DATA ITEM DESCRIPTION

Form Approved
OMB NO.0704-0188

1. TITLE

CMSC 447 GROUP 5 SOFTWARE DEVELOPMENT PLAN (SDP)

2. IDENTIFICATION NUMBER

CMSC-447-5

3. DESCRIPTION/PURPOSE

3.1 This Software Development Plan (SDP) is for CMSC 447 Group 5's Fall 2015 semester project and involves modifying/supplementing an existing code base, *The Parable of the Pentagons: Micromotives and Macrobehaviour*, according to the customer's (Russ Cain's) preference. In the performance of this project, the group will document, produce, and review various software development artifacts in order to facilitate the desired enhancements as provided by the customer. The team will also demonstrate its progress to all of its software stakeholders periodically as each project part comes to fruition.

3.2 The SDP and all the pieces of the software development project will be available at https://github.com/Joho95/Polygons-Project as a tool for monitoring growth. The team will be in day-to-day communication via Skype and applying changes to the project through version control via the Github repository (above). The group will be meeting weekly as dictated through our schedule, and resources shall be provided by the developers accordingly.

4. APPROVAL DATE (YYMMDD)	5. OFFICE OF PRIMARY RESPONSIBILITY	6a. DTIC APPLICABLE	6b. GIDEP APPLICABLE
151012	EC		

7. APPLICATION/INTERRELATIONSHIP

7.1 The Data Item of the project will be a website formatted in HTML and JavaScript and prepared through any editor (as seen fit). The code base and software should be maintained in the Github repository, and changes to such needs to be recorded in versions as commits are made.

7.2 The team will delineate all software tasks accordingly to all of its developers and tested thoroughly on an organized, periodical basis.

7.3 This plan only applies to the specific process of this Data Item (the website).

7.4 The deliverable data shall be delivered online via Blackboard, or emailed to Professor Birrane in PDF/Word format unless otherwise specified by the customer or Professor Birrane. Permission to the Github repository shall be given to all stakeholders, to facilitate any appropriate accesses to the deliverables and/or software head.

7.5 This Data Item simply modifies the existing website code base at http://ncase.me/polygons/

8. APPROVAL LIMITATION	9a. APPLICABLE FORMS	9b. AMSC NUMBER
Limited Approval from 10/12/15 through 11/12/15		N7070

10. PREPARATION INSTRUCTIONS

10.1 General instructions. Follow general instructions from MIL-STD-498.

10.2 Content Requirements. Content requirements begin on the following page. The numbers shown designate the paragraph numbers to be used in the document. Each such number is understood to have the prefix "10.2" within this DID. For example, the paragraph numbered 1.1 is understood to be paragraph 10.2.1.1 within this DID.

11. DISTRIBUTION STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

1. Scope. This section shall be divided into the following paragraphs.

1.1 Identification. The system and software shall consist of a modified code base from https://github.com/ncase/polygons, as well as the documentation, use cases, and other software articles encasing the project in the following repository: https://github.com/Kirkas1/polygons. This software is based from Vi Hart and Nicky Case's *The Parable of the Pentagons: Micromotives and Macrobehaviour*, and has adaptations that the customer and the developer team have previously discussed (later detailed). The presented product will be an HTML page that will be functional in Internet Explorer, FireFox, and Google Chrome. All releases and versions shall be noted in the Github repository as they procure.

1.2 System overview. The system and software are to be used for the interests of the customer (Russ Cain) who has stated that he would like a functioning game based off the original "Parable of the Pentagons" site. The following alterations need to be addressed in the software as required by the customer: implementation of a red circle along with the other pentagons, new algorithms to sort the polygons to supplement the one already in place (detailed later), and radio buttons for each algorithm for the user to select the one to be used. The history of the software before this current project is documented at https://github.com/ncase/polygons, which is an open source repository. The current stakeholders of the system and software include the customer Russ Cain and Professor Birrane and the developers will include CMSC 447 Group 5 (Joseph Peterson, Ian Kirk, Brooke Washington, Christopher Vaughn, and James Hastings). The operating site will include Github and the final HTML page which is open source to the public.

