Yaroslav Kechkin

Software Development: Text Encryption Program

Gulf Coast State College

MAN 4900

Wendy Payne, Guy Garrett

April 21, 2012

Product Introduction

The program that I've created is the simple program to encrypt text files using ASCII alphabet. I used 128 regular ASCII and 128 extended ASCII characters in the encryption process. Program has a simple menu design that prompts the user to enter their choice of action and enter the path to the text file selected and the program will encrypt the file using either user specified key or generated randomly. The program saves output in a separate text file in the directory where the *.exe was ran. Using 256 characters to encrypt the text provides better security as using 26 alphabetical characters like regular Vigenere cipher uses. The study done by professor Kallam and Dr. Kumar shows that using 256 ASCII characters in encryption algorithm is safe and it would take 2,530,000,000,000,000,000,000 years to brute force it using 1 decryption pre microsecond (Kallam, Kumar, Vinaya, & Kumar, 2011). It is a long time and it does not worth the trouble of hacking. This program is perfect for personal use, but when using it for multiparty usage it raises a few problems discussed later in risk management section.

Cryptography

The design of the program is based on polyalphabetic substitution technique. In this case I used an array 256 by 256 of ASCII characters – modified Vigenere cipher. The key, either provided by user or generated randomly using ASCII alphabet, is extended to match the length of the plaintext so every character is encrypted. If using randomly generated, then user must provide the length of the key to be generated. This technique is preferable because the strength of the key is better than the one provided by user, which can be broken using dictionary attack since the user will most likely use the words that are easy to remember. Random number generator function used is a simple function that is based on the system time. This choice was based on my programing experience which is virtually non-existent, but later I am planning on updating the

program and use multiple encryption cycles and use more sophisticated pseudorandom number generator technique like true random number generator based on source of entropy that can produce random data.

Software Development Life Cycle

Software development life cycle (SDLC) is the process of creating or altering software systems. It consists of five steps: Analysis, Design, Implementation, Testing, and Evaluation. In analysis phase I was gathering needed tools and information needed to start working on the project. Microsoft Visual Studio 2010 Express was used to write the code of the program and www.cplusplus.com website was used to resolve problems that I encountered during the implementation phase. I created the outline for schedule and a work breakdown structure to divide the process on deliverables and allocate time for each step of the project. I created the outline of how the program should look and work to create the architecture in the design phase. The program was required to encrypt the text and save the output in the file for later use and decryption.

Design phase started with creating a pseudo code of the program outlining each function and its role in the program (See Diagram 1 in Appendix B). The design of the program requires user to run the program from the beginning for new iteration because it causes problems with saved values of key and input text (minor flaw which will be eliminated in later versions).

The source code was written during implementation phase. It was written in C++ language using Microsoft Visual Studio 2010 Express.

During testing phase I was carefully checking that every function performs its role without mistakes. During testing phase I found several mistakes with encryption and decryption

modules which were not producing viable results and had to be modified. (See Appendix A for Testing Checklist)

During evaluation phase I was checking the integrity and usability of the program. Criteria were: easy to use, simple and user friendly interface, fast generation of ciphertext or plaintext from ciphertext, use of small amount of memory. Specified criteria were met and program is easy and fast to use even on big amounts of text.

Documentation

User guide to the program:

Run Capstone.exe. The window will open showing the menu with five choices:

1 – Encrypt

2 – Encrypt using Auto Key

3 – Decrypt

4 – Decrypt using Auto Key

5 – Exit

If you will press any other number the program will show the message “Invalid input” and will prompt you to enter a number again.

Option 1

If you will select option 1, the program will ask you to enter the file name. Enter the file name if it is in the same directory (folder) as the Capstone.exe or enter the path to the file if it is in different directory. Use “\\” instead of “\” in a path name. Example: “E:\\New Folder\\File.txt” (without quotes). The program will show the content of the text file. Then you will be prompted to enter the key – you may enter any length key using special characters (!,.?’” etc), alphabet characters (a-z, A-Z), numbers (0-9). The program will show the key and the extended key – the

key that is the same length as a plaintext. And then the program will create the file named "outputEncrypt.txt" in the same folder as Capstone.exe. Program will terminate.

