Tests
Software Quality Plan

4/1/2022

Purpose:

Authors: Adam Amott, Jared Steadman, David Swearingen, Alec Swainston

Approved By: Brother Clements

Revision History

Date	Authors	Revision	Description
Week 5	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	0.05	Added Sections 1-3
Week 6	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added section 9
Week 7	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added section 7
Week 8	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added section 6
Week 9	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added sections 3 and 11
Week 10	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added sections 10 and 12
Week 11	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added sections 8 and 14
Week 12	Adam Amott, Jared Steadman, David Swearingen, Alec Swainston	6	Added section 13

Table of Content

Section 1: Purpose

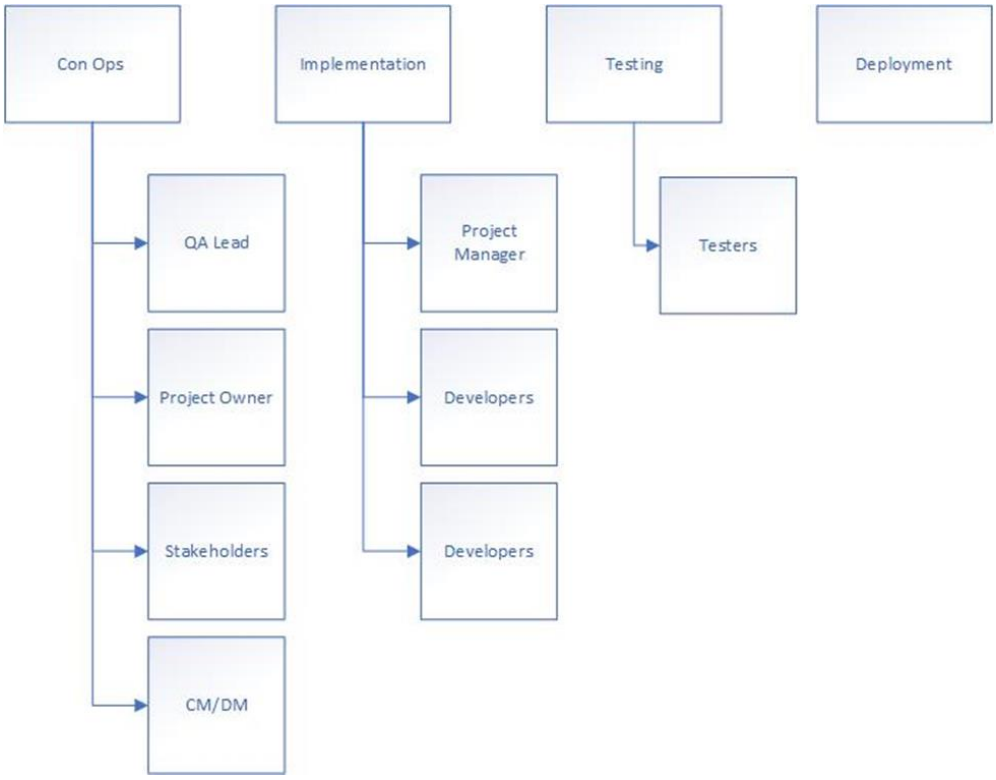
We are designing a new android app for a local store. They would like to give their customers the ability to do their shopping directly in the app. They referenced Walmart and Target apps as references for features and design ideas.

