**Professor Panic: The Academic Exodus**

Software Quality Assurance Plan

Version 1.0.2

12/1/2023

**Professor Panic: The Academic Exodus**

**SOFTWARE QUALITY ASSURANCE PLAN**

**12/1/2023**

**WESTERN NEW ENGLAND UNIVERSITY**

**CPE -425, 525 SOFTWARE ENGINEERING**

**Team #1 Project**

DOCUMENT APPROVALS

|  |  |  |
| --- | --- | --- |
| **NAME** | **SIGNATURE** | **DATE** |
| David Melanson (Project Lead) |  |  |
| Kalyan Bhagavan |  |  |
| Matthew Dingman |  |  |
| Philip Brehart |  |  |
| Logan Paranto |  |  |
| Sydney Adams-Jones |  |  |

# DOCUMENT CONTROL / REVISION HISTORY

|  |  |
| --- | --- |
| **TITLE** | Quality Assurance Plan |
| **REPOSITORY LOCATION** | GitHub |
| **URL** | https://github.com/PJB01/Academic-Exodus |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/28/2023 | 1.0.0 | Initial Draft | KBT |
| 11/06/2023 | 1.0.1 | Minor Revisions | DM |
| 12/1/2023 | 1.0.2 | Minor Revisions | All |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[DOCUMENT CONTROL / REVISION HISTORY 2](#_Toc929045097)

[1. INTRODUCTION 3](#_Toc2027945125)

[1.2 Scope 3](#_Toc181392276)

[1.3 Overview 3](#_Toc738688021)

[1.4 Definitions 3](#_Toc1589051152)

[2. QUALITY OBJECTIVES 3](#_Toc872237076)

[2.1 Product Quality 3](#_Toc389732973)

[2.2 Process Quality 3](#_Toc1220530992)

[3. QUALITY RELATED DOCUMENTATION 3](#_Toc538610689)

[3.1 Quality records 3](#_Toc1188712439)

[3.2 Meeting Minutes 3](#_Toc1279307525)

[3.3 Formal Deliverable Review and Acceptance 3](#_Toc1527755252)

[3.4 Code Quality Audits (IV&V) 3](#_Toc832408724)

[3.5 Defect Status and Severity Summaries 3](#_Toc1063009942)

[3.6 Testing Standards, Evaluation Summaries and Reports 3](#_Toc781583211)

[4. STANDARDS AND GUIDELINES 3](#_Toc1362127859)

[5. REVIEWS AND AUDITS 3](#_Toc1638244206)

[5.1 Project Reviews 3](#_Toc1548669749)

[5.2 Formal Reviews 3](#_Toc436114788)

[5.3 Informal Reviews 3](#_Toc956557630)

[6. TOOLS, TECHNIQUES, AND METHODOLOGIES 3](#_Toc960693047)

[6.1 Tools 3](#_Toc385108595)

[6.2 Techniques 3](#_Toc1725476734)

[6.3 Methodologies 3](#_Toc653433453)

[7. METRICS 3](#_Toc965240797)

[8. EVALUATION AND TEST 3](#_Toc1952389300)

[8.1 Integration Testing 3](#_Toc1419676840)

[8.1.1 Objective 3](#_Toc1857311061)

[8.1.2 Benefits 3](#_Toc468623679)

[8.1.3 Responsibility 3](#_Toc1733338613)

[8.1.4 Activities 3](#_Toc729242437)

[8.2 Regression Testing 3](#_Toc1633931084)

[8.2.1Objective 3](#_Toc1920509173)

[8.2.2 Benefits 3](#_Toc1095751706)

[8.2.3 Responsibility 3](#_Toc50624460)

[8.2.4 Activities 3](#_Toc501299685)

[8.3 User Acceptance Testing (UAT) 3](#_Toc877243038)

[8.3.1 Objective 3](#_Toc206938891)

[8.3.2 Benefits 3](#_Toc1551053011)

[8.3.3 Responsibility 3](#_Toc1203577126)

