1. product Backlog with tasks

The product backlog consist of user stories broken down into smaller tasks for development.

User story 1: Login Functionality

"As a user, I want to log in securely so that I can access my account."

Task:

1. Design UI for the login page.
2. Implement user authenticate (email, password)
3. implement secure password hashing and stora
4. Integrate OAuth (Google, Facebook) authentication
5. validate user input (email format, password s
6. implement error handling (wrong credentials, locked account)
7. create unit tests for authentication functio
8. deploy the login feature and perform secure testing.

user story-2: product search by category.

"As a user, I want to search for products by category to find items easily".

Tasks:

1. Design UI for the search functionality.
2. implement the backend API for category based search.
3. optimize database queries for better performance.
4. implement auto complete suggestions for categories.
5. display search research with filter (price, rating)
6. Ensure mobile responsiveness of the search feature.
7. conduct testing for search accuracy and performance.
8. Deploy and monitor search functionality in production

2. Prioritization in Sprint Planning:-

During Sprint planning, the development team prioritizes user stories based on customer value and technical feasibility.

- customer value:

the login feature in high priority as users

must authenticate before accessing their accounts. the search functionality is also essential but comes after authentication since users need to log in to personalize search results.

Technical feasibility: The login feature requires security consideration. It has moderate complexity but must be implemented first.

Sprint planning Decision: login tasks will be completed first in the sprint as it is a fundamental feature. search implementation will begin after the core login. is functional to ensure smooth integration.

3 tracking tasks on a Scrum Board.: A scrum board visualizes task progress using three columns

to Do	In progress	Done
Design login UI	implement authentication	login UI completed
Implement Oauth	validate user input	password hashing done
Design search UI	Backend API Development	Oauth integration completed
Optimize search queries	testing search functionality	User input validation completed
Create Unit tests	Deploy login feature	Backend API search completed

- To DO: tasks that are planned but not started.
- In Progress: Tasks currently being worked on by the team.
- Done: completed and tested tasks ready for deployment.

comparision of spiral, Agile and extreme programming in Risk management and adaptibility.

⊛ Spiral model:

• Risk management: Design for high-risk-projects, the spiral model explicitly includes risk assessment in each phase. use prototyping to identify uncertainties.

⇒ Adaptibility: Allows iterative refinement, but changes may be costly due to its heavy planning and documentation requirement.

⇒ Suitability: Best for projects with high technical risks but not ideal for rapidly changing requirement due its structured nature.

⊛ Agile model:

⇒ Risk management: Reduce risk through short development cycles and continuous client feedback. frequent testing and incremental

delivery prevent costly late-stage failures.

⇒ Adaptability: highly flexible, allowing quick adaptation to evolving client needs. encourages collaboration and incremental improvements making it cost-effective.

⇒ suitability: Best suited for projects with evolving requirements and uncertainty since it allows frequent course correction.

* Extreme programming:

Risk management: focuses on continuous testing, pair programming and frequent release reducing failures risks. emphasizes constant communication with the client to ensure alignment with needs.

adaptibility:- high adaptibility through short iterations and refactoring, but may face challenges in large terms.

suitability: works well in projects where

customer involvement is high and requirement change frequently, but may struggle with high risk technical components requiring deep analysis.

(3)

Comparision of Development Methodology

Methodology	predictability	Customer Collaboration	Risk management	Adaptability
Waterfall	High	Low	Low	Low
Agile	moderate	High	High	High
Extreme programming	Low-moderate	very high	moderate	very high
Spiral	moderate-High	moderate	very high	moderate

* Issues Related to professional Responsibility.
- software Reliability & safety: Engineers must develop error-free and secure software.
- privacy & security: protecting user data from breaches and unethical use.
- Intellectual property: Respecting copyrights patents and licenses.
- Fairness & non-discrimination: Ensuring unbiased software design.

* Role of ACM/IEEE code of ethics.

The ACM/IEEE code of ethics provides ethical guidelines including

1. prioritizing public welfare: over business or personal gains
2. maintaining honesty and transparency in professional work.
3. Ensuring fair treatement of all

rapid iterations with high customer involvement and constant testing.