Section 2: Reference Documents

Name	Location
730-2002 Mobile QA Model	https://drive.google.com/file/d/1a55Pk8LLGatqDmKKiwU31JgkKV5H2gwW/view?usp=sharing
IEEE Standard for System, Software, and Hardware Verification and Validation," in IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017) , vol., no., pp.1-260, 29 Sept. 2017, doi: 10.1109/IEEESTD.2017.8055462	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/8055462
IEEE Std 730-2014 (Revision of IEEE Std 730-2002) , vol., no., pp.1-17, 13 June 2014, doi: 10.1109/IEEESTD.2014.6835311.	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/6835311
IEEE Standard Software Reviews and Audits," in IEEE Std 1028-1988 , vol., no., pp.1-36, 30 June 1989, doi: 10.1109/IEEESTD.1989.81607.	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/29123
IEEE Std 730-2002 (Revision of IEEE Std 730-1998) , vol., no., pp.1-8, 23 Sept. 2002, doi: 10.1109/IEEESTD.2002.94130.	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/1040117
L. Westfall Certified Software Quality Engineer Handbook, ASQ Quality Press, 2009-09-01,	https://ebookcentral.proquest.com/lib/byui/detail.action?docID=3002591
90003-2018 - ISO/IEC/IEEE International Standard - Software engineering -- Guidelines for the application of ISO 9001:2015 to computer software	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/8559961
C. Redzic and Jongmoon Baik, "Six Sigma Approach in Software Quality Improvement," Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06), 2006, pp. 396-406, doi: 10.1109/SERA.2006.61.	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/1691407
K. N. and H. Naganna, "CMMI and Six Sigma-Relationship & Integration," 2009 International Conference on Signal Acquisition and Processing, 2009, pp. 174-176, doi: 10.1109/ICSAP.2009.59.	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/5163849
E. Bersoff, "Elements of software configuration management," IEEE Trans. on Softw. Eng., vol. SE-10, no. 1, pp. 79-87, Jan. 1984	https://content.byui.edu/items/8bcbc45e-012a-48e0-800d-7082dd962f15/1/?.vi=file&attachment.uuid=ebf74848-b4af-4336-8a5b-c5f1c7aab5fc
12207-2008 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Software life cycle processes	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/6042287
"IEEE Standard for Software Configuration Management Plans," in IEEE Std 828-2005 (Revision of IEEE Std 828-1998) , vol., no., pp.1-30, 12 Aug. 2005, doi: 10.1109/IEEESTD.2005.96464.	https://ieeexplore-ieee-org.byui.idm.oclc.org/document/1502775

Section 3: Management

3.1 Organization



3.2 Tasking

- Concept of Operations
  - Verify the concept is in line with the project.
  - Verify that the objectives are reasonable and relevant.
  - Evaluate the risks involved and mitigation steps for them.
- Requirements
  - Verify the system requirements are up to date and relevant to the projects parameter.
  - Evaluate software version(s) compatibility.
  - Build a test environment to test the project architecture works properly.
  - Conduct requirement elicitation to verify the requirements match expectations.
- Design
  - Analyze any existing competition to be distinguished from other products.
  - Test the program design with test groups.
  - Plan and execute prototyping stages.
  - Conduct questionnaire tests for product validation.
- Implementation
  - Set agile sprints for each feature of the application
  - Plan and conduct tests of modules of the code.
  - Build a test environment to test the build of the product.
- Testing
  - Conduct unit testing.

    - Plan and execute simulation tests.
    - Plan and execute integration tests.
    - Plan and execute system and acceptance testing.
- Deployment
    - Plan and test version-release.
    - Decide time periods for each version release.

3.3 Roles and Responsibilities

QA Lead: Validation of requirements and processes.

Project Owner: Review documents for coverage of risks.

Stakeholder: Provide resources for the project.

Project manager: Help coordinate and bring information on any new customer requests

Developers: Work closely with QA as they design each section of the project

QA: Provide input on the direction to help match the desired outcome of the customer.

Testers: Write and conduct tests for the code to verify its performance and report failure and bugs.

3.4 Estimated Resources

Positive costs

Positive costs are any cost that bring benefit to the project in the way of keeping massive costs from happening in the future.

- Appraisal costs
    - Appraisal costs will cover items that are related to preparation and planning. Appraisal costs will help us make informed decisions when setting project goals
- Prevention costs
    - Any cost that directly ties to risk mitigation will be labeled under prevention

Negative costs

These are costs that come from items of failure inside or outside the company

- Internal costs
    - Internal costs come from inside the project such as an employee not completing their work which leads to technical debt
- External costs
    - External costs cover items such as changes in licensing fees for third party software or accidents outside our control.

Section 4: Documentation

4.1 Purpose

This section lists the documents required by the project.

