CALIFORNIA STATE UNIVERSITY, SAN BERNARDINO

School of Computer Science
And Engineering

CSE 455

{AlgorithmA }; 2011

Software Quality Assurance Plan
First Iteration

CSE 455, Inc.
CEO: Dr. Concepcion
Project Manager: Anthony Thomas
Assistant Manager: Erwin Soekianto
Team Leads: Matthew Compean and Patrick Ridge

Table Of Contents

1 Purpose

The purpose of this Software Quality Assurance Plan (SQAP) is to ensure that the product described in the Software Requirements and Specifications (SRS) is the product delivered by AlgorithmA 2011. Additionally, this document serves to ensure that this software be delivered on time to the Clients, Dr. Concepcion and Computer Science students and faculty at CSUSB. The Extreme Programming development will be used as the software life cycle model while design aspects of AlgorithmA 2011 will be implemented using the Unified Modeling Language (UML) diagrams.

Reference Documents

The following is a list of references used to create this SQAP document:

Tom S. Lee, Tim Berger, Joriz DeGuzman, Ken Lewis. (2002). Software Project Management Plans, Iteration 14.

Fowler, M.; Scott, K (2000). UML Distilled. (2nd ed) Boston, MA: Addison-Wesley.

IEEE SA Standards Board (200798). Software Project Management Plan IEEE Std 1058-200798. New York, NY: IEEE.

Ritchie, D. (2000). The Development of the C Language, retrieved from <http://cm.bell-labs.com/cm/cs/who/dmr/chist.html> (25 Jan 2003)

Dictionary of Algorithms and Data Structures, retrieved from <http://www.nist.gov/dads/ > (25 Jan 2003)

O'Connor, Patrick; Vargas, Danny; Kamel, Abdelrahman (2010). AlgorithmA 2010 SQAP.

Management

The management team's responsibilities include:

- Construct the final build of the AlgorithmA project.
- Supervise the overall design, implementation, and framework for AlgorithmA 2011.
    - Manage all teams using synch-and-stabilize development approach.
- Allowing for efficient communication between all team members of the project team.
    - The team leaders will serve as the main contact for each team, and team members are urged to speak to their appropriate team leader.
    - However, software engineers are not limited to only communicating through the team leaders. Open communication will always be encouraged during this project, and team members can talk to the manager in charge of their team if need be.
- Creating and overseeing a schedule that results in the on time delivery of a quality product that satisfies all the requirements described in the SRS.

- Overseeing and supervising each team's progress throughout the project. The project managers will be responsible for supervising the team leads, ensuring that they keep to the adopted project schedule.
- Each team lead will be responsible for delegating the work to his two teams in order to complete the tasks required to fulfill the project schedule and comply with the SRS.

The following is a listing of the organization of the members of the project team for work on the AlgorithmA 2011 project:

CEO

Dr. Conception

Project Manager

Anthony Thomas

Assistant Manager

Erwin Soekianto

Server Team

Angel Zepeda(Team Lead)

Timothy Castelli

Nate Parrish

Programming Team 1

Patrick Ridge (Team Lead)

David Cushman

Sean Kooyman

Grover Wimberly

Gabriel Matthews

Kenneth Williams

Matthew Tonyan

Programming Team 2

Matthew Compean (Team Lead)

Carlos Maldonado

Anthony Phillips

Esmirna Nolaro

Andrew Hutchinson

Crystal Bauza

Wes Rockwood

Christopher Huerta

Documentation Team

Idalia A Flamenco (Team Lead)

Juan Vidart

Michael Basilious

The management team assigned team members to 3 separate teams: Programming Team A, Programming Team B, and Documentation Team. The team members were categorized based on a skills survey, and placed in the appropriate teams. A hierarchical structure was put into place in order to facilitate a consistent means of communication between teams and members.

Documentation

The documentation that exists in addition to the SRS, SQAP and SPMP is as follows:

Software Requirements Review (SRR)

The Software Requirements Review (SRR) will be used to determine the adequacy of requirements in SRS. At the end of the iteration, all teams will make sure the following criteria are upheld:

Have all necessary components been delivered?