1.3 Document overview. This SDP shall address all the logistical, design, schedule, and software aspects critical to the completion of the project. The goal of such documentation is to explicitly dictate a plan of action to carry out Russ Cain and Professor Birrane's requirements for the "Polygon game" and the rest of the project criteria. The project shall explicitly give access to its developers and stakeholders via an open source repository, which does mean that anyone can see the project (although its access will not be distributed directly to the public). The use of the system and software is not for commercial use and will not be privatized.

2 Referenced Documents.

1. Parable of Polygons, Revised Apr 18, 2015, http://ncase.me/polygons/, Vi Hart and Nicky Case
2. Parable of Polygons Source Code, Revised Oct 25, 2015, https://github.com/ncase/polygons, Vi Hart and Nicky Case
3. Polygons, Revised Dec 9, 2014, https://github.com/dncnmcdougall/polygons, Duncan McDougall

3 Overview of Required Work.

a. Requirements and constraints on the system and software to be developed.
   - The system will have a new shape, red circle, added to the polygons.
   - The system will update the polygon ratio sliders to support the new polygon.
   - The system will have two new algorithms added the will sort the polygons more efficiently.
   - The system will have radio buttons that will allow the user to select which algorithm the system uses to sort the polygons.
   - The system will function correctly on the most recent stable versions of Firefox, Chrome, and Internet Explorer.

b. Requirements and constraints on project documentation.
   - The documentation will be uploaded on the project's GitHub page for version control.

c. Position of project in the system life cycle.

- The project is in the planning section. The team is planning weekly meetings and setting up document and version control infrastructure. The team is also still meeting with the client to establish and clarify expectations.

d. Requirements and constraints on project schedules and resources.
   - The project's use cases and software development plan must be complete by October 12, 2015.
   - The project's software requirements specification and design document must be complete by October 26, 2015.
   - The project's software test plan, description, and results must be complete by November 9, 2015.
   - The project must be presentable with a workable demo by November 30, 2015.

4.1 SOFTWARE DEVELOPMENT PROCESS. The software development process implemented in the creation of the software is what is known as Rational Unified Process , with the general stages of inception, elaboration, construction, and transition. A diagram describing this process in detail is provided below.

Table 4-1, 'The Parable of The Polygons' Software Development Process

General Activity	Brief Description
Inception	Getting the general project initiated and acquainting oneself with the client. In this stage, the client shall describe the requirements to be met in the software; or, in the case that a requirement cannot be met, the team and client negotiate and reach a compromise.
Elaboration	Discussion within the team (and client) when deadlines can be realistically met, who shall do which aspect of the project, what deadlines will be in place, and how the team will keep track of itself (biweekly meetings, weekly meetings, etc.) Architecture, use case diagrams and so on may be included
Construction	Creating the software based off what was chosen in the modeling section above. Testing the code (to see if the shapes actually segregate, it runs properly on IE/FireFox/Chrome, etc.) also is conducted. For our concerns, if the client wants a new requirement during this stage, we shall try to see if our code will permit such a change; and if not able to, we shall try to reach a compromise.
Transition	Presenting the software to the client, and receiving feedback from our client.

In order to create our client's desired rendition of 'The Parable of the Polygons,' our team shall adhere to the above guidelines in the development process. The Rational Unified Process was chosen for our project due to each phase being due at a specific time, creating a linear progression in work. Furthermore, since our project uses open source code, the main objective of our rendition is to alter the

code to the client's needs. All members of our group are responsible for various aspects of the project, but all will be involved when it comes to the coding/testing phase of the project.

4.2 GENERAL PLANS FOR SOFTWARE DEVELOPMENT. As described above, the approach to be used while creating the specialized 'The Parable of the Polygons' software is a generalized Rational Unified Process. The following sections describe how it will be implemented in the context of the project.

4.2.1 Software Development Methods

I. Communication: In this stage, our team will meet with the client in order to discuss requirements, possible limitations (not necessarily but including constraints), expected delivery dates, and so on. Any past, current and possible future requirements will be recorded in a notebook specifically designated for the class, and a simple text file with the same descriptions will be kept on Google Drive.

Our first initial meeting with our client will be done in person. Further correspondence regarding things both regarding software and regarding meetings will be conducted over (university) email and in-person discussions. The point of contact shall be Brooke.

II. Elaboration: Our schedule shall first and foremost be determined by the appropriate due date as described in the course schedule. Each phase represents a specific aspect of our software build. Phase 1 covers communication and planning, Phase 2 modeling, Phase 3 construction (and some modeling if necessary), and Phase 4 deployment.

