Software Engineering

①

(i) product backlog (Breaking down User Stories into tasks)

User-1 : I want to log in securely so that I can access my account:

→ login page UI

→ User authentication (Email/password)

→ OAuth for third party login (Google, Facebook)

→ encrypt user credential and stored safely

→ session management

→ Error handeling for incorrect login

→ Unit and integration test

→ security test (SQL injection, brute force attacks)

User-2 : I want to search products by category to find items easily

→ Search bar UI

→ product categorey filter

→ a search API for fetch product based on categorey

→ optimize database quairies for efficient searching

→ Implement sorting feature (price, rating etc

→ mobile responsiveness

→ Unit and integration testing

→ Usability testing.

(ii) In sprint meeting team can prioritize considering customer value and technical feasability

Customer value:

→ login needs for user access, which is high priority.

→ search improves user experience, is low priority.

Technical feasability:

→ login needs for security which needs more efforce.

→ search needs for database quires.

Sprint prioritization:

1) User 1 story should be completed first with high priority

2) parallely user 2 story can be completed with low priority.

(iii) •

Scum board:

Task	To Do	In progress	Done
Design login UI	✓		
User authentication (Email/password		✓	
oAuth for third party login (Google, facebook)			✓
Design search bar UI	✓		
product category filter		✓	
a search API			✓

How the spiral, Agile and Extreme methodologies address risk management and adaptibility

spiral	Agile	Extreme methodology
risk driven approach focus on risk analysis	iterative and incremental approach focus on customer collaboration, software, changes. Framework like scrum, kanban allow teams to deliver software in short cycle called sprints.	adaptive methodology focus on pair programming, TDD(Test driven Development, continious integration.
manage risks by risk assessment for which need experienced personnel	mange risk by continious ~~integreation~~ testing and ~~into~~ itereation	mange risk by automated testing continious integration.

Changes are expensive and time consuming	Change quickly	change quickly

so, more suitable is agile.

3

	waterfall	Agile	XP	Sprial
approach	sequential-phase based	iterative and incremental	iterative and engineering focused	iterative and risk driven
risk management	low occurs at the stard	~~high~~ medium	medium	high
Testing	occures at the end	continous testing in iteration	TDD (Test Driven Development) automated test	risk assessment
Flexibility in chang customer collaboration	low	high	very high	medium
project	well defined requirement and fixed scope	evolving requirement and continious feedback	rapidly changing requirement with customer collaboration	High-risk projects with evolvin requiremen
predictability	high	medium	medium	high

Issues Related to professional Responsibility:

1) security Risks: cyberattacks.

2) privacy violation: misuse of personal data

3) Algorithmic bias: Developing software that discriminates based on race, gender or other factors.

4) Intellectual property theft: using unauthorised code or software components.

5) Software reliability issues:- Releasing untested or defective software that causes harm.

6) whistleblowing - Reporting unethical practices in an organization.

ACM/IEE code of Ethics and Ethical Decision - Making:

1) public Interest and well being

2) Honesty and Integrity

3) fairness and equity

4) confidentiality and privacy

5) professional competence and Responsibility

(5)

Functional requirements:

1) User authentication and authorization

→ Users must login securely using credentials or biometrics

→ contribution: enhances security

2) Flight search and Booking

→ user able to search flights based on date, destination and airline

contribution: improves usability

3) Payment processing:

→ multiple payment options (credit/debit cards, paypal, digital wallets)

→ contribution: ensures performance

and user satisfaction

4) seat selection & reservation management:

→ passengers must be able to select seats and modify booking before departure

contribution: usability and flexibility

5) Real time Notifications and updates:

→ booking confirmation, flight delays

contribution: improves user experience

Non-functional requirements

1) performance and scalability

2) security and data protection

3) usability and assessibility

4) availability and reliability

5) maintainability and modularity

V-Model in software testing:

V-model is a plan driven software development model.

Key phases:

1) Requirements analysis → Acceptence testing
2) system Design → system testing
3) Architecture Design → Integration Testing
4) Module Design → Unit testing
5) Implementation → Unit testing

Benefits:

→ ensures early testing and early defect detection.