- Does the Software do what it's supposed to do?
- Does it conform to the MVC Architecture?
- Does the product have all required aspects, as stated by CEO?
- Does the product implement the requirements described in the SRS?

Functional Audit

Functional audits will have the sole purpose of ensuring that all requirements outlined within the SRS have been met. Programming Teams will make sure the following criteria are upheld:

- Have all necessary components been delivered?
- Does the Software do what it's supposed to do?
- Does it conform to the MVC Architecture?

Physical Audit

The physical audit will be utilized to verify the software product and corresponding documentation. Assurance of internal consistency will be the main focus. The following criteria will be considered followed by the teams assigned to each task:

- Is the code well documented within the source?
    - Programming Team assigned to that particular source.
- Does the external Documentation adhere to Management set Standards?
    - Documentation Team
- Are all associated Bugs logged in the Bug Database (BugZilla)?
    - Programming Team Leads
- Is the organization and structure of the code in logical placement and does it adhere to set standards?
    - Programming Team assigned to that particular source.

In-Process Audits

The in-process audits to be conducted in the duration of AlgorithmA 2011 will be used to verify the consistency of the design of AlgorithmA 2011. This includes ensuring that the code base maintains consistency with the MVC architecture. In-Process Audits shall be conducted through a joint effort that includes both the Management Team and the Programming Team.

Managerial Reviews

Managerial reviews will be held biweekly during the course of work on AlgorithmA 2011. The Management team will meet with the CEO to discuss the progress of AlgorithmA 2011. During these meetings, the execution of all of the actions and items identified in the SQAP will undergo assessment. If any of the items listed in the SQAP do not apply to work on the project, no longer encourage progress on the project, or are no longer deemed feasible, the Management team will seek modification or removal of those items in question. The SQAP will always be open to modification at any of these meetings. The purpose will be to ensure the SQAP remains a definitive document outlining the procedures the project team will take to uphold SQAP.

Standards, Practices, Conventions, and Metrics

The purpose of this section is to provide a coding convention that will be followed in order to increase code readability and maintenance for the programmers of AlgorithmA 2011 and future

iterations. The conventions described in this section have been defined based upon these criteria:

- To make the source code complies with the MVC architecture.
- To assist in the maintenance of future iterations.
- For any documentation to be clear and concise.

JavaScript Coding Standards

Unlike previous iterations of AlgorithmA, we no longer maintain a format that works with the application JavaDoc in mind. Many of the same conventions carry over, but for the most part, coding standards now must take on the form of readability and well documented code.

Classes and Functions

All Classes and Functions will be no longer than 20 characters in length and be descriptive. No short hand terms may be used. They must all begin with a capitol letter and have a capitol letter for each first letter of a new word in the name. Any and all parenthesis must be spaced after the object name and within the parenthesis themselves. Any arguments contained within the parenthesis must also be spaced, allowing for a space between the first and last arguments and the parenthesis.

Examples:

function ShowBox (ID)

function MouseEvent (Event, nXPosition, nYPosition)

Variables

Variables shall be no longer than 10 characters (if it can be helped). Due to the conventions of JavaScript, a variable can be any type you set it to (from strings to numbers to whole functions and classes). Due to this, the first letter of each variable must indicate what that variable is used for. If the variable is for a number (integer, float, double) indicate it by starting with “n” for number. If the variable is a string, use “s”. If the variable is a function or class object, use “o”. If the variable contains HTML, use “h”. If the variable is an array, you must first use the type prefix then add an “a”. Using it in the other way may cause confusion such as “anArrray” or “asName”. Matrices, unlike arrays, must only use “m” for a prefix. This need not apply for any temporary single use variables such as for iterators. Specialty variables such as identifiers (ID), events (keyboard and mouse), etc must be named justly to indicate it’s purpose, foregoing the prefix notation for conventional variables.

The first letter following the indicator variable must be uppercase. Unlike classes and functions, short hand terms may be used, but only if they are obvious as to their meaning. You must also try to keep variables either local or a part of a class. Due to the conventions of JavaScript, always use the “var” name identifier for each new variable created. Not doing so will create a variable automatically, making it hard to understand.

