Enigma Machine Project
CMSC 495 Team Enigma

Adolph, Walter Gene
Ikley, Jessica Lyn
Montanez, Rosana
Ohlmacher, Ellen
Winstead, Bryan Matthew
# Contents

1. **Enigma Machine Project Plan** Page 3  
   1.1. Introduction  
   1.2. Project Organization  
   1.3. Managerial Process  
   1.4. Technical Process  
   1.5. Work Packages, Schedule, and Budget  
   1.6. Additional Components  
2. **Mechanical Technical Specification** Page 15  
   2.1. Terms  
   2.2. General  
   2.3. Enigma Version and Components  
   2.4. Enigma Physical Layout  
   2.5. Enigma Circuit  
   2.6. Rotors  
   2.7. Reflectors  
   2.8. Plugboard  
   2.9. Setting the Machine  
   2.10. Example of Complete Enciphering Process  
   2.11. References  
3. **Requirements Matrix** Page 26  
4. **Software Design** Page 27  
   4.1. Introduction  
   4.2. System Overview  
   4.3. System Architecture  
   4.4. Data Design  
   4.5. Component Design  
   4.6. Human Interface Design  
   4.7. Appendices  
5. **User’s Guide** 57  
   5.1. Introduction  
   5.2. Information for User of this Guide  
   5.3. Concept of Operation  
   5.4. Procedures  
   5.5. Information on Software Commands  
   5.6. Error Messages and Problem Resolution  
   5.7. Glossary  
   5.8. Related Information Services  
6. **Test Plan and Results** 69  
   6.1. References  
   6.2. Introduction  
   6.3. Test Items  
   6.4. Software Risk Issues  
   6.5. Features to be Tested  
   6.6. Features Not to be Tested
1 Enigma Machine Project Plan

1.1 Introduction

1.1.2 Project Overview

Historical Overview

Enigma Machines, a series of electro-mechanical machines used for encrypting and decrypting messages, are most famous for their use by Nazi Germany in World War II. Some believe the successful decryption of the ciphers created by these machines contributed significantly to the victory of the Allies. Following their invention in the 1920’s by Arthur Scherbius, the machines increased in complexity by version as the German Military attempted to thwart cryptanalysis. Decryption was achieved initially through the reverse engineering efforts of the Polish Cipher Bureau, and by exploiting mistakes made by German operators. In 1939, the Poles shared their techniques and equipment (Cryptographic Bombe) with the French and British. British intelligence at Bletchley Park were able to build more sophisticated Bombe devices to combat the increasing complexity of the Enigma machines.

Objective

In this project, the team will create a Java program which will emulate an Enigma Machine and produce accurately encrypted or decrypted results from a given input. This software can be used as an educational device, as a historical piece, or as a tool for puzzle creation and decryption in intellectual social gatherings. The team will also spend a portion of the project researching and implementing possible algorithms for decrypting communications that have been encrypted with the Enigma Machine software. These methods may include the following:

1. Completed Goal – Decryption with a known or partially known cryptographic key. (Machine configuration)
2. Completed Goal – Brute force, statistical, or other modern techniques not available to WWII cryptographers
3. Stretch Goal (Researched, but not implemented) – Decryption with a “discovered” codebook (List of possible cryptographic keys)
4. Stretch Goal (Researched, but not implemented) – Decryption by comparison to a different message encrypted with the same key
5. Stretch Goal (Researched, but not implemented) – Decryption by exploiting known flaws in the machine

Design and User Interface

The Enigma Machine version will determine the complexity of the project. An early machine may have three rotors and no plug board. A later version could have four rotors of a possible subset of eight to ten and a plug board. The goal of this project is to provide an Enigma Machine of variable version, allowing the user to select the number and type of rotors used and if the plug board and ring settings will be required. The user will be provided multiple options for
plain or cipher text input into the machine. These may include the following:

1. Completed Goal – Copy/Paste or typed input into a text box
2. Completed Goal – Submit a text file for encryption/decryption
3. Completed Goal – Manual key input simulating a machine
4. Completed Goal – The ability to switch back and forth between the Enigma GUI and a Cryptanalysis GUI via the use of a tabbed pane
5. Completed Goal – A machine version dropdown that restricts rotor and reflector choices based on historically accurate Enigma Machine types
6. Completed Goal – Dropdown that allows the user to choose output text space (groups of four letters, five letters, no spaces, etc)
7. Completed Goal – A reset button
8. Stretch Goal (Researched, but not implemented) – Accurate machine sounds

**User Required Specifications**

**Software:**
1. JRE 7.0 or higher
2. Browser capable of running JRE 7.0+
3. Requirement: Software shall correctly perform encryption and decryption of text using the Enigma scheme.
4. Requirement: Software must provide a UI to input plaintext and output cipher text, and vice versa.
5. Mac, Windows 7+, and Linux

**Hardware:**
6. Machine capable of supporting a web browser and JRE 7.0.

**1.2 Project Deliverables**

All Deliverables will be submitted to Dr. Duchon on the provided delivery dates. Documentation and Code where applicable will be submitted to the CMSC 495 WebTycho Assignments folders of each student. Progress updates and documentation will also be posted to the CMSC 495 Web Tycho Conference topics dedicated to the appropriate weekly assignment.

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and submit a draft Project Plan. Language, IDE, Collaboration, and</td>
<td>November 3, 2013</td>
</tr>
<tr>
<td>primary design objectives defined.</td>
<td></td>
</tr>
<tr>
<td>Compile and submit draft Design documentation.</td>
<td>November 17, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 1 Documentation. Demonstrable Rotor, Reflector,</td>
<td>November 24, 2013</td>
</tr>
<tr>
<td>and Stepping Mechanism code complete, reviewed, and tested. UI Wireframes</td>
<td></td>
</tr>
<tr>
<td>and Block-out UI in progress. Ring Setting and Plugboard code complete.</td>
<td></td>
</tr>
<tr>
<td>Compile and submit Phase 2 Documentation. Demonstrable Ring Setting, Plugboard, Light Board, Txt File Encryption, and Text Entry checking reviewed and tested. Blockout UI complete, reviewed, and tested. Art direction selected</td>
<td>December 1, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 3 Documentation. Demonstrable decryption/encryption, Cryptanalysis, and machine versioning code complete, reviewed, and tested. UI Art complete and reviewed. Stretch goal progress evaluated.</td>
<td>December 8, 2013</td>
</tr>
<tr>
<td>Compile and submit final Project code and Documentation. Stretch goals implemented only if reviewed and tested. Successful and unsuccessful investigations documented.</td>
<td>December 15, 2013</td>
</tr>
</tbody>
</table>

### 1.1.3 Evolution of the SPMP

The SPMP will evolve over the course of the project. All team members have access to the document in Google Drive and will make changes and corrections. Versioning will be managed using a combination of Google Drive’s version recording and comments functionality.

11/22/2013: Changes to Deliverables: Ring Settings and Plugboard Settings code completed early. Moved Stretch goals for UI and Cryptanalysis into phase 2 and 3.

### 1.1.4 Reference Materials

**Reference for historical and mechanical breakdowns**


Reference for document and technical designs


13. The contents of the Practical Cryptography site were used as a basis for our Cryptanalysis QuadBomb. http://practicalcryptography.com


http://www.gutenberg.org/

1.1.5 Definitions and Acronyms
Definitions of uncommon words and acronyms used in this document.

1. Substitution – A cipher algorithm which, in this project, substitutes one alphabetic character for another.

2. Shift – A cipher algorithm which, in this project, replaces a letter in a known list with the letter one place down in the list, wrapping the first letter around to the end of the list.

3. Rotor – A gear-like component which is hard-coded for a specific substitution permutation. There are 10 possible rotors, the number, combination, and order of which determine the output and complexity of the final result.

4. Reflector – A machine component which performs a substitution and returns the result to the rotors.

5. Stepping Mechanism or Carry Notch – A specific pre-determined character on each rotor which will trigger a shift of the rotor to the left.

6. Ring Setting – A setting which shifts the initial permutation and carry notch position of a rotor.

7. Plugboard or ‘Stecker board’ – A manually hard-wired substitution method added to the Enigma Machine in an attempt to thwart decryption.

8. PT or Plaintext – The raw, unencrypted user input.

9. CT or Ciphertext – The result of the full encryption on the plaintext.

10. Key – For the purposes of this project, a message which indicates the rotor, reflector, ring, and plug board settings necessary to decrypt a message that has been encrypted.

11. NLP (Natural Language Processing) – Machine interpretation and understanding of natural language input.

12. Quadgram Analysis (or Quad Bomb) – Frequency/Statistical analysis of blocks of 4
characters from a cryptext. The cryptext is decrypted with multiple keys, and the most likely key is chosen based on the result which produces the most english-like groups of 4 characters.
13. Bigram Analysis – Statistical/frequency analysis much like the Quadgram, but analyzing groups of only two characters for most common syllables.
14. N-gram Analysis – Statistical/Frequency character analysis. In the case of this program, single letter to three-letter syllabic English language frequency analysis. The Enigma Machine Project does not analyse character groups over four.
15. Corpus – A large body of texts which can be processed to build a database of letter, word, and phrase frequency.

1.2 Project Organization

1.2.1 Process Model

The end product of this project will be divided into smaller demonstrable milestones (See Project Deliverables). Each component must go through the following checklist before it is considered “complete”:

**Code**

1. Requirements gathered
2. Design complete
3. Design Lead sign-off
4. Code Complete
5. Code Review Passed
6. Functionality documented
7. QA Tested
8. Bugs Fixed
9. Testing Lead sign-off
10. Code Lead sign-off
11. Requirement lead sign-off

**UI**

1. Requirements gathered
2. Wireframes Complete
3. Design Lead sign-off
4. Block-out UI Implemented
5. Functionality documented
6. Block-out UI QA tested
7. Bugs fixed
8. Testing lead sign-off
9. Final art direction approved by team
10. Final art implemented
11. Requirement lead sign-off

Documentation
1. Documentation updated
2. QA verified for accuracy
3. QA Lead sign-off
4. Planning lead sign-off

1.2.2 Organizational Structure
Roles and Responsibilities
Every team member will participate in each phase of the project, with the following members taking on the role of lead in their listed section:
1. Ellen Ohlmacher – Project – planning
2. Jessica Lyn Ikley – Testing
3. Bryan Matthew Winstead – Definition and Requirements
4. Rosana Montanez – Design
5. Walter Gene Adolph – Coding

1.2.3 Organizational Boundaries and Interfaces
Tasks and Planning
Group planning will be conducted in 3 locations. On days when group schedules differ, a Google Group has been created for lengthy conversations. On determined meeting days, a text chat room will be opened up for direct person-to-person communication. Task tracking, deadlines, and progress will be managed on a Trello board. This board has been made public so that the Professor can view progress. https://trello.com/b/9iYOHSr

1. 2.4 Project Responsibilities
All team members will contribute to every phase of the project, with team leads taking final responsibility for their respective departments. These departments are divided as follows:
1. Project / Planning – Responsible for project “creative vision,” development method and strategy, overseeing task tracking and organization, project submission sign-off, resource management, and risk management.
2. Testing – Responsible for the creation of test plans and strategies, code testing, UI testing, documentation verification, bug fix verification, and sign-off on all phases of the project.
3. Definition/Requirements – Responsible for project specifications and requirement gathering before any phase of the project, and sign-off at the end of any phase to verify requirements were met.

4. Design – Responsible for design documentation, diagrams, wireframes/mock-ups, and sign-off on initial and revised designs on every stage of the project.

5. Coding – Responsible for setting and upholding coding standards, implementing code, reviewing code, and signing-off on submitted code before it is “complete.”

1.3 Managerial Process

1.3.1 Management Objectives and Priorities

The team as a whole will set major goals and milestones together. Individuals will break those goals down into tasks and provide time estimates for those tasks. The project manager will oversee the progress and negotiate any goal slippage or bring in stretch goals where appropriate. Team members will update their tasks on Trello as often as time permits, meeting once per week for at least one hour to discuss progress openly as a team.

1.3.2 Assumptions, Dependencies, Constraints

Assumptions upon which this project is based.

1. The machine itself should be capable of encrypting any language using the 26 characters of the modern Latin alphabet, but the UI, documentation, and code comments will be in English.

2. Program performance will vary based upon the size of the input text.

3. Program access will depend upon the user having access to an appropriate OS and Browser.

1.3.3 Risk Management

Risk factors and Contingency plans

<table>
<thead>
<tr>
<th>Risk Factor</th>
<th>Contingency Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline, Estimation, or Goal Slippage</td>
<td>Re-Evaluation of goals and stretch goals. Re-design of project to eliminate complexity or reduce optional steps. Re-balancing of workload to team members with more availability.</td>
</tr>
<tr>
<td>Team member fails to meet expectations</td>
<td>Direct discussion of issues with team member in a mature and professional manner. Give the team member an opportunity to improve. Re-distribute workload if improvement is not seen.</td>
</tr>
<tr>
<td>Team member is lost due to unforeseen circumstances</td>
<td>Re-Evaluation of goals and stretch goals. Re-design of project to eliminate complexity or reduce optional steps. Re-balancing of workload to remaining team members.</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Media storage or disk drive failure</td>
<td>Redundancy among team members. Locally saved copies of code and documentation.</td>
</tr>
</tbody>
</table>

### 1.3.4 Monitoring and Controlling Mechanisms

1. Individual tasks will be tracked in Trello.
2. Designs will be updated in Google Drive/Docs, using the built-in versioning information and comments.
3. Code will be updated and version controlled via GitHub.
4. Bug tracking will be conducted in GitHub.

### 1.3.5 Staffing Plan

Five team members will adequately provide staffing for all departments during this project.