Option 2

If you will select option 2, the program will ask you to enter the file name. Enter the file name if it is in the same directory (folder) as the Capstone.exe or enter the path to the file if it is in different directory. Use "\\" instead of "\" in a path name. Example: "E:\\New Folder\\File.txt" (without quotes). The program will show the content of the text file. Then you will be prompted to enter the number that will reflect the length of the key to use. The program will show the key and the extended key – the key that is the same length as a plaintext. Program then will encrypt the text using this generated key. Finally, the program will create the file named "outputEncryptAutoKey.txt" and the file named "AutoKey.txt" in the same folder as Capstone.exe and will terminate.

Option 3

If you will select option 3, the program will ask you to enter the file name of the encrypted file (outputEncrypt.txt). Enter the file name if it is in the same directory (folder) as the Capstone.exe or enter the path to the file if it is in different directory. Use "\\" instead of "\" in a path name. Example: "E:\\New Folder\\outputEncrypt.txt" (without quotes). The program will show the content of the text file. Then you will be prompted to enter the key that you used to create the cipher text (key used in option 1). The program will show the key and the extended key – the key that is the same length as a ciphertext. Program then will decrypt the text using this key. Finally, the program will create the file named "outputDecrypt.txt" in the same folder as Capstone.exe and will terminate.

Option 4

If you will select option 4, the program will ask you to enter the file name of the encrypted file (outputEncrypAutoKey.txt). Enter the file name if it is in the same directory (folder) as the Capstone.exe or enter the path to the file if it is in different directory. Use "\\" instead of "\" in a path name. Example: "E:\\New Folder\\ outputEncrypAutoKey.txt" (without quotes). The program will show the content of the text file. Then you will be prompted to enter the file name of the key (AutoKey.txt). The program will show the key and the extended key – the key that is the same length as a ciphertext. Program then will decrypt the text using this key. Finally, the program will create the file named "outputDecryptAutoKey.txt" in the same folder as Capstone.exe and will terminate.

Option 5

If you will select option 5 the program will exit.

Quality Management (Six Sigma: DMAIC)

Quality – the degree to which the software satisfies stated and implied requirements. Quality management is continuous process of improving on different steps of the project. During creation of this program I used the process called DMAIC – the Six Sigma quality improvement methodology. DMAIC stands for Define, Measure, Analyze, Improve and Control. DMAIC methodology ties tightly with Software development life cycle.

After finishing the development, I went back and started to look for improvements on the program. First I started over from the beginning and looked over the requirements users might have. I added two more options – auto key encryption and decryption. After redefining user requirements I started gathering information required for me to perform this change. I was reading on pseudorandom number generator function that uses the system time as seed and the

fills out the string with ASCII characters derived from integer values. Then I wrote separate program to test the new part and after it passed initial unit testing I implemented it into a full program. After implementation was successful I tested all part of the program separately to check for unit errors and then tested the whole program to see that all parts are working together. As a last phase of DMAIC methodology I documented all changes and added changes into a user manual. Also lessons learned were documented after each step of the improving process.

After first stage of quality improving, I went back and started the DMAIC process again. I started with identifying the problem. The first version of the program was written in a single file and was very bulky and it was easy to get lost in it. I started studying on functions and constructors. This helped me to divide the program on 3 files: header, source code and main. This helped to break the whole program on separate elements which are easy to look through and understand what part does what. Implementation process was hard and required many hours of testing but it was completed on time before the project deadline. After two cycles of quality improvement, the program now does two types of encryption with decryption options and the outline of the program is easier to understand and follow.

Risk Management

A risk can be defined as a consideration that has some degree of probability of compromising the success of a software development project (Karolak, 1995). Formally, risk can be described as project variables that designate project success (Capers, 1994). Risk defines the probability that the software development project will experience unwanted and unacceptable events, such as termination, delays in project schedule and overrun of project resources.

Main areas of risk that are encountered during the development process are: requirements, architecture of software, performance, non-functional area of software

development – schedule, resources, support. Software requirements are a common term for all users' needs regarding software system functionality and quality of service. It is often very hard to develop the right software solution that absolutely meets users' expectations. These requirements can be divided into two groups: functional and non-functional. Functional requirements define behaviour of the software, while non-functional requirements define qualities and constraints to which the software must conform.

Software architecture must be defined in the early development phases in order to build a quality software solution. It is possible that software architecture defined in the early development phases does not satisfy all of the requirements set on a software solution. The software architecture can be verified only with a software prototype, which can be realized only in later development phases. "Due to the importance of software architecture, many development processes focus directly on software architecture in order to build a quality software solution according to the defined requirements" (Royce, 1998).