Strings and HTML code will all use double quotes “ “. This is to allow for HTML code to have

strings within strings using single quotes ‘ ‘.

Example:

```
var sName = ”Williams”;
var nPosition = 100;
var hDivBox = “<div onClick=’DoSomething()’></div>”;
```

Methods (Member Functions)

All methods shall be descriptive of their functionality and be no longer than 20 characters. All methods shall be prefixed by a description of the action they are performing (excluding functions that are extended or implemented form of an existing interface). Functions that modify data shall be prefixed with “set”. Functions that query for data shall be prefixed with “get”. Functions that return a Boolean value shall be prefixed with “is”.

Example:

```
getBoxID ( );
setBoxID ( ID );
isColliding ( );
```

Code Blocks & Iteration

All code blocks shall conform to the same basic principles; this includes functions. All blocks of code (the statements in between the curly braces) must indented within the block itself by one tab. Anything that can make use of the code block must place the beginning curly brace one line below its statement line. The end curly brace must be place after the last statement within the code block, moving back one tab for a clean finish. Encapsulated code blocks will continue the tabbed trend, no matter how many are used.

Example:

```
while ( true )
    {
        getBoxID();
}
```

Iteration statements such as “for” loops that make use of variables are free to use single letter variables. The conditions and statements that make up a “for” in particular must use proper spacing to make the statement readable as a whole. This includes spacing, like a function, for each parenthesis and arguments after each semicolon.

Example:

```
for ( var i = 0; i < 10; i++ )
{
        document.innerHTML += i "<br />";
}
```

Only C/C++ styles of iterations may be used. The C# method ("for (x in Array)") will not be used.

Arrays

JavaScript has three ways to declare arrays; first is the regular method, the second is called a condensed array, the third is the literal array. All arrays can follow either the traditional or condensed array styling. Literal arrays can be cryptic to understand at first glance.

Example:

```
// Regular Array
var saCars = new Array ( );
saCars[0] = "Saab";
saCars[1] = "Volvo";
saCars[2] = "BMW";

// Condensed Array
var saCars = new Array ( "Saab", "Volvo", "BMW" );
```

Matrix arrays must be created using the regular array method.

Example:

```
// 2 x 2 matrix with 1s and 0s
var mBinary = new Array ( 2, 2 )
mBinary[0][0] = 0;
mBinary[0][1] = 1;
mBinary[1][0] = 1;
mBinary[1][1] = 0;
```

Commenting

Due to the change to JavaScript, we can no longer rely on JavaDoc. We must therefore comment code in a clear and concise manner. Comments are to make use of both the traditional one line

comment "//" and the multiline comment box "/**/". Single line comments are only to be used for simple comments where needed. After the double slash, use a space after for easier reading. Comment blocks are to be used to make in depth explanations, examples, and variable makeup.

Each JavaScript file for any particular code must contain the following information at the top of each page, following the indentation using tabbing.

```
        /*****************************
        *       Name:           John Doe
        *       File:           [Filename].js
        *       Project:        {AlgorithmA}; 2011
        *       Date Created:   February 2, 2011
        *       Date Modded:    February 3, 2011
        *       Purpose:
*               [List the purpose of this file. Comment
*               where and how it is used. Be sure to
*               give any details of what it changes]
*       Dependencies:
*               [Files, functions, and variables required
*               to run the code in the file.]
*       Notes/Changelog:
*               [List any notes such as, what has been
*               changed and when]
*       Functions:
*               [Function name]     :       [One line description]
*               […]
*****************************/
```

Each portion of this is made so as to let anyone know at a quick glance what they are getting into with this particular file. The modified date must be done after each successful change to the code is made. The purpose must provide an in depth reasoning for the file. Dependencies must list each file, function, and variable used. The exception for this is for common utility files such as animation libraries. Notes must let the reader know what has changed. Functions is a list of all functions and classes found within the file. List their name, arguments if possible, and a short one line description of its task.