(9)

Principles of software Engineering Ethics:

software engineering ethics ensures responsible professional conduct. Key principles include:

1. public interest: ~~Print~~ prioritizing user safety, privacy and societal well-being.
2. integrity: Honesty in project estimates, reporting issues and avoiding misrepresentation.
3. Confidentiality: protecting sensitive client and user data.
4. competence: Engaging only in work where expertise is sufficient.
5. professional Responsibety: Avoiding conflicts of interest and ensuring accountability for software failures.

Best methodology for each project

project A : well defined requirement, strict deadline.

Best choice : waterfall or spiral.

⇒ Waterfall : ideal for well-defined, stable requirements with a clear roadmap. Ensures timely delivery with structured phases.

⇒ Spiral : Suitable is risks need to be managed at each stage while maintaining a structured plan.

project B : Evolving requirements, uncertain timeline, continuous feedback.

Best choice : Agile or Extreme programming

⇒ Agile : Best suited for projects with changing requirements and continuous client involvement. Ensures flexibility and risk mitigation through incremental delivery.

⇒ XP : works well if the team can handle

stakeholders.

4. Upholding privacy and security of sensitive information.

5. continual learning to maintain professional competence.

By following these principles.

(5)

Functional Requirement:

1. user authentication: Ensures secure login for passengers, airline staff, and administrators.

2. Flight search and Booking: Allows users to search flights by destination, date and price ensuring a seamless reservation process.

3. payment processing: Enable ensuring online payments via credit card, debit card or digital wallets.

4. Ticket management: user can view, modify

or cancel reservations, improving flexibility and user experience.

5. Real time flight status update: provide live flight information, enhancing passenger convenience.

Non-functional requirement:

1. performance and scalability: Handles multiple simultaneous user request with-out lag, ensuring fast response times.

2. Security and Data Protection:- implement encryption and secure authentication to protect user data from breaches.

3. Usability and accessibility:- offers a user-friendly interface with a multi-language support for global accessibility

. System availability:- Ensures 99.9% uptime for uninterrupted service, critical for

flight reservation.

5. maintainability & support: Allows easy software updates and bug fixes, ensuring long term reliability.

These requirements collectively enhance system efficiency, security and user satisfaction, making the airport reservation system robust and user friendly.

⑥

V-Model of testing in plan-Driven Software Development:

The v-model (verification & validation model) is a software development lifecycle where testing phases run parallel to development stages. it emphasizes early defect detection by aligning each

development phase with a corresponding testing phase.

phases and their relationship:

1. Requirement analysis ↔ Acceptance Testing.
   - Ensures the system meets business needs and user expectations.
2. System design ↔ System testing
   - validates system functionality, performance and security.
3. Architectural Design ↔ integration testing
   - check data flow and interaction between software modules.
4. module design ↔ Unit testing
   - Test individual components for correctness and reliability.

. Implementation (coding phase)

- The only development phase, involving

to unit testing.

Key benefits:

- Early defect detection reduces costly fixes later.
- Structured ~~approach~~ approach ensures quality at each stage.
- parallel ~~stage~~ testing improves efficiency and system reliability.

Thus. The V-Model ensuring rigorous testing making it ideal for high-quality, risk-sensitive projects.

(7)

Prototype Development in Software engineering.

prototyping: it is an iterative approach where a working model of the software is built to refine requirements before full scale development.

Key stages in Prototype Development

1. Requirement gathering: Basic user needs are identified.
2. Quick design: A rough UI and basic functionality are outlined.
3. prototype Development: A working model is created with limited features.
4. User evaluation & feedback: users interact with the prototype, providing insights
5. Refinement & iteration: Based on feedback the prototype is improved until requirement are clear.

Benefit of prototyping:

1. User feedback: Helps stakeholders validate features and useability early.
2. Risk Reduction: Identifies issues before

full development, saving cost and time.

3. iterative Development: Encourages incremental improvements, ~~requiring~~ refining requirement efficiently.