The performance of a software solution could be tested only on a real and realized software solution. Thus, it is necessary to make predictions about software system performance in the early development phases. These predictions are very important because it is possible to develop a software solution that satisfies functional requirements, but it is too slow to fulfil performance requirements (Karolak, 1995). The first version of the program, which utilized arrays instead of strings, was very slow and monitoring revealed that it was taking up almost 1GB of memory and it took 5 minutes to complete the encrypting process. This issue was addressed in later version of the program with switching arrays with strings.

The last risk impact area is the non-functional area. The risks in this area could be described as organizational problems and problems related to the project resources and schedule.

“Organizational problems may affect the realization of a software solution since only the efficient organization of software development leads to a successful software development project” (Sertić, 2002). A defined project schedule could become a risk because there are many unpredicted events that could cause a delay in software realization. In order to fulfil defined project deadlines, resources given to the project should be sufficient. This means that every software development project should have enough project members with the right competencies and all the required resources for the planned completion. This area was a problem for this software project because the knowledge of team members was insufficient to produce fast result and it took longer time to produce the code than was intended.

Lessons Learned

It is important to study the issue you are going to address before the start. It will help to get the general knowledge of the functionality of the software.

Do not use arrays where strings could be used. Arrays take a lot of memory space and are slow to work with. Do not use arrays for PRNG generating numbers for keys because after that you will have to put array into a string. It is easier to fill the string with characters generated then use an array to pass values to string.

Write pseudo code in the beginning of the development process and not in the middle, it will help to make development process easier.

Make sure that strings that you use are cleared before using them again; otherwise it will create problems later in the process.

Make sure that your code is easy to follow, use comments.

Known Issues

The issue with this program is the use of keys. It is perfect for personal use because it keeps your files encrypted with the keys that you know. It becomes an issue when the program is used between multiple parties because it raises the issue of sharing the keys. The parties can agree on one key to use that they know, but if they use randomly generated key it becomes an issue to transmit the key to other party. This issue can be addressed with the use of encryption algorithm for the key or use of public-key cryptography. With public key cryptography the exchange of key becomes easier because each party has private and public keys that they use to encrypt/decrypt the message. The next version of the program will have an option for PKI encryption for multiple parties.

References

Capers, J. (1994). Assessment and Control of Software Risks. Prentice Hall PTR, Upper Saddle River NJ.

IT.Toolbox.com (February 9, 2008). Development Testing. Retrieved from http://it.toolbox.com/blogs/enterprise-solutions/development-checklist-22370

Kallam, R. B., Kumar, S. U., Vinaya, A., & Kumar, V. S. (2011). A Contemporary Polyalphabetic Cipher using Comprehensive Vigenere Table. *World of Computer Science and Information Technology Journal (WCSIT),* 1(4), 167-171.

Karolak, W. (1995). Software Engineering Risk Management. Wiley-IEEE Press, San Francisco

Royce, W. (1998). Software Project Management: A unified framework. Addison-Wesley, New York.

Sertić, H. (2002). Applying Unified process on complex software system development. Master Thesis, Economic Faculty, University of Zagreb, Zagreb.

Appendix A: Testing Checklist (it.toolbox.com, 2008)

CHECKLIST FOR DEVELOPMENT				
Questions	**Yes**	**No**	**DP**	**Comments**
Unit Testing				
The unit test environment has been equipped with utilities for backup and recovery of multiple versions of the database.				
The unit test environment has been kept up-to-date with the most current version of each release of the software and the database.				
The test cases documented in the Unit Test Plan have been executed.				
If a code error was detected, the demotion/promotion procedures documented in the Standards and Procedures Manual were followed.				
A Promotion Request form has been completed for each module which has achieved the Unit Test Plan expected results.				
The Build Status in the automated software item directory has been set to "Unit Tested" for each module which has passed unit testing.				
Integration Testing				
The Integration Test Environment has been established as defined in the Standards and Procedures Manual and in the Integration Test Plan. It is a separate test environment.				
The integration test environment has been equipped with utilities for backup and recovery of multiple versions of the database.				
The integration test environment has been kept up-to-date with the most current version of each release of the software and the database.				
The personnel who conducted Integration Test				

are different from those who coded and unit tested the modules.				
The test cases documented in the Integration Test Plan have been executed.				
Documentation of test results has been maintained as stipulated in the test plan.				
Fault Reports have been generated and classified for detected errors.				
An Integration Test Execution Log has been maintained.				
Corrected modules have been re-unit tested before being promoted to the Integration Test environment.				
A Promotion Request form has been completed for each module which has achieved the Integration Test Plan expected results.				
The Build Status in the automated software item directory has been set to "Integration Tested" for each module which has passed Integration Testing.				