Before each function must be an in depth explanation of the task the function performs. This is to cut down on the number of single line comments made within the code.

Single line comments must be made on blocks of code difficult to read or requiring a special operation to complete. Iterations must be documented as to their functionality and so forth.

HTML and PHP Coding Standards

The accompanying code following this text will implement every detail that is defined in this section.

General HTML

At the beginning of each HTML file, a five-line header will be implemented like the JavaScript files above. The first line will have the name of the Software Engineer. The second line will contain the Date which this file was created. The third line will contain the file name and line four will contain the revision number of the file. The final line of the header will be a detailed description of the file. It will explain the functionality and possible outcome. The header will be enclosed within comments of the form "<!-- *** -->". This header will be implemented before the major HTML tag. Indenting with tab will make the overall header easier to read, following the same context as the JavaScript header.

Unlike previous versions of AlgorithmA, we will compact the HTML the overall design of the site by not using a separating line between each and every major tag used. Spacing will be used before the use of a major tag (<HTML>, <HEAD>, <BODY>) and after. Indentation will be used heavily, treating them as you would code blocks in JavaScript. This does not apply however to the major tags, however all tags placed within must be indented appropriately. All singular elements will conform to the latest standard of HTML using self-closing tags (ie: "
<img />"). These closed tags will be spaced from the rest of the tag itself as shown.

<!--

- Name: John Doe
- File: Simple.html
- Project: {AlgorithmA}; 2011
- Date Created: February 2, 2011
- Date Modded: February 3, 2011
- Description: A basic intro to HTML for CSCI 455
- Notes:
- - Notes usually placed here.

-->

```
<html>

	<head>
		<title>Welcome AlgorithmA Team!</title>
		<!-- Insert scripts and CSS here -->
	</head>

	<!-- Sets background to a light grey, text to white -->
<body bgcolor = ”#E9E9E9” text = “#FFF”>
	<!-- Unordered list -->
<ul>
		<li>How do you do?</li>
		<li>What are you doing?</li>
		<li>Happy New Year!!!</li>
	</ul>
	<!-- centered statement paragraph -->
	<p align=”center”> Good luck with AlgorithmA 2011!!!</p>
</body>

</html>
```

General PHP

For PHP, the same methodology as HTML and JavaScript must be employed. At the beginning of each PHP file, a five line header will be implemented. The first line will have the name of the Software Engineer. The second line will contain the date when this file was created. The third line will contain the file name of the file. The fourth line will contain the revision number of the file. The final line of the header will be a detailed description of the file. It will explain the functionality and possible outcome. The header will be enclosed within comments of the form “/* --- */”. This header will be embedded within the pre-defined PHP tags (“<?php” for opening and “?>” for closing).

PHP Variables

Each variable will start with a dollar sign ($) and will be no longer than eight (8) characters long. The variable will be named according to the values that will be stored in it, starting with lowercase for the first word and uppercase for the second.

String values will be enclosed in single quotes, also called ‘apostrophes’ (‘ and ‘). Every control structure within the PHP code will contain its truth parameters and the body to be executed.

PHP Code Blocks

All control structures containing only one line, or many lines, of code in its code body will utilize an opening brace “{“ before, and a closing brace “}” after, the code body in that section. All lines of code within each block will be indented by a single tab (4 spaces) to simplify and increase the readability of the code. For encapsulated code blocks, another indent will be used and will continue for every code block within that one.

PHP Functions

Functions will be used to simplify the creation of larger bodies of organization on a web page and will not exceed fifteen (15) characters long. The function name shall fully describe the functionality and purpose of that function. Each word of the name will be capitalized, starting with the first name and every subsequent name following. Function bodies will be enclosed in curly braces.

PHP Comments

Much like with JavaScript, comments will follow the same guidelines issued before. Single quotes “//” must be used to describe brief functionality descriptions before major lines of code. A space must be put after the double slash for easier reading. Comment blocks “/**/” will be used to go into detail on how much larger portions of code operate. Before each function, you must document in a block comment what the function performs. This is to cut down in the amount of commenting you need to accomplish within the code itself.