→ provides a structured approach

→ improves system quality

prototyping in software engineering:

prototyping is an iterative development approach where a working model of the software is created, tested and refines based on user feedback.

Key stages:

1) requirements gathering.
2) prototype Development
3) User Feedback
4) Refinement
5) iteration

Benefits:

1) user Feedbuck
2) Risk Reduction
3) iterative Development
4) faster time to market.

process Improvement cycle in software engineering is a iterative approach aimed at enhancing software development processes.

<u>key stages</u>:

1> planning : Define goals and gather baseline data.

2) Analysis: identify bottlenecks and areas for improvement

3) improvement: implement solutions and process changes.

4) control: Monitor and ensure the sustainability of improvements

5) Feedback and iteration. use data to iterate and further refine the process

<u>common process Metrices</u>

1) Defect Density: Measures defects per unit of code

2) cycle time : Tracks time to compute a piece of work

3) <u>lead Time</u>: Measures the time it com

request to delivery.

4) <u>test coverage</u>: Indicates the extent of automated testing.

5) <u>velocity</u>: Tracks the amount of work completed per iteration

3

<u>SEI capability Maturity model (CMM)</u>

The CMM is a framework that helps organization asseses and improve their software development processes through 5 maturity levels:

1) <u>Level 1: initial</u> - process are unpredictable and ad hoc

2) <u>Level 2: Managed</u>: Basic project management practices are established for better predictability

3) level 3 : Defined - standardized and documented processes are applied across the organization.

4) level 4 : Quantitatively managed : processes are measured and controlled using data-driven metrics

5) level-5 : optimizing

continuous process improvement is focused on innovation and optimization.

(10)

Agile principle:

1) customer collaboration: focus on customer feedback

2) Responding to change: adapt to changes Quickly.

3) Individuals and Interactions: emphasize teamwork.

4) working software: prioritize functional software over documentation.

Agile in Environments:

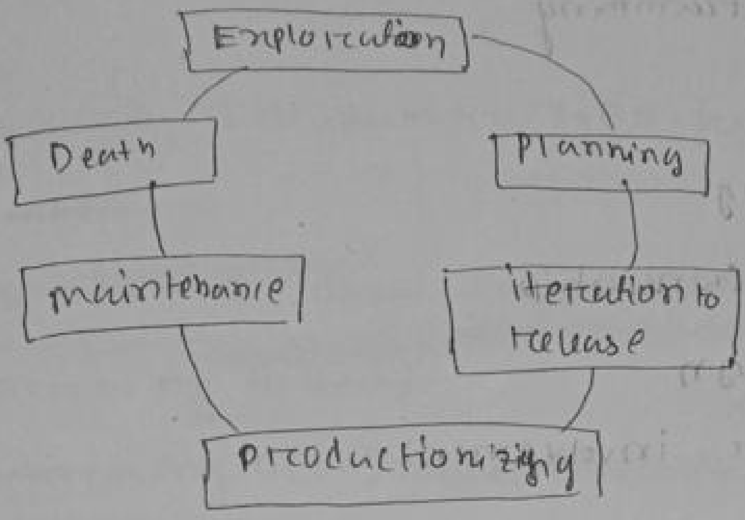
*Startups: fast, flexible iteration.

→ Benefints: speed

→ Challenges: limited resources

* Enterprises: scaled frameworks (SAFe/LeSS)
Benefits: coordination.
Challenges: complexity.

* Regulated Industries: adapt to compilance.
Benefits continious feedback.
Challenges: Balancing agility and regulation