As of this writing, the group shall hold an official meeting every Friday evening to discuss the current state of the project and to ensure everyone is completing their task within reasonable time. However, team members will be able to discuss the project over a Skype chat, as well as every Monday and Wednesday.

With the above dates in mind, we plan to have each deliverable done at the latest a day before the due date (so, Phase 1: October 11, 2015; Phase 2, October 25, 2015; Phase 3, November 8, 2015; and Phase 4, November 28, 2015). Finishing the phases before the due date will allow us to thoroughly review our material and ensure it meets the standards as described by this template and class standards. We estimate we will need the full time between each phase, with this time frame accounting for discussion, coding, debugging, and requesting the client's thoughts.

The client will hold meetings as needed, but the proposed dates are on Thursday evenings.

As in the above section, the group will be in correspondence via (university) email, as well as Skype.

III. Modeling: Pseudocode tackling the issues will be created in the modeling phase. Each team member will evaluate the pseudocode and will come to a majority rule of what is the most efficient and feasible for the given task. The team will also discuss architecture, use case diagrams, risks (if any), and so on.

Collaborative ideas will be discussed over Skype, but ultimate judging will be done in person.

IV. Construction: The code itself originates from an open source file, so much of the code will most likely find basis in pre-established standards. For instance, to implement another shape, its code will be heavily based on the previously given code of the original two shapes.

Testing will include running it on multiple browsers if necessary. Each person on the team will help to inspect and debug another team member's code, as well as properly integrate and test it (to make sure it functions properly). Team members may or may not work in a pair of 2, depending on how much work the assigned issue requires. If the issue requires a pair of people, the pair will be the ones who proposed it in the modeling section. Team members can help others even if they weren't 'assigned' in solving a specific issue.

Any and all written documentation will be distributed on Skype, as well as edited on Google Drive. The code itself will be hosted on a GitHub created by Joseph, and Brooke will provide drawings, made in SAI and saved in MS Paint of any new shapes. Previous versions of anything (code and drawings) will also be kept in the case of having to restart from a previous version.

V. Deployment: The team will present the software, in whatever state it is, on December 2nd 2015. Feedback will come in the form of classmate's opinions, the instructor's final grade, and the client's final opinion.

4.2.3 Reusable Software Products. Our project is based on open source code, 'The Parable of The Polygons', which can be found on the project description document on Blackboard. Since a majority of our project is simply modifying the source itself, this aspect will make integration easier, since we shall all work from the same code.

However, with all open source code, a limitation is how much we can change before completely breaking the code; and with this in mind, its being open source code inhibits us from making certain changes. Another drawback is dealing with any and all bugs found in the original source code. It may not prove time-effective to fix a certain bug, so our code may have to work around bugs.

A team member may find another type of source code unrelated to the project, such as an art site using a similar slider feature, useful in modifying the project. In such scenarios, the finder and other team members will ensure that the code will work with 'The Parable of the Polygon' code; the main criterion will be easy integration, which implies its own criteria such as the same version of JavaScript. Proper credit will be given to additional source codes.

4.2.4 Handling of Critical Requirements. Our software, in order to be relatively safe to use, mustn't completely make a browser take an excessive amount of time to run a script, eventually rendering the current window of the browser unusable. Each member will test the code on various browsers to make sure it does not break one or more browsers completely.

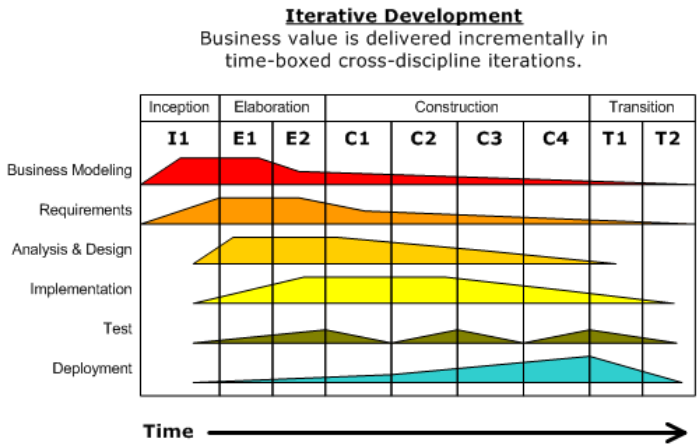
Related to the above, the modified software must also work with browsers such that any one library function does not cause major security issues. Any other source code used must also be safe to involve and not have any hidden exploits in it. Any bugs that could otherwise risk browser safety must be removed as soon as possible.