At the beginning of each PHP file will contain the creator’s name, filename, project, creation date, modified date, description, notes, a list of dependencies and a list of functions with a brief description of each found in the file.

```
<?php
        /*****************************
        *       Name:           John Doe
        *       File:           [Filename].php
        *       Project:        {AlgorithmA}; 2011
        *       Date Created:   February 2, 2011
        *       Date Modded:    February 3, 2011
        *       Purpose:
*               [List the purpose of this file. Comment
*               where and how it is used. Be sure to
*               give any details of what it changes]
*       Dependencies:
```

```
*               [Files, functions, and variables required
*               to run the code in the file.]
*       Notes/Changelog:
*               [List any notes such as, what has been
*               changed and when]
*       Functions:
*               [Function name]       :       [One line description]
*               Create_Layout                 :       Create layout depending on choice
*****************************/

// This variable stores choice of user for selected layout.
$choice = $_POST['choice'];

// Counter used in for loop.
$count = 1;

// Will be used in debugging later on.
$check = 'Debugger!';

// Calls function Create_Layout
Create_Layout($choice);

//Debugging Line
print $check;
print "<br /><br />";

//If loop checks if inputted choice is correct.
if ($choice != '1' || $choice != '2')
{
print "You have chosen wrongly! <BR>";
}
```

```
//Defines the layout that was chosen by user.
function Create_Layout($variable)
{
if ($choice == '1') //Choice One: Three Frames
{
print "You Have Chosen Layout One! <br />";
print "Thank you for choosing! <br/ >";
print "Layout one contains: A Logo, A Body, and A Menu<BR>"; //Message
}
if ($choice == '2') //Choice Two: Three Frames
{
print "You Have Chosen Layout Two! <br />";
print "Thank you for choosing! <br />";
print "Layout two contains: Two Logo, Body, and A Drop-Down Menu<br />";
}
} //End function

//Testing arithmetic
for ( $i; $i < 5; %i + 1 )
{
print "Testing the operations";
print "<br />";
$multiplication = 20 * $i;
print $multiplication; //Prints result
}
?>
```

Pseudocode Standards

All coding teams will use a standard system of pseudocode to convert interactivity diagrams to Java code. The pseudocode language "AL," will be used for all such purposes for clarity and consistency.

The “AL” pseudocode language is fully described in self contained documentation elsewhere and is available upon request.

In general, the “AL” pseudocode language is logically similar to the actual code, but does not require the syntactic technicalities of any actual coding language.

The pseudocode submitted by each team or engineer will be consistent with all interactivity diagrams submitted for the same programming task, and will be used to generate the actual source code for that task.

To maintain the required consistency, any changes to actual programming code (for any reason, including “bugs,” change in rationale, or new configuration requirements) that deviates from, or in any way modifies, the underlying logic, will also be changed in the associated pseudocode. Therefore, for maintenance purposes, all supporting documentation will consistently represent the executable code.

The pseudocode will also adhere to the same logical construction as the programming language, relating to repetition structures, condition (“decision”) structures, and assignment structures.

File Storage Standards

This section is defined to prevent the ambiguity of file and directory names that can arise from multiple files or directories of the same name, such as

MyFile.CPP

MyFile.cpp

myfile.cpp

MyDirectory

Mydirectory

This is caused by a case sensitive OS that distinguishes between upper and lower case letters in file names, allocating separate storage for files whose names would, except for the case of the characters used, be the same file (or directory) name.

Therefore, all file and directory names will be defined in all lower case letters. The structure itself will follow the MVC model of design, with names corresponding to their respective section and what type of files they contain (ie: “image” for images and “javascript” for JavaScript). Files will also be lowercase, using names to actively describe what they represent and are used for.

Metrics

The metrics used by each team are as follows:

Quality Metrics

- Bugzilla error and exception tracking software will be used to manage code faults
    - Faults will be logged and tracked by members of any team not working on that

particular section of the code.

- Faults will be managed by the Team Leads.