### 1.4 Technical Process

#### 1.4.1 Methods, Tools, and Techniques

**Required Specifications and Tools for Project Development**

1. Language: Java JDK 7.0 or higher
2. IDE: Eclipse
3. Collaboration: GitHub
4. Documentation: Microsoft Office Suite, Google Drive
5. Communication: Google Chat, Google Groups
6. Task Tracking: Trello
7. Testing: Internet Browser with JRE 7.0+
8. Automated Builds: Drone.IO

#### 1.4.2 Software Documentation

Documentation will be updated in a Word document or documents, following a similar format to the planning documentation. Informative code comments and headers will also be required.

#### 1.4.3 Project Support Functions

Quality Assurance test plans will be created for every applicable component of the project. Each component must be tested. Any bugs discovered must be resolved and that resolution must be verified by the tester who discovered the bug. No code or document is considered “complete” if it has not been approved by the Testing Lead.
1.5 Work Packages, Schedule, and Budget

1.5.1 Work Packages

<table>
<thead>
<tr>
<th>Phase</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase I</td>
<td>Code implemented, reviewed, and tested for a first generation Enigma Machine. This includes code representing the rotors, reflectors, and stepping mechanism. UI should be block-out, and functional enough to test and verify the current stage of development. Documentation should be compiled for submission.</td>
</tr>
<tr>
<td>Phase II</td>
<td>Code implemented, reviewed, and tested for a later generation Enigma Machine. This includes code representing the plug board and ring setting. UI should be block-out, complete, and QA tested. Documentation should be compiled for submission. Cryptanalysis should be in development.</td>
</tr>
<tr>
<td>Phase III</td>
<td>Code implemented, reviewed, and tested for a variable generations of Enigma Machines. Decryption/encryption should be accurate, and the user should be able to select the machine version. UI should be art complete. Documentation should be compiled for submission. Stretch goals and decryption algorithms should be reviewed. Cryptanalysis should be Demonstrable and tested.</td>
</tr>
<tr>
<td>Final Submission</td>
<td>Bugs should be resolved and sign-offs complete. Stretch goals if implemented should be approved and QA tested. Investigations should be documented. Final project should be compiled for submission and approved by all team members.</td>
</tr>
</tbody>
</table>

1.5.2 Dependencies
Each phase relies on the completion of the phase before it to continue.

1.5.3 Resource Requirements
This project relies on all five members of the current team having access to machines with Eclipse, GitHub, Java SDK 7.0+, and an internet connection for five to twelve hours per week.

1.5.4 Budget and Resource Allocation
All software used in development of this project is freeware.

1.5.5 Schedule

<table>
<thead>
<tr>
<th>Phase</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase I</td>
<td>24-Nov-13</td>
</tr>
<tr>
<td>Phase II</td>
<td>1-Dec-13</td>
</tr>
<tr>
<td>Phase III</td>
<td>8-Dec-13</td>
</tr>
<tr>
<td>Final</td>
<td>15-Dec-13</td>
</tr>
</tbody>
</table>
1.6 Additional Components

1.6.1 Index
Index currently covered under 1.5 – Definitions and Acronyms.

1.6.1 Appendices
This section contains supporting information related to the project plan which may be too complex or distracting for the sections above.

Enigma Machine Physical Components

3 rotors, each a wheel of letters representing a substitution cipher.

1. Three to four physical wheels with hard coded values, selected from an optional set of up to 10 possible rotors depending upon the version of the machine.
2. The values on each wheel represent a specific permutation of the alphabet.
3. Offsetting the starting location of the wheel shifts the substitution on the wheel. If A is aligned with E and B with K, shifting the starting location by 1 would align A with K.
4. Substitution is processed for a single character of the encoded message at a time, substituting against each wheel, right to left. In a 3-wheel configuration using wheels I, II, and III, if the entry letter is G, the substitution would be G->C, C->D, D-> F. The output is sent to the reflector.
5. After returning from the reflector, current flows back through rotors and performs inverse shifts. If the output from the reflector is S, then the inverse substitution using rotors I, II, III is S->S, S->E, E->P.
6. The rightmost rotor rotates 1 position with each button press.

Below is the substitution table used by each rotor:

| Rot or I | E | K | M | F | L | G | D | Q | V | Z | N | T | O | W | Y | H | X | U | S | P | A | I | B | R | C | J |
| Rot or II | A | J | D | K | S | I | R | U | X | B | L | H | W | T | M | C | Q | G | Z | N | P | Y | F | V | O | E |
| Rot or III| B | D | F | H | J | L | C | P | R | T | X | V | Z | N | Y | E | I | W | G | A | K | M | U | S | Q | O |
| Rot or IV | E | S | O | V | P | Z | J | A | Y | Q | U | I | R | H | X | L | N | F | T | G | K | D | C | M | W | B |
| Rot      | V | Z | B | R | G | I | T | Y | U | P | S | D | N | H | L | X | A | W | M | J | Q | O | F | E | C | K |
2-4 Reflectors, each also representing substitution-swap.

1. Swappable connections of one letter to another.
2. Input comes from the rotor output. For example, if the rotor output into the reflector is F, the output is S.

Below is a table of the reflector substitutions:

| reflector B Dünn (Thin) | (AE) (BN) (CK) (DQ) (FU) (GY) (HW) (IJ) (LO) (MP) (RX) (SZ) (TV) |
| reflector C Dünn (Thin) | (AR) (BD) (CO) (EJ) (FN) (GT) (HK) (IV) (LM) (PW) (QZ) (SX) (UY) |
| reflector B | (AY) (BR) (CU) (DH) (EQ) (FS) (GL) (IP) (JX) (KN) (MO) (TZ) (VW) |
| reflector C | (AF) (BV) (CP) (DJ) (EI) (GO) (HY) (KR) (LZ) (MX) (NW) (TQ) (SU) |

Below is a table of the reflector substitutions:

| reflector B | (AY) (BR) (CU) (DH) (EQ) (FS) (GL) (IP) (JX) (KN) (MO) (TZ) (VW) |
| reflector C | (AF) (BV) (CP) (DJ) (EI) (GO) (HY) (KR) (LZ) (MX) (NW) (TQ) (SU) |

| reflector B Dünn (Thin) | (AE) (BN) (CK) (DQ) (FU) (GY) (HW) (IJ) (LO) (MP) (RX) (SZ) (TV) |
| reflector C Dünn (Thin) | (AR) (BD) (CO) (EJ) (FN) (GT) (HK) (IV) (LM) (PW) (QZ) (SX) (UY) |
**Stepping Mechanism**

1. Each rotor contains a specific character which, when it comes up, has a carry notch that will trigger the rotor to its left to shift. (Like a regular calculator would carry from nine to ten.)

2. At specified intervals, the carry notch will cause the middle and eventually the rightmost rotors to turn.

3. A flaw of this system – The carry notch was different on each rotor, helping diagnose the rightmost ring.

4. Positions:

<table>
<thead>
<tr>
<th>Rotor I</th>
<th>at R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotor II</td>
<td>at F</td>
</tr>
<tr>
<td>Rotor III</td>
<td>at W</td>
</tr>
<tr>
<td>Rotor IV</td>
<td>at K</td>
</tr>
<tr>
<td>Rotor V</td>
<td>at A</td>
</tr>
<tr>
<td>Rotors VI, VII and VIII</td>
<td>at A and at N</td>
</tr>
</tbody>
</table>

**Ring Setting**

1. A shift upon the initial position of the output. (A = K instead of A = E)

2. Affects carry notch based decryption for stepping mechanism by changing the output in relation to the viewed character, this avoiding the flaw that would allow diagnosis of the rightmost rotor.

**Plug board**

1. A manually configurable direct mapping of one letter to another. A simple mono-alphabetic substitution cypher that could be used to increase the complexity of the rotors.

2. Up to 13 pairs could be matched, but ten were typically used.

3. A flaw of this system – no letter could be enciphered to itself.

**Keys & Lamps**

1. Typewriter keys are used to type input.
2. Lamps light up to display output – P would light up in example.

3. Keyboard layout =

```
Q W E R T Z U I O
A S D F G H J K
P Y X C V B N M L
```

**Example Walkthrough**

Say we want to encrypt the word ENIGMA using Rotors I, II, and III, with reflector B (for simplicity, we will assume that the rotors do not step in this case). We see that the substitution after processing through Rotors III, then II, and finally I is ENIGMA -> JNRCZB-> BTGDEJ-> KPDFLZ, using the above table. Next, the transformed word is processed through the reflector. So, now we have KPDFLZ -> NIHSGT. Finally, the word is sent back through the rotors in reverse, and the output is the encrypted word. For our example, we have NIHSGT -> KVPFSL -> DXUEWK -> BKWPRU. BKWPRU is our encrypted word for ENIGMA.

## 2 Mechanical Technical Specification

This section’s intent is to provide a summary of key mechanical functions of the Enigma machine. This section serves as a guide to develop a software Enigma machine simulator. To facilitate further research by the reader, source references are provided at the end of the section. This section is not, in itself, a software design document and does not contain information related to the Enigma simulator being developed by Team Enigma. Rather, it is a collection of information about the actual Enigma Machines that Team Enigma found useful to reference during the design and programming of the simulator.

Topics that are covered in other project sections are abbreviated and the reader is referred to the appropriate section. English terms are used to describe each component to provide a standardized term nomenclature. The component German equivalent words are listed in section 2.1 to facilitate term standardization and translation.

### 2.1 Terms

1. Plug board. *Steckerverbindung* in German. Also refer as the ‘Stecker board’
2. Rotors, *Walzen* in German
3. Reflector, *Umkehrwalze or UKW* in German
4. Ring Setting, *Ringstellung* in German
5. Rotor order for that date, *Walzenlage* in German
6. Discriminant for that date, *Kenngruppen* in German
7. Indicator-setting, *Grundstellung* in German (Section 1.1.4, Reference 8)
8. Date, *Datum* in German
2.2 General

Enigma is a modern cryptographic machine that uses the principle of polyalphabetic substitution to encrypt and decrypt messages. The machine rotor stepping mechanism provides further keyed permutation to prevent frequency analysis, a known weakness of all substitution ciphers. For more details, see Section 1.1.4, Reference 1, 7, 11, and Section 4, Enigma Machine User’s Guide.

2.3 Enigma Version and Components

The following table provides a list of the different Enigma Versions and their respective components, sources from Section 1.1.4, Reference 2.

<table>
<thead>
<tr>
<th>Machine Type</th>
<th>Rotor</th>
<th>Reflector</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3 Army</td>
<td>I</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>II</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>III</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VII</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VIII</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gamma</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Beta</td>
<td>B-Thin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C-Thin</td>
</tr>
<tr>
<td>M4 Naval</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(all)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>M4 R2</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>M4 R1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Enigma 1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Special Note:
1. Gamma and Beta Rotors can only be used in the fourth rotor position
2. Thin or Dünn reflectors can only be used in conjunction with the Gamma and Beta Rotors.

2.4 Enigma Physical Layout
Key Features

1. Rotors and reflectors are accessible by lifting the cover of the wheels and lamp (see right top corner Fig. 2). To remove the reflectors and rotor the operator must lower the wheel release lever.
2. Plug board. Located at the bottom of the machine. This component allows the operator hardwire swapping of pair of letters.
3. Light board. After a letter has being input, it corresponding output letter lights up on the board.
4. Wheel Cover. Component that sits on top of the rotors (Fig 3 – Center and Right). It provides windows that show each rotor alphabet ring. Lifting the wheel cover allows access to change the values that display in the window; this is an operation required when changing indicator and message setting.

### 2.5 Enigma Circuit

For more details on this section refer to Section 1.1.4, Reference 1.

![Enigma Machine Circuit Diagram](image)

The keyboard was laid out as follows:

```
Q W E R T Z U I O
A S D F G H J K
P Y X C V B N M L
```

The same arrangement was used for the lamp panel and the plugboard.

*Figure 4: Enigma Machine Circuit Diagram [Section 1.1.4, Reference 1]*

### 2.6 Rotors
2.6.1 Physical Components

Information and illustrations are excerpts from Section 1.1.4, Reference 10.

The rotors are the most important elements of the machine. These round disks [9], approximately four inch in diameter, are made from metal or bakelite and have a core with 26 spring-loaded contacts [6] on the right side, scramble wired [5] to 26 flat contacts [4] on the left side, with a hollow axle in the center [8]. On the outside of the wiring core there’s a movable ring [3] with 26 numbers or letters and a notch [1]. This ring is rotatable and is locked with a spring-loaded pin (Wehrmacht) [7] or two spring-loaded arcs (Kriegsmarine) into any of the 26 positions. Changing the position of the ring will change the position of the notch and alphabet, relative to the internal wiring. This setting is called the ring setting or Ringstellung and its position is visible by a dot marking [2]. Each rotor has on its left a notch [1] and on its right a ratchet [10]. These are used by the stepping mechanism to advance the rotors. The internal wiring is different for each rotor. This wiring represents a substitution encryption. The combination of several rotors, in ever-changing positions relative to each other, is what makes the encryption so complex.

*Figure 5: Rotor Components*

[1] Notch. Use for the stepping mechanism
[2] Dot Marking. Used to indicate the setting of the ring.
[5] Scramble. Wired to the 26 flat contacts
[6] Spring loading contact
[8] Rotor axle
[9] Finger notch used to turn the rotors to a start position. The position of the rotor will display in the window inside the wheel cover.
[10] Ratchet
2.6.2 Rotor Wiring Tables

The internal wiring is responsible for the encryption. The wiring of the rotor depends on the type of the rotor. For more details on wiring, see Section 1.1.4, Reference 1 and 10, and Section 4, Enigma Machine User’s Guide.

