.  
Output: Instruction sent to delivery partner.

* Users can write instructions like “Leave at door” or choose default options.

R-9. Refund Processing

Type: Functional  
Input: Refund request.  
Output: Refund confirmation or rejection.

* Refunds are initiated for missing items or cancellation as per the policy.

R-10. Edit Address/Details

Type: Functional  
Input: Updated details from the user.  
Output: Profile updated successfully.

* Users can edit saved addresses and profile details at any time.
* Edits during an ongoing order are restricted to a certain time window.

R-11. Feedback System

Type: Functional  
Input: Ratings and review text.  
Output: Feedback saved and visible to admins.

* Users can rate the delivery partner and purchased items.

R-12. Cancel Order

Type: Functional  
Input: Cancel request.  
Output: Confirmation and refund process if eligible.

* Cancellation is allowed within a limited time after placing the order.

R-13. Wishlist

Type: Functional  
Input: Add/remove product.  
Output: Updated wishlist for the user.

* Users can save or remove products for future purchases.

**6. Non-Functional Requirements**

R-1. Delivery Time Guarantee

* The system must ensure delivery in 10–12 minutes in eligible areas using optimized route planning and nearby warehouses.

R-2. Performance Requirements

* The application should load completely within 2–5 seconds on standard networks.
* Smooth animations and loading indicators must be used to improve UX.

R-3. Scalability

* The system should support a high number of concurrent users without performance degradation, especially during peak times like festivals or flash sales.

R-4. Availability

* The application must maintain 99.9% uptime and have redundant failover systems for services like payments and order dispatch.

R-5. Usability

* The app should offer intuitive UI/UX with features like auto-suggestions, saved preferences, and minimal click ordering.
* Even first-time users should be able to complete an order within a few steps.

R-6. Security

* All user data must be stored securely using encryption.
* The app should comply with privacy regulations like GDPR and use secure payment protocols.

R-7. Compatibility

* The application should work on a wide range of Android and iOS versions (e.g., Android 8+ and iOS 12+).
* It should also function on tablets and smaller screen devices.

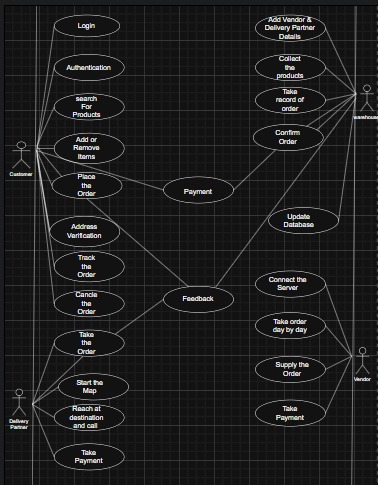
R-8. Localization

* The app should support multiple languages based on the user's location.
* Product recommendations should be localized based on city or region.

R-9. Customer Support Integration

* The app should include real-time chat or helpline for issues related to orders, refunds, or delivery.
* Support should be available 24/7 or during working hours depending on region.

**EXPERIMENT – 4**

****