- Faults per line of code
  - There shall be 5 or fewer per every 100 lines of code
  - This is due in part of the nature of JavaScript not needing to be large to be useful
- Compliance with defined standards
  - The product shall pass reviews and audits, and be in compliance with pre-defined SQAP standards
- The number of functional requirements
  - This method will be used to gauge the completeness of the project at the time of the review.

Management Metrics

- How many features have been implemented
- Personnel allocation.
  - How much they are able to accomplish as reported by the Team Leads.
- How many tasks have been completed/in-progress

Teams will use the tools provided by SVN, reviews, and audits to measure progress and track goals. Standards and regulations for completed products will be adhered to as defined in the SPMP. The metrics collection plan will implement the above metrics, while considering the standards and regulations governing the finished product.

These metrics shall be determined using status meetings, milestones, and code inspections. The status meeting will allow the management to gauge to overall progress of the project. Milestones will allow the management team to set specific dates for product submission, and based on the time of submission can be used to gauge the progress of this project, and to predict future milestone dates. Code review will allow the management team to analyze the product's overall functionality and gauge the areas of the product where most errors are occurring and resolve any issues discovered.

Reviews and Audits

See Section 4.

Software Requirements Review (SRR)

The Software Requirements Review (SRR) will be used to determine the adequacy of requirements in SRS. At the end of the iteration, all teams will make sure the following criteria are upheld:

- Have all necessary components been delivered?
- Does the Software do what it's supposed to do?
- Does it conform to the MVC Architecture?
- Does the product have all required aspects, as stated by CEO?
- Does the product implement the requirements described in the SRS?

Functional Audit

Functional audits will have the sole purpose of ensuring that all requirements outlined within the SRS have been met. Programming Teams will make sure the following criteria are upheld:

- Have all necessary components been delivered?
- Does the Software do what it's supposed to do?
- Does it conform to the MVC Architecture?

Physical Audit

The physical audit will be utilized to verify the software product and corresponding documentation. Assurance of internal consistency will be the main focus. The following criteria will be considered followed by the teams assigned to each task:

- Is the code well Documented within the source?
    - Programming Team assigned to that particular source.
- Does the external Documentation adhere to Management set Standards?
    - Documentation Team
- Are all associated Bugs logged in the Bug Database (BugZilla)?
    - Programming Team Leads
- Is the organization and structure of the code in logical placement and does it adhere to set standards?
    - Programming Team assigned to that particular source.

In-Process Audits

The in-process audits to be conducted in the duration of AlgorithmA 2011 will be used to verify the consistency of the design of AlgorithmA 2011. This includes ensuring that the code base maintains consistency with the MVC architecture. In-Process Audits shall be conducted through a joint effort that includes both the Management Team and the Programming Team.

Managerial Reviews

Managerial reviews will be held biweekly during the course of work on AlgorithmA 2011. The Management team will meet with the CEO to discuss the progress of AlgorithmA 2011. During

these meetings, the execution of all of the actions and items identified in the SQAP will undergo assessment. If any of the items listed in the SQAP do not apply to work on the project, no longer encourage progress on the project, or are no longer deemed feasible, the Management team will seek modification or removal of those items in question. The SQAP will always be open to modification at any of these meetings. The purpose will be to ensure the SQAP remains a definitive document outlining the procedures the project team will take to uphold SQAP.

Test

Testing will occur using the following methods:

- Menu testing will be a comprehensive step by step of each menu item of AlgorithmA 2011 for the identification of current faults.
- Unit Test
    - The code test plan that each software engineer will use prior to integration.
- Integration Test Plans
    - The plan that is utilized to ensure that codes function along with the page elements and design.
- Team Integration Test
    - The compilation testing of the separate components of the project by each team member
- System Integration Test
    - The compilation test of the separate components of the project by each team.
- Black Box Testing
    - Testing for behavior of program without making reference to the internal structures or the algorithm used.
- White Box Testing
    - Structural testing that focuses on the internal knowledge of the software.
- Clean Room Technique
    - Technique where a module is not compiled until it passes inspection. Also includes an incremental life cycle, formal techniques for specification and design, and code-walkthroughs and inspections.

