Super Voters 

Software Quality Plan

Version 1.0
2/2/2022

David Kelly
Garhgaj Singh
Jonathan Coronado
Ya Wang

Document Control / Revision History

Title	**Software Quality Plan**
Project Name	Super Voters
Repository Location	GitHub
URL	https://github.com/leiawang89/SuperVoters

Date	**Version**	**Description**	**Author**
2/2/2022	1.0	Initial draft	Ya Wang
2/3/2022	1.1	Updated draft	David Kelly,Garhgaj Singh,Jonathan Coronado,Ya Wang
2/4	1.2	Final draft	David Kelly,Garhgaj Singh,Jonathan Coronado,Ya Wang

Table Of Contents

1.Introduction

The purpose of this plan is to specify how Software Quality Assurance(SQA) and Cost of Software Quality(CoSQ) will be performed throughout the Super Voters project. This plan describes the software requirements, software testings, SQA activities to be performed and designates a set of goals and metrics for performing the project.

2. Software Requirement

2.1 Project Description

The product we are building is an electronic voting application named SuperVoter. SuperVoter can be installed in many different remote locations such as a library, post office or schools. The application will enable all users to vote securly for an election by keeping the data isolated from other voters. High participation is possible due to a simple design and responsive navigation and processing. The current results will be quickly displayed when a user has finished voting.

2.2 Project Scope

This project's main application usage is for Presidential voting at a desktop environment (at a government disclosed location and moderated).
There are four actors defined in this project, 1) Candidates (Resource Actor), 2). System Administrator, 3) Developers, and 4) Users(Voters)

2.3 Project Requirement

This desktop application allows users to vote for the candidate they prefer in a presidential election. More specifically, 1) the system shall be user friendly, implementing modern standards of UX design and implementation, 2) the system shall have a robust and reliable backend to ensure data is reliable, 3) the system shall have a high level of security and encryption utilizing RSA or AES as per government standard, and 4) the system shall run efficiently to allow for high volume and intensity voting

2.4 Project Constraints

1) Windows 7 Systems or Greater
2) Project completion time 04/29/2022
3) 4 developers
4) Candidate information

2.5 Project Milestones

1) Documentation

2) Base code
3) Base Testing
4) Backend Implementation and Database
5) Backend Testing
6) User Interface Implementation
7) User Interface Testing
8) Quality Control Review and Acceptance Testing

2.6 Final Deliverables

1) Software Executable Application
2) Full Software Documentation

3. Software Quality Plan

3.1 Quality Goals

1) The application should be user-friendly and a new user should be able to understand how to use it the first time.
2) The application should be reliable by being available 99% of the time.
3) The application should be efficient by handling at least 10 simultaneous submissions at a time.
4) The submission process should take no more than 5 seconds to return a response.
5) The results should be shown no more than 10 seconds after a user submits their vote.
6) The system is robust and if it fails, a message is sent to the support staff. It must recover within 5 mins.
7) The login process should authenticate the user within 5 seconds.

3.2 Quality Metrics

1) The application tracks how often the help system or tour is used. Based on the usage, we can determine how user-friendly the application is (Goal 3.1.1)
2) The application tracks the overall error count to make sure it is reliable (Goal 3.1.2).
3) The application tracks when the user submits their vote. This tracks the successful count for voting (Goals 3.1.3, 3.1.4)
4) The application tracks how long it takes for the results to be shown to the user (Goal 3.1.5).
5) The application tracks when a user cannot login due to forgot login or failure (Goal 3.1.6, 3.1.7)
6) The application tracks when an error has occurred during submission or showing results. This helps in seeing if our system is robust (Goal 3.1.6)

4. Review Activities

4.1 Scope

Team members first need to review user interface design documents. UI specification documents will help the team ensure if all important elements are included and make sure if functions operate properly. The document will go into detail about what the application should have as well as how the user will interact while using the app. By using this document, team members will be consistent and will receive help to avoid any confusions.

After reviewing documents, team members will maintain the code so it is readable and organized. In order to maintain the code, common practice will be to write useful comments and use meaningful variable names. Team members will also ensure if code is running properly with no memory leaks or overflow.

4.2 Type

Team members will conduct a technical review of code. Technical review will help members detect any defects while working on the code. Each member will review the other's code and provide each other with strong feedback.

4.3 Schedule

For code reviews and examining design documents, the team will meet once a week via zoom on Wednesdays and go over code and see if any improvement is needed in design documents or if there is a need to add/ implement new features.

4.4 Procedures

The process that the team will use is called “over the shoulder”, where every member will review the other members' code.

Basically in this process:

- Each member is assigned some specific tasks.
- After completing those tasks, each member will meet via zoom and share their code.
- Each person will get a chance to explain what they are trying to do.
- Team members will ask questions and try to fully understand the code.
- After each meeting a report will be prepared with a list of issues.
- Members will provide suggestions if any improvements are needed.

4.5 Reviewers

Team members Ya Wang, David Kelly, Garhgajpal Singh and John Coronado will review code.