4.1 Industry Standards

- Internal company standards and procedures
- External company standards and procedures

4.2. Company Policies

- Internal company policies and standards
- External policies

4.3 Project Plans

- Project Charter
- Software Requirements Specification (SRS)
- Software Design Document (SDD)

4.4 Project Process

- Project Management Plan
- Software Quality Assurance Plan (SQAP) – This document
- Software Configuration Management Plan

4.5 Other Documentation

- Copyright Documentation
- Patent Documentation
- User Documentation

Section 5: Standards, Practices, Conventions, and Metrics

5.1 Purpose

- Overview – brief description of this section
    - Standards –
        - 1012-2004 Standard for Software Verification and Validation
        - 730-2002 Standard for Software Quality Assurance Plans
        - 828-2005 Standard for Software Configuration Management Plans
        - 1028-2008 Standard for Software Reviews and Audits
    - The standards outlined in this document must be reviewed whenever a change is proposed. This will keep the standards in line with our project goals.

5.2 Content – describe this section in the frame of the project.

- Documentation standards
    - Acceptable Use Policy (AUP)
        - Comply with Legal Requirements
        - Monitor all employees
        - Notify of Monitoring
    - Change and Configuration Management Policy
        - Track changes of hardware, software, and infrastructure configurations
    - Privacy Policy
        - All Personally Identifiable Information will be kept secure and not shared.
    - User Management Policies
        - Limit network access.
        - Limit Software Configuration
    - Password Policies
        - There should never be the same password for two different systems
        - Passwords never contain words
        - Users must change passwords every year
        - Do not use part of name or email address.
- Design standards
    - IEEE 1016-1998
    - IEEE Software Design V&V Standards
- Coding and Commentary standards
    - Imports
        - Use imports for modules and packages only. Not classes or functions.
    - Packages
        - Import each package using the full pathname from modules
    - Exceptions
        - Exceptions are allowed but must be used carefully
    - Global Variables
        - Avoid Global Variables
    - Line-Length
        - 80 character maximum

    - Comments
        - Always use block comments for tricky areas of code.
        - Make sure any other developer can understand comments.
- Testing standards and practices
    - Please refer to Section 7 of this document to review testing standards and practices.
- Selected software quality assurance product and process metrics
    - Week 04 Model and Characteristics
    - CMM: 2 SQA, 4 QPM SQM

Section 6: Software Reviews

6.1: SRS and SDD reviews

- These are included so we know if the software conforms to the requirements set in these documents

6.1.1 Requirements Plan

- Based on the requirements laid out at the beginning of the project is the SRS, SDD, and CONOPS we will be using multiple plans to cover each section.

6.1.2 Release Plan

- Review the current release and plan what needs to be added and when.
- Create an outlined schedule for updates.

6.2 Required Reviews

6.2.1 Software Specifications Review

Based on the requirements laid out at the beginning of the project is the SRS and CONOPS we will be using multiple plans to cover each section. Once each plan is complete it will be reviewed to ensure it meets the requirements we laid out for the project.

- Below are the plans that will be created for each section a Fagan's inspection will take place upon each of their completion to ensure quality and compliance of the plans.
    - Software Qualification Test Plan
    - Software Acceptance Test Plan
    - Software Component Test Plan
    - Software Integration Test Plan

6.2.3 Detailed Design Review

Again, based on the requirements laid out at the beginning of the project is the SRS and CONOPS we will be using multiple plans to cover each section. Once each plan is complete it will be reviewed to ensure it meets the requirements we laid out for the project.

Throughout the design process developers and the QA team will perform inspections without meetings. This ensures quality is being maintained daily and lowering technical debt.

Active design reviews ensure that the design of components and features align with the goals of the project and what we have completed thus far. Active Design Reviews will be used for the following plans:

- Software Component Test Plan
- Software Integration Test Plan

Verification and Validation process that will be performed include:

- Interface Analysis
- Traceability Analysis

- Criticality Analysis
- Hazard Analysis
- Security Analysis
- Risk Analysis

6.2.4 Verification and validation plan review

Verification and validation shall be conducted as Active Design Review. Component testing shall be conducted as each unit (a unit being the smallest portion of code able to be tested) is completed and sent to Quality Assurance for testing. Integration testing shall take place before implementing a unit of code into the staging environment. Acceptance testing shall be conducted at the end of construction prior to release.