[8.3.4 Activities 3](#_Toc1666028687)

[9. PROBLEM RESOLUTION AND CORRECTIVE ACTION 3](#_Toc280809101)

[10. CONFIGURATION MANAGEMENT 3](#_Toc335643751)

[11. TRAINING 3](#_Toc390398)

# INTRODUCTION

* 1. **Purpose**

The purpose of this plan is to specify how Software Quality Assurance (SQA) and Project Quality Assurance (PQA) will be performed throughout **Professor Panic: The Academic Exodus Project.** This plan describes the SQA activities to be performed and designates a set of standardized techniques for performing those activities.

# Scope

This plan applies to all stages of the Professor Panic: The Academic Exodus Project. Software Quality Assurance Plan will continue throughout the project as needed, based on the results of project execution and in accordance with best practices. The plan will be reviewed and updated, if needed, on at least a bi-weekly basis by SQA.

# Overview

The Software Quality Assurance Plan is organized into the following sections:

1. **Quality Objectives -** Describes the objectives of the project with regard to product and process quality.
2. **Review Plan -** Describes the type of reviews to be conducted to ensure product quality.
3. **Quality-Related Documentation -** Describes the documentation produced to support quality assurance activities.
4. **Metrics -** Defines high-level metrics employed during the project.
5. **Evaluation and Test -** Describe the type of testing to be performed to evaluate and verify product (system) quality.
6. **Problem Resolution and Corrective Actions –** Describes the management and resolution of problems and issues.
7. **Tools, Techniques, and Methodologies –** Identifies the tools to be used for quality-related activities, techniques for quality assurance, and methodologies or procedures to be adhered to when executing and evaluating project activities.
8. **Quality Records -** Describes the records that will be maintained to track formal Quality Assurance activities performed.
9. **Training -** Describes training necessary to support Quality Assurance activities.

# Definitions

The following terms are used in this document.

* + - * **Formal Artifact Review (FAR)** - A review process used to obtain formal acceptance of project artifacts. This is described in the Deliverable Acceptance Plan.
      * **Integration Testing** - Testing of separately developed components that have passed unit test to ensure they function as designed when integrated together.
      * **Build and Deployment Testing** - Validation of build and deployment procedures to ensure they are correct.
      * **Regression Testing** - Regression testing is executed in a dedicated testing environment, which replicates production environment setup. Testing starts on a scheduled date after functional code is merged with the existing code base. After code merger, no new builds are accepted, which allows testing to occur on a stable code base. If a defect is encountered, depending on the severity, we either postpone it or request another build. If postponed, the resolved build will be added to a future release. If another build is accepted, we then start our regression cycle over again. Once regression testing is completed and signed off, the code would then be production-ready and approved for deployment.
      * **User Acceptance Testing (UAT)** - User Acceptance Testing is performed to ‘accept’ the system prior to moving to production. Acceptance test includes the exercise of functional test cases to ensure there are no significant functional problems present in the system, such as interactions and movement not operating as designed to.
      * **Independent Verification and Validation (IV&V)** - The process of independently checking that a software system meets specifications and that it fulfills its defined requirements and specifications.

# QUALITY OBJECTIVES

Quality management addresses the two aspects of system development: Product Quality and Process Quality.

For this project, product “quality” means “conformance with requirements”, i.e., software and non-software products adhere to the defined business and technical specifications.

# Product Quality

Ultimately, the product quality goal is to implement a solution that adheres to approved requirements. The software is the primary ‘product’ being provided; however, artifacts needed to develop, implement and maintain the software, are also part of the product. The quality of these artifacts is directly relatable to the quality of project outcomes. Assessment of artifacts will be conducted throughout the project lifecycle. Two primary factors are to be used to assess each artifact:

* + - How well it meets the defined acceptance criteria.
    - How it measures against established standards or guidelines defined for the project

The Product Quality objective is to implement a solution that operates per the agreed-upon requirements. This will be achieved through effective requirement management, quality processes, formal and informal reviews, evaluation and testing as well as other monitoring and controlling activities, as defined in this plan.