Table 1: Rotor Permutation Table [Section 1.1.4, Reference 1]

| INPUT | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Rotor I | E | K | M | F | L | G | D | Q | V | Z | N | T | O | W | Y | H | X | U | S | P | A | I | B | R | C | J |
| Rotor II | A | J | D | K | S | I | R | U | X | B | L | H | W | T | M | C | Q | G | Z | N | P | Y | F | V | O | E |
| Rotor III | B | D | F | H | J | L | C | P | R | T | X | V | Z | N | Y | E | I | W | G | A | K | M | U | S | Q | O |
| Rotor IV | E | S | O | V | P | Z | J | A | Y | Q | U | I | R | H | X | L | N | F | T | C | K | D | C | M | W | B |
| Rotor V | V | Z | B | R | G | I | T | Y | U | P | S | D | N | H | L | X | A | W | M | J | Q | O | F | E | C | K |
| Rotor VI | J | P | G | V | Q | U | M | F | Y | Q | B | E | N | H | Z | R | D | K | A | S | X | L | I | C | T | W |
| Beta rotor | L | E | Y | J | V | C | N | I | X | W | P | B | Q | M | D | R | T | A | K | Z | G | F | U | H | O | S |
| Gamma rotor | F | S | D | K | A | N | U | E | R | H | M | B | T | I | Y | C | W | L | Q | P | Z | X | V | G | J | D |

2.6.3 Rotor – Stepping Mechanism

The stepping mechanism provides a way to advance the rotors while the message is input. This results in a different substitution each time for a given letter. Rotors are advanced according to their position in the wheel. The initial stepping design is as such, (extract from Section 1.1.4, Reference 10):

1. The first rotor, on the right side, steps on each depressed key before the input is processed.
2. The middle rotor advances once on every 26 advances of the first rotor.
3. The third, slowest, rotor, on every 26 advances of the middle rotor.
4. In configurations that allow a fourth rotor, this rotor is not part of the stepping process.

The way in which this features was implemented allowed to map the stepping process to letter display in each rotor window. Table 2 shows the location of the notch in respect to the letters in the rotor. In the case of Rotor Type I, the stepping notch is by the letter Y. The notch has to make contact with the pawls for the stepping to take place; when this happens letter Q will be visible in the window. When the rotor goes from Q to R the rotor to the left is increased one step further.

Illustrations below are excerpts from Section 1.1.4, Reference 10.
This mechanism also has an additional feature called double stepping. Double stepping occurs when two adjacent rotor notches are in contact with their respective pawls at the same time, which will be noted by the operator as two consecutive stepping values shown in adjacent rotor windows. The following sequence of rotors III – II – I will result in double stepping.

KDO, KDP, KDQ, KER, LFS, LFU

### 2.6.4 Rotor Encryption Process

Information and illustrations are paraphrased excerpts from Section 1.1.4, Reference 10.

This example provides detailed explanation of the encryption process for Rotor I in the right position. In Figure 5, notice the ring setting is identified by a black mark by the movable ring (grey). In both rotors, the letters outside of the rotor are the input (right) and output (left) respectively. Although not visible, the letters in the alphabetic ring (white) are wired internally according to Rotor I wiring table. The letter in black box is the position of the rotor; this letter would display on the rotor window.

In this example, the letter ‘A’ is input twice. Initially, the ring and indicator settings are set to ‘A’. The left picture indicates the input of the first letter ‘A’ that maps to
‘E’, and then on the way out it is mapped to ‘E’ again.

Then the alphabetic ring (white) moves one step forward. This has the effect mapping the input ‘A’ first to ‘K’ and then to ‘J’ on the way out. Table 4 shows the same process but in table form.

Notice that this example does not take into account the fact that stepping takes place before an input letter is processed.
2.7 Reflectors

After the letter has gone through the rotor, it reaches the reflector. The reflector provides simple substitution. Like the rotors, each reflector substitution matches a particular letter. Unlike the rotors, input and corresponding output are not affected by other mechanism, like a stepping.

Table 4: Reflectors Permutation Table [Section 1.1.4, Reference 1]

<table>
<thead>
<tr>
<th>reflector B</th>
<th>(AY) (BR) (CU) (DH) (EQ) (FS) (GL) (IP) (JX) (KN) (MO) (TZ) (VW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>reflector C</td>
<td>(AF) (BV) (CP) (DJ) (EI) (GO) (HY) (KR) (LZ) (MX) (NW) (TQ) (SU)</td>
</tr>
<tr>
<td>reflector B Dünn</td>
<td>(AE) (BN) (CK) (DQ) (FU) (GY) (HW) (IJ) (LO) (MP) (RX) (SZ) (TV)</td>
</tr>
<tr>
<td>reflector C Dünn</td>
<td>(AR) (BD) (CO) (EI) (FN) (GT) (HK) (IV) (LM) (PW) (QZ) (SX) (UY)</td>
</tr>
</tbody>
</table>

2.8 Plug Board

The plug board provides a mechanism for hard wiring and rewiring two keys; this provides an additional key substitution method (similar to the reflector). Notice that the
substitution affects both input and output letters. Connecting two letters using a cable, swaps the two involved letters before entry into the entry disk or before entry into the light board.

For detailed description, refer to Section 1.1.4, Reference 1 and 10.

2.9 Setting the Machine

This is an extract from Section 1.1.4, Reference 8 “How the Enigma was Setup and Operated”. Some terms have been omitted for document standardization.

The operator was provided with a codebook, which he consulted at midnight, which was when the new key for that day came into effect.

The codebook listed the five parameters for setting up the Enigma:

1. The date.
2. The rotor order for that date. For example: IV, I, V.
3. The ring setting for that date. For example: 23 02 17.
4. The plugging for that date. For example: AR KT MW LC XD EJ ZB UY PS HN.
5. The Kenngruppen, or discriminant for that date. For example: TXM.

According to this example, the operator would take Rotor IV and turn its ring until the 23 (W) position was next to a zero mark on the rotor and then clip the ring into position. He would repeat this process with Rotor I setting its ring in the 02 (B) position and with Rotor V setting its ring in the 17 (Q) position.

The operator would then put Rotor IV on the spindle in the left position, followed by Rotor I in the middle position and then Rotor V in the right position, and then slot the spindle and rotors into the Enigma and secure them in place with a lever.

Next, the operator would set the plug board by plugging A to R, then K to T, then M to W, and so on.

The operator would then think of three letters at random, say RNF, for the indicator setting. He would then manually rotate the left rotor until it had R uppermost, the middle rotor until it had N uppermost and the right rotor until it had F uppermost.

Next, the operator would think of another three letters at random, say JRM, for the message setting. He would then press the J key, and B, say, would light up, he would then press R, and K, say, would light up, next he would press M, and T, say, would light up. The operator’s assistant would make a note of the enciphered message setting (BKT in this example). The operator then set the rotors to JRM.

The Enigma is now set for enciphering or deciphering.
2.10 Example of Complete Enciphering Process

Many of the online samples omit some part of the machine in their implementation. The following example covers most part except that it omits the stepping process. The following is an abbreviated version of Section 1.1.4, Reference 1: “An Example of the Basic Enigma”.

Rotor Order: (left) I, (middle) II, (right) III
Reflector: B
Ring and Indicator Setting: A A A.
Plug Board Settings: None

The right-hand rotor R affects the substitution as such (in italic, input letter; bold output letter)

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| B | D | F | H | J | L | C | P | R | T | X | V | Z | N | Y | E | I | W | G | A | K | M | U | S | Q | O |

The middle rotor M affects the substitution as such (in italic, input letter; bold output letter)

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| A | J | D | K | S | I | R | U | X | B | L | H | W | T | M | C | Q | G | Z | N | P | Y | F | V | O | E |

The left-hand rotor L affects the substitution as such (in italic, input letter; bold output letter)

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| E | K | M | F | L | G | D | Q | V | Z | N | T | O | W | Y | H | X | U | S | P | A | I | B | R | C | J |

If we input the letter G, the following substitution takes place, (R) G -> C, (M) C -> D and (L) D -> F; where the letter in parenthesis indicates the rotor.

In Reflector B, a letter is substitute for its pair. Pairs are:

- {AY}, {BR}, {CU}, {DH}, {EQ}, {FS}, {GL}, {IP}, {JX}, {KN}, {MO}, {TZ}, {VW}

The reflector effect, F-> S

Now the path through the rotors is traverse in reverse (L -> M-> R); the substitution values are inverted (the input becomes the output and vice versa).

Left-hand rotor inverse substitution, (in italic, input letter; bold output letter)

| E | K | M | F | L | G | D | Q | V | Z | N | T | O | W | Y | H | X | U | S | P | A | I | B | R | C | J |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

Middle rotor inverse substitution, (in italic, input letter; bold output letter)

| A | J | D | K | S | I | R | U | X | B | L | H | W | T | M | C | Q | G | Z | N | P | Y | F | V | O | E |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

Right–hand rotor inverse substitution, (in italic, input letter; bold output letter)
The following substitution then takes place (L) S -> S, (M) S -> E and (R) E -> P.
RESULT. Input key = G; Output Lamp = P.

2.11 References

For a listing of references used in the completion of the Enigma Machine Project and associated documents, please see section 1.1.4, Reference Materials.

3 Requirements Matrix

Requirements Matrix

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Software Components</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GUI Component</td>
</tr>
<tr>
<td>It shall accurately encrypt and decrypt messages according to the specifications of one or more historic Enigma machines.</td>
<td></td>
</tr>
<tr>
<td>Only alphanumeric characters will be processed.</td>
<td></td>
</tr>
<tr>
<td>Incompatible input will be translated into compatible input (a-A, b-B, etc…) if possible, otherwise discarded.</td>
<td></td>
</tr>
<tr>
<td>Numbers will be</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>X</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>enciphered as Q=1, W=2, E=3, R=4, T=5, Z=6, U=7, I=8, O=9, P=0</td>
<td></td>
</tr>
<tr>
<td>Ensure that stepping and double stepping processes are correctly applied</td>
<td>X</td>
</tr>
<tr>
<td>It shall provide an intuitive graphical user interface to facilitate ease of use.</td>
<td></td>
</tr>
<tr>
<td>The GUI will provide the user drop-down boxes to select rotors, select rotors, set rotor starting values, set plug board settings, and select a reflector.</td>
<td>X</td>
</tr>
<tr>
<td>It shall ensure that the selected rotor and reflector combinations are valid. When a fourth rotor is used, only the Thin reflectors can be used.</td>
<td></td>
</tr>
<tr>
<td>It shall ensure that plugboard combinations are valid. When Enigma 1 type machine is selected, no plugboard setting should be allowed.</td>
<td></td>
</tr>
</tbody>
</table>

### 4 Software Design

#### 4.1 Introduction
4.1.1 Purpose

This software design section outlines the architectural, component, and interface design of the Enigma Machine Project. It is intended for use as reference by the collaborative software engineering team during development and maintenance of the project. It will also be used by quality assurance to verify software accuracy.

4.1.2. Scope

The Enigma Machine Project is a Java program which emulates an Enigma machine and produces accurately encrypted or decrypted results from a given input. This software can be used as an educational device, as a historical piece, or as a tool for puzzle creation and decryption in intellectual social gatherings. The user can select machine settings based on physical machine components of the original historical devices.

These components are:
1. Rotor Wheel Selections (Includes Carry Notch Positions)
2. Rotor Wheel Starting Positions
3. Ring Settings
4. Reflector Selection
5. Plug Board Combinations

See Section 2 of this document for description of historical components.

The software also offers modern methods of text entry and extraction for convenience, including a text entry input box, text file upload capability, and output text box for easy copy/paste functionality.

A second tab in the program GUI offers an Cryptanalysis section. In this section, cryptext can be entered into a text box for decryption. The text will be analysed by the software for known natural language patterns. The best possible solution will be provided based on language pattern frequency analysis. The user can increase odds of success by providing the machine with any known Enigma Machine selection inputs, or by increasing the number of keys the program tests to find the most likely solution.

4.1.3. Overview

This section will:
1. Outline the System Architecture of the project, including descriptions, diagrams, and explanation of rationale.
2. Describe the Data Design, including data structures and storage.
3. Delve into individual system components.
4. Provide an overview of the user interface, including imagery.
5. Cross-reference components and data with requirements specifications, where applicable.

4.1.4. Reference Material

For a listing of materials referenced in the completion of this project, please see
section 1.1.4 Reference Materials.

4.1.5. Definitions and Acronyms

For definitions of uncommon words and acronyms used in this document, please see section 1.1.5, Definitions and Acronyms.

4.2 System Overview

The Enigma Machine Project has two sections. The first section is a basic Enigma Machine Emulator, which allows the user to encrypt and decrypt messages as a machine operator would, using machine settings and advanced “ease of use” features. The second Section is used for automated Cryptanalysis. In this section, the user can enter a cryptext and the machine will attempt to identify the settings used to encrypt it.

The basic functions of the systems are:

Main Tab

1. Encrypting and Decrypting Messages. Because of the design of the Enigma Machine, these are both the same process.
2. Allowing the user to select one of four available reflectors and one reflector
3. Allowing the user to select rotors and place them in the desired order - up to four rotors per configuration, each placed in one of four positions: left, middle, right, fourth rotor.
4. Allowing the user to set the rotor’s rings and wheel start positions.
5. Allowing the user to customize the settings on the plug board.
6. Bulk text, single-character, or file input for text entry.
7. Advanced features which allow the user to set character spacing, choose from historical machine type GUI restrictions, and reset input text and settings.

Cryptanalysis Tab

1. Decryption of cryptexts with unknown or partially known settings using statistical language analysis.
2. Allowing the user to set any known machine settings for faster or more accurate decryption.
3. Allowing the user to select the number of threads used for cryptanalysis, or number of keys the machine will try.
4. A progress bar with abort functionality if the process its taking longer than the user wishes to wait.

This application is developed to run in an environment that supports a browser capable of running JRE 7.0+. This project uses the Agile Development Process.