4. Better Requirement clarity: Reduces misunderstanding between developers and users.

5. Enhanced stackholder confidence: provides a visual model to access feasibility before full investment.

prodotyping is ideal for projects with evolving requirements, ensuring better alignment with user needs and minimizing development risks.

(8)

rocess Improvement cycle in software
ngineering.

he process improvement cycle in an iterative
pproach to enhancing software development

efficiency and quality.

Key stages:

1. process assessment
2. process Modeling
3. process implementation
4. process monitoring
5. process optimizing

commonly used process metrics

1. Defect Density
2. cycle time
3. Effort variance
4. customer satisfaction Index
5. code churn.

These metrics help in monitoring productivity, identifying bottlenecks and ensuring continuous improvement in software process.

(9)

Software Engineering institute Capability Maturity Model (SEI CMM): CMM is a framework developed by SEI to access and improve software development process. It defines five maturity levels that organizations progress through to enhance progress quality and efficiency Five levels of CMM & their Contributions:

1 Level-1: initial:

Ad-hoc, chaotic processes with unpredictable outcomes.

Level-2 Repeatable:

Basic project management is in place with defined process.

Level-3 - Defined: standardized processes are established accross the organization improvement enhance consistency and documentation.

Level-4-managed:

Quantitative metrics are used to measure and controll process.

Level-5 optimizing:

Focus on continuous process improvement and innovation. improvement ensures agility and long term- efficiency.

impact on software efficiency Development & organizational performance.

1. Reduce defects and improves software quality.
2. Enhances predictability and project success rates.
3. promotes efficiency and better resource management.
4. encouraging innovation and long term growth.

(10)

core principles of Agile Software Development.

1. customer collaboration over contact negotiation.
2. working software over comprehensive Documentation.
3. Responding to change over following plan.
4. individual interactions over process and tools.
5. incremental Delivery.
6. sustainable Development.

Application in Different Development Environments.

⇒ startups - Agile enables rapid prototyping and quick marketing
⇒ Enterprise systems
⇒ Embedded systems

benefits of Agile methods:

1. Faster time to market.
2. improved customer satisfaction through

continuous feedback.

3. Greater flexibility to enhancing requirements.

4. Enhancing team collaboration.

Challenges of Agile Methods:

1. Difficult to scale in large, highly regulated projects.

2. Requires high customer involvement which may not always feasible.

3. Less suited for projects with fixed scope and rigid deadlines.

(11)

Release Cycle of Extreme Programming (XP)

The XP release cycle follows an iterative and incremental approach to software development. The key stages includes:

1. planning
2. iteration
3. Design and coding
4. Testing
5. Release
6. Feedback and improvement.

influential programming practice in XP:

1. pair programming
2. Test-Driven Development (TDD)
3. continuous Integration (CI)
4. Refactoring improves code maintainability without changing functionality.
5. simple Design: Focuses on minimalism. avoiding unnecessary complexity.

6. Small releases: Frequent implementation deliveries provide early feedback.
7. Collective code ownership: Any developer can modify the codebase increasing flexibility
8. Sustainable Pace: Avoids burnout by maintaining a steady development speed.

XP's iterative cycle and best practices enhance software quality, adapting and developer collaboration in fast-changing environments.

(12)

Entity relationship Diagram (ERD) for Digital Library Management System.

Entities and their Attributes:

1. Book (Book_ID, title, Author, ISBN, GENRE Availability status)
2. Member (Member_ID, Name, contact_Details membership type)
3. Borrowing (Borrowing_id, Borrow_Date, Return_Due_Date, Return_status, Fine_Amount)

④. overdue_Books (overdueID, Fine_Amount, overdue_days)

Relationships:

⇒ Member → Borrows → Book

⇒ Book – Has → Borrowing Record

⇒ Borrowing → Generates → overdue_Books

explanation:

⇒ Books are identified by unique Book_ID and track availability.

⇒ member are unique by identify by member-ID and store contact details

⇒ Borrowing Records connect books and member tracking dates and return status.

⇒ Overdue Books maintain fines based on the number of overdue days.

