TU856/3
SOFTWARE TESTING

ASSIGNMENT 1 - DEVELOPMENT PLAN

18/03/2025

PAULINA CZARNOTA C21365726

1. Introduction

TunePal is a cross-platform music library and streaming service currently in its early development phase. The main objective of TunePal is to provide users with an efficient and seamless music streaming experience, including access to a shared song library, private song uploads, and premium song purchases.

To ensure high-quality software, testing will be an integral part of the Software Development Lifecycle (SDLC) using an Agile Scrum methodology. This document outlines the testing activities, required inputs, expected outputs, and justifications for each phase.

2. Testing Throughout the SDLC

Testing will be integrated into every phase of the SDLC, ensuring that functional, non-functional, security, and performance requirements are properly verified. Both automated and manual testing methods will be applied.

2.1 Requirements Analysis Phase

- **Inputs:**
  - Business requirements document
  - System specifications from the business analysis team
  - User stories describing core functional requirements
  - User acceptance criteria
- **Testing Activities:**
  - **Validate system requirements:** Ensure that all requirements are clear, concise, and testable by transforming them into actionable test cases.
  - **Identify ambiguous or missing requirements:** Work with the business analysts to identify any gaps in the requirements or unclear functionality descriptions.
  - **Define acceptance criteria:** Establish measurable acceptance criteria for each functional and non-functional requirement, making sure each requirement is testable.
  - **Confirm test coverage for edge cases and error handling:** Ensure thorough test coverage, including validation for special scenarios.
- **Outputs:**
  - **Refined system requirements document**: Any ambiguities identified during the requirements review will be addressed, and the document will be updated to include detailed, specific requirements.
  - **Test plan**: A high-level test plan will be created, outlining the coverage of the requirements, which includes functional, non-functional, security, and performance tests. This plan will also define edge cases and provide a roadmap for the subsequent testing phases.

Justification: Early involvement of testing ensures that all requirements are testable and that there is alignment between business goals and the testing objectives, avoiding costly misunderstandings later in the process.

2.2 Design Phase

- **Inputs:**
  - System architecture diagrams
  - UI/UX wireframes
  - Prototypes
  - Technical design specifications

- **Testing Activities:**
  - **Usability and accessibility reviews:** Ensure that the user interface (UI) meets accessibility standards (e.g., WCAG 2.1) and provides a user-friendly experience.
  - **Static analysis of UI flow:** Use tools such as SonarQube for static analysis to evaluate the UI design for any issues that may hinder performance or usability.
  - **Prototype testing:** Test the initial designs and prototypes to gather user feedback regarding UI/UX, identifying any design flaws that may impact later phases.
  - **Validate design alignment with business requirements and system feasibility:** Ensure that all components align with functional and technical expectations.

- **Outputs:**
  - **Usability feedback report**: Document and provide recommendations based on the usability and accessibility testing results.
  - **Design validation report**: A report verifying that the design meets the business requirements and is feasible for implementation. This will also include feedback on potential design improvements.

Justification: By addressing design flaws and ensuring accessibility during the design phase, the product will be more user-friendly and meet necessary legal requirements, reducing the cost of changes in later phases.

2.3 Development Phase

- **Inputs:**
  - Incremental software builds (e.g., versions 1.0, 1.1, etc.)
  - Initial database setup (e.g., song library database)
  - Backend API and front-end components

- **Testing Activities:**
  - **Unit testing:** Implement automated unit tests for the API using unittest framework and PyTest to validate the core functionalities of each method (e.g., adding a song, retrieving paginated songs).
  - **Bug tracking:** Use tools like YouTrack/Jira to document and track defects discovered during unit testing.
  - **Code reviews and static analysis:** Conduct code reviews and use static analysis tools to ensure adherence to coding standards and best practices (e.g., refactor for readability and efficiency).
  - **Ensure exception handling and error messages are robust:** Verify proper error handling to prevent crashes and improve user experience.
  - **Validate search functionality for case-insensitivity and special character handling:** Confirm that search queries return accurate results regardless of input case or special characters.

- **Outputs:**
  - **Unit test reports**: Reports containing details on which tests passed or failed, along with code coverage metrics.
  - **Defect tracking documentation**: Log of defects discovered during testing, their severity, and steps for fixing them.

Justification: Unit testing is critical to validate individual functionalities and methods early in the development phase, catching bugs before integration. This reduces the cost of fixing issues later in the cycle.