# Process Quality

Process Quality refers to the degree to which an accepted process is implemented and followed in order to produce project artifacts and deliverables. Quality products require establishing and following processes that build quality during development. Through the use of quality methods, tools, and techniques, development can be planned, measured, and improved as necessary. The success of process quality is ultimately measured both by the degree to which processes are adhered to, as well as the level of quality of the artifacts (and ultimately the product) produced by those processes.

The Process Quality objectives for the project are:

* + - Ensure that processes are defined, and project participants follow those processes.
    - Ensure the work performed follows approved project plans.
    - Pre-define and agree on acceptance criteria for project artifacts.
    - Ensure that processes defined in this plan adequately measure and track product quality across all project activities.

# QUALITY RELATED DOCUMENTATION

A variety of documentation will be produced during the project to satisfy the requirements of the project. Each of these documents is created to support the goals of the project.

# Quality records

The following quality records, detailed in the following sub-sections, will be maintained:

* + - Meeting Minutes
    - Formal Deliverable Review and Acceptance documentation
    - Code Quality Audits (Independent Verification and Validation)
    - Defect Status and Severity Summaries
    - Test Evaluation Progress Status Summaries

# Meeting Minutes

Meeting minutes document discussions, decisions, and action items from project meetings. They provide a historical record of project progress, issue resolution, and key discussions. These records help ensure that all stakeholders are informed and aligned.

# Formal Deliverable Review and Acceptance

This set of records includes documentation related to the formal review and acceptance of project deliverables. It provides evidence that deliverables meet the specified requirements and have been approved by stakeholders.

# Code Quality Audits (IV&V)

1. **Code Quality Audits:**

Code Quality Audits focus on the examination of the source code to assess its quality, maintainability, and adherence to coding standards. These audits can be both manual and automated. The objectives of Code Quality Audits include:

* *Identifying Coding Errors:* Detecting syntax errors, logical errors, and other issues in the source code.
* *Adherence to Coding Standards:* Ensuring that the code complies with coding standards, style guides, and best practices.
* *Performance and Efficiency:* Assessing code for performance bottlenecks, memory leaks, and suboptimal algorithms.
* *Documentation*: Checking the presence and quality of comments and documentation within the code.

2. **Benefits of Code Quality Audits:**

* *Mitigate Risks:* Audits help identify and mitigate risks early in the development process, reducing the likelihood of costly issues in later stages.
* *Quality Assurance*: Ensure that the software aligns with quality standards, requirements, and customer expectations.
* *Process Improvement*: Audits provide insights into the development process, enabling continuous improvement and adherence to best practices.
* *Compliance*: Verify compliance with industry standards, regulations, and organizational guidelines.

# Defect Status and Severity Summaries

Defect records document the identification, tracking, and resolution of software defects or issues. Summaries of defect status and severity help prioritize and manage the resolution process.

# Testing Standards, Evaluation Summaries and Reports

Records related to test progress and evaluation summarize the status of testing activities, including test plans, test cases, test execution, and the identification of defects. They help ensure that testing is on track and issues are addressed.

# STANDARDS AND GUIDELINES

Project activities will use existing standards and guidelines for developing work products. The establishment and adherence to standards and guidelines strengthens the quality of analysis, design, development and implementation efforts. Standards and guidelines help project team members deliver consistent, high-quality artifacts and provide project reviewers with criteria for assessing those artifacts.

At minimum, the following standards and guidelines will be established for the project and followed by the project team:

* Documentation Guidelines
* Programming Standards and Guidelines
* Test Standards and Guidelines

Within each standard or guideline, checklists will be established to verify artifacts meet the appropriate standards and/or guidelines.

# REVIEWS AND AUDITS

It is a goal of the project to build in quality through multiple types of reviews and audits. There are three types of reviews, and the type used will depend on the artifact.