4.3 System Architecture
4.3.1 Overall Architecture

4.3.1.a Description

There are four main packages in the Enigma Machine program. These are:

1. main.java.GUINew – This is the user interface component of the program. Details of user experience functionality can be found in section 4.6 Human Interface Design.
2. main.java.enigma – This is the machine emulation component of the program. It handles the basic machine encryption or decryption based on user selected machine settings.
3. main.java.cryptanalysis.nlp – This component handles statistical analysis of cryptext using a large corpus database.
4. main.java.cryptanalysis.quadbomb – This component tests combinations of consecutive four characters groups at a time from the cryptext, scores each group by determining likelihood of the characters matching English phrases. When the most likely candidate is found, it returns the keys and solution to the GUI.

Together these packages allow for message processing, application configuration and message deciphering through cryptanalysis.

The main.java.GUINew and main.java.enigma packages together replicate the operations of World War II era Enigma Machines. The GUI package is responsible for processing user rotors and reflector selection, rotor initialization setting, ring setting and plug board setting. It is also responsible for displaying the processed message to the user. The GUI communicates with the Enigma package in order to pass user configuration settings and input text.

The main.java.enigma package is responsible for configuring the software according to the parameters passed by main.java.GUINew, processing the enciphering and deciphering of messages, and passing the processed messages back to the GUI. The program allows a user to encipher and decipher a message, using the same machine configuration for both processes. Test cases are generated using already existing Enigma simulators and are compared between multiple simulators, using the rationale that it is unlikely that three or more Enigma Simulators will all produce the same "wrong" results. A user should be able to use the Enigma Machine Project software to decrypt text generated with another Enigma simulator, provided they have the correct settings to do so. The Enigma package communicates with the GUI to pass processed messages.

The main.java.cryptanalysis.nlp package is the language analysis component of the program. This package stores a corpus database which contains tables of n-grams, bigrams, trigrams, quadgrams, and full words along with their statistical frequency of appearance in literature as compiled from a large digital library. This package runs statistical analysis against entered text, attempting to match full or partial words. The NLP package is referenced by main.java.cryptanalysis.quadbomb, providing language reference and analysis to help the QuadBomb detect the most likely setting candidates for a cryptext submitted by the GUI for decryption.

The main.java.cryptanalysis.quadbomb package attempts to find the most likely setting and plaintext candidates for a given cryptext which is passed to it via a Cryptanalysis tab in the
main.java.GUINew package. It does this by testing setting combinations, comparing sections of four characters at a time against the corpus in main.java.cryptanalysis.nlp. It returns the most likely candidates to the GUI, providing the user not only the most likely solution, but also the settings used to decrypt the solution.

4.3.1.b Overall UML Diagram

The following UML diagram represents the entire architectural design of the Enigma Machine software.

The diagram below represents the Main Tab Machine flow.

4.3.1.c Flow Diagram

System High Level Functional Flow Diagram

4.3.1 EnigmaGUI Architecture
4.3.2.a Description

The EnigmaUI consist of two parts: 1) MainUI and 2) Cryptanalysis UI

The MainUI is the front-end component that allows the user to interact and set the Enigma machine emulator. Once the user selects the settings and the input message, the UI sends this information to the EM emulator. The UI is also responsible for displaying the output messages that the EM emulator returns. Details of each operation are covered in Section 5.

To accomplish this, the user is provided with the all components, and most common version of each component, and allowed to combine them as desired. The components for the following EM versions are available: 1) M3 Army, 2) M4 Naval, 3) M4 R2, 4) M4 R1 and 5) Enigma 1. In addition to selecting different components, the user is able to set the mechanical part of the components, which allows for a wide range of possible configurations. For more details on the machine components and versions, refer to Mechanical Technical Specification.

The MainUI allows the user to select settings that are required for the EM Emulator to operate. The UI allows the user to:

1. Select Machine type and appropriate rotors.
2. Place the rotor in the desired position. Acceptable positions are Right, Middle and Left. Some machine types allow for a fourth rotor.
3. Set the ring of each rotor in the desired position
4. Set the rotor indicator setting
5. Select one reflector
6. Configure the Plug Board to allow for up to 10 letter swapping pairs.
7. Enter a message or upload a text file to be processes.

A second tab in the application window gives access to the Cryptanalysis UI. The configuration is similar to the MainUI except that the user would enter an encrypted message that wants to decrypt. Through the CryptanalysisUI the user can also define a possible decryption setting (called candidate) to attempt decryption with. The final decrypted message is posted in the CryptanalysisUI output message box.

4.3.2.b UML Diagram
4.3.2.c Flow Diagram
4.3.3 Enigma Main (Machine) Architecture

4.3.3.a Description

Enigma Machine (EM) Emulator is responsible for applying the settings, processing the
message and returning the processed message back to the UI. Once a message is processed by EM emulator the UI outputs the message letter-by-letter to the screen and to the light board. The EM emulator is responsible for setting the machine according to the setting passed by the UI and processing a message. It consist of two subcomponents,

1. A Rotors class, with a subcomponent Rotor class, which simulates the operation of the rotors
2. A Plugboard class which simulates the plug board functionality.

The Plugboard class is responsible for processing the letter swapping before and after the Rotors class performs its operations.

The Rotors class simulates the operation of the rotors as a unit. It is also responsible for performing the stepping operation, handling inverse operations and representing the reflectors. The Rotor class defines the internal of each rotor, its appropriate wiring, applies the ring and indicator setting, and define the notch for the stepping process.

Once the configuration is in placed, the message is processed. Because the machine implements a substitution cipher, cipher and decipher letters map to one another. Therefore the same settings that are used to enciphering a word are also used to decipher it. See Section 4.3.3.c for a decomposition of the encryption/decryption process.

### 4.3.3.b UML Diagram
Enigma Machine
4.3.2.c Flow Diagram

Module 2 (EnigmaMachine) Functional Flow Diagram – Component Level

Encryption/Decryption Process

4.3.3 Enigma Cryptanalysis Architecture

4.3.3.a Description

The Enigma Cryptanalysis (EC) component performs code-breaking functionality. Three majors activities performed by the EC are, 1) selection of possible machine configurations, also refer to as selection of candidates, 2) message decryption, and 3) decrypted message optimization to increase legibility.

Possible machine configurations, or candidates, are selected using a quadgram statistics algorithm. Quadgram are groups of four-letters that are part of a word. For example, acceptable quadgram for the word 'ATTACK' would be, 'ATTA', 'TTAC', and 'TACK'. Quadgram statistics is a fitness measure to determine the likelihood that a candidate can correctly decrypt a message. To do this, a portion of the text is deciphered using a combination of settings. The process text is then compared to database of known valid English quadgrams. Quality of the candidate is calculated based on how closely the output matches known quadgrams. This process is repeated with different combinations of rotors and reflector types, indicator and ring.
settings, and plugboard pairing until a good candidate is found. The user can enter a partial setting combination for testing. The range of settings tested is restricted by the user’s partial combination setting, if any.

The message is then processed through the EnigmaMachine Emulator using the machine configuration settings produced by the quadgram statistic process. This process works exactly as the decryption/encryption process of any message.

The output from the EM Emulator is then analyzed to identify sets of words and add appropriate spacing. This process increases the legibility of output message.

### 4.3.3.b UML Diagram

#### 4.3.3.b.1. Enigma Cryptanalysis QuadBomb

#### 4.3.3.b.2. Enigma Cryptanalysis NLP
Enigma Machine
4.3.3.c Flow Diagram

Module 3 (CryptoAnalysis) Functional Flow Diagram – Component Level

- From Cryptanalysis UI
- Set and Initialize Values
  1. User define setting candidates
  2. Message
- Determine Rotor and Reflector Type and Settings candidates
- Trim candidate list
- Determine Ring Setting candidates
- Trim candidate list
- Determine Plugboard candidates
- Trim candidate list
- Use candidate list and EnigmaMachine to decipher message
- For accuracy, verify output against database
- Output decrypted message to Cryptanalysis GUI

4.4 Data Design
4.4.1 Data Description

User entered data in the Enigma Machine Main tab is not stored indefinitely. It is stored only long enough to process a single encryption, with the exception of rotor position settings which can be stored locally and used for multiple encryptions. Text input is first processed to strip characters that the emulator can not process, and is then passed to the Enigma Machine emulator for encryption. Machine settings are passed along with the text data as encryption instructions. The encrypted text is returned along with settings information which updates the rotor positions and light board.

Likewise, user entered text and settings information in the Cryptanalysis tab is stored for single use. However, a log is created during cryptanalysis which stores the original input text, attempted and selected keys, and the chosen output.

Additionally, two major data entities are used for the functionality of the application, 1) the machine configuration, and 2) the corpus database.

The machine configuration consist of multiple data elements that are used to define the settings for the EM Emulator. These data elements are defined in the EnigmaMachine class constructor. The text decryption/encryption and cryptanalysis processes rely on this data entity for correct messages processing.

A large corpus database file is used as reference for frequency analysis. This file contains a frequency-based listing of character groupings from single characters to complete words. A large online library was used as reference material for the frequency analysis. This can be found in Section 1.1.4, Reference 15.

4.4.2 Data Dictionary

1. MachineConfiguration – This data structure is used by all three classes. It is used to store the user defined machine configuration settings to be passed to the EnigmaMachine class. Its members are:
   a. selectedRotor – Integer array that stores user selected rotor. Rotor position is defined as follow: left at index 0, middle = index 1, right = index 2 and fourth = index 3
   b. selectedReflector – Integer value that stores the user selected reflector
   c. ringSetting. Character array that the ring position of each rotor following the indexing describe in a.
   d. indicatorSettings – Character array that the indicator position of each rotor following the indexing describe in a.
   e. plugboardSetting – Two-dimensional character array that stores the user defined plugboard pairs.

2. Rotor – This data structure is used by the EnigmaMachine class. It provides a mechanism to capture the main characteristics of an EM rotor. Its members are:
   a. forwardWiring – A character array that depicts the substitution scheme of the
Enigma rotor.

b. reverseWiring – A character array that depicts the inverse of the substitution scheme of the Enigma rotor.

c. stepOffset – An integer type variable that stores the indicator setting position for that rotor.

d. ringSetting – An integer type variable that store the ring setting position for that rotor.

e. notchPosition – A character array that depicts the value of the notch for step up to take place.

f. size – An integer value that store the length of the input message.

The following functions are included in the Rotor class

g. Rotor – Class constructor

h. getNotchPosition() – Returns the values of the variable notchPosition.

i. getPosition() – Returns the character value of the current rotor position.

j. setRingPostion() – Takes a character input and modify the values in ringSetting

k. setStartPosition() – Takes a character array input and modify the values in stepOffset.

l. cycleRotor() – Return a boolean variable to indicate if rotor stepping should be applied

m. forwardEncrypt() – Takes a character type value and returns it’s character type substitution value.

n. reverseEncrypt() – Takes a character type value and returns it’s inverse character type substitution value.

3. Plugboard – This data structure is used by the EnigmaMachine class. It is used to define the plugboard settings. Its members are:
   a. plugboard – Two-dimensional character array that store the user defines plugboard substitution pairs.

   b. numPairs – Integer type variable internally used by the Plugboard class to define the length of the two-dimension array plugboard.

The following functions are defined in the Plugboard class

a. Plugboard() – Class constructor takes a String input.

b. Plugboard() – Class constructor that takes a two-dimensional character array.

c. matchChar() – Takes a character type variable and return a character type values of its substitution.

4. Corpus - This class provides the database construction to character grouping and whole words. The database is used during the cryptanalysis process to test the efficiency of candidates and for message optimization process after a message has being analyze. Its
members are,

a. unigramTable - Map container of String and Integer pairs that stores a collection of unigram values.
b. bigramTable - Map container of String and Integer pairs that stores a collection of bigram values.
c. trigramTable - Map container of String and Integer pairs that stores a collection of trigram values.
d. quadgramTable - Map container of String and Integer pairs that stores a collection of quadgram values.
e. wordTable - Map container of String and Integer pairs that stores a collection of words.
f. unigramCount - integer value of the element count in unigramTable.
g. bigramCount - integer value of the element count in bigramTable.
h. trigramCount - integer value of the element count in trigramTable.
i. quadgramCount - integer value of the element count in quadgramTable.
j. wordCount - integer value of the element count in wordTable.

The following functions are defined in the Corpus class

a. Corpus() - Class constructor.
b. addBigram() - Takes a String input and add it to bigramTable.
c. addQuadgram() - Takes a String input and add it to quadgramTable.
d. addTrigram() - Takes a String input and add it to trigramTable.
e. addUnigram() - Takes a String input and add it to unigramTable.
f. addWord() - Takes a String input and add it to wordTable.
g. getBigramCount() - Takes a String input and returns the frequency count for specified bigram.
h. getQuadgramCount() - Takes a String input and returns the frequency count for specified quadgram.
i. getTrigramCount() - Takes a String input and returns the frequency count for specified trigram.
j. getUnigramCount() - Takes a String input and returns the frequency count for specified unigram.
k. getWordCount() - Takes a String input and returns the frequency count for specified word.
l. getBigramTestQueue() - return a PriorityQueue of the values in the bigramTable sorted in descending order by frequency count.
m. getQuadgramTestQueue() - return a PriorityQueue of the values in the
quadgramTable sorted in descending order by frequency count.

n. getTrigramTestQueue() - return a PriorityQueue of the values in the trigramTable sorted in descending order by frequency count.

o. getUnigramTestQueue() - return a PriorityQueue of the values in the unigramTable sorted in descending order by frequency count.

p. getWordTestQueue() - return a PriorityQueue of the values in the wordTable sorted in descending order by frequency count.

q. getTotalBigramCount() - return the value in the bigramCount.

r. getTotalQuadgramCount() - return the value in the quadgramCount.

s. getTotalTrigramCount() - return the value in the trigramCount.

t. getTotalUnigramCount() - return the value in the unigramCount.

u. getTotalWordCount() - return the value in the wordCount.

v. trimCorpus() - removes low frequency occurring items from the each table.