4.6 Responsibilities

All of the reviewers are responsible for checking each other's code every week. Each member will provide strong suggestions and come up with solutions to perfect the code. Members then will further examine if they need to add or improve anything on design documents/code.

5.Tests

5.1 Test Scope

Team members will do unit testing for this program. Unit testing is a really good choice because team members will get to test individual components of the software and see if certain methods are working.

After the unit test, members will conduct an integration test. Integration testing is useful in order to combine all methods and components to check the functionality of the whole application. Sometimes there are issues that are missed while doing unit testing so to ensure that won't happen, team members need to test if program interfaces are working properly.

5.2 Types of Test

Team members will manually input tests to examine if the program is working. Members will write test cases which will test the functionality, security, and usability of the code.

5.3 Test Schedule

Team members will create classes and after they are running properly,they will create test cases to see if each component is meeting the requirements for the code.

5.4 Test Procedures

- Examine the code.
- Find important functions or methods that can be tested.
- Come up with test cases and conditions to determine if the methods/functions are running properly.
- Test the code.
- Make any changes or adjustments to the code if needed to meet the test requirements.

5.5 Testers

All team members are going to test the code. Everyone in the team will contribute by creating their own pieces of code. Whenever someone creates a method or function, they need to write test cases to determine if it is meeting all the conditions. If someone forgets to write their test cases, other team members will notify them through email or messages. Testing will be done while creating the code.

6. Tools and Techniques

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

6.1 Tools

1) This project is a Windows Form app and will run in Visual Studio with C#.
2) Defects and Change Requests will be managed in Google Doc.
3) Testing will be maintained in NUnit Test Project in Visual Studio with C#
4) Meeting Minutes will be recorded using Google Doc.
5) Project Schedules will be maintained in MS Project

6.2 Techniques

1) In reviewing documents, a combination of reviews and walkthroughs will be conducted. Prior to submission of documents, authors are required to evaluate documents against corresponding guidelines and associated checklists.
2) Developers will use Unit Testing and Integration Testing to prove the initial quality of the application and interfaces.
3) Defects will be recorded in document for problems identified though any form of testing.

7. Quality Cost Estimation

7.1 CoSQ Model

7.1.1 Formula For CoSQ Model

The American Society of Quality(ASQ) uses the following formula to calculate the Cost Of Quality(CoQ):

$$Cost\ of\ Quality(CoQ) = Cost\ of\ Poor\ Quality(CoPQ) + Cost\ of\ Good\ Quality(CoGQ)$$

This project adopts the same formula for the Cost of Software Quality(CoSQ).

7.1.2 Cost of Good Software Quality(CoGQ)

CoGQ is broken down into several categories. These are the costs related to producing high-quality work products, systems, or services.

1) Prevention Costs
   The prevention costs are the investments made to prevent the occurrence of errors in various stages during the delivery process, such as design, development, production, and shipment.
2) Appraisal Costs
   The appraisal costs are the costs of verification or evaluation of a product or service during the various stages of delivery. The appraisal costs determine the degree of conformance to requirement and quality standards.Table 1 gives the subcategories, definitions, as well as typical costs for prevention and appraisal costs.

Major Categories	Subcategories	Typical tasks
Cost of Good Software Quality(CoGQ)	Prevention cost	Definition of release criteria for acceptance testing and related quality standards,metric collection and analysis, training of employees.
	Appraisal cost	Quality assurance, inspection/Testing, Setting up effective test environments, design and code reviews

Table 1 Categories and typical tasks of CoGQ

Note: This table references a table(table2) from the article "Measuring the Cost of Software Quality of a Large Software Project at Bombardier Transportation:A Case Study" and has been modified to meet requirements for current project

7.1.3 Cost of Poor Quality(CoPQ)

CoPQ are the costs associated with providing poor-quality work products, systems or services.CoPQ can be divided into two types:

1) Internal failure costs
   Internal failure costs are the costs associated with defects found before the customer receives the product or services
2) External failure costs
   External failure costs are the costs associated with defects found after the customers receive the product or services.

Major Categories	Subcategories	Typical tasks
Cost of Poor Quality(CoPQ)	Internal failure costs	Rework(e.g., recode, retest, rereview, redocument, etc.), failure analysis
	External failure costs	Complaint handling, recalls, sales reductions, returns, company reputation damage, company devaluation

Table 2 Categories and typical tasks of CoPQ

Note: This table references a table(table2) from the article "Measuring the Cost of Software Quality of a Large Software Project at Bombardier Transportation:A Case Study" and has been modified to meet requirements for current project

Management control cost is another major part of the CoSQ, but it is not included in this project due to the small size of the project and the lack of managerial resources.

7.1.4 Tasks Weighting and Cost Calculation

The typical tasks are sorted as follows: implementation, evaluation, prevention, and rework(interna, and external failures). Table 3 gives weight of each typical task and the effort(in hours) under each stage of development. The total effort of all tasks is 1,212 hours.