# Project Reviews

Project reviews are systematic assessments of the entire software development project at key milestones or critical phases of the project's lifecycle. These reviews serve as comprehensive evaluations to ensure that the project is on track, adheres to its goals and objectives, and complies with established standards and requirements. They involve a structured examination of project documentation, deliverables, schedules, budgets, and overall progress. Project reviews help identify any deviations from the project plan, assess the quality of work performed, and make informed decisions regarding project scope, budget, and timeline. The outcomes of project reviews can lead to corrective actions, adjustments to project plans, or re-evaluation of project objectives, ultimately ensuring successful project delivery.

# Formal Reviews

Formal reviews are structured, planned, and documented evaluations of software work products and processes. These reviews are typically conducted at predefined stages of the software development lifecycle, such as requirements, design, code, and testing phases. They involve a systematic examination of project artifacts, including requirements documents, design specifications, code modules, and test plans. Formal reviews follow a defined set of procedures and criteria, and they are typically led by designated reviewers who assess the quality and compliance of the work product with specified standards and requirements. Common types of formal reviews include Requirements Review, Design Review, Code Review, and Test Plan Review. The key objective of formal reviews is to identify and rectify issues, inconsistencies, and non-compliance early in the development process, contributing to improved software quality and reduced rework.

# Informal Reviews

Informal reviews are less structured, less formal assessments of software work products and processes. They are often ad-hoc and may occur more frequently during the development process compared to formal reviews. Informal reviews are generally less rigorous and may not follow a predefined set of procedures. They involve discussions, feedback, and assessments of software artifacts, such as code snippets, design sketches, or user interface mock-ups. While they lack the formality of their formal counterparts, informal reviews offer flexibility and the advantage of quick feedback. They are particularly useful for identifying minor issues, facilitating collaboration among team members, and encouraging knowledge sharing. Informal reviews can be an integral part of agile development practices, promoting continuous improvement and timely issue resolution.

# TOOLS, TECHNIQUES, AND METHODOLOGIES

This section details the different tools and techniques that are to be utilized in performing the Quality Assessments for this project.

# Tools

In the context of a software development project, tools refer to the software applications and systems used to facilitate various tasks and activities. These tools can encompass a wide range of functions, from project management and version control to code development and testing. Tools like Jira, Bitbucket, Confluence, GitHub, Microsoft Word, and Excel are commonly utilized for project management, documentation, version control, and data analysis. Selecting the right tools is crucial for streamlining project workflows, enhancing collaboration, and ensuring the efficient management of project artifacts and resources.

# Techniques

Techniques in a software project context are the specific methods, practices, or approaches used to perform certain tasks or achieve project objectives. These techniques can vary across different project phases, such as requirements gathering, design, coding, and testing. For example, software development techniques may include test-driven development and agile methodologies. The choice of techniques depends on the project's goals, requirements, and the development team's expertise. Adopting effective techniques can lead to improved code quality, faster development cycles, and more efficient problem-solving.

# Methodologies

Methodologies are overarching frameworks or sets of guidelines that prescribe how a project should be planned, executed, and controlled. They provide a structured approach to managing the entire project lifecycle. Common software development methodologies include Agile, Scrum, Waterfall, and DevOps. The selection of a methodology depends on project characteristics, such as size, complexity, and customer requirements. Each methodology has its own set of principles and practices that guide project teams in decision-making, communication, and the delivery of project outcomes. Choosing the right methodology is critical for aligning the project with organizational goals and ensuring successful project delivery.

# METRICS

Quality metrics play a crucial role in ensuring that a software development project maintains a high standard of quality throughout its lifecycle. Defining these metrics upfront is vital to establish clear expectations and objectives. Here are some quality metrics commonly employed during the software development lifecycle:

1. **Defect Density:**

Defect density is calculated as the number of defects identified in the software divided by the size of the code or other relevant units (e.g., lines of code, function points). It provides insight into the quality of the code and helps track improvements over time.

1. **Test Pass Rate:**

The test pass rate shows the percentage of test cases that pass successfully. It reflects the software's overall quality and readiness for release.