The team will continuously keep track of the software to ensure that no risks have a probable chance of transpiring. While our project may not have any critical requirements that a medical program may have, nevertheless it is dire to assure the client that the software does not fail to answer to critical requirements.

4.2.5 Computer Hardware Resource Utilization. Each team member will possess their own personal laptop and/or desktop to work on a given assignment. All computers will have access to GitHub to allow concurrent work on the project. The team and client will decide how many people will be able to access the modified website at a given time. If it is determined more hardware such as flash drives is needed, it is up to the discretion of the individual and possibly other teammates, depending on the situation.

5 Plans For Performing Detailed Software Development Activities. The life cycle of the program applies Iterative development strategies as listed in the figure below. Planning through approved documentation for each of our three requirements will be applied as the project progresses through the SDP. The specific activities performed during the Inception, Elaboration, Construction, and Transition phases are

listed in this document in that order, according to the Iterative Development figure. We are Soft Where It Counts or SWIC, and our Customer is Professor Cain.



Each SWIC team member will have a specific role. Project training will be specific to the tasks assigned to a specific role. We will contribute to a Data Centered Architecture based on GitHub. The tasks are as follows:

Project Designer: maintain the fulfillment of requirements through UML and in person technical reviews. Update UML and project design as needed. Give instruction and direction to the Programmers as needed.

Lead Programmer: have a good understanding of HTML and javascript when development starts. Help put the UML into context with the program itself.

Project Manager: Meet with the Customer and the Professor. Stay available via Email and in person with the team. Be familiar with protocol in meetings and be thorough. Be familiar with the project, and stay in touch with Documentation. Direct protocol according to this SDP.

Documentation: Complete the OCD. Complete software documentation and header files. Direct software team based on concepts learned in class.

Quality Assurance: Join weekly formal meetings to keep track of changes in the code, and have a good enough understanding of javascript to make or revoke changes. Design test cases. Test code from the Development branch to push to Master.

5.1 Project Planning and Oversight. The Software Project Manager's direction shall be followed throughout the project in person during weekly meetings and through email and other modes of communication through this SDP. Approved Operational Concept Document (OCD), and project approval from the customer and professor Birrane's suggestions for improvement are required for task completion. The tasks are listed in the following subsections.

5.1.1 Software Development Planning. SWIC will configure the OCD document template to control requirement significance and changes throughout the project based on requirement changes, and resulting risk factors and resource management.

Requirements will be divided into use cases and illustrated by the Software Designer. The diagrammed use cases will then be used to create sequence diagrams. Their purpose will be to analyze the required classes, methods and attributes. With this, a class diagram outlining the structure of the system will be documented. All of the UML will be saved to the sharepoint on the UMBC BOX server.

5.1.2 CSCI Test Planning. CSCIs will be tested upon completion by a designated Quality Control manager. Each method will be tested up to three steps with other methods using the Google Testing open source software which they will prepare.

5.1.3 System Test Planning. After each CSCI is unit tested, integration testing among multiple CSCIs will commence and SWIC will select a peer with no knowledge of computer science to test the working alpha version of the system using the UI. We will have planned use cases as steps for the outside tester to perform.

5.1.4 Software Installation Planning. SWIC will upload the completed system to GitHub and construct a README.md based on the original which addresses and attributes the purpose of this system. The main HTML UI which our product will run on

5.1.5 Software Transition Planning. The system will comprise of code which can be reused for the purpose of simulation or the addition of other features. It will be posted as open source software and as such a good example of a modified version of Parable of Polygons.

5.1.6 Following and Updating Plans, Including the Intervals for Management Review. Reviews of the system will take place throughout development. Some will be informal as peer reviews. These will be between the main Coder and the Quality Control Manager, and the Project Manager and Project Designer. The Project Manager shall be available to do desk checks as well. Formal reviews will take place once a week at a designated time and all available members of SWIC will attend.

5.2 Establishing a Software Development Environment. Each member of SWIC shall have access to the Project in an environment which they are comfortable with and supports their role on the team.