4.5 Component Design

This subsection will step through the components of each of the four main packages in the Enigma Machine Program. This information is also reflected in the program Java Docs packaged with the source code. Please see the Java Docs for a more detailed breakdown of all components.

4.5.1 main.java.GUINew

1. CaGuiPrototype – User Interface for Cryptanalysis components. Contains all cryptanalysis fields and options. The user can enter the message in the input field and set specific constraints if part of the key is known (for example, some or all of the rotors are known, but not the reflector). After processing is complete, the GUI displays the QuadBomb results and what it believes to be the most likely combination of settings and decryption. A four-rotor decryption attempt using no constraints can potentially take a considerable amount of time. The user is warned about this in the instructions, and an abort option is provided.

2. ComponentLinker – Draws the lines that illustrate which letters are linked together in the plugboard.

3. ConfigureOutput – Class to format the input for processing by the Enigma Machine. Automatically replaces numbers with their “code” letters, removes all other non-letter characters, and converts everything to upper case.

4. EnigmaApplet – The applet driver for the Enigma GUI, for running it as an applet instead of as an app.

5. EnigmaGUI – Driver for the EnigmaGUI, so that it can be used as an app in addition to run as an applet.

6. EnigmaSingleton – Uses the Singleton and Observer design patterns to enable data sharing across the various modular GUI pieces. Maintains state consistency for the instance of EnigmaMachine and provides conveniency methods for manipulating it and
performing encryption. Is thread safe. The rotor options are represented as an integer numbered 0-9. The rotors are mapped as follows: 0 - Rotor I; 1 - Rotor II; 2 - Rotor III; 3 - Rotor IV; 4 - Rotor V; 5 - Rotor VI; 6 - Rotor VII; 7 - Rotor VIII; 9 - Rotor Beta; 10 - Rotor Gamma

Reflector options are also represented as an integer. The options are numbered 0-3, and are mapped as follows: 0 - Reflector B; 1 - Reflector C; 2 - Reflector B thin; 3 - Reflector C thin

Ring and rotor settings are represented using characters. Plugboards are represented through a string that indicates their replacement mapping. Letters are swapped with their adjacent letters. For example, a string of "ABCD" swaps A's with B's (and vice-versa) and C's with D's. The updateType is an integer indicating what type of update or reset is to be performed. 0 means that only the rotor indicators are to be updated. The rotor positions change with every character encryption. 1 means that the GUI is to be fully reset to its default position. 2 means that only the text boxes are to be cleared. The spacesOption is an integer used to determine the spacing in the output. The default setting is no spaces, indicated by a 0. 1 indicates "words" of four letters, 2 indicates "words" of five letters, and 3 indicates that the original spacing should be maintained. This information is maintained here because the original spacing processing must be performed during the input processing. The machineTypes is an integer from 0 - 5 indicating the available machine type options. The machine type are mapped as follows: 0 - No Restrictions, 1 - Enigma I; 2 - Enigma M3 Army; 3 - Enigma M4 Naval; 4 - Enigma M4 R1; 5 - Enigma M4 R2;

7. IOPanel – Combines all of the IO functionality of the Enigma, providing ways to encrypt characters, long strings, and entire files. When encrypting characters it provides a graphical representation of the "lightboard", lighting up the encrypted character, reminiscent of how the original machines operated.

8. Lightboard – Provides a visual "Light Board" display of the last output character following encryption. This simulates the "Light Board" output of the historical Enigma Machines.

9. PlugboardDialog – PlugBoard dialog window pop-up for character pair selection. Allows the user to create the plugboard map to be used by the EnigmaMachine for encryption.

10. PrimaryGUIPanel – The primary panel for the EnigmaGUI. The Enigma GUI interface allows users to encrypt and decrypt messages using a simulation of the Enigma devices used by the Germans during World War II.

11. ResetPanel – Encapsulates the functionality to reset the machine state in one of three different ways. Also includes the functionality for configuring the text output. ResetPanel.java

12. ResultsPanel – Panel which displays the results of a Cryptanalysis attempt.

13. RotorPanel – Top panel of the modular GUI for the Enigma Machine. Allows setting of all Enigma settings, and helps propagate those settings to other classes that need them through the use of the Enigma Singleton.

4.5.2 main.java.enigma

1. EnigmaMachine – The complete "guts" of the Enigma Machine, a simulation of the encryption device used by the Germans during World War II. Includes all rotors, reflector, and plugboard. Allows for the encryption of a single character or a string. The rotor options are represented as an integer numbered 0-9. The rotors are mapped as follows: 0 - Rotor I; 1 - Rotor II; 2 - Rotor III; 3 - Rotor IV; 4 - Rotor V; 5 - Rotor VI; 6 - Rotor VII; 7 - Rotor VIII; 9 - Rotor Beta; 10 - Rotor Gamma

Reflector options are also represented as an integer. The options are numbered 0-3, and are mapped as follows: 0 - Reflector B; 1 - Reflector C; 2 - Reflector B thin; 3 - Reflector C thin
an integer. The options are numbered 0-3, and are mapped as follows: 0 - Reflector B; 1 - Reflector C; 2 - Reflector B thin; 3 - Reflector C thin

Ring and rotor settings are represented using characters. Plugboards are represented through a string that indicates their replacement mapping. Letters are swapped with their adjacent letters. For example, a string of "ABCD" swaps A's with B's (and vice-versa) and C's with D's.

2. **EnigmaSettings** – EnigmaSettings.java Wrapper class to store an Enigma machine's full settings. Used to capture a candidate state for cryptanalysis testing, or other points when it is useful for passing all Enigma Machine settings at once. The rotor options are represented as an integer numbered 0-9. The rotors are mapped as follows: 0 - Rotor I; 1 - Rotor II; 2 - Rotor III; 3 - Rotor IV; 4 - Rotor V; 5 - Rotor VI; 6 - Rotor VII; 7 - Rotor VIII; 9 - Rotor Beta; 10 - Rotor Gamma

Reflector options are also represented as an integer. The options are numbered 0-3, and are mapped as follows: 0 - Reflector B; 1 - Reflector C; 2 - Reflector B thin; 3 - Reflector C thin

Ring and rotor settings are represented using characters. Plugboards are represented through a string that indicates their replacement mapping. Letters are swapped with their adjacent letters. For example, a string of "ABCD" swaps A's with B's (and vice-versa) and C's with D's.

3. **Plugboard** – This class simulates the plugboard in the Enigma encryption machine. The plugboard was a simple two-way replacement cipher that was available on some Enigmas. The substitutions were performed both during initial letter input (before processing by rotors and reflectors) and output (after processing). Plugboards are represented through a string that indicates their replacement mapping. Letters are swapped with their adjacent letters. For example, a string of "ABCD" swaps A's with B's (and vice-versa) and C's with D's.

4. **Rotor** – This class simulates a single rotor of the Enigma encryption machine. It stores both the forward and reverse wiring to allow for O(1) encryption and decryption. It also stores the Ring offset (in the physical machine, this was adjusted before the rotors were placed in the Enigma) and the stepOffset, which indicates the current position of the rotor. It also stores the positions of the carry notch(es), which impact when the rotor to the left of it will rotate.

5. **Rotors** – Contains 3-4 instances of the Rotor class. Also acts as the reflector in an Enigma machine. Fully encrypts a character (except for Plugboard Substitutions) and handles the proper cycling of the individual Rotor instances. The rotor options are represented as an integer numbered 0-9. The rotors are mapped as follows: 0 - Rotor I; 1 - Rotor II; 2 - Rotor III; 3 - Rotor IV; 4 - Rotor V; 5 - Rotor VI; 6 - Rotor VII; 7 - Rotor VIII; 9 - Rotor Beta; 10 - Rotor Gamma

Reflector options are also represented as an integer. The options are numbered 0-3, and are mapped as follows: 0 - Reflector B; 1 - Reflector C; 2 - Reflector B thin; 3 - Reflector C thin

Ring and rotor settings are represented using characters.

4.5.3 main.java.cryptanalysis.nlp

1. **Corpus** – Implements a set of tables containing character unigrams, bigrams, trigrams, quadgrams, and whole words, with frequency counts of each. After loading all the desired grams, it is recommended that trimCorpus() is called to remove very low frequency count occurrences. N-grams and words are added by calling the appropriate add(ngram)
method. Retrieving a frequency count of a n-gram or word is done by `get(ngram)Count()`. Retrieving the total count of a n-gram is done by `getTotal(ngram)Count()`. Convenience methods to get sorted priority queues of grams is via `get(ngram)TestQueue()`. This class is not thread safe if retrieving frequency counts and/or ngram queues while adding ngrams. If not adding words, then concurrent calls to get frequency counts and ngram queues is safe.

2. **CribDetector** – Analyzes a decrypted message and attempts to identify words and add spaces. Calling `parseMessage` recursively splits the message by words in the corpus, starting with the most common word (“THE” in most cases). For each piece of the message, a word probability and a count of unidentified characters is done, and compared for best fit. Once a list of word matches is found, spaces are added, starting with the lowest probability word. This class is not thread safe, as adding items to the corpus at the same time as accessing state values can lead to inconsistent state. However, this class is designed to be called by a single thread.

3. **StatisticsGenerator** – This class computes a variety of statistics, provided at the below references: Each generator can be initialized to use a particular statistic. List of available statistics:
   - 0 - Unigram character probability. (Sinkov's statistic)
   - 1 - Bigram character probability. (Sinkov's statistic)
   - 2 - Trigram character probability. (Sinkov's statistic)
   - 3 - Quadgram character probability. (Sinkov's statistic)
   - 4 - Unigram character probability. (Index of Coincidence)
   - 5 - Bigram character probability. (Index of Coincidence)
   - 6 - Trigram character probability. (Index of Coincidence)
   - 7 - Quadgram character probability. (Index of Coincidence)
   - 8 - Unigram character probability. (Chi-Squared Statistic)
   - 9 - Bigram character probability. (Chi-Squared Statistic)

4. **TextParser** – This parser takes a database reference and scans a text file (UTF-8 is proven to work, other formats are unknown) for character grams and whole words. Nonalphanumeric characters are skipped for character grams, and specific rules are applied for parsing words.

**4.5.4 main.java.cryptanalysis.quadbomb**

1. **IndicatorDetector** – Worker thread to determine best indicator settings for a given rotor / reflector configuration. This thread performs an exhaustive search of all possible indicator settings within defined constraints. Each candidate indicator setting is applied to the provided machine settings, and scored with a defined statistic. The best number of candidates (number specified by the user) is saved in `resultsList` for further use. Return values indicate if the thread completed its computation. `IndicatorDetector.java`

2. **PlugboardDetector** – Worker thread to determine plugboard settings. Each combination of possible plugboard settings are tested and scored. If any combinations score better than the pre-tested message, then the best combination is saved and then applied in the next iteration of this check. The process iterates until no improvement in score is found during an iteration. The best result is returned.

3. **QuadbombManager** – This manages the thread work list and process flow of QuadBomb. QuadBomb attempts to decrypt an Enigma message using quadgram statistics algorithm as described at the linked websites: Step 1: Determine best rotor wheel order and indicator settings, saving each consecutive best result. Step 2: Determine best ring setting by cycling through rotor combinations from the top number
of candidates saved in step 1. Step 3: Determine plugboard connections a pair at a time, saving best results of each pair combo, until no further improvement can be found.

Improvements: Incorporated multiple threading to improve CPU utilization. Limitations: This method does not guarantee a correct result. Essentially, this algorithm is equivalent to a ensemble local maxima search in that it first searches the wheel order and indicator settings and saves a set number of best matches. Once that search is exhausted, then it searches for the best ring setting using the list of best rotor settings previously constructed. This search is limited in that the ring search is restricted to the candidate rotor settings. It is possible that the correct result is a combination of suboptimal rotor and ring settings, and in these cases the algorithm is expected to fail. Furthermore, in cases of messages encrypted with a large number of plugboard pairs, the search space scores are very similar until most of the correct settings are recovered. In these cases, it is very likely that the correct settings are lost in a given step due to a large number of slightly better scoring 'incorrect' setting combinations clogging the candidate lists.

4. QuadBombSettings – QuadBombSettings.java convenience class to control how QuadBomb will test candidates. Array definitions rotorSettings / ringSettings / indicatorSettings: 0 - fourthRotor 1 - leftRotor 2 - middleRotor 3 - rightRotor Flag values: -2 or '!' - Do not test this item. -1 or '?' - Test all values. Any positive value or character - test that value only.

5. RingDetector – Worker thread to determine best ring settings for given rotor / reflector / indicator configurations. This thread performs an exhaustive search of all possible ring settings within defined constraints. As rings are tested, the indicators are offset to keep alignment. Because of this, if a full test is done, the leftmost ring does not need to be stepped. Each candidate ring setting is applied to the provided machine settings, and scored with a defined statistic. The best number of candidates (number specified by the user) is saved in resultsList for further use. Return values indicate if the thread completed it's computation. RingDetector.java

4.6 Human Interface Design

4.6.1 Overview of User Interface

The User Interface consists of two component, 1) Main component which provides Standard Enigma Machine component interface and 2) Cryptanalysis with provides cryptanalysis component interface.

4.6.1.a EnigmaMachine User Interface
The Main (Machine) Tab User Interface consists of five different sets of machine settings, three advanced feature settings, three input options, and two output options. These features are as follows:

1. Four rotor selection drop-down menus. Each drop-down allows the user to select a different rotor wheel out of a possible ten wheels, each representing a different substitution permutation.