This ERD ensures efficient tracking of books, members and overdue fines, optimizing Library operations.

(19)

What is testing?

Answer: Testing is the process of evaluating a software application to ensure it function correctly, meets requirements and is free of defects.

Difference between Validation and Verification

Aspect	Verification	Validation
Defination	Ensures the software meets specifications	Ensures the software meets user needs
Focus	Process-oriented (Are we building the product right?)	product oriented
methods	Reviews, inspirations, walkthroughs	functional & user acceptance testing
Timing	Done before implementation	Done after development
Outcome	Ensure correctness of design & requirement	Ensure final product usability & functionality

Both are crucial for delivering high-quality software with minimal defects.

(14)

layered architecture model for an online judge system.

1. presentation layer (UI layer)
   ⇒ Responsibility: Handles user interactions (submission, results, login)
   ⇒ Ensures: Responsives UI, multi-device support.
2. Application layer (controller layer)
   ⇒ Responsibility: manages HTTP requests, authentication and session handling.
   ⇒ Ensures: Secure access and smooth user experience.
3. Business logic layer (Execution & Evalution layer)
   ⇒ Responsibility: stores problems submissi

user profiles and results.

⇒ ensures: Efficiency data retrieval backup and indexing.

How this architecture Helps:

⇒ Scalibility: Layer allow independent scaling

⇒ Maintainability: Modular structure simplifies updates and debugging.

⇒ performance: optimized query handling and caching enhance response time.

this model ensures a robust, scalable and maintainable online judge system.

(15)

UML class diagram for the given sequence Diagram

Based on provided UML sequence Diagram, the system consists of the following main classes:

1. student – Represents the user interacting with the system
2. login screen – Handles user input and initiates authentication.
3. Validate user – verifies credentials.
4. Database – store user information and class list.

class Diagram structure & Relationship:

1. student class
   - Attributes : studentID, name, password.
   - Methods : click login(), display class list()
2. login screen class

- Validate user (userid, password), showsuccess Message()
- Association: Interacts with validate user.

3. Validate User class.
   - Methods: checkCredentials (userid, password) getClasslist(), returnStatus()
   - Association: calls Database for authetication

4. Database class.
   - Attributes: UserID, password, classlist
   - Methods: matchUserID(), fetchClasslist().

Student
-studentID -name -password
+clicklogin() +displayclasslist()

Explanation of class Diagram

⇒ student interacts with Login Screen to initiate authentication.

⇒ LoginScreen communities communicates with validate User to check credentials.

⇒ Validate User queries the database for user authentication and class list retrieval.

⇒ once validated, the class list is displayed back to the student.

This is UML class Diagram.

(16)

Data flow Diagram (DFD) for Hospital Management system.

Level-0 DFD (context Diagram)

⇒ Actors: patient, Receptionist, Hospital Database.

⇒ processes: patient Registration, appointment scheduling, payment processing.

⇒ flow:

1. patient → provides Details → Reception

2. Reception → stores Data → Hospital Database.

3. Reception → Scheduling appointment → Doctor

4. Reception → Generates payment Receipt - patient.

Level-1 DFD (Detailed process Breakdown)

⇒ process in Detail:

1. Register Patient
2. Scheduling Appointment
3. Admit patient
4. process payment
5. file insurance & Reports

UML use case Diagram:

1. Register Patient
2. schedule Appointment
3. Admit inpatient
4. process payment
5. File insurance & reports

Diagram Explanation:

⇒ Actors interacts with reception to manage hospital service.

⇒ Receptionist connects with the hospital database to store & retrieve data.

⇒ Ensures efficient patient handling & documentation.

(17)

Quality Assurance (QA) vs Quality control (QC).

Explanation Description:

QA: focuses on preventing defects by improving processes. It ensures that quality standards and procedures are followed during development.

QC: is about detecting defects in the final product through testing and inspections.