Problem Reporting and Corrective Action

Whenever a problem is found, that individual shall perform the following steps:

- Use a web browser to open the URL for bugzilla
- Click the link: "Enter a new bug report".

    - Login to bugzilla
- Fill in these required fields:
    - Platform, Component, OS, Severity, URL, Summary, and Description.
- In the Description field be sure to input a detailed description of the problem and how to recreate it.
- The user should end the Description with their name and team.
- Submit bug

Team Leads of the respective algorithm will be notified automatically. They will then notify the one in charge of that algorithm and development process will begin anew on the bug. The Documentation Team will be responsible for follow up testing, outcome documentation, and tracking. Resolution and rational documentation is to be included in the maintenance manual.

Tools, Techniques, and Methodologies

The AlgorithmA 2011 project will require particular tools, techniques, and methodologies in order to support SQA. They are as follows:

- UML - Unified Modeling Language. The AlgorithmA 2011 project follows the principle of object-oriented design. UML will enforce that principle, serving as the tool used by every team to specify, visualize, construct and document any and all artifacts present in the AlgorithmA project.
- Rational Rose or Dia - Each team will use these UML modeling tools in order to accomplish the tasks outlined above.
- System Architect - Alternative to Rational Rose or DIA as UML modeling tools.
- Tortoise SVN - For programmers in the programming team and compiler team who works on windows, this is a good tool to interact with SVN. Refer to http://tigris.tortoisesvn.org for more information.
- Bugzilla - All teams will use this bug tracking system. The Server team will install and configure Bugzilla on the server. The Documentation and Testing teams will report bugs to the system. The remaining teams will refer to Bugzilla to fix the documented bugs. After fixing bugs, the appropriate team will update the bug report in Bugzilla. Refer to http://www.bugzilla.org/ for more information.
- AlgorithmA Project Team Hierarchy – The structure of the adopted AlgorithmA Project Team Hierarchy will uphold SQA through strong communication, close supervision, and micromanagement. The hierarchy follows a top-down model, with which tasks and assignments are handed down from higher hierarchical levels to lower levels. In other words, managers would hand down tasks and assignments to team leaders, who in turn delegate that work to their team members. Communication will also follow the same principle, but can travel up or down the model.

- GIMP - A program for editing photos and graphics. This can be useful to the animation team and the web team for creating and editing graphics.
- Inkscape – A vector graphic editor and creator. Useful for creating logos or graphics that must be “scaled” as a functional requirement.
- SVN - A tool the AlgorithmA project will utilize for various functions including verification of completed work, implementation of project tasks, documentation, and retention.
- OOA/D - Object Oriented Analysis and Design, a methodology that will ensure that the software product will comply with Object-Oriented Programming design principles.
- Object-Oriented Programming (OOP) - A computer programming paradigm that emphasizes the following aspects:
    - Objects - packaging data and functionality together into units within a running computer program; objects are the basis of modularity and structure in an object-oriented computer program.
    - Abstraction - combining multiple smaller operations into a single unit that can be referred to by name.
    - Encapsulation - separating implementation from interfaces.
    - Polymorphism - using the same name to invoke different operations on objects of different data types.
    - Inheritance - defining objects data types as extensions and/or restrictions of other object data types.
- JavaScript Animation Library – To be determined.

Code Control

The work to be performed for AlgorithmA 2011 will be subject to code control. Each version of identified software will undergo maintenance, storage, security, and documentation during each phase of the software life cycle. The process of maintenance will be performed and controlled by the respective Programming Team. All work will be stored using the SVN. All software products will be developed by the appropriate teams, and uploaded to these tools. All managers will back up copies of all code in an independent system, preferably on a password protected terminal. Security measures dealing with the AlgorithmA 2011 project will be handled by the Server team. For the specific tasks, responsibilities, and methods of execution of the Server team, refer to the SPMP.