1. **Requirements Traceability:**

Ensuring that each requirement is traceable to design, implementation, and testing activities helps maintain alignment with project goals, the user’s manual, and customer expectations.

1. **Customer Satisfaction:**

Customer satisfaction surveys and feedback are valuable metrics for assessing the software's quality from the end-users perspective.

These quality metrics help in objectively evaluating the quality of the software throughout the project's lifecycle. Defining and agreeing upon these metrics upfront provides a clear framework for quality assurance and reduces potential debates and delays during critical project milestones.

# EVALUATION AND TEST

The objective of the activities in this section is to validate the application meets requirements.

# Integration Testing

## Objective

Integration testing is performed to validate that different components or modules of the software interact correctly when combined. It focuses on identifying issues related to the interaction between software components.

## Benefits

* Component Interaction: It ensures that components integrate seamlessly, supporting the integration of features and functionalities across the software.
* Early Issue Detection: Detecting integration issues early reduces the risk of complex defects in later phases of development.

## Responsibility

Developers are responsible for conducting integration testing.

## Activities

1. Plan testing approach.
2. Identify and address issues related to data flow, communication, and interface compatibility.
3. Execute integration test cases to validate correct interactions.

# Regression Testing

## 8.2.1 Objective

Ensure the delivered application release meets all the functional requirements as defined in the requirements specification documents.

## Benefits

1. Defect Prevention: By verifying that code changes do not break existing functionality, regression testing helps prevent the introduction of new defects.
2. Software Reliability: It ensures that the software remains reliable and maintains its expected behavior.

## Responsibility

Developers analyze and resolve logged defects and depending on severity move it future releases.

## Activities

1. Re-run previously executed test cases that cover various aspects of the software.
2. Compare new test results with historical results to detect deviations.
3. Prioritize testing efforts on areas affected by recent code.

# User Acceptance Testing (UAT)

## Objective

User Acceptance Testing (UAT) validates that the software aligns with user requirements and is ready for release to the end-users. It focuses on ensuring that the software meets the expectations of business stakeholders and end-users.

## Benefits

1. User Satisfaction: UAT ensures that the software aligns with user expectations, leading to high user satisfaction.
2. Readiness for Release: It validates that the software is ready for deployment and meets business objectives.

## Responsibility

End-users, business stakeholders, and quality assurance teams are responsible for conducting UAT.

## Activities

1. Define test cases based on user requirements, scenarios, and real-world use cases.
2. Execute test cases to validate that the software meets user expectations.
3. Gather user feedback and make necessary adjustments based on their acceptance.
4. Determine the software's readiness for release based on user feedback and acceptance criteria.

# PROBLEM RESOLUTION AND CORRECTIVE ACTION

Problem resolution and corrective action are essential aspects of quality management and project management. They involve identifying and addressing issues, defects, and deviations from project requirements or standards to ensure that the project stays on track and produces a high-quality outcome. Here's an overview of problem resolution and corrective action:

**Problem Resolution:**

Problem resolution involves the identification, documentation, and analysis of issues, defects, or discrepancies that arise during the project's execution. These issues can pertain to various aspects of the project, such as requirements, design, development, testing, or project management. The key steps in problem resolution include:

* + 1. **Issue Identification:** Recognizing and documenting issues as they arise. This can be done through issue-tracking systems, regular meetings, or project reports.
    2. **Issue Analysis**: Investigating the root causes of the issues. This may involve examining project documentation, code, and communication records.
    3. **Prioritization**: Assigning priorities to issues based on their severity and impact on the project. High-priority issues may require immediate attention.
    4. **Resolution Planning**: Developing action plans for addressing issues, including defining corrective measures and assigning responsible individuals or teams.
    5. **Issue Resolution**: Implementing the defined corrective actions to address the issues. This may involve code changes, process improvements, or rework.
    6. **Verification**: Ensuring that the corrective actions are effective and that the issues have been resolved. This may involve retesting, reviews, or inspections.