These items are to be reviewed at the above-mentioned times during Software Construction V&V to ensure the product meets stakeholder needs:

- Software component testing
- Software integration testing
- Software qualification testing
- Software acceptance testing
- Software component test procedure
- Software integration test procedure
- Software qualification test procedure
- Software acceptance test execution
- Hazard analysis
- Security Analysis
- Risk Analysis

6.2.5 Post-Implementation Reviews

These items are to be reviewed during Integration and Acceptance V&V to ensure the product meets stakeholder needs:

Acceptance:

- System and Software requirements Validation plan.
- Each test design, case, procedure, and result relates to and can be traced back to a specific requirement.
- Acceptance requirements documentation
- Conformance to acceptance requirements.
- Review user documentation and ensure it is sufficient for user needs.

Integration:

- Test tasks and results documentation.
- Conformance to functional requirements at each stage of integration
- Performance at boundaries and under stress conditions.
- Requirements test coverage.

6.3 Software Management Plan

- The overarching plan of how the app will be made.

6.3.1 Risk Register

- A record of risks that can affect the production and functionality of our app.

6.3.2 Risk Mitigation Plan

- A plan of how we can mitigate and manage the risk is created for every risk found in the Risk Register.

6.4 Stakeholder Reviews

- Meeting with Stakeholders to go over what has been produced and what more needs to be added to meet stakeholder requirements.

Section 7: Testing

INTRODUCTION

Different methods will be used to test effectively different regions. White-Box testing is testing in a way that you can see the code, interacting with targeted parts. Black-Box testing, or data-driven testing, only involves checking what outputs come from what inputs. Gray-Box testing is a middle ground, testing parts of the software like Black-Box.
Test-Driven Design is writing tests to build the code around. Simulation is creating an artificial environment to fully test the software. Test Automation is making tests run automatically on certain events.
Every input cannot be reasonably accounted for, so we use Risk-Based Testing, or testing items based on their risk. Time-Box testing is simply allocating a set amount of time to testing so it does not run over, running the most important tests first. Good-Enough Software is the concept of knowing when to stop because the software fulfills its need.

7.1 TESTING PLANNING SECTION

Test planning is all about setting goals for testing. This means when you are doing your test planning you will have to cover the test objectives, strategies to use, how you will approach each level of testing and all the other details that go into a good test. Now according to the IEEE there are some key features of a good test plan.

First of all, you need to make a list of what you want to test so that you can point all your energy and tests in the right direction. Now looking at the items you listed, find the key features you want tested and ignore those that will not be tested.

Now the most detailed section of a good test plan is how you detail your approach and methods for each level of testing you have. This can include deciding on manual vs automated testing, interface configuration testing, data base testing, and regression testing. Now a test is only as good as its intended goal or the pass-fail criteria. You need to know what you are looking for and how you know whether you found it or not.

A lot of the information you need to build a good test plan can be found in your CONOPS, SDD, or SRS and other planning items that should have been completed before you started your testing plan. Also consider details such as Correctness, Consistency, Completeness, Accuracy, or Feasibility as you design your test plan.

Test Plan outline

- List feature to be tested
- List test type
- List of test beds to be used
- What test cases are to be used
- Recorded test results
- Test conclusions and notes

7.2 TESTING DESIGN SECTION

Each test is created based on one or more requirements and documented. To make testing more efficient, test data can be grouped into categories (Equivalence Class Partitioning) including invalid and valid data that can be reused. Boundary value testing is used to identify bugs at the edges of valid data and just past. Fault error handling helps the software fail gracefully, when needed. Error handling should be designed around what is most likely to fail.

Fault Insertion is a method of estimating how many bugs are left. By putting bugs into the code intentionally and looking for them with tests then comparing the number found to the number not found, the number of bugs left in the code can be estimated.