   1.1. The leftmost rotor (fourth rotor) is optional, as it is only available in specific Enigma Machine versions.

   1.2. When a rotor is selected in one drop-down, it can not be selected in a different drop-down.
1.3. The machine will default to a three-rotor setting using Rotors I, II, and III when it is first started. Users can change these settings as desired.

1.4. It is not possible to encrypt or decrypt a message unless the three rightmost rotors are selected.

2. One reflector selection drop-down menu. This menu allows the user to select between four possible reflectors, each representing a different substitution permutation.

2.1. The final two reflector settings, B Dünn (Thin) and C Dünn (Thin), should be used if the fourth rotor is selected.

2.2. The first two reflector settings, B and C, should be used if the fourth rotor is not selected.

2.3. The machine will default to use Reflector B when it is first started. Users can change these settings as desired.

3. Four ring setting drop-down menus. Each drop-down allows the user to select a letter representing a setting between A and Z. This shifts the transmitted output from the rotor.

3.1. The leftmost ring setting (fourth drop-down) is optional, and will not be set if the user does not set the leftmost rotor.

3.2. The machine will default to use only the three rightmost settings, set to A A A, when it is first started. Users can change these settings as desired.

4. Four rotor window spinners. Each spinner displays a letter between A and Z, which indicates the current rotor position.

4.1. The machine will default to use only the three rightmost settings, set to A A A, when it is first started. Users can change these settings as desired.

4.2. The rotor positions displayed in the windows will change as the machine processes text.

4.3. The rightmost rotor will change once every time a button is pressed.

4.4. The second rotor will change once every 26 rotations of the rotor to the right, plus an additional turn when the third rotor changes.

4.5. The third rotor will change once every 26 rotations of the rotor to the right.

4.6. The fourth rotor, when selected, will not change.

5. A plug board settings button which launches a plug board selection window.

5.1. The menu consists of twenty-six radial buttons representing letters A-Z which can be selected in pairs. Selecting two buttons one after another will link them and draw a line between them.

5.2. After pressing the button to exit this window, the selected pairs will be added to a list for processing in link order. This list will be displayed on the Main GUI tab.

5.3. The machine will default to settings where no letters are swapped, resulting in no swapped letters.

5.4. The plug board and list can be cleared with the “Reset Plugboard” button.

6. Twenty-six light board image icons. When a user types a single character into the text box at
a time, the Light Board character representing the proper output will change color as if lighting up.

7. One bulk input text box. A user can type a text message directly into this box in order to encrypt or decrypt the message.
   7.1. Special characters will be ignored.
   7.2. Numbers will be converted directly into representative letters, which will be encrypted as those letters would be. Those letters are: 0 = P, 1 = Q, 2 = W, 3 = E, 4 = R, 5 = T, 6 = Z, 7 = U, 8 = I, 9 = O.

8. One single-character input text box. A user can type a message in this box one character at a time.
   8.1. Lighboard, rotor position spinners, and single-character output box will update after every character input.

9. One bulk output text box. The output after encryption or decryption will be displayed in this box.
   9.1. A user can copy the result from this box using Ctrl+C.

10. One single-character output text box. The results of single-character encryption will display here immediately after a character is entered into the single-character input text box.

11. One browse button. Pressing this button will bring up a File Chooser which will allow the user to pick a local txt file for encryption/decryption.
    11.1. The selected file will display in a text field to the right of the browse button.
    11.2. Only txt files can be selected.

12. One encrypt button. Pressing this button will submit the contents of the selected txt file or bulk input text box for encryption/decryption.
    12.1. The results will be output to the output text box.
    12.2. This button will first check for a file in the file input box. If no file is chosen, it will check for text in the bulk input text box.
    12.3. The lightboard and rotor wheel positions will be updated after every press of the encrypt button.

13. One output space options drop-down menu. This setting offers the user four options for spacing in the output text. These options are no spaces, four spaces, five spaces, and original spaces.
    13.1. Use of the final option is discouraged in the User’s Guide, as it increases likelihood of decryption.

14. Three reset options buttons. These buttons offer the user a choice of three methods for resetting machine settings. The user can revert the entire program to default configurations, reset only the indicators to the positions they were in before encryption, or clear only the text from the text boxes.

15. One machine type drop-down. This offers the user a choice of historical machine types. Selecting one of these options will restrict the GUI to only those features which were
available on the selected version of the Enigma Machine.

15.1. The options available are “No Restrictions”, Enigma I, Enigma 3 Army, Enigma 4 Naval, Enigma M4 R1, and Enigma M4 R2. See section 2.3 for machine version settings.

16. Minimize, maximize, and close buttons. These will be at the upper right of the program, and will Minimize the window to the system tray, Maximize the window to full screen, or close the program respectively.

4.6.1.b Cryptanalysis User Interface

The Cryptanalysis Tab User Interface consists of five different sets of machine settings, two advanced feature settings, one input option, and abort option, and six output options. These features are as follows:

1. One input text box. The user can enter a cryptext into the box for decryption. Only alphanumeric and whitespace characters are allowed. Any other characters will prompt an error dialog.

2. Four rotor selection drop-down menus. If the user knows any of the rotor settings used to create the cryptext, these can be entered for more accurate and faster decryption.
   a. If it is known that the message was encrypted with a three-rotor machine, the fourth-rotor can be set to 'none' to skip four-rotor checks.

3. One reflector selection drop-down menu. If the user knows the reflector setting used to create the cryptext, it can be entered for more accurate and faster decryption.
4. Four ring setting drop-down menus. If the user knows any of the ring settings used to create the cryptext, these can be entered for more accurate and faster decryption.
5. Four indicator selection drop-down menus. If the user knows any of the rotor position settings used to create the cryptext, these can be entered for more accurate and faster decryption.
6. One plugboard setting input text box. If the user knows any of the plugboard combinations used to create the cryptext, these can be entered for more accurate and faster decryption. Only alphanumeric characters and whitespace characters are permitted. If other characters are found, or the listed number of pairs is odd, an error message will prompt.
   a. Settings must be entered in pairs. EG: {AX YM}
7. One thread limit spinner. The user can set the number of threads used in decryption for faster processing.
8. One candidate size spinner. The user can adjust this value to analyse additional potential keys for more accurate results.
9. One decrypt button. This button begins the decryption process using the user settings and input text.
10. One abort button. If the decryption is taking too long, the user can abort the process and adjust settings.
11. One progress bar. Because the decryption process can take a very long time with a long input or higher candidate number, a progress bar is necessary to inform the user of the program’s status. Otherwise, it would appear as if the machine were frozen or not functional.
12. Status text box. A list of the number of completed operations is shown against the total number of operations.
13. One output text box. The machine’s best guess solution will be displayed here when the process is complete.
14. Four rotor output text boxes. The program’s best guess of rotor selections used to encrypt the input text are displayed here.
15. One reflector output text box. The program’s best guess of reflector selection used to encrypt the input text are displayed here.
16. Four ring output text boxes. The program’s best guess of ring selections used to encrypt the input text are displayed here.
17. Four indicator output text boxes. The program’s best guess of rotor position selections used to encrypt the input text are displayed here.
18. One plugboard output text box. The program’s best guess of plugboard pairs used to encrypt the input text are displayed here.

### 4.6.2 Screen Images

The image below is a screen capture of the User Interface with all possible setting options.
4.6.3 Screen Objects and Actions
Screen navigation is executed through drop-down menus, spinner menus, direct text input into spinners and text boxes, File Chooser pop-up navigation, and button presses for text file decryption. Navigation options require both a mouse and keyboard.

### 4.7 Appendices

Appendix 1: High Level Message Passing
Appendix 2: Stepping Process

#### 4.7.1 Appendix 1: High Level Message Passing
4.7.2 Appendix 2: Stepping Process

Module 2 (EnigmaMachine) Functional Flow Diagram
Subcomponent Level: Rotors class

Stepping Process in Detail

Rotors:
R – Right
M – Middle
L – Left

From Encryption Process

Is the rotor R in stepping position?

Update EnigmaUI rotor M

Step M rotor

Is the rotor M in stepping position?

Update EnigmaUI rotor M and L

Step M and Step L

Return to Encryption Process
5 User’s Guide

Note: Many sections in the User’s guide contain information found elsewhere in the Project Document. This is deliberate, as this guide is intended to be a stand-alone document provided to users on the project website.

5.1 Introduction

Enigma Machines, a series of electro-mechanical machines used for encrypting and decrypting messages, are most famous for their use by Nazi Germany in World War II. Some believe the successful decryption of the ciphers created by these machines contributed significantly to the victory of the Allies. Following their invention in the 1920’s by Arthur Scherbius, the machines increased in complexity by version as the German Military attempted to thwart cryptanalysis. Decryption was achieved initially through the reverse engineering efforts of the Polish Cipher Bureau, and by exploiting mistakes made by German operators. In 1939, the Poles shared their techniques and equipment (Cryptographic Bomba, or Bombe) with the French and British. British intelligence at Bletchley Park were able to build more sophisticated Bombe devices to combat the increasing complexity of the Enigma machines.

This program is intended for use by students, historians, and curious intellectuals. It is designed to emulate the basic cryptographic functionality of the original Enigma Machines. It is not an exact replica of an Enigma Machine, but instead mimics the functionality with a modern GUI and added features for ease of use. Examples of such features include the ability to select from multiple Enigma versions, select from multiple rotors, select from multiple reflectors, enter blocks of text, enter single characters, or enter .txt files for bulk encryption or decryption. In this way, the single emulator can be used to encrypt documents for historical recreations, lessons, or puzzle challenges.

The program has a second tab which can be selected if the user wishes to attempt decryption of an existing Enigma Machine cryptext. Due to the nature of encryption, successful decryption is not guaranteed and the process may take several minutes to complete. This section is not a part of the original machine. It is to be used for user experimentation with cryptanalysis.

To operate this program from the web, the user must have a web browser running Java JRE 7.0 or above. To acquire a copy of the latest Java version, visit http://java.com/. A standalone executable Jar file will also be available for download, but still requires the use of Java JRE 7.0.

5.2 Information for Use of This Guide

This guide is created for customer assistance. It provides an overview of features, program use, and error resolution. The sections are broken down into the following:

1. Concept of Operations – An overview of software process and workflow
2. Procedures – Software access/launching, navigation, and functions
3. Information on Software Commands – Data entry and Import/Export instructions
4. Error Messaging and Problem Resolution – For use in troubleshooting problems
5.3 Concept of Operations

This software is designed for access via a web browser, although a stand-alone application may be downloaded. Functionality is primarily GUI based, using drop-down selection boxes or direct text input. A user is meant to select three to four rotor options, ring settings, rotor starting positions, and a reflector type from drop-down options. Plug-board entries (pairs of alphabetic characters for a substitution swap) should be selected by clicking the “Plugboard Settings” button and selecting the bubbles in pairs. The user will then enter the plaintext or ciphertext message manually into the Manual Input section or via copy-paste functionality into the bulk input section. Manually inputed text will be encrypted as the user types, but the bulk input section requires the use of the “Encrypt” button. The encrypted or decrypted output text will then be displayed in an output text field. Alternatively, the user can select “upload file” to pull the input text from a file.

5.4 Procedures

1. Software Access/Launching – The web software can be accessed by visiting http://www.enigmamachine.info or http://www.enigmamachine.org in a web browser. An executable Jar file will also be available which does not rely on a browser, but does require Java version 7.0.
2. Software Navigation and functions – The following walkthrough defines the GUI options, their usage, and results.

5.4.1 GUI Layout

The figure below shows the GUI interface. The function of each control is described in the paragraphs that follow.
5.4.2 Main Tab: Top Right: Component Settings
Three to Four Rotor Options

In the Enigma machine, rotors are hard-coded rotating wheels which each represent a specific permutation of Substitution Ciphers that can be performed upon each character of the text input. There are a total of ten possible rotors. Users can select up to four to use, but can only use the same rotor one time per message. Each box in the GUI contains a drop-down selection of available rotors, with the fourth box containing a blank option for the three-rotor version of the Enigma Machine. As each character is processed, it will be substituted with the corresponding character from each of the three to four wheels twice. Once against the initial permutation, and once against its inverse.

Below is the substitution table used by each rotor:

| ROT I | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| E     | K | M | F | L | G | D | Q | V | Z | N | T | O | W | Y | H | X | U | S | P | A | B | R | C | J |
| ROT II| A | J | D | K | S | I | R | U | X | B | L | H | W | T | M | C | Q | G | Z | N | P | Y | F | V | O | E |
| ROT III| B | D | F | H | J | L | C | P | R | T | X | V | Z | N | Y | E | I | W | G | A | K | M | U | S | Q | O |
| ROT IV| E | S | O | V | P | Z | J | A | Y | Q | U | I | R | H | X | L | N | F | T | G | K | D | C | M | W | B |
| ROT V | V | Z | B | R | G | I | T | Y | U | P | S | D | N | H | L | X | A | W | M | J | Q | O | F | E | C | K |
| ROT VI| J | P | G | V | O | U | M | F | Y | Q | B | E | N | H | Z | R | D | K | A | S | X | L | I | C | T | W |
| ROT VII| N | Z | J | H | G | R | C | X | M | Y | S | W | B | O | U | F | A | I | V | L | P | E | K | Q | D | T |
| Beta | L | E | Y | J | V | C | N | I | X | W | P | B | Q | M | D | R | T | A | K | Z | G | F | U | H | O | S |
| Gamma| F | S | O | K | A | N | U | E | R | H | M | B | T | I | Y | C | W | L | Q | P | Z | X | V | G | J | D |

60 | Enigma Machine
Three to Four Ring Setting Options

Each of the four Ring Setting GUI options contains a drop-down from which a user can select the initial shift position of the Stepping Mechanism. As each character is processed, the rightmost rotor will rotate/shift one place, changing the substitution. Each rotor contains a specific character which, when it comes up during encryption/decryption, will cause the rotor to its left to shift. The ring setting will shift the initial position of the “carry notch” character, making the encrypted message harder to decipher. In four rotor Enigma, the leftmost rotor does not rotate.