Media Control

The media in which the computer product will be stored will be a hard drive and backed up at least one per week to another independent terminal to prevent data loss or corruption. All access is to be password protected. The Server team is to do incremental backups of the entire AlgorithmA 2011 project every night and full backups of the entire AlgorithmA 2011 project weekly. The versions shall be backed up and logged separately in distinct folders, to show all phases of the evolution of

the software life cycle. In addition, the Management team will burn the final build of AlgorithmA 2011 on multiple media, including USB flash drives, CD, and SourceForge. This will also have a backup, doubling the number of media for it to be available on.

Supplier Control

We hereby break the chain of SQAP derivation with this iteration of AlgorithmA (2011). Suppliers of the project are to adhere, from this iteration forth, to 2011's SQAP. Subsequent teams of AlgorithmA, follow this guide like a bible. But unlike the bible, make changes according to your needs.

Supply of AlgorithmA is in the sole hands of the head of CS455 inc.

Records Collection, Maintenance, and Retention

SVN allows for multiple users to maintain a single version of the product by checking out portions of the code and then combining it back into the product. This tool documents revisions and changes. The SPMP includes provisions for requiring team members to full document their rational in revisions.

Maintenance is provided in full by the Server Team. Please refer to the SPMP for their methods of execution.

Retention is handled by both the Team Leads and Project Managers. No other may commit any changes to the SVN at any given time.

Training

Training in JavaScript, SVN, MVC, BugZilla, and Wiki must be arranged and will take place during the first sessions of the quarter. The training should primarily be held during designated lab hours by means of demonstration. In addition, personal guidance from management is imperative.

JavaScript Training

In order to prepare the JavaScript and Documentation Teams to complete their assigned tasks, the following training shall be performed:

- Instructional overview of the differences between C++ and JavaScript.
    - Be sure to note variables, functions, and classes
- Demonstration of JavaScript in action
    - Simple animation test using a "DIV" box and 2 buttons to play and reset.
- Crash course training in HTML and CSS
    - Used heavily in all JavaScript animations
- Referral to sites well known for information
    - w3schools.com

- jquery.com

- Documentation and examples readably available through GoogleGroups.
- Assignment of a training to be done in between lab sessions before the real project begins (roughly 3 weeks).

Server/Web Training

In order to prepare the Web and Server teams to complete their assigned tasks, the following training shall be performed:

- Consult demonstrations and tutorials in these areas
- SVN use and setup
- GoogleGroups use and setup
- Server maintenance and cron jobs
- Repository structure exposure

General Training

This is for everyone to follow and understand to allow teams to complete their assigned tasks:

- SVN user login and checking out
- BugZilla use and integration
- General Wiki syntax, layout, and use

Risk Management

During the course of the assigned work for AlgorithmA 2011, potential risks or problems may arise. These may threaten the assurance of SQAP. The following is a table that outlines possible risks and an appropriate response to eliminate that risk:

Risk	Category	Response
Massive Scope of Project	Project	Only construct semi-functional skeleton apps that maintain necessary aspects of MVC Architecture. Leave View elements primitive but functional
Lack of time	Project	Managers will utilize a schedule, adhere to it, and strictly enforce it. If necessary, reorganize teams for better efficiency. Restructure team goals if necessary.
Lack of JavaScript	Project	Coding based team leaders and

knowledge		project managers will conduct tutorial sessions, use Internet resources, and encourage independent study.
Accidental loss of source code	Process	The use of SVN will keep archived records of source code from all previous versions. Server Team will create backups of that just in case.
Lack of teamwork	Project	Managers will implement a management archetype in which team oriented goals are encouraged. Team leaders will enforce and implement this archetype. Teamwork will be demonstrated with a top down model starting from the Management team.
Server crashes	Process	The Server Team will be contacted to address the issue. If one member is not available, the second member of the server team will be contacted. The purpose of having a two-man team for server maintenance prevents the absence of a knowledgeable server curator within the AlgorithmA team. The use of SVN will help alleviate any data loss that may occur.
Security	Project	The Server Team will ensure that all available security updates are applied as soon as they are available.
Quantity of defects	Product	The Documentation will categorize and analyze the severity of defects to ensure that critical defects are found before completion date. The Bugzilla system will be utilized to aid all applicable teams in this task.