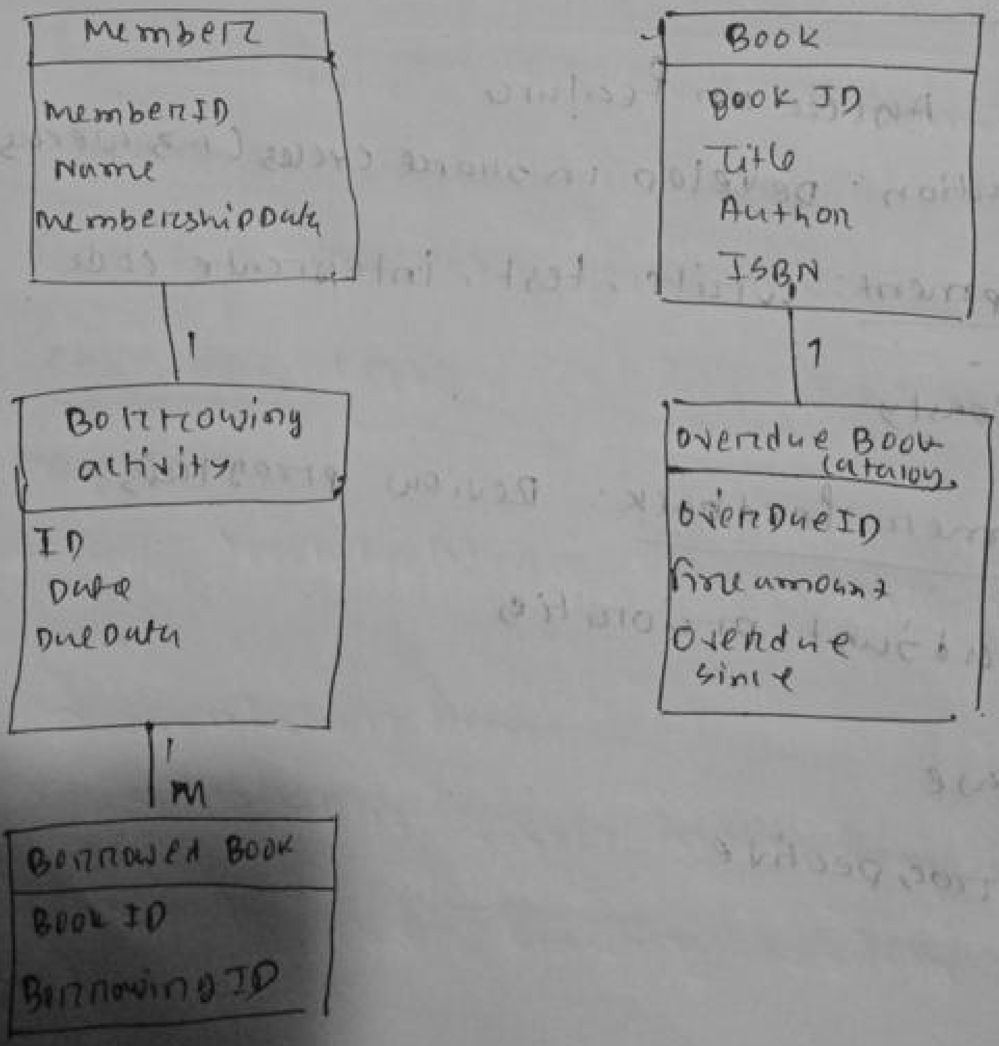
Xp release cycle

1) planning : Agree on feature
2) Interation : Develop in short cycles (1-3 weeks)
3) Development : write, test, integrate code continiously
4) Customer feedback : Review progress and adjust priorities
5) Release
6) Retrospective

Key XP practices:

1) paire programming
2) TDD
3) CI
4) Refactoring
5) collective ownership
6) simple design
7) customer involvement

12



Digital library management system ERD

Entities and attributes:

1) Book
- BookID, Title, Author, ISBN, Genre

2) Member
- MemberID, Name, ContactDetail

3) Borrowing Activity:
- BorrowingID(PK), BorrowDate, DueDate,
- Links Member and Book through foreign keys.

4) OverDue book catalog.
- overdue, fineAmount, Overduesince.
- Tracks overdue books and fines

<u>Relationships</u>

1) Member ↔ Borrowing activity (1:N)

2) Book ↔ Borrowing activity (1:N)

3) Borrowing activity ↔ overdue Book catalog (1:1)

(13)

Aspect	verification	Validation
Definition	Ensures the product is built corrrectly	ensures the right product
Focus	conuformance to specifications and design	meeting user needs and requirements
Activities	Reviews, inspection, static analysis	user testing, system testing, accepteance
Nature	static	Dynamic

(14)

Layered Architechture for online Judge system:

1) presentation layer:

→ responsibilities: user interfaces for submitting code, viewing resultes and interacting with the system.