Deficiency Evaluation (to be completed by Project Reviewer)

Demerit Points Description (DP)

0 Not applicable (the statement does not apply to the project being reviewed) or no deficiency (the letter and the spirit of the statement is completely true).

1 Minor deficiency (the statement is mostly true).

3 Major deficiency (the statement is partially true).

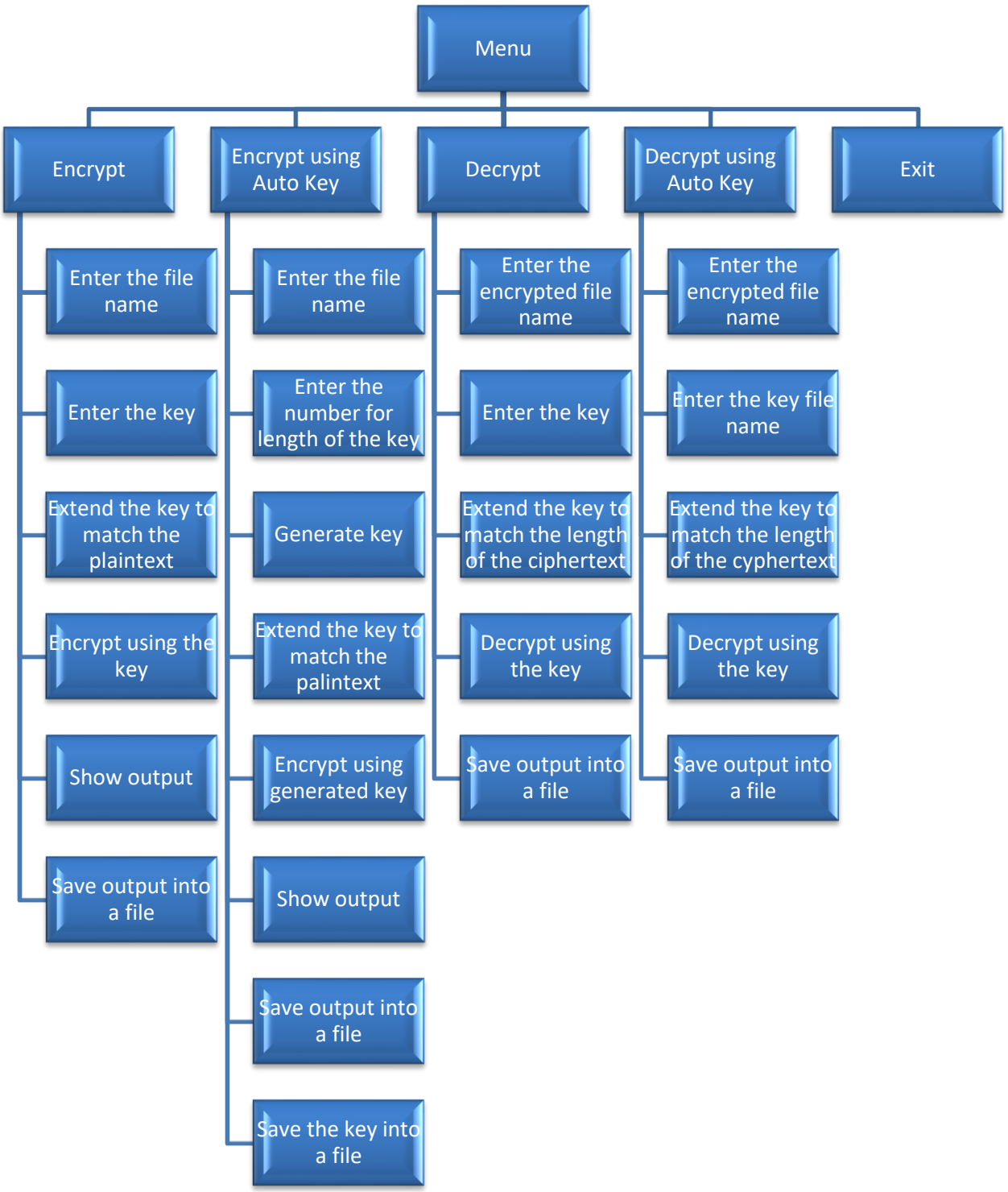
5 Serious deficiency (the statement is largely not true).