5.2.1 Software Engineering Environment. The JavaScript code and the HTML interface will be modified in a free API similar to Dreamweaver. It will be selected by the Main Coder and the selection will be justified with a description of the advantages and disadvantages it has over the other APIs.

5.2.2 Software Test Environment. Unit testing will run on an external testing framework Google Test, or similar framework which can unit test JavaScript code. The Quality Control manager will be responsible for selecting it and explaining their decision.

5.2.3 Software Development Library. The Parable of Polygons webpage is uploaded to GitHub. The CSCI which we are modifying from it is called Automatic. Its dependencies are in the folders which share the same directory as the automatic directory.

5.2.4 Software Development Files. The two requirement-based items which we are adding to the system will have their own files in directories next to the automatic directory. The interface of automatic.js will be modified to accommodate these items.

5.2.5 Non-deliverable Software.

Each CSCI can be considered a deliverable in this project.

5.3 System Requirements Analysis. The system requirements have been established by the customer, but may change throughout the course of development. Any changes to the requirements must be met with a review of the OCD and scrutiny of the purpose of the change and the OCD should be crafted with the purpose of analyzing the purpose for the requirements and efficient allocation of resources.

5.3.1 Analysis of User Input. The User will have the existing UI of the Automatic object with modified attributes. The slider will accommodate the new shape and the user will have the option to run the efficiency algorithm through radio buttons. All other UI elements will remain the same as the original code.

5.3.2 Operational concept. The method of segregation will be modified to accommodate the new shape, 2 individual preferences, and the movement determination algorithms.

5.3.3 System requirements. The system requirements are currently to accommodate a new shape in the form of a red circle, and to enable the user to select a more efficient movement method or the original. The customer has

5.4 SYSTEM DESIGN. The current system is a webpage demonstration of the automatic.js object. Our system is based specifically on this object and its attributes.

5.4.1 System-Wide Design Decisions. The Project Manager will direct the Designer if the system needs to be redesigned in any way. The current architecture is portable enough for our purposes.

5.4.2 System Architectural Design. The Project Designer will create the UML of the existing system and add the attributes we require to the current system accordingly.

5.5 Software Requirements Analysis. The software requirements are the same as those listed in section 5.3. They will be analyzed at this point with the creation of a Use Case diagram for the current system, and then a modified version to suit our requirements.

5.6 Software Design. The Use Case diagrams formed in the previous section will be used to create Sequence diagrams to identify the required CSCIs, their attributes, and their methods.

5.6.1 CSCI-Wide Design Decisions. The Project Designer will lead the group in creating a Class diagram based on the sequence diagrams to create a map of the relationship between items. This will be approved by the Project Manager or edited by Quality Control.

5.6.2 CSCI Architectural Design. The architecture of each CSCI will be designated by the class diagram's classes, and their methods and attributes.

5.6.3 CSCI Detailed Design. As the coder develops the CSCIs, the lead Writer will give detailed descriptions for each attribute and method and upload their changes to the repository.

5.7 Software Implementation and Unit Testing. The Quality Controller will prepare for and execute implementations of the code and unit testing with the purpose of finding bugs which the Coder should fix. They will add their testing software to a folder in the repository.

5.7.1 Software implementation. The Coder will run builds of the project as development progresses and commit to a branch for Quality Control. The code on this branch will not reach the Master branch without Quality Control's thorough observation through testing and implementation.

5.7.2 Preparing for Unit Testing. Existing testing software such as Google Test will be modified to suit the needs of unit testing the CSCI, their methods, and their attributes.

5.7.3 Performing Unit Testing. The scripted unit tests will run on the code in the Quality Control branch. If they pass the thorough unit tests, the code will be committed to the Master branch and deleted from the QC branch. Otherwise:

5.7.4 Revision and Retesting. The Quality Control manager will send the code back to the development branch after analyzing the errors and giving the Coder guidance to debug based on the errors.

5.7.5 Analyzing and Recording Unit Test Results. The thorough unit tests should be created with understandable output so that a log can be forwarded to the Master branch to prove unit testing occurred or to identify future bugs due to the risk of under-testing.

5.8 Unit Integration and Testing. SWIC shall integrate the system after the independent CSCIs have been prepared.

5.8.1 Preparing for Unit Integration and Testing. When the individual CSCIs have been created and committed, they will all be pulled back to the development branch for integration. The Coder will find any bugs after integrating and building the code and will then send it to the Quality Control branch of the git repository. Quality Control will prepare tests for each use case based on the developed program.