Cause-Effect Graphing is graphing specific inputs with their expected outputs. It can only reasonably be used with limited input. Every possible input combination is mapped. This helps

7.3 TEST CASES

Tests can be run at different levels of a system or software. Beginning with unit tests to test the smallest portion of the system. Testing multiple units or the interaction of separate units together is called integration testing. To test the whole system is a system test case. After the system test confirms that it is operational according to the requirement document, an acceptance test is conducted with the acquirer (customer) of the software to demonstrate that the software performs in accordance with the acceptance criteria.

The following will be different tests and what they are used to identify. Tests will be listed in order of priority.

Functional testing: This focuses on the functional requirements of the software. Answering the question "what is this *supposed* to do?".

Usability Testing: The objective of this testing is to ensure that the software matches the work style and highlights any areas of concern. Characteristics of a usability test are accessibly, responsiveness, efficiency, comprehensibility, aesthetics, ease of use, and ease of learning.

Performance, Environmental Load, Volume, and Stress Testing: The purpose of this is to determine if the system has any issues meeting its performance requirements for throughout. This is typically done at the system level.

Regression Testing: This is used to verify that new releases are not going to cause more issues with previous or current versions of software.

Worst-Case Testing: There are two types of worst-case testing. The first is an extension of boundary testing. Investigating the boundaries of each input variable separately by exploring the results of the minimum value, one below the minimum value, the maximum value, and one above the maximum. The other is testing the resource condition in a worst-case scenario.

Exploratory Testing: Also known as artistic testing, the testers design and execute tests at the same time based on the knowledge gained as they are testing the software. This is not just wandering around

the software randomly; it is intended to be a systematic exploration of the software based on experience.

7.4 PROCEDURES

7.4.1 Test Coverage Techniques

Test coverage aims at measuring the completeness of the testing. Test coverage maps tests attributes of the software or component being tested. Requirements testing maps out tests to individual requirements in the specification and is in line with the Requirements V&V procedure. Functional coverage makes sure that all requirements have been met and is in line with the Requirements V&V procedure. State coverage looks at the mapping of tests to various states that the software can be in or transition to. Data domain coverage maps test the various data domains in software. Date and time coverage maps tests to various special date or time domains in the software. Interface coverage makes sure various interfaces are being tested. Security coverage maps tests to various security issues that have been identified. Platform configuration coverage maps test the possible platforms that the software may be operating on. Internalization configuration coverage maps tests to possible geographic locations to be sure software will operate appropriately at each location; this can be done in line with integration V&V.

State, data domain, date & time, interface coverage can all be done in the software design V&V process.

7.4.2 Code Coverage Techniques

Code coverage is a way of making sure that all code has a purpose and is being used and there is no pointless or bulk code. This consists of statement, decision and condition coverage which all are done within the Software Design and Construction V&V processes.

Statement coverage (also known as line coverage) is the extent to which a software's component is exercised by tests. To have complete statement coverage each statement (or line) must be executed at least once in the code.

A decision is what determines what path the code takes. Decision coverage (also known as branch coverage) measures both statement coverage and makes sure every branch or decision takes all possible outcomes at least once.

A condition is the state that a decision is based on. For full condition there must be full statement and decision coverage in addition to all conditions being used at least once in some way. If you have full condition coverage then naturally you have full decision coverage and full statement coverage.

7.5 EXECUTION.

7.5.1 Test Beds

Test beds are environments where tests can be executed as if they were in the real environment. The environment shall include:

- **Hardware**: What the software executes on. In the case of a phone app, this can be an emulator or a real phone.
- **Instrumentation:** Tools that monitor the software's test execution.
- **Simulators**: Software that simulates the components that will not be available, such as users.
- **Software tools:** includes automation tools, stubs, and drivers or other software tools used in testing**.**

- **Support elements:** Manuals, facilities, and other needed elements.

7.5.2 Stubs

A software unit or component that simulates a lower-level component so individual components can be tested before they are implemented or to change their behavior. These stubs can be used to test specific parts of the software.