2.4 Integration Phase

- **Inputs:**
  - Integrated front-end and back-end modules
  - API endpoints
  - Testing environment setup (e.g., staging server)

- **Testing Activities:**
  - **API testing with Postman:** Validate that all API endpoints (e.g., add song, search songs) return the expected responses and handle errors correctly.
  - **Integration testing:** Test the interaction between the front-end and back-end services to ensure proper data flow between the two layers.
  - **Performance testing:** Ensure API responses meet the defined benchmarks (e.g., song details should load in under 2 seconds).
  - **Verify error handling and logging functionality:** Confirm that system logs accurately capture errors and performance issues.
  - **Test pagination to confirm proper data retrieval:** Validate that users can navigate through song listings without issues.

- **Outputs:**
  - **API test reports**: Detailed reports of each API test, including success or failure and response times.
  - **Integration test logs**: Detailed logs highlighting any issues with the interaction between the front-end and back-end.

Justification: Integration testing ensures that the components of TunePal, including the front-end and back-end, work together as expected, allowing for early detection of issues between system modules.

2.5 System Testing Phase

- **Inputs:**
  - Fully integrated system (UI, back-end, database)
  - Test cases derived from the system requirements

- **Testing Activities:**
  - **Functional testing:** Test that all features of the system work as expected. This includes testing song streaming, login functionality, song search, and other features.
  - **Performance testing:** Measure system performance to ensure that songs download within 5 seconds and the system performs under expected user load.
  - **Security testing:** Verify the system's compliance with security standards (e.g., user data protection, authentication).
  - **Validate compliance with non-functional requirements:** Ensure API response times remain below 200ms and scalability meets requirements.
  - **Confirm correct handling of invalid inputs:** Validate input validation to prevent incorrect data entry or crashes.

- **Outputs:**
  - **System test reports**: A comprehensive set of reports from functional, performance, and security testing, with detailed results and any defects found.
  - **Defects and resolutions**: Log any defects found during system testing and document their resolutions.

Justification: System testing validates that all features and performance requirements are met and ensures that the system is stable, functional, and secure before release.

2.6 User Acceptance Testing (UAT)

- **Inputs:**
  - Staging environment that mimics production
  - Real-world user scenarios

- **Testing Activities:**
  - **Beta testing:** Engage real users to test the application in a controlled environment, simulating real-world scenarios.
  - **Collect feedback:** Collect user feedback on expected vs. actual behaviour, and use that to identify any discrepancies between user expectations and system performance.
  - **Identify areas for final refinements before deployment:** Ensure that any necessary changes are made before the official release.
- **Outputs:**
  - **UAT feedback report**: Summarize user feedback, including any critical issues or improvements suggested by the users.
  - **Go/No-Go decision**: Based on UAT results, the project team will decide whether to release the system into production or make further improvements.

Justification: UAT ensures that the system meets user expectations in real-world conditions. This step is critical for ensuring user satisfaction and avoiding issues in production.

3. Risk Assessment

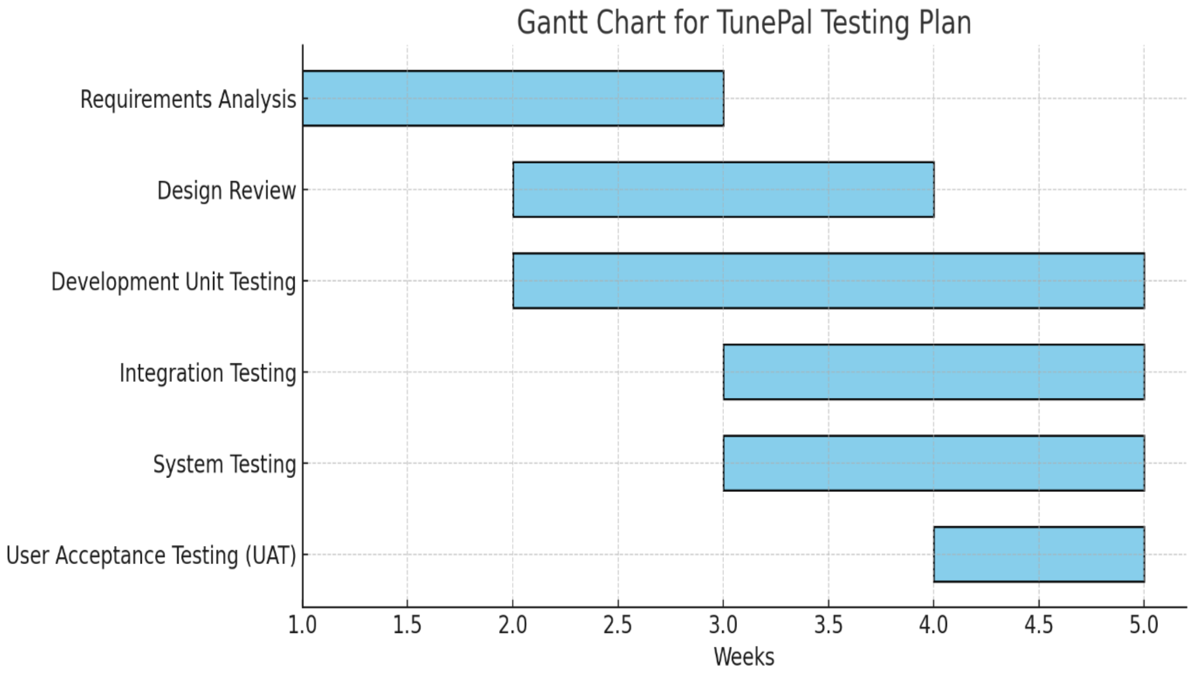
Risk	Impact	Mitigation
Incomplete Requirements	High	Early stakeholder engagement, requirement refinement, and continuous feedback loops to prevent gaps.
Unclear or Missing Test Cases	Medium	Cross-team collaboration, requirement traceability matrix, and regular test case reviews to ensure full coverage.
API Failures	High	Automated API testing, continuous integration of API changes, and real-time monitoring of API endpoints.
Performance Bottlenecks	High	Load testing with JMeter, performance profiling, and database query optimizations to resolve potential issues early.
Security Vulnerabilities	High	Regular security audits, penetration testing, input validation, and adherence to OWASP security best practices.