Key Differences

Aspects	Quality Assurance (QA)	Quality control (QC)
Focus	process-oriented	product-oriented
Goal	Prevent defects	identify & fix defects
Approach	Proactive	Reactive
Methods	Audits, process improvement	testing, inspecting
Responsibility	Entire development	Dedicated QC/ testers

Impediments to QA & QC:

challenges in QA:

⇒ poorly defined processes

⇒ Lack of term training

⇒ Resistance to change.

challenges in QC:

Diagram Explanation:

⇒ Actors interacts with reception to manage hospital service.

⇒ Receptionist connects with the hospital database to store & retrieve data.

⇒ Ensures efficient patient handling & documentation.

(17)

Quality Assurance (QA) vs Quality control (QC).

Explanation Description:

QA: focuses on preventing defects by improving processes. It ensures that quality standards and procedures are followed during development.

QC: is about detecting defects in the final product through testing and inspections.

conclusion:

QA ensures quality at every stage, reducing risks, improving reliability and delivering a defect-free product

(19)

Rapid Application Development Model.

Defination:

The RAD model is an iterative and adaptive software development approach that emphasizes rapid prototyping, feedback-driven refinement, and fast delivery of functional software.

Key phases:

1. Business Modeling
2. Data modeling
3. process modeling
4. Application generation
5. Testing & Deployment.

principles:

⇒ user involvement & continuous feedback
⇒ component reusability & automation.
⇒ iterative development with Quick release.
⇒ parallel development by multiple teams.

advantage:

⇒ Faster delivery
⇒ High user-satisfaction
⇒ Flexibility → Easily accomodates changes
⇒ Reduced Risk → Early Error detection through iterations.

Conclusion:

The RAD model accelerates software development, ensures high quality through constant feedback, and is ideal for projects needing rapid adaptation to changing requirements.

(20)

Decision	x input	y input	Expected output
if (y==0)	5	0	"y is zero"
else if (x==0)	0	3	"x is zero"
for(int i=1; i<=x; i++)	0	2	no output
if (i%y==0) (true)	4	2	"2", "4"
if (i%y==0) (false)	4	3	"3"
Edge case: Negatively y	5	-2	"2", "4"
Edge case: Negative x	-3	2	No output

JUnit Test Class in Java:

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import java.util.ArrayList;
import java.util.List;
```

```
public class DecisionTest {
    private List<String> output;

    @Before
    public void setup() {
        output = new ArrayList<>();
    }
    private void println(String message) {
        output.add(message);
    }
private void process(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(String.valueOf(i));
}
}
}
}
```

```
@Test
public void TestYIsZero(){
    process(5,0);
    assertEquals(List.of("y is zero"), output);
}
@Test
public void testXIsZero(){
    process(0,3);
    assertEquals(List.of("x is zero"), output);
}
@Test
public void testLoopDoesNotRun(){
    process(0,2);
    assertEquals(List.of(), output);
}
@Test
public void testNumberDivisibleBy(){
    process(4,2);
    assertEquals(List.of("2","4"), output);
}
```

This JUnit test class follows the python structure and ensures full decision coverage using white box testing principles.

(21)

You can write JUnit 4 test case using Exception handling, setup function and timeout Rules as follows

Example production Code (simple Math operation)

code:

```
public class calculator {
public int divide(int a, int b) {
  if (b==0) {
throw new ArithmeticException("cannot
divide by zero");
  }
return a/b;
}
```

```
public int slowfunction() throws InterruptedException{
    Thread.sleep(2000);
    return 1;
  }
}
```

JUnit 4 code:

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

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

```
@test(expected = ArithmeticException.class)
public void testDivideByZero() {
    calculator.divide(10, 0);
}

@Rule
public Timeout globalTimeout = Timeout.millis
                               (1000);

@Test
public void testSlowFunction() throws
    InterruptedException {
    calculator.slowFunction();
}

@Test
public void testValidDivision() {
    assertEquals(5, calculator.divide(10,2));
}
}
```

Key features Used:

1. Setup Function (@Before): Initializes calculator before each test.
2. Exception Testing (@Test(expected)) Ensures division by zero throws an exception.
3. Timeout Rule (@rule Timeout): Fails if Execution exceeds the specified time.

This ensures early error ~~correction~~ detection in unit testing before system integration.

—o—