Typical tasks (effort measured in hours)	Implementation (weight/cost in hours)	Evaluation (weight/cost in hours)	Prevention (weight/cost in hours)	Rework (weight/cost in hours)
Normal project task, coding(200)	100% / 200			
Software quality Plan(24)			100% / 24	
Metric collection and analysis(12)			100% / 12	
Training of new resources(48)	20% / 10		80% / 38	
Requirements traceability(12)			100% / 12	
Prototype(66)			100% / 66	
Configuration management(14)			100% / 12	
Requirement audit(24)		100% / 24		
Design review(24)		100% / 24		
Code review(74)	70% / 52	30% / 22		
Setting up test environment(24)		100% / 24		
Software unit testing activities(100)	60% / 60	40% / 40		
Software integration testing activities(30)		100% / 30		
Software validation, verification(30)		100% / 30		

Problem corrections, debugging and final coding(112)	40% / 45	60% / 67		
Acceptance, debug and test prototype(80)	100% / 80			
Baseline acceptance(48)		50% / 24	50% / 24	
Rework and maintenance (24)				100% / 24
Software problem correction(58)	15% / 9			85% / 49
Failure analysis(12)				100% / 12
Redocument , SQA writing and update(46)	30% / 14		40% / 18	30% / 14
Update traceability(48)	50% / 24			50% / 24
Follow-up and validation(80)	85% / 68			15% / 12
Follow-up and validation(24)				100% / 24
Total efforts: 1,212 hours	**Total: 562 hours**	**Total: 285 hours**	**Total: 206 hours**	**Total: 159 hours**

Table 3 Task weighting and effort calculation

Figure 1 shows the distribution of development costs in the various categories of software quality and implementation cost. The figure reveals that the cost of rework is 13.1%,the cost of prevention is 17% , and the cost of evaluation is 23.5% of the total cost of development.

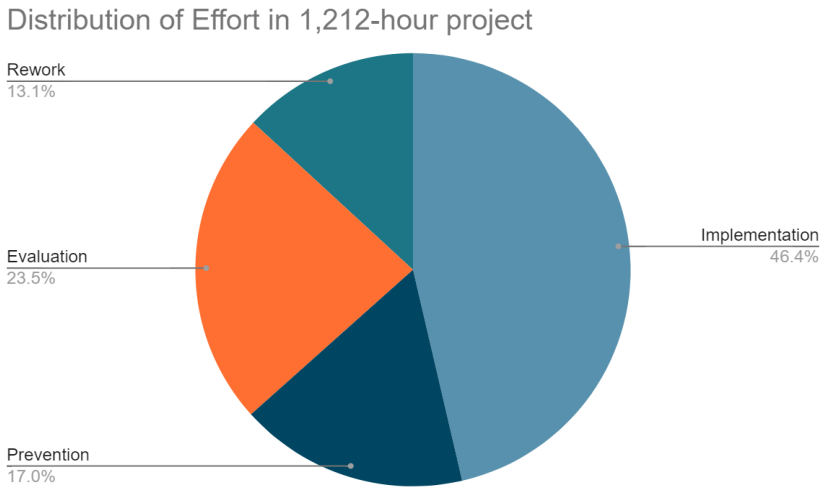

Figure 1 Distribution of effort in 1,212-hour project

7.2 COCOMO Model

This project uses the COCOMO model to estimate the number of Man-Months required to develop this project.

7.2.1 Choose the Mode

According to the COCOMO model, there are three modes of software development projects that depend on complexity, such as organic mode, semi-detached mode, and embedded mode. This project adopts organic mode since it is a small application with low complexity and high flexibility.

7.2.2 Effort Equation

The COCOMO model makes its estimates of required effort(measured in Man-Month) based primarily on the estimation of the software project's size. The equation is as follows,

$$E = a(KDSI)^b * EAF(MM)$$

Where
KDSI: Thousand(K) lines of Delivered(D) Source(S) Instructions(I)
EAF: Effort(E) Adjustment(A) Factor(F)

7.2.3 Schedule Equation

The COCOMO model schedule equation predicts the number of months required to complete the software. The duration for a project is based on the effort predicated by the effort equation.The equation is as follows,

$$D = c(E)^d(months)$$

Where
D: Development time
E: Effort from the effort equation

7.2.4 Parameters

Since this project development is under organic mode, the corresponding value for each parameter(a, b, c, d) was picked as follows,

Mode	**a**	**b**	**c**	**d**
Organic Mode	2.4	1.05	2.5	0.38

7.2.5 Results

1) Effort(MM)
   KSDI = 4,000 lines of code

EAF = 1.00

$$E = 2.4 * (4)^{1.05} * 1 = 10.29(Man - Months)$$

2) Development Time(in months)

$$D = 2.5 * (10.29)^{0.38} = 6\ Months$$

3) People Required

$$P = E/D = 10.29/6 = 1.7 \approx 2\ people$$

The result yields an estimate of 6 month, and an average staffing of two people.