4. Test Prioritization

- **Critical Tests:** Prioritize testing user login, song search, and payment functionality, as these are core features essential for system usability.
- **API Testing:** Perform thorough API testing before front-end testing to ensure stability and reduce the risk of failures in data processing.
- **Integration Testing:** Validate seamless communication between front-end and back-end components before system-wide testing.

- **Performance and Security Testing:** Conduct load testing, stress testing, and security assessments after functional testing to verify system scalability and data protection.
- **Edge Cases and Error Handling:** Ensure graceful handling of invalid inputs, special characters, and boundary conditions to improve system robustness.

5. Testing Timeline



5.1 Testing Timeline Explanation

The testing plan for TunePal follows a 5-week structured schedule, ensuring a comprehensive approach to validating system functionality, performance, and security. This timeline aligns with the Software Development Lifecycle (SDLC) and Agile Scrum methodology, allowing for continuous testing alongside development.

Breakdown of the Testing Phases:

1. **Weeks 1-3: Early Testing and Development**

    - **Requirements Analysis (Weeks 1-3**): Ensures test cases are clear, complete, and aligned with business objectives before development begins.
    - **Design Review (Weeks 2-4):** Validates UI and UX compliance, accessibility, and design feasibility before moving into full development.
    - **Development Unit Testing (Weeks 2-5):** Starts early and continues iteratively, identifying and fixing issues during development.

2. **Weeks 3-5: System Integration and UAT**

    - **Integration Testing (Weeks 3-5):** Verifies seamless front-end and back-end communication, ensuring API stability and data consistency.

- **System Testing (Weeks 3-5)**: Performs functional, security, and performance validation before deployment.
- **User Acceptance Testing (UAT) (Weeks 4-5)**: Engages real users in beta testing, ensuring the final product meets user expectations before release.

5.2 Justification and Risk Mitigation

By following this structured timeline, TunePal undergoes rigorous validation before its release, ensuring all defects are identified early. This approach minimizes risks while maintaining efficiency in development, reducing post-release issues and ensuring a high-quality final product.

6. Testing Tools & Resources

Testing Type	Tool
Unit Testing	unittest, PyTest
API Testing	Postman
Bug Tracking	YouTrack, Jira
Performance Testing	JMeter
Code Review	SonarQube
Security Testing	OWASP ZAP

Justification:

- **PyTest and unittest** ensure automated and reliable unit testing, allowing for early detection of defects.
- **Postman** simplifies API request validation and response verification.
- **YouTrack and Jira** provide efficient bug tracking, helping teams manage issues effectively.
- **JMeter** is used for performance benchmarking to identify bottlenecks.
- **SonarQube** enhances code quality by identifying potential vulnerabilities and bad practices.
- **OWASP ZAP** conducts security vulnerability scanning to ensure application security compliance.

7. Conclusion

This development plan provides a structured, Agile-compliant approach to testing TunePal at every stage of the software development lifecycle. With rigorous validation in each phase, the product is ensured to meet functional, performance, and security requirements. This structured approach minimizes risks, enhances quality, and guarantees a seamless user experience upon release.