7.5.3 Drivers

A driver is a software unit/component that usually minimally simulates the actions of a calling (higher-level) unit/component that has not yet been integrated during bottom-up testing. Drivers help establish values that get passed around before tests get called.

7.5.4 Harnesses

Test harnesses are used to execute and automate testing. They can be a full framework to guide tests. A harness has two parts: the test execution engine and the repository of test cases and scripts.

7.5.5 Test Libraries

Test libraries are libraries that are used to keep track of changes to test plans, case procedures, scripts, and data. This helps control the flow of information in a way that allows the reuse of data and designs in a much more efficient way.

7.5.6 Controlled Test Environments

Controlled Test Environments are used so that you can return the environment to the base settings which is useful for performing multiple tests using the same variables. This allows you to generate reliable data with your tests.

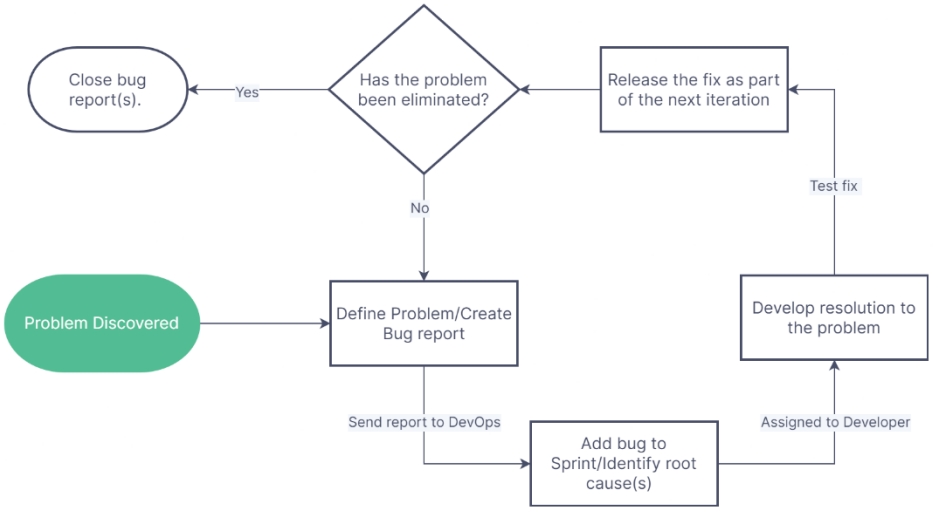
Section 8: Problem Reporting and Corrective Action

8.1 Problem Reporting

Problems shall be generated in two ways: an internal agent generates a report based on an issue or problem in the product. A report automatically generates when a product user experiences an error or chooses to report an error that was not detected automatically. These reports are used to track and resolve problems in the product. Once the problem is fixed in the product, the bug report is closed.

8.2 Corrective Action

The bug report generation shall be the responsibility of a Project Manager. The evolution of the report, including updates regarding the root causes and possible solution, updating the stage of the report, and closing the report shall be the responsibility of developer the report is assigned to.



Section 9: Tools, Techniques, and Methodologies

INTRODUCTION

This covers the different tools and techniques that will be used for analyzing the data we receive. We will be using the seven basic quality tools such as histograms, fishbone diagram, pareto charts, and more to organize the data into readable charts and graphs. Once that has been done, they will be organized into more comprehensive plans to help us tackle problems.

9.1 Tools

- Gantt Chart
  Keep track of project schedule and dependencies.
- Scatter Diagrams
  When cause analysis is going to be performed with paired numerical data.
- Stratification
  Like a control chart, but specifically used for grouping data points together.
- Histograms
  When data distribution is important.
- Control Chart
  Best when change is being tracked over time
- Fishbone Diagram
  Finding root causes of most bugs.
- Check Sheet
  Used when gathering simple data in one place or from one person.