**Corrective Action:**

Corrective action is the process of taking steps to rectify and prevent the recurrence of issues or defects identified in the project. The primary objectives of corrective action are to ensure the project gets back on course and that similar issues do not surface in the future. Key elements of corrective action include:

* + 1. **Root Cause Analysis**: Identifying the fundamental reasons behind the issues. This analysis aims to understand the underlying factors that led to the problem.
    2. **Action Planning**: Develop a corrective action plan that outlines the steps to be taken to address the root causes. This plan should include specific tasks, responsible parties, and deadlines.
    3. **Implementation**: Executing the corrective actions as per the plan. This may involve process improvements, changes to project tasks, or additional testing.
    4. **Monitoring**: Continuously tracking the progress of corrective actions to ensure they are being carried out as planned.
    5. **Preventive Action**: In addition to resolving existing issues, preventive action involves putting measures in place to avoid the recurrence of similar issues in the future. This might include process enhancements, training, or updated documentation.
    6. **Documentation**: Properly documenting the entire process, including issue identification, root cause analysis, corrective action plans, and their outcomes. This documentation serves as a valuable reference for future projects and audits.

Problem resolution and corrective action are crucial for maintaining project quality and ensuring that the project stays aligned with its objectives and requirements. They promote continuous improvement and contribute to the overall success of the project. Effective problem resolution and corrective action processes lead to higher-quality deliverables and more efficient project management.

# CONFIGURATION MANAGEMENT

Configuration management (CM) is a fundamental discipline in software engineering and project management that involves tracking and controlling changes to the project's components, ensuring consistency, and enabling efficient management of the project's configuration items. Here's a comprehensive overview of configuration management:

Key Components of Configuration Management:

* + 1. **Configuration Items (CIs):** CIs are the individual components of a software project that need to be managed. This can include source code, documentation, design specifications, hardware components, and more. Each CI is uniquely identified and tracked.
    2. **Baseline**: A baseline is a snapshot of the configuration items at a specific point in time. It represents a stable state of the project and is used as a reference for future development. Common baselines include the initial baseline, functional baseline, and product baseline.
    3. **Change Control**: Change control involves the processes for requesting, reviewing, approving, and implementing changes to the project's configuration. It ensures that changes are well-managed and don't negatively impact the project.
    4. **Version Control**: Version control is a subset of configuration management that focuses on tracking changes to individual files or code components. It provides the ability to manage different versions of the same CI.

Key Activities in Configuration Management:

* + 1. **Identification**: Identifying and naming configuration items, ensuring each has a unique identifier.
    2. **Change Management**: Managing changes to the configuration items, including assessing the impact of changes, tracking change requests, and making decisions on whether to approve or reject changes.
    3. **Configuration Status Accounting**: Maintaining a record of the current status of all configuration items and baselines. This includes information on versions, changes, and release status.
    4. **Configuration Audits**: Conducting periodic reviews and audits to ensure that the configuration items and documentation align with the project's requirements and standards.
    5. **Baseline Management**: Creating and maintaining different types of baselines to represent various project milestones or stages.

**Benefits of Configuration Management**:

* + 1. Consistency: CM ensures that all project components are consistent and that changes are managed in a controlled manner.
    2. Traceability: CM provides traceability, allowing you to track changes back to their sources and understand their impacts.
    3. Quality Assurance: By ensuring that all project artifacts and components are well-managed, CM contributes to improved software quality.
    4. Efficiency: CM reduces the risk of configuration-related errors and enhances the efficiency of project development and maintenance.
    5. Compliance: It helps maintain compliance with industry standards and regulations by providing a clear record of project activities.

Tools for Configuration Management:

Configuration management is often supported by tools and software, such as version control systems like Git, Jira, Bitbucket, and Github and integrated development environments (IDEs) with built-in CM features.

# TRAINING

Training is a fundamental component of the Quality Assurance plan. As roles and responsibilities are identified over the course of the project technical and business training will be provided to all staff associated with the project. Additionally, “how to” user guides will be created and distributed to support the training.