Tech: HTML, CSS, Javascript (React, Angular)

2) Application layer:

Responsibilities: manages API requests, authentication and communication between UI and business logic.

Tech: Rest APIs, JWT authentication.

3) Business logic layer:

Responsibilities: code evalution (compiling, testing) manging user and problem data, calculating results.

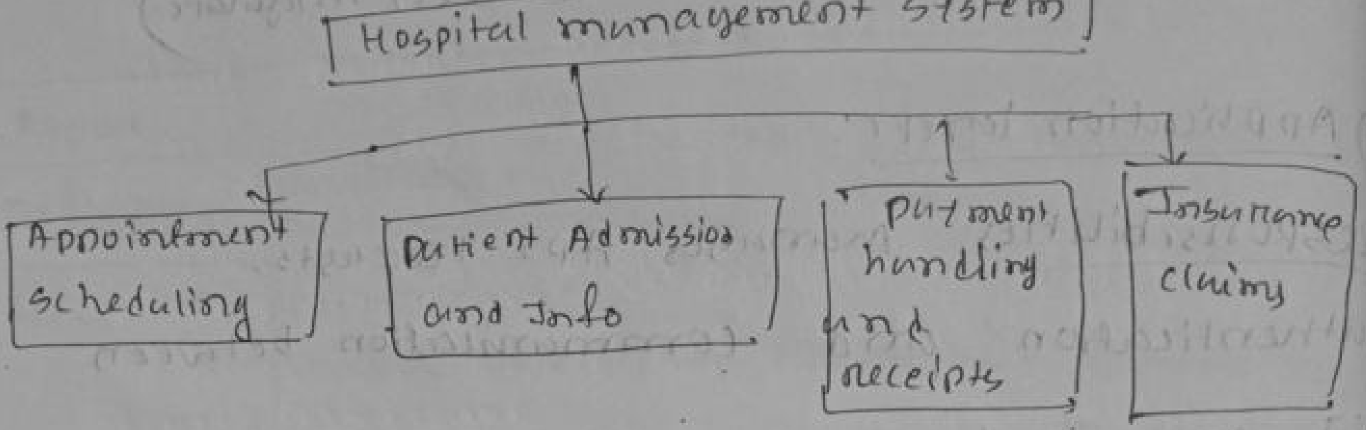
Tech: spring Boot, Docker for isolated code execution, message queues

4) Data layer:

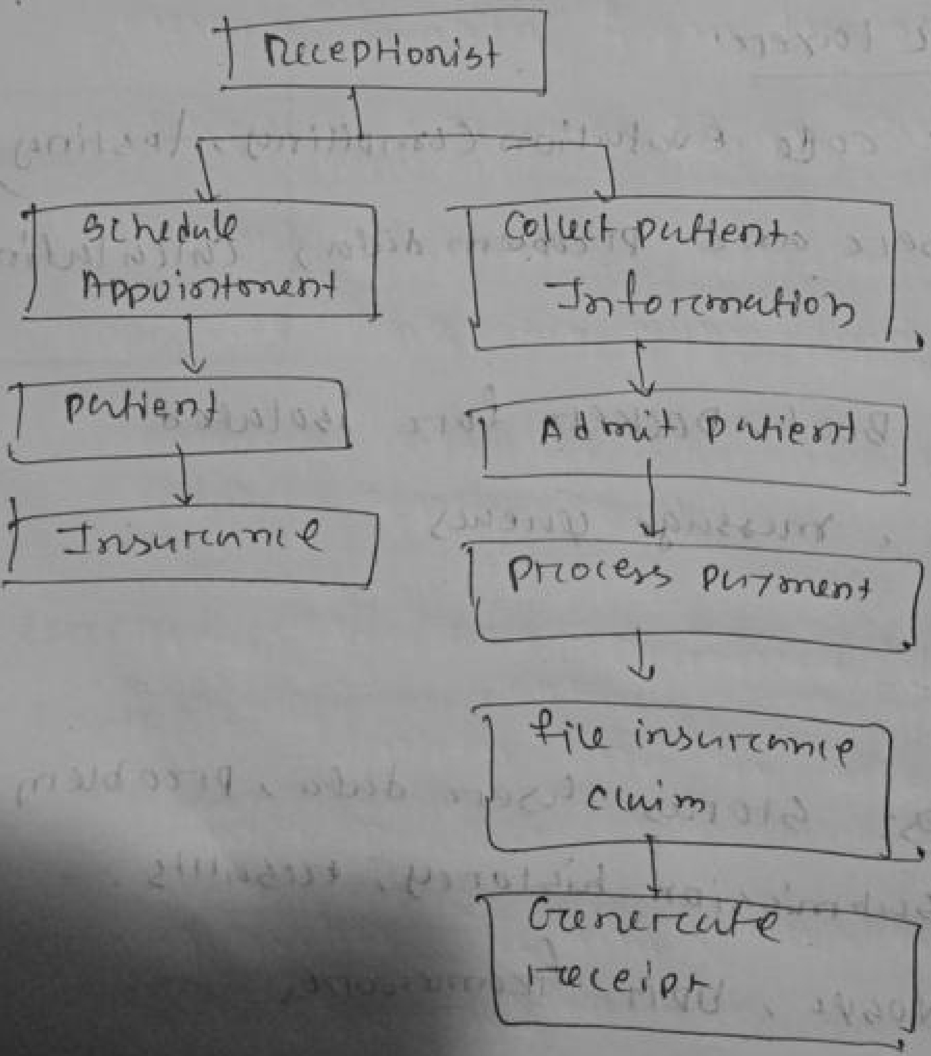
Responsibilities: stores user data, problem definition, submission history, resullts.

Tech: SQL, NoSQL, ORM frameworks.

DFD level:1


Hospital management system
Appointment scheduling
patient Admission and Info
payment handling and receipts
Insurance claims

UML case Diagram:



DFD fore Hospital Managment System

<u>level 0 (content Diagram)</u>

→ <u>Entities</u>: patient, Receptionist,

Flow.
· patient interacts with receptionist fore appointments and info.

· receptionist communicates with th Hospital system and Database.

<u>level 1 (substystme)</u>

· <u>processes</u>:
· Appointment scheduling, patient admission, payment handeling.

· flow: Each process interacts with the database.

<u>UML use case Diagram (Receiption Module)</u>

· Actors: Receptionist, patient, Insurance provider:

. use cases:

Schedule Appointment, collect info, admit patient, process payment

(17)

Quality assurance vs Quality Control

quality assurance

· Focus: process oriented

· Goal: Improve development processes to ensure quality from the start

· Activities: Process design, audits.

· Nature: proactive

· Tools: Agile, six sigma, lean.

Quality control (QC)

Focus: Product-oriented (detects defects)

· Goal: Identify and fix defects in the final product.

· Activities: Testing, inspection, Validation.

nature: reactive

Tools: Testing tools (Junit, Selenium)

Role of QA in Each SDLC Phase.

1) Requirements Gathering:

· Role: Ensure requirements are clear, complete and testable.

· Activity: Analyzed and validate requirements.

2) System Design

· Role: Ensure design aligns with requirements and is testable.

· Activity: Review design, prepare test plans.

3) Development:

· Role: Ensure quality coding practices and early testing.

Activity: Develop test cases, perform code reviews, static analysis.

4) Testing:

· Role: Validate that the software meets requirements and identify defects.

· Activity: execute test, report bugs and conduct feedback loops.

5) Deployment:

· Role: Ensure stable deployment and readiness for production.

· Activity: conduct UAT, verify deployment process.

(17)

RAD (Rapid Application Development Model):

is a fast, iterative software development approach focused on prototypes, user feedback, parallel development.

Key phases:

1) Requirement phases: Define requirements

2) User Design: Create and refine prototypes

3) construction: Build system with parallel development.