5.8.2 Performing Unit Integration and Testing. Quality control will use sequence diagrams to run tests on each of the program's use cases. They will record any bugs or update the program header to confirm that it is a working version.

5.8.3 Revision and Retesting. If Quality control can manage to fix a found bug, they will mark their modification with a comment. Otherwise, Quality control will report bugs and possible revisions to the Coder for guidance, who will then fix the bug and repeat 5.8.1.

5.8.4 Analyzing and Recording Unit Integration and Test Results. The tests will be analyzed qualitatively as success or failure on the header of the program they correspond to. Bugs and the explanation for them will be recorded on the programs themselves as a comment with FIXME attached.

5.9 CSCI Qualification Testing. SWIC has tested the integrated system at this point. Our customer requires that a common user should be able to use the system however. We need to ensure this through protocols which are more formal and unbiased. Therefore, we will introduce a student with no CS background or any knowledge of the System.

5.9.1 Independence in CSCI Qualification Testing. Qualification testing will be more formal than previous unit tests. It must cover many possible step that the CSCI will take in the system.

5.9.2 Testing on the Target Computer System. The computer system for testing should have commodity hardware, such as a notebook laptop. The specs of the system will be recorded by Documentation.

5.9.3 Preparing for CSCI Qualification Testing. The Designer will construct tests based on the methods of each CSCI in detailed steps that any user could understand regardless of their knowledge of the System. A tester with no computer science background or knowledge of the system will perform the prepared tests to ensure that the requirements are met and exceptions are handled properly.

5.9.4 Dry Run of CSCI Qualification Testing. The tests will be performed on the system by Quality Control before contacting an outside source to minimize the possibility of finding a bug with an outside tester since they may not make themselves as available as a member of the team.

5.9.5 Performing CSCI Qualification Testing. The Outside Tester will run through the detailed CSCI tests. Documentation will record the results of each test as success or failure.

5.9.6 Revision and Retesting. If the tester finds a bug in the system, then see section 5.9.2 again after SWIC debugs the integrated system using the documentation and design.

5.9.7 Analyzing and Recording CSCI Qualification Test Results. Documentation will record the results of each test on an excel document listing each test which was previously carefully selected by Quality Control. Results will not be quantified.

5.10 CSCI/HWCI Integration and Testing. System consists purely of CSCIs with the exception of the computer used to run it. To save the resource of time, We shall skip to 5.11 and run it after 5.9 while the outside tester is on site.

5.11 System Qualification Testing.

5.11.1 Independence in System Qualification Testing. Qualification testing will be more formal than previous tests. It must cover every possible use case for the system in order to ensure it is stable and robust.

5.11.2 Testing on the Target Computer System. The computer system for testing should have commodity hardware, such as a notebook laptop. The full specs of the system will be recorded by Documentation.

5.11.3 Preparing for System Qualification Testing. The Designer will construct tests based on every use case of the System with detailed steps that any user could understand regardless of their knowledge of the System. A tester with no computer science background or knowledge of the system will perform the prepared tests to ensure that the requirements are met and exceptions are handled properly.

5.11.4 Dry Run of System Qualification Testing. The tests will be performed on the system by Quality Control before contacting an outside source to minimize the possibility of finding a bug with an outside tester since they may not make themselves as available as a member of the team.

5.11.5 Performing System Qualification Testing. The Outside Tester will run through the detailed CSCI tests. Documentation will record the results of each test as success or failure.

5.11.6 Revision and Retesting. If the tester finds a bug in the system, then see section 5.11.2 again after SWIC debugs the integrated system using the documentation and design.

5.11.7 Analyzing and Recording System Qualification Test Results. Documentation will record the results of each test on an excel document listing each test which was previously carefully selected by Quality Control. Results will not be quantified at this point.

5.12 Preparing for Software Use. The system should be tailored for the common user regardless of their knowledge about computers or the system design.

5.12.1 Preparing the Executable Software. A main HTML file similar to the Parable of Polygons should be prepared with an instance of automatic.js and instructions for use.

5.12.2 Preparing Version Descriptions for User Sites. As new versions of the program are uploaded to GitHub, descriptions of the changes made must be added. This and the git log command should provide enough detail on changes made between versions.