Initial Carry Notch Positions:

<table>
<thead>
<tr>
<th>Rotor</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotor I</td>
<td>at R</td>
</tr>
<tr>
<td>Rotor II</td>
<td>at F</td>
</tr>
<tr>
<td>Rotor III</td>
<td>at W</td>
</tr>
<tr>
<td>Rotor IV</td>
<td>at K</td>
</tr>
<tr>
<td>Rotor V</td>
<td>at A</td>
</tr>
<tr>
<td>Rotors VI, VII and VIII</td>
<td>at A and at N</td>
</tr>
</tbody>
</table>

Three to Four Rotor Starting Position Options

The user is able to set the initial starting position of each of the four rotors. This is displayed as spinners in the GUI. As the machine operates, these windows will show the current position of each rotor following each “Encrypt” button press or entered character.

Reflector

The reflector drop-down gives the user one of four options for a one-time substitution which occurs at a point between encryption of a character through the three to four rotors, and encryption through the inverse permutations of the same rotors (The Reflection).

Reflector Options:

<table>
<thead>
<tr>
<th>Reflector</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>reflector B</td>
<td>(AY) (BR) (CU) (DH) (EQ) (FS) (GL) (IP) (JX) (KN) (MO) (TZ) (VW)</td>
</tr>
<tr>
<td>reflector C</td>
<td>(AF) (BV) (CP) (DJ) (EI) (GO) (HY) (KR) (LZ) (MX) (NW) (TQ) (SU)</td>
</tr>
</tbody>
</table>
5.4.3 Main Tab: Center Right: Plugboard Settings

Plugboard Wiring

When the Plugboard Settings button is selected, a popup window appears allowing the user to input their plugboard pairs.

A pair is chosen by clicking the letters in “swap order.” For example, in order to swap “O” with “K”, the user would select “O” and then “K”. A line appears on the plugboard to indicate that the letters have been paired. To return to the main GUI, click the “X” in the bottom right of the Plugboard.

Reset Plugboard

When selected, “Reset Plugboard” clears all Plugboard settings. Note: “Reset Plugboard” is the only means available to undo a Plugboard selection.

5.4.4 Main Tab: Bottom Right: Text Input and Output

Input Text: Manual Input

The GUI has a text field into which the user can hand-type a message to encrypt or decrypt. This field has a 500 character limit. As the user types, the encrypted text will appear in the “Encrypted Output” field, the rotor positions will advance, and the corresponding output letter
The larger text field on the left is for bulk text encryption. To use it, the user can paste the text to be encrypted (or decrypted) into the field and select the “Encrypt” button. The encrypted text will appear in the field to the right.

The GUI has an “Upload File” button which can be used to import a text file of up to 500 characters for encryption/decryption. As with bulk text input, the “Encrypt” button is used once the file is chosen.

The GUI has a text box in the Output section which will display the output of the encryption/decryption session as a single message. The message appears in the box as the user types the original text into the “Manual Input” field.

The Bulk Output Text Box (bottom right) shows the encrypted results of either a bulk input cut and paste or a file upload after the “Encrypt” button has been selected.

The Light Board output is for Aesthetic purposes only. In the original Enigma Machine, the lights would illuminate to show the enciphered/deciphered output of each typed character. The operator would then hand-copy the results to paper for transmission. It is not necessary to do this with the emulator, as the output is provided in easy to copy text format.

5.4.5 Main Tab: Leftmost Panel: Additional Options

Machine Version Selection

The Machine Version Selection drop-down allows the user to pick from a list of historical machine versions. This feature will restrict the user to only the settings which would have been available on that version of the Enigma Machine. The options are:

1. No Restrictions – The user can enter any combination of rotor and reflector settings from any machine version, even if those settings are not historically accurate.
2. Enigma I – The user can only select from Reflector B or C. Rotor 4 is not an option, and only Rotors I, II, and III are available in rotor selection drop-down menus.
3. Enigma M3 Army – This has all of the features of Enigma I, plus an additional two rotors in the rotor selection drop-down menus (Rotors IV and V).
4. Enigma M4 Naval – This has all of the features of Enigma M3 Army, plus an additional three rotors in the rotor selection drop-down menus (Rotors VI, VII, and VIII).
5. Enigma M4 R1 – This has all of the features of Enigma M4 Naval, plus Reflector B Thin and C Thin and the fourth rotor option with Rotors Beta and Gamma.
6. Enigma M4 R2 – This is identical to Enigma M4 R1.

Output Space Options

The Output Space Options drop-down provides a list of options for the placement of spaces in the output text. These options are:

1. No Spaces – The output will be a solid line of text with no spaces or line breaks.
2. 4 Spaces – The output will have one space after every four characters.
3. 5 Spaces – The output will have one space after every five characters.
4. Original Spaces – The output will use the same spacing as the original input. (NOTE: This is easier to decrypt, because original spacing exposes world length.)

Reset Options

Three buttons exist under Reset Options. These are:

Default Config – Acts like the reset button on the page linked. Clears everything to the default configuration.

Reset Indicators – Sets the indicators to their initial configuration for the machine being used, clears out the text boxes. Useful for creating multiple messages using the same key and the same indicators.

Clear Text – Leaves the machine alone, clears the text boxes in case they have become cluttered.

5.4.6 Cryptanalysis Tab: Input Settings Panel

Note: The cryptanalysis algorithm is designed to analyze messages that were generated with historical machine configurations (i.e. different rotors, special reflectors for four-rotor machines). Attempting to decrypt a message generated using invalid setting combinations will likely result in failure.

Text Input Box

The Text Input box accepts user text which is directly entered or copy/pasted for analysis.

Rotor Selection

The rotor selection section offers four drop downs for the four rotor selections. These provide identical options to their counterparts in the Main tab, but are instead used to assist in cryptanalysis if one or more rotors are already known by the user.

Reflector

The reflector setting can be set manually by the user if the reflector used in initial encryption is known.

Ring Settings

Similar to the rotor selection option, the ring settings option reflects it’s Main tab counterpart, but
is used to assist in cryptanalysis if one or more ring setting is known by the user.

**Indicator Settings**

The indicator settings are the rotor starting positions. If the user knows one or more of the rotor starting positions, these can be entered to assist with cryptanalysis.

Note: For best results use the ring and indicator settings on the same rotor, as opposed to differing rotors.

**Plugboard Settings**

The plugboard settings box accepts pairs of two-letter plugboard matches if the user knows one or more of these pairs. The format is identical to the format which appears in the plugboard box on the Main Tab. EG: AB LM OR.

**Thread Limit**

The thread limit spinner is used to indicate how many threads to use in cryptanalysis in order to take advantage of multithreading for processing speed.

**Candidate Size**

The candidate size spinner tells the cryptanalysis program how many best results from each part to carry to the next. Increasing this number increases the time to decrypt, but also typically gives more accurate results.

**Decrypt Button**

The Decrypt button should be pressed after all other Input Settings are set. This will begin analysis on the input.

**Progress**

The progress bar shows cryptanalysis progress. This is an important feature because process time increases as the input increases based on the size of the input.

**5.4.7 Cryptanalysis Tab: Output Settings Panel**

**Text Output Box**

The text output box occupies the center of the Cryptanalysis Tab. This box will display the most likely decryption of the entered text based on analysis and any settings entered by the user.

**Rotor, Reflector, Ring, Indicator, and Plugboard Boxes**

The rotor, reflector, ring, indicator, and plugboard boxes will display the “best guess” selections which the cryptanalysis program has chosen when decrypting the output. In this way the user knows not only what the program assumes the output to be, but what settings were used to create that output.
5.5 Information on Software Commands

The following commands can be used within the program:

1. Ctrl+C/Ctrl+V – These commands can be used to copy and paste text into the Input Text Box, or copy it from the Output Text Box.

2. F5 – This command can be used in a web browser to completely refresh/reset the program.

3. Ctrl+F4 – This command can be used to manually close the browser window if the program is unresponsive.

5.6 Error Messages and Problem Resolution

1. Problem: Java Applet will not run in browser, or displays incorrectly.

   1.1. Resolution:

      1.1.1. Make sure your browser itself is updated to the latest version, and Java is enabled in your browser’s user settings. A list of browsers that support Java can be found at this link. [http://java.com/en/download/help/enable_browser.xml](http://java.com/en/download/help/enable_browser.xml) Visit the following link to ensure Java is enabled. [http://java.com/en/download/help/enable_browser.xml](http://java.com/en/download/help/enable_browser.xml)

      1.1.2. If that does not resolve the issue, visit [http://www.java.com/verify](http://www.java.com/verify) and verify that you are running the latest version of Java.

2. Problem: Output text has different or fewer characters than input text.

   2.1. Resolution 1 - Missing characters:

      2.1.1. The program automatically strips special characters from input text, using only Latin alphanumeric characters for processing.

      2.1.2. Verify that the input text uses only alphanumeric characters and attempt the operation again.

      2.1.3. Try spelling out any characters that are stripped, or replace them with representative words or characters.

   2.2. Resolution 2 - Different characters:

      2.2.1. The program automatically converts numbers to letters for processing, using the following rule set: 0 = P, 1 = Q, 2 = W, 3 = E, 4 = R, 5 = T, 6 = Z, 7 = U, 8 = I, 9 = O.

      2.2.2. If the automatic conversion is not desirable, try spelling out the numbers in the input text.

3. Error Message 100: No valid data in input text.

   3.1. Resolution 1 – Invalid or no characters in input:
3.1.1. Verify that the input text or document contains only Latin alphanumeric characters for most accurate encryption. (a-z, 0-9)
3.1.2. The program will automatically disregard any incompatible characters in the input text.
3.1.3. If only invalid characters are entered, output message will read “Input text must contain at least one letter or number.”

3.2. Resolution 2 – Invalid file type:
3.2.1. Verify the input file type is .txt.
3.2.2. If the file type is .txt, open the file in a text editor such as Notepad and remove special (non-alphanumeric) characters from the document.

4. Error Message 101: Selected rotor is already in use.
4.1. Resolution:
4.1.1. Ensure all rotors selected are unique and there are no duplicates.
4.1.2. Check for the same rotor used in a different drop-down box, EG: {II}, {IV}, {II}.

5. Error Message 102: With the fourth rotor inactive, you can only choose reflector B or C.
5.1. Resolution:
5.1.1. The physical Enigma Machine would only work with reflectors B or C if the fourth rotor was inactive. Ensure the correct reflector is selected for your choice of rotors.
5.1.2. The machine will automatically default to a valid reflector option after this message is displayed.

6. Error Message 103: With the fourth rotor active, you can only choose reflector B Thin or C Thin.
6.1. Resolution:
6.1.1. The physical Enigma Machine would only work with reflectors B Thin or C Thin if the fourth rotor was active. Ensure the correct reflector is selected for your choice of rotors.
6.1.2. The machine will automatically default to a valid reflector option after this message is displayed.

5.7 Glossary
Definitions of uncommon words and acronyms used in this document.

1. Substitution – A cipher algorithm which, in this project, substitutes one alphabetic character for another.
2. Shift – A cipher algorithm which, in this project, replaces a letter in a known list with the letter one place down in the list, wrapping the first letter around to the end of the list.
3. Rotor – A gear-like component which is hard-coded for a specific substitution permutation. There are ten possible rotors, the number, combination, and order of which determine the output and complexity of the final result.
4. Reflector – A machine component which performs a substitution and returns the result to the rotors.
5. Stepping Mechanism or Carry Notch – A specific pre-determined character on each rotor.
which will trigger a shift of the rotor to the left.

6. Ring Setting – A setting which shifts the initial permutation and carry notch position of a rotor.

7. Plugboard or ‘Stecker board’ – A manually hard-wired substitution method added to the Enigma Machine in an attempt to thwart decryption.

8. PT or Plaintext – The raw, unencrypted user input.

9. CT or Ciphertext – The result of the full encryption on the plaintext.

10. Key – For the purposes of this project, a message which indicates the rotor, reflector, ring, and plug board settings necessary to decrypt a message that has been encrypted.

11. NLP (Natural Language Processing) – Machine interpretation and understanding of natural language input.

12. Quadgram Analysis (or Quad Bomb) – Frequency/Statistical analysis of blocks of 4 characters from a cryptext. The cryptext is decrypted with multiple keys, and the most likely key is chosen based on the result which produces the most English-like groups of 4 characters.

13. Bigram Analysis – Statistical/frequency analysis much like the Quadgram, but analyzing groups of only two characters for most common syllables.

14. N-gram Analysis – Statistical/Frequency character analysis. In the case of this program, single letter to three-letter syllabic English language frequency analysis. The Enigma Machine Project does not analyse character groups over four.

15. Corpus – A large body of texts which can be processed to build a database of letter, word, and phrase frequency.

5.8 Related Information Sources

Reference for historical and mechanical breakdowns

   http://www.codesandciphers.org.uk/enigma/index.htm


6 Test Plan and Results

6.1 References

1. Enigma Machine Project Plan (EMPP)
2. Enigma Machine Users Guide (EMUG), version 0.9

6.2 Introduction

The purpose of this test plan is to provide an overview of the deliverables and requirements to be tested during development of the Enigma Machine emulator. It will also outline the testing and regression procedure, criteria, team’s needs, and risks. This version (0.9) is pre-Alpha and subject to revision.