4) cutover: final testing and deployment.

Principles:

· prototyping and user involvement.
· Iterative Development and time boxed phases.
· component reuse.

Advantages:

· Faster delivery, flexibility and cost efficiency
· High user satisfaction through continious feedback.
· Early issue detection esures Quality.

(20)

White Box Testing (Java)

-The RAD approaches focus on testing key decision points like if, else if and for loops in the provided code.

Decision	x input	y input	Expected output
y==0	Any	0	"y is zero"
x==0	0	Any	" x is zero"
loop(i<=x)	Positive	positive	Numbers divisible by y
loop(i<=x)	positive	larger than x	No output
loop(i<=x)	Negative	positive	No output

Junit Test class:

```
import org. Juint . Jupiterr. api . BeforceEach;
import org. Junit . Jupiterr. api . Test ;
import static org. Junit. Jupiterr . api. Assertions .*;
import Java. util . Arrcary list ;
import Java. util. List;

class DecisionTest{
private List < String > output ;
private void println(String message){
 output.add (message);
}

private void prcocess (int x, int y){
 if (y==0){
 Println( "y is zero"); }
 else if (x==0){
 Println ("x is zero");
 else{
```

```
for (int i=1 ; i<=x ; i++){
if (i%y==0){
  println(Integer.tostring(i));
  }
 }
}
}
@BeforeEach
public void setup(){
output = new ArrayList<>();}

@Test
public void test_y_is_zero(){
 process(5,0);
 assertEquals(output, List.of("y is zero")); }

@Test
public void test_x_is_zero(){
 process(0,3);
 assertEquals(output, List.of("x is zero")); }

@Test
 public void test_loop_does_not_run(){
 process(0,2);
 assertEquals(output, List.of()); }
```

```
@Test
public void test_numbers_divisible_by_y(){
process(4,3);
assertEquals(output, List.of("3")); }

@Test
public void test_edge_case_y_negative(){
process(5,-2));
assertEquals (output, List.of ("2", "4")); }

@Test
public void test_edge_case_x_negative(){
process(-3,2));
assertEquals(output, List.of()); }
```

(21)

Junit4 allows for effective black-box unit testing using exception handling, setup fuctions and timeout rules.

1) Exception Handling:

Use @Test(expected = Exception.class) to ensure methods throw exceptions under specific conditions.

2) setup Function:
use @Before to set up objects or prepare the environment before each test.

3) Timeout Rule:

Use @Rule with timeout to limit test execution time.

production code(calculator):

```
public class calculator{
public int divide(int a, int b){
 if(b==0) throw new ArithmeticException("cannot
 divide by zero!");
return a/b;
}
public String getGreeting (String Name)
}
if (name==null) ~~throw new~~
throw new IllegalArgumentException( "Name
cannot be Null");
return "Hello," + name; }

public int longRunningMethod()
throws InterruptedException {
   Thread.sleep(1000);
   return 42;
  }
 }
```

Junit Test class:

```
import org.Junit.Before;
import org.Junit.Test;
import org.Junit.Rule;
import org.Junit.rules.Timeout;
import static org.Junit.Assert.*;
```

```
private Calculator calculator;
@Before
public void setup(){
 Calculator = new calculator(); }

@test(expected = ArithmaticException.class)
 public void testDividedByZero(){
  Calculator.divide(10,0); }
@Test
 public void testDivide(){
 assertEquals(5, Calculator, divide(10,2))); }
@test(expected = IllegalArgumentException.class)
public void testGreetingWithNullName(){
 Calculator.getGreeting(null); }

@Test
 public void testGreeting(){
  assertEquals("Hello, John", Calculator.
                               getGreeting("John")
                                                 }

@Rule
public Timeout globalTimeout = Timeout.seconds(2
@Test
public void testLongRunningMethod()
throws InterruptedException {
assertEquals (42, calculator.longRunningMethod
);                                         }}
```

Key points:

- Exception Handling: Test if methods throw expected exception.
- setup Function: Use @Before to prepare the test environments.
- Timeout Rule. Ensure tests complete within a specific time limit.