5.12.3 Preparing User Manuals. A README.md file along with the instructions on the HTML main file should suffice for directions to use our UI. It should be that user friendly and intuitive. If time permits, Documentation can write more detailed instructions in the HTML file.

5.12.4 Installation at User Sites. The program is a web page and may be downloaded from GitHub and run on a web browser.

5.13 Preparing for Software Transition. Throughout development, the role of Quality Assurance and the Project Designer is to ensure that the system is portable and extendable through encapsulation.

5.13.1 Preparing the Executable Software. The executable file will be tailored with instructions for the user after System Qualification testing in the way 5.12.1 describes.

5.13.2 Preparing Source Files. Source files will be documented with headers and method descriptions by Documentation throughout development.

5.13.3 Preparing Version Descriptions for the Support Site. Version descriptions will be recorded before every commit, and GitHub will keep track of the changes each version makes to the software.

5.13.4 Preparing the "as built" CSCI Design and Other Software Support Information. The Project Designer will update the UML based on the updated CSCI design as requirements change throughout development.

5.13.5 Updating the System Design Description. Documentation will update design description based on the updated CSCI design as requirements change throughout development.

5.13.6 Preparing Support Manuals. Support for using the program will be underneath automatic.js on the main HTML site.

5.13.7 Transition to the Designated Support Site. The project will remain on GitHub for now so that it can be downloaded and run on any machine with a web browser.

5.14 Software Configuration Management. CSCIs in this project are Classes defined by javascript files.

5.14.1 Configuration Identification. The CSCIs in this System are automatic.js and the two algorithms. The shapes only exist as attributes to automatic.js. Other CSCIs will be encapsulated in .js files should requirements change.

5.14.2 Configuration Control. Each CSCI will correspond to the attributes and methods of it on the class diagram. All other changes should be noted in the header of the CSCI javascript file.

5.14.3 Configuration Status Accounting. Configuration status must depend on the version control branch it is in. In the development branch, the configuration is not complete or only partial. The Testing branch will contain all or part of the configuration for testing. The Master branch will have a version of the whole configuration which is executable.

5.14.4 Configuration Audits. Quality Control will audit the configuration on the Testing branch. Working code will be pushed on to the Master branch, and bugged code will be pushed back to the Coder on the Developing branch.

5.14.5 Packaging, Storage, Handling, and Delivery. The whole project will be stored on GitHub. The package will be outlined by the UML and the current structure of the Configuration. Delivery involves polishing the executable for user friendliness and instructions.

5.15 Software Product Evaluation.

5.15.1 In-process and Final Software Product Evaluations. Product evaluations will take place later in development when the requirements are met during the formal weekly code reviews. The final one will be with the Customer to prove our product is finished. It will assess the completeness of each requirement based on the use cases which the System can perform fully.

5.15.2 Software Product Evaluation Records, Including Items to be Recorded. Documentation will record each CSCI and the use cases it can and can't perform. These documents will be uploaded to Box.

5.15.3 Independence in Software Product Evaluation. Evaluation should be thorough enough to rule out the possibility of a bug in the Software. Test logs should be included in the evaluations to confirm the information is correct.

5.16 Software Quality Assurance. SWIC will give a member of the Group the role of Quality Assurance. Throughout the project they will work on the Testing branch and give detailed bug and change descriptions in the comments to the Coder or push to Master.

5.16.1 Software Quality Assurance Evaluations. Quality assurance looks for bugs and checks for proper encapsulation and programming style so the program can expand. If no bugs are found and the program runs properly under unit testing, modification and refactoring is considered, and if done the System is tested again. The working system is then pushed to Master.

5.16.2 Software Quality Assurance Records, Including Items to be Recorded. Quality assurance records will be recorded to GitHub through this version control plan. Commits must have a detailed message about refactors.

5.16.3 Independence in Software Quality Assurance. Quality assurance is distinguished from product evaluation because it is more detailed. Other tests ensure whether or not the code works. The quality of the code itself is assessed in this section.

5.17 Corrective Action. Corrective action is the responsibility of Quality Assurance. Documentation will write these reports as the project progresses.