10 Critical deficiency (the statement is not true) or the statement cannot be evaluated at this time (which is given a demerit point rating of 20 because of the risk(s) which could be associated with the review item and to encourage the review process to evaluate all statements of compliance).

Maximum possible demerit points (A)	180

Total demerit points assigned (B)	
Rating (1-B/A)	

Appendix B: Diagram 1



Appendix C: Schedule

#	Task Name	Duration	Start	Finish	Predecessors
1	Scope	6 days	2/1/2012 8:00	2/7/2012 17:00	
2	Determine project scope	1 day	2/1/2012 8:00	2/1/2012 17:00	
3	Secure project sponsorship	1 day	2/2/2012 8:00	2/2/2012 17:00	2
4	Define preliminary resources	1 day	2/3/2012 8:00	2/3/2012 17:00	3
5	Secure core resources	2 days	2/4/2012 8:00	2/6/2012 17:00	4
6	Scope complete	1 day	2/7/2012 8:00	2/7/2012 17:00	5
7	Analysis	9 days	2/8/2012 8:00	2/17/2012 17:00	1
8	Conduct needs analysis	1 day	2/8/2012 8:00	2/8/2012 17:00	
9	Draft preliminary software specifications	1 day	2/9/2012 8:00	2/9/2012 17:00	8
10	Develop preliminary budget	1 day	2/10/2012 8:00	2/10/2012 17:00	9
11	Review specifications and budget with team	1 day	2/11/2012 8:00	2/11/2012 17:00	10
12	Incorporate feedback on software/hardware specifications	1 day	2/13/2012 8:00	2/13/2012 17:00	11
13	Develop delivery timeline	1 day	2/14/2012 8:00	2/14/2012 17:00	12
14	Obtain approval to proceed with the project (prepare concept, timeline, budget)	1 day	2/15/2012 8:00	2/15/2012 17:00	13
15	Allocate and secure required resources	1 day	2/16/2012 8:00	2/16/2012 17:00	14
16	Analysis complete	1 day	2/17/2012 8:00	2/17/2012 17:00	15
17	Design	6 days	2/18/2012 8:00	2/24/2012 17:00	7
18	Develop functional specifications	1 day	2/18/2012 8:00	2/18/2012 17:00	

19	Develop prototype based on functional specifications	1 day	2/20/2012 8:00	2/20/2012 17:00	18
20	Review functional specifications	1 day	2/21/2012 8:00	2/21/2012 17:00	19
21	Incorporate feedback into functional specifications	1 day	2/22/2012 8:00	2/22/2012 17:00	20
22	Obtain approval to proceed	1 day	2/23/2012 8:00	2/23/2012 17:00	21
23	Design complete	1 day	2/24/2012 8:00	2/24/2012 17:00	22
24	Development	25 days	2/25/2012 8:00	3/24/2012 17:00	17
25	Review functional specifications	1 day	2/25/2012 8:00	2/25/2012 17:00	
26	Identify design parameters	1 day	2/27/2012 8:00	2/27/2012 17:00	25
27	Assign development staff	1 day	2/28/2012 8:00	2/28/2012 17:00	26
28	Develop Code	20 days	2/29/2012 8:00	3/22/2012 17:00	27
29	Developer testing (debugging)	1 day	3/23/2012 8:00	3/23/2012 17:00	28
30	Development complete	1 day	3/24/2012 8:00	3/24/2012 17:00	29
31	Testing	17 days	3/26/2012 8:00	4/13/2012 17:00	24
32	Develop unit test plans using product specifications	1 day	3/26/2012 8:00	3/26/2012 17:00	
33	Develop integration test plans using product specifications	1 day	3/27/2012 8:00	3/27/2012 17:00	32
34	Unit Testing	16 days	3/27/2012 8:00	4/13/2012 17:00	32
35	Review modular code	1 day	3/27/2012 8:00	3/27/2012 17:00	
36	Test component modules to product specifications	1 day	3/28/2012 8:00	3/28/2012 17:00	35
37	Identify anomalies to product specifications	1 day	3/29/2012 8:00	3/29/2012 17:00	36
38	Modify code	10 days	3/30/2012 8:00	4/10/2012	37