9.2 Techniques

- Plan
  - Brainstorming
  - Benchmarking
- Do
  - Project Planning
  - Project Management
  - Tree diagram
  - Arrow diagram
- Check
  - Data collection
  - Histograms
  - Check sheets
  - Control sheets
- Act
  - Cause and Effect Diagram
  - Pareto Diagrams
  - Scatter Plot Diagram
  - Analysis Tools

- Flowchart

9.3 Methodologies

- We will be using the PDCA cycle for the project which stands for Plan, Do, Check, and Act. The PDCA cycle is all about Improvement and high quality. It breaks down a project into four phases that will guide the project in the best direction possible. Each of the tools and techniques listed above will be great benefits for this kind of cycle. To mention a few examples Brainstorming and benchmarking for Plan, Tree and arrow diagrams for Do, Histograms and control sheets for Check, and Pareto Diagram and Scatter Plot Diagrams for Act.

Section 10: Media Control

The requirements listed in the SRS document shall be configured in the Version Control System (VCS) GitHub. This shall be established as the baseline.

Changes to the baseline shall be sent in the form of a Change Request (CR). Any changes must be clearly communicated in a timely manner via documentation in the VCS.

The baseline shall be copied to parallel versions known as branches for implementing changes. Once changes are made to the baseline within a branch, the branch must be merged with the baseline. If a change causes adverse effects a regression may be performed back to a working point on the baseline. Prior to merging with the baseline, the branch must perform a copy process to ensure the merge will not overwrite a change from a separate branch. These merges must be accompanied by clear documentation describing what is being changed or added with the merge.

Section 11: Supplier Control

11.1 Supplier Provisions:

The work that will be subcontracted will be defined and planned. All important information will be documented. Requirements that need to be met must be clear and defined so that any subcontracted work will meet the requirements for the project.

11.2 Supplier Requirements:

A subcontract manager will be chosen, and they will be responsible for thorough communication of the requirements. The manager and any others involved in the subcontracting will go through orientation ensuring understanding of technical aspects.

11.3 Previous Developed Supplier Software:

The software development plan must be reviewed and approved by the prime contractor. Documentation will be set for the subcontract along with any following review deemed necessary by the manager.

11.4 New Supplier Software Development:

A SQAP will be developed for the subcontracted work including items such as test cases, documentation, reviews, and audits. What each SQAP contains will differ from subcontract to subcontract.

11.5 Supplier Compliance:

The manager of the subcontract will conduct periodic status reviews to ensure the SQAP is being followed. Documentation will be reviewed, and any deliverable will be audited against the stated standards in the SQAP

Section 12: Records Collection, maintenance, and retention

Every team and/or section of the project is required to maintain documentation of their work and plans. In an effort to keep all the documents organized and available the following plan has been laid out.

Section 12.1 Records collection

Each team and section are responsible for the collection of any records they produce. For collection they will be placed in the corresponding section of the provided document library. Each document must be created and labeled according to the documentation guidelines.

Section 12.2 Record maintenance

All records, versions, and documentation will be housed in a git hub. This will keep all items in an easy to track location along with the bonus items of keeping track of who applied what changes and when. Each team inside the project should review and approve any changes to the files in the library for their section. Each section should be reviewed or audited at a set interval.

Section 12.3 Record retention

Certain documents are only required or useful for certain periods of time and will then be removed from the documents to avoid clutter. Any items that are marked for removal will be reviewed and approved for removal to prevent any required information from being deleted. This will apply for code and any other form of record or documentation.

Section 13: Training

List of training that will be required with associated roles and phases.

13.1 Requirements Documentation

Train on effective requirements documentation and organization and team standards.
Phase: Design
Roles: Product Owner, QA

13.2 Cost Tracking

Train on company policies and standards for cost tracking and schedule management.
Phases: Analysis, Implementation
Roles: Project Manager

13.3 Version Control

Company and team standards for code and document version control.
Phases: Design, Implementation
Roles: Developers, QA

13.4 Performing Audits

Company and legal standards on performing audits, as well as team standards for preparing for audits.
Phases: Analysis, Design, Implementation, Testing, Deployment, Maintenance
Roles: QA, Project Manager

13.5 Testing Automation

Company and team standards on testing automation, as well as specific training for any and all software used for testing automation.
Phases: Analysis, Design, Implementation, Testing
Roles: QA, Testers

13.6 Checklist

Essential training checklist items

- ❑ Effective requirements documentation and organization and team standards
- ❑ Company policies and standards for cost tracking and schedule management.
- ❑ Company and team standards for code and document version control
- ❑ Company and legal standards on performing audits, as well as team standards for preparing for audits.
- ❑ Company and team standards on testing automation, as well as specific training for any and all software used for testing automation.