5.17.1 Problem/Change Reports, including items to be recorded (candidate items include project name, originator, problem number, problem name, software element or document affected, origination date, category and priority, description, analyst assigned to the problem, date assigned, date completed, analysis time, recommended solution, impacts, problem status, approval of solution, follow-up actions, corrector, correction date, version where corrected, correction time, description of solution implemented)

Documentation will create a history of changes in the sharepoint. They will use GitHub to record the date, changes made, problems found, and whether Quality Assurance could solve it and push to Master or push it back to Development.

5.17.2 Corrective Action System. The Testing branch will be used by Quality Assurance to make corrections as the code is created. If needed, the Project Designer may get involved in debugging if the problem is too hard to solve or if the Coder is unable to fix a bug on the Development branch.

5.18 Joint Technical and Management Reviews. Reviews will check for the fulfillment of requirements throughout the project during separate specified meetings discussed below.

5.18.1 Joint Technical Reviews. Peer reviews shall be conducted on the code, documentation, and testing of the product during informal gatherings. These are important mainly for bug detection and removal. Management reviews involve the Customer and the Professor. The Project Manager will be responsible for meeting with the management. The Management will be involved in reviews:

- To approve or update the OCD with regards to requirement changes
- To confirm the proposed system design
- To confirm that the requirements have been satisfied upon Software Product Evaluation.
- To help with or report a problem with the code such as a bug.

Status reviews among all available project members will be held weekly. Periodic status reviews with the customer will be held biweekly. Status meetings with the Professor will be held whenever necessary, at least biweekly at a given day for his office hours.

5.19 Risk Management. The following risks and contingency plans are the most significant:

1) Requirements Stability

- Metrics: Count number of requirements
- Risk Reduction Plan: Review requirements in SRS with customer upon establishing them at first. If customer requests to change one, require another SRS review to change the requirement.
- **Contingency Plan:** Reduce functionality to meet cost and schedule or increase schedule / resources to include new / changed requirements.
- **Entrance Criteria:** If / when the customer changes their requirement criteria in the short time we have to code.

2) Overall system testability

- Metrics: Percentage of interfaces tested and verified as correct
- Risk Reduction Plan: Develop a test plan which addresses three step tests on the integrated system. Design and perform complete unit tests. Follow the lecture notes on testing.
- Entrance Criteria: If any required interfaces cannot be tested by their due date.

3) New Technology

- Metrics: Count team members qualified for each discipline necessary for project deployment.
- **Risk Reduction Plan:** Provide training in required languages. Use and enforce design and coding standards.
- **Contingency Plan:** Seek help in the new technology from a professor. Find training online.

6 Schedules and activity network.

a. Schedule

10/16	Start planning/elaboration
10/23	Finish planning/elaboration
10/30	Start developing red circle
11/6	Finish developing red circle
11/13	Start developing sorting algorithm #1
11/20	Finish developing sorting algorithm #1
11/27	Start developing sorting algorithm #2
12/4	Finish developing sorting algorithm #2
12/11	Finish Testing

b. Activity Network

Activity	phase	Time	Needs
Plan project	planning	1 week	n/a
Model project	modeling	1 week	n/a
Plan circle	Planning	1 day	n/a
Develop circle	construction	12 days	n/a
Show circle to customer	deployment	1 day	Circle done
Plan algorithm 1	Planning	1 day	Circle done
Develop algorithm 1	Construction	12 days	Circle done

Show algorithm 1 to customer	Deployment	1 day	Circle, algo 1
Plan algorithm 2	Planning	1 day	Circle, algo 1
Develop algorithm 2	Construction	12 days	Circle, algo 1
Show algorithm 2 to customer	Deployment	1 day	Circle, algo 1, 2
Class presentation	Deployment	1 day	Everything done

7 Project Organization And Resources.

7.1 Project organization. This project will be organized with a direct link between the customer and the development team. The customer, Russ Cain will give guidelines for the project, and we the software development team will carry them out. The customer will be required to give direction based on what he wishes and the development team will be responsible for making these wishes a reality. There will be one point of contact that will serve as the go between for the team and the customer.

7.2 Project resources. During this project there will be the five of us, the development team fulfilling all roles in the project, as well as the client. Within that there will be a manager in charge of making sure members complete their tasks (as well as their own), as well as standard developers. Work will be done over GitHub which the customer will have access to and be able to view at any time, meetings will be held on campus at least once a week, on Fridays. Resources required include the source code as well as an IDE for HTML that will be standardized for each worker on the development team.