				17:00	
39	Re-test modified code	3 days	4/11/2012 8:00	4/13/2012 17:00	38
40	Unit testing complete	0 days	4/13/2012 17:00	4/13/2012 17:00	39
41	Integration Testing	6 days	3/28/2012 8:00	4/3/2012 17:00	33
42	Test module integration	1 day	3/28/2012 8:00	3/28/2012 17:00	
43	Identify anomalies to specifications	1 day	3/29/2012 8:00	3/29/2012 17:00	42
44	Modify code	2 days	3/30/2012 8:00	3/31/2012 17:00	43
45	Re-test modified code	1 day	4/2/2012 8:00	4/2/2012 17:00	44
46	Integration testing complete	1 day	4/3/2012 8:00	4/3/2012 17:00	45
47	Documentation	9 days	3/26/2012 8:00	4/4/2012 17:00	24
48	Develop Help specification	1 day	3/26/2012 8:00	3/26/2012 17:00	
49	Develop Help system	1 day	3/27/2012 8:00	3/27/2012 17:00	48
50	Review Help documentation	1 day	3/28/2012 8:00	3/28/2012 17:00	49
51	Incorporate Help documentation feedback	1 day	3/29/2012 8:00	3/29/2012 17:00	50
52	Develop user manuals specifications	1 day	3/30/2012 8:00	3/30/2012 17:00	51
53	Develop user manuals	1 day	3/31/2012 8:00	3/31/2012 17:00	52
54	Review all user documentation	1 day	4/2/2012 8:00	4/2/2012 17:00	53
55	Incorporate user documentation feedback	1 day	4/3/2012 8:00	4/3/2012 17:00	54
56	Documentation complete	1 day	4/4/2012 8:00	4/4/2012 17:00	55
57	Post Implementation Review	4 days	4/14/2012 8:00	4/18/2012 17:00	31
58	Document lessons learned	1 day	4/14/2012 8:00	4/14/2012 17:00	
59	Distribute to team members	1 day	4/16/2012 8:00	4/16/2012 17:00	58

60	Create product maintenance team	1 day	4/17/2012 8:00	4/17/2012 17:00	59
61	Post implementation review complete	1 day	4/18/2012 8:00	4/18/2012 17:00	60
62	Project Complete	1 day	4/19/2012 8:00	4/19/2012 17:00	57,47
63	Software product complete	1 day	4/19/2012 8:00	4/19/2012 17:00	
64	Presentation	1 hr	5/2/2012 8:00	5/2/2012 9:00	63

Appendix D: Risk Matrix

Task	Risk Definition	Impact	Severity	Rank	Contingency Plan
Scope					
Determine project scope Secure project sponsorship Define preliminary resources Secure core resources Scope complete					
Analysis (Software and Hardware Requirements)					
Conduct needs analysis Draft preliminary software specifications Develop preliminary budget Review specifications and budget with team Incorporate feedback on software/hardware specifications Develop delivery timeline Obtain approval to proceed with the project (prepare concept, timeline, budget) Allocate and secure required					

resources Analysis complete					
Design					
Review preliminary hardware specifications Develop functional specifications Develop prototype based on functional specifications Review functional specifications Incorporate feedback into functional specifications Obtain approval to proceed Design complete					
Development					
Review functional specifications Identify design parameters Assign development staff Develop Tablet	Functional specification may not be updated Product developing may take longer than planned				Make sure that specifications that are used are up to date Make changes to budget and schedule and allocate more resources if

					needed
Developer testing (primary compatibility between software and hardware)					Test pilot product's functionality
Development complete	Test results were not satisfactory				Send the product back to make changes in functionality
Testing					
Develop unit test plans using product specifications					
Develop integration test plans using product specifications					
Unit Testing					
Review design					
Test hardware components to product specifications					
Identify anomalies to product specifications					
Modify design					
Re-test modified design					
Unit testing complete					
Integration Testing					
Test software compatibilities					

Identify anomalies to specifications					
Modify software specifications to be compatible with hardware					
Re-test modified specifications					
Integration testing complete					
Training					
Develop training specifications for end users					
Develop training specifications for helpdesk support staff					
Develop training materials					
Conduct training usability study					
Finalize training materials					
Training materials complete					
Documentation					
Develop Help specification					
Develop Help system					
Review Help documentation					
Incorporate Help documentation feedback					
Develop user manuals specifications					

Develop user manuals					
Review all user documentation					
Incorporate user documentation feedback					
Documentation complete					
Deployment					
Determine final deployment strategy					
Develop deployment methodology					
Secure deployment resources					
Train support staff					
Deploy Tablet PC on the market					
Deployment complete					
Post Implementation Review					
Document lessons learned					
Distribute to team members					
Create product maintenance team					
Post implementation review complete					
Project Complete					
Product development template complete					