Section 14: Risk Management

14.1 Format

When a risk is identified during or after the risk assessment must be formatted as such for clarity.

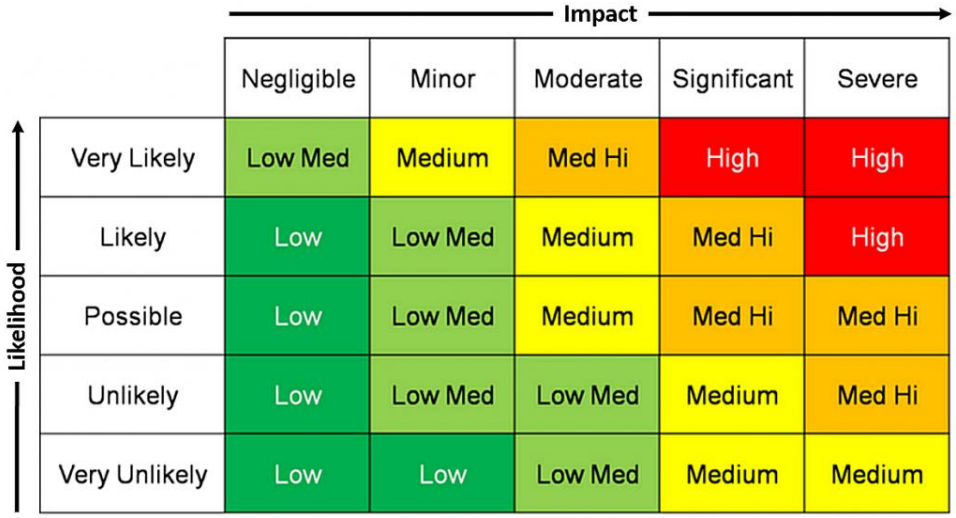
Given the <Condition> there is a possibility that <Event> may occur, causing <Consequence>

Or

Given the <Conditions>, if this <Event>, then these <Consequences>

14.2 Scale

Low = 1 Medium = 2 High = 3



Likelihood \ Impact	Negligible	Minor	Moderate	Significant	Severe
Very Likely	Low Med	Medium	Med Hi	High	High
Likely	Low	Low Med	Medium	Med Hi	High
Possible	Low	Low Med	Medium	Med Hi	Med Hi
Unlikely	Low	Low Med	Low Med	Medium	Med Hi
Very Unlikely	Low	Low	Low Med	Medium	Medium

14.3 Plan

Finally, 1 to 2 risk mitigation plans will be produced based on one or more of the four mitigation techniques Acceptance, Transference, Mitigation, or avoidance. This plan must include the value based on the following equation.

Value = risk/cost

Value is rated on a 1- 5 scale

14.4 Ex

Given the Requirements there is a possibility that the customer may change requirements, causing inaccurate requirements that don’t represent the customer's needs.

Risk level: 2

Mitigation technique: Transference

Value: 2

Plan: As customer change requests are a common any request must be reviewed by a team to evaluate the potential setback and then present that information to the customer and let them decide if the setback is worth the change transferring any potential blowback from the customer back to them.

Glossary and Change Procedure

Terms

Acceptance: a mitigation technique where the project just accepts that the risk might happen.

Transference: a mitigation technique focused on taking a risk and placing the effect somewhere else.

Mitigation: a mitigation technique focused on reducing the probability that a risk will occur.

Avoidance: a mitigation technique focused on making sure risks do not happen all together.

Review: informal check against project procedure and standards as the item is being worked on

Audit: A final formal check of a completed item ensuring that it meets project standards and meets goals

Change Procedure

Changes to this document will be listed in the revision history with names and date.

Appendixes