6.3 Test Items
The following is a list of items to be tested.

1. Phase 1 Deliverables:
   1.1. Rotor version 0.9 (Pre-Alpha)
   1.2. Plugboard version 0.9 (Pre-Alpha)
   1.3. Rotors version 0.9 (Pre-Alpha)
   1.4. EnigmaMachine version 0.9 (Pre-Alpha)
   1.5. EnigmaGUI version 0.7 (Pre-Alpha, partial functionality)

2. Phase 2 Deliverables
   2.1. EnigmaGUI version 0.8 (Pre-Alpha, Features Added)
   2.2. Cryptanalysis GUI version 0.9 (Pre-Alpha)

3. Phase 3 Deliverables
   3.1. Enigma Simulator with complete GUI and Cryptanalysis functions, version 1.0

6.4 Software Risk Issues

Most of the third party software and services the team intends to use to support product testing, which have been listed in the EMPP, are already available and have been verified as stable. The project is not yet far enough along to test the applet version on the externally hosted website. If that should prove unviable for some reason, the user will be able to make use of a locally stored version of the program and any Java 7.0 capable browser.

6.5 Features to be Tested

The following is a list of areas to focus on while testing:

1. The “key” setting process, including rotor settings and plugboard wiring
2. The text input feature
3. The file upload feature
4. Lightboard output
5. Accuracy of Text output
6. Specific Machine Selection feature
7. Automated decryption results

6.6 Features Not to be Tested

The following feature will not be specifically addressed in testing: sounds, and their
relative level of realism. While a nice feature, this is not currently part of our requirements.

6.7 Approach

The testing of the Enigma simulator will consist of Unit and System. Testing will be performed by all team members, with final verification and sign-off by the Testing Lead. Testing will be performed on systems meeting the requirements listed in the EMPP.

Team organization and communication will follow the procedures outlined in section three of the EMPP.

6.7.1 Unit Testing

Unit testing will test the outputs of the major system classes (as defined in the Software Design documentation) and verify that the class methods provide expected outputs when provided with both valid and invalid inputs. The Enigma Team will rely primarily on JUnit open source framework and the Eclipse IDE for these tests. Preferably, JUnit tests for a given class will be written by a different team member than the one who authored the bulk class.

6.7.2 System Testing

SYSTEM testing will verify the Enigma simulator’s ability to adhere to each system requirement. Different team members may be given responsibility for the testing of different settings, input/output combinations, and any code-breaking features included in the final product. The system tests will require a browser capable of running Java 7.0 applets. The general procedure for the System Test follows:

1. Launch the program in accordance to the EMUG. Note: While initial tests can be conducted with locally stored copies of the Applet, the final test should be accomplished through http://www.enigamachine.info or http://www.enigamachine.org.
2. Begin to enter the first test phrase in the INPUT field. Verify that the user is prompted to select rotors first.
3. Using the REFLECTOR dropdown menu, select the reflector listed as the key for the given test phrase.
4. Using the ROTOR, Rotor POSITION, and RING SETTING dropdown menus, set the left and middle rotors according to the key for the given test phrase.
5. Set the right rotor as a duplicate of the left rotor.
6. Attempt to enter the first test phrase and verify that the user is warned that all rotors must be different and prompted to select different rotors.
7. Set the remaining ROTORS AND RING SETTINGS according to the key for the giving
test phrase.
8. Using the PLUGBOARD dropdown menus, attempt to set the first plugboard option to “AA”.
9. Attempt to enter the first test phrase and verify that the user is warned that the same letter cannot be used twice in the PLUGBOARD and then prompted to choose a different setting for the PLUGBOARD.
10. Finish setting the key provided for the test phrase.
11. Enter the test phrase. Verify that the appropriate LIGHTBOARD letters light up as each letter is typed, and verify that the expected output appears in the OUTPUT text field.
12. Enter the remaining manual-entry test phrases and keys, verifying expected outputs each time. Ensure the Enigma simulator is set before you begin to enter its associated test phrase.
13. Click the UPLOAD FILE button and attempt to choose something besides a text file. Verify that you are unable to do so.
14. Select reflectors, rotors, etc., according to the key for the first test file to be uploaded.
15. Using the file chooser, select the first prepared test file.
16. Click PROCESS and verify that the expected output appears in the OUTPUT field.
17. Repeat steps 14-16 for all remaining “valid input” test files.
18. Attempt to upload the “invalid input” test file and verify that the correct error messages are displayed. The final phase of testing will involve each team member creating their own messages complete with keys and sending their Cipher Text (CT) plus keys to the other members of the team. The other members should then set the Enigma according to the key and enter the CT in order to retrieve the original message. This will demonstrate the Enigma’s ability to encrypt and decrypt spontaneously generated messages in addition to predetermined test cases.

6.8 Item Pass/Fail Criteria

The test process will be completed when the Test Lead has signed off on the final project and all deliverables have been submitted to Dr. Duchon. Stretch Goal features will only be included in the final project if they are completed and operational. Most Stretch Goals are code-breaking methods (decryption through some method where the user lacks the key), and will be considered acceptable if they can produce the original message in at least 80% of test cases. For the basic system, the tests will be considered passing if we yield the predicted results from the provided setting/input combinations 100% of the time.

6.9 Suspension Criteria and Resumption Requirements

This project has no suspension criteria or resumption requirements.
6.10 Test Deliverables

All Deliverables will be submitted to Dr. Duchon on the provided delivery dates.

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Test Plan (draft)</td>
<td>November 9, 2013</td>
</tr>
<tr>
<td>Text files to be used to test encryption and decryption functions</td>
<td>November 9, 2013</td>
</tr>
<tr>
<td>Test report containing unit test results for the Plugboard Class, Rotor Class, Rotors Class, and EnigmaMachine Class, including program files used to run the JUnit tests</td>
<td>November 24, 2013</td>
</tr>
<tr>
<td>Preliminary test report containing the system test results for the Phase 2 GUI as well as the unit tests for those code-breaking methods that have been completed.</td>
<td>December 1, 2013</td>
</tr>
<tr>
<td>Test report containing the system tests of the Phase 3 system, which should have a fully functional GUI plus a codebreaking GUI (incorporated into the master document).</td>
<td>December 8, 2013</td>
</tr>
<tr>
<td>Final system test report (incorporated into the Master document)</td>
<td>December 15, 2013</td>
</tr>
</tbody>
</table>

6.11 Remaining Test Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Assigned To</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create text files for System testing</td>
<td>WGA</td>
<td>Complete</td>
</tr>
<tr>
<td>Define Unit Test rules and procedures</td>
<td>JLI, BMW, RM</td>
<td>Complete</td>
</tr>
<tr>
<td>Create Test Report prototypes</td>
<td>JLI</td>
<td>Complete</td>
</tr>
</tbody>
</table>
6.12 Environmental Needs

There are no unusual requirements, special test equipment, or restrictions required for testing.

6.13 Staffing and Training Needs

Due to the small size and brief time frame of the project, all five team members will also act as testers. This project has no special training needs.

6.14 Responsibilities

The Testing Lead is responsible for the creation of test plans and strategies, unit testing, system testing, test reports, and bug-fix verifications.

6.15 Schedule

Most of the testing schedule is detailed in the EMPP as part of the overall plan schedule. Those items listed in section twelve (Remaining Tasks) but not explicitly mentioned in the EMPP schedule will be completed by the dates listed in the table following, so that they can be referenced by the rest of Team Enigma for the performance of their tests. Note that these are not the delivery dates, but the internal completion dates.

<table>
<thead>
<tr>
<th>Item</th>
<th>Completion Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text files to be used in testing</td>
<td>November 9, 2013</td>
</tr>
<tr>
<td>Definition of Unit testing rules and procedures</td>
<td>November 16, 2013</td>
</tr>
<tr>
<td>Template for reporting Unit test results</td>
<td>November 16, 2013</td>
</tr>
<tr>
<td>Finalization of System testing rules and procedures</td>
<td>December 1, 2013</td>
</tr>
<tr>
<td>Template for reporting System test results</td>
<td>December 1, 2013</td>
</tr>
</tbody>
</table>

6.16 Planning Risks and Contingencies
The following table contains planning risks identified by Team Enigma as having the potential to impact product delivery, as well as proposed solutions.

<table>
<thead>
<tr>
<th>Risk Description</th>
<th>Proposed Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network or online service failure could inhibit communication or code sharing</td>
<td>Each member keeps track of their personal tasks offline and switches to a task that can be done individually until service is restored</td>
</tr>
<tr>
<td>Document or code loss due to technical difficulties or user error</td>
<td>Users keep personal backup copies of important documents to prevent total data loss in the case of a failure or mistake</td>
</tr>
<tr>
<td>Personnel or velocity loss</td>
<td>Active load balancing between team members, and re-evaluation of goals if necessary</td>
</tr>
<tr>
<td>Features committed to cannot be completed by due date due to poor estimation of team capabilities in the provided timeframe</td>
<td>Active time estimate updates. Identifying of issues early and re-evaluation of goals. Update goals and documentation to reflect changes. Do not wait until the last minute to identify problems</td>
</tr>
<tr>
<td>“Feature Creep,” i.e. attempting to add too many new features at the end of the project</td>
<td>Estimate all time into tasks, including QA, Review, and Sign-Off. Do not add a stretch goal if all necessary team members cannot commit. Do not add a stretch goal if existing features have outstanding bugs</td>
</tr>
</tbody>
</table>

6.17 Approvals

Final sign-off on all tests will be the responsibility of the Testing Lead, and will be done through the Enigma Trello board.

6.18 Unit Test Guidelines (TE-UTPS0.9)

6.18.1 References

1. Enigma Machine Project Plan (EMPP)
2. Enigma Machine Users Guide (EMUG), version 0.9
3. Enigma Machine Test Plan (EMTP), TE-MTP0.9

6.18.2 Purpose

The purpose of this subsection is to provide a set of consistent standards and guidelines to use while conducting Unit level testing on the Enigma Machine simulator. This documentation
does not contain a set of detailed procedures for each individual unit, as these will largely consist of JUnit tests.

6.18.3 Special Requirements
The JUnit framework will be the primary means of testing at the Unit level.

6.18.4 Guidelines
The following guidelines shall be followed whenever reasonable when conducting unit tests on the Classes to be used by the Enigma Machine simulator.

1. The primary means of testing individual classes will be the JUnit framework.
2. The programmer who creates the final acceptance JUnit class should not be the primary programmer who created the class in question.
3. Classes are to be tested at the most basic possible level first before any Class that incorporates them is tested.
4. All testing classes should have “Test” as the last four characters of their class names.
5. All methods that produce output are to be tested against expected output, and the results are to be recorded in a Unit Test Report.
6. Multiple classes can be included in a single Test Report.
7. Each output-producing method should have at least one JUnit method focused on it, with the most basic method checked first.
8. Each method should be checked using multiple possible cases.
9. A class must pass all test cases provided to be considered acceptable.

6.18.5 Classes To Be Tested

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Primary Programmer</th>
<th>Tester</th>
<th>Testing Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plugboard</td>
<td>BMW</td>
<td>JLI</td>
<td>19 Nov 2013</td>
</tr>
<tr>
<td>Rotor</td>
<td>WGA</td>
<td>JLI</td>
<td>19 Nov 2013</td>
</tr>
<tr>
<td>Rotors</td>
<td>BMW</td>
<td>JLI</td>
<td>19 Nov 2013</td>
</tr>
<tr>
<td>EnigmaMachine</td>
<td>BMW</td>
<td>JLI</td>
<td>19 Nov 2013</td>
</tr>
</tbody>
</table>

6.19 Unit Test Report

6.19.1 Summary
This report contains the results of the unit testing conducted by Team Enigma in order to verify the operation of individual classes used in the final Enigma simulator. This document will be updated as new classes are created. Thus, this document is subject to heavy revision. Unit tests will generally use the JUnit framework and will, when reasonable, adhere to the guidelines laid out in the EMUTG and follow the schedule put forth in the EMTP and EMPP. Testing results are to be approved by the individual tester (when different from the Testing Lead), the Testing Lead, and the Project Planning Lead.
6.19.2 Test Items

Classes to be tested are as follows.

1. Plugboard
2. Rotor
3. Rotors
4. EnigmaMachine

A basic version of the GUI has been completed, although not all features and validation are complete. GUI testing will be performed as part of the full system test.

While some of the codebreaking classes have been completed, they have not yet been verified. The results of these unit tests will be included in the next phase’s progress update.

6.19.3 Environment

Unit testing is to be conducted using the Eclipse IDE with the JUnit testing framework.

6.19.4 References

See section 6.18.1

6.19.5 Variances

Team Enigma found different testers for each class to be largely unnecessary, and the Test Lead authored most JUnit classes used. The Code and Requirements Leads aided in logic checking of the JUnit tests. Otherwise, test classes have been written in accordance with the EMUTG.

6.19.6 Comprehensiveness Assessment

Unit testing verified the functioning of the four components of the EnigmaMachine back-end, although there were some complications that arose during testing. Several of the classes accept different kinds of invalid input (for example, the Plugboard will allow its client to substitute a letter for a punctuation mark). The team decided that this was acceptable provided it was included in the documentation of the classes, due to the fact that input validation was designated as part of the GUI’s functionality during the design phase.

The Plugboard performs simple two-way substitutions (that is, if “A” is substituted for “B”, “B” must also be substituted for “A”). Testing for this was relatively simple, although the constructor was adjusted to convert any lowercase into uppercase first, and was adjusted to prevent a runtime error when presented with a character map String that contained an odd number of characters.

Although the Rotor class, in essence, perform substitutions as well (in this case, one forward and one reverse), it proved difficult to construct tests for many of its related methods, such as the cycling, on its own. To address this, heavier testing of the Rotor class was moved up a level, to the Rotors class (which contains three to four rotors plus handles the Reflector function).

The Rotors class essentially acts as a full Enigma minus the Plugboard. It contains three to four instances of the Rotor class (depending on configuration) and, as mentioned, performs the Reflector function. It contains the information to build the Reflectors and Rotors most often
used by the German military in World War II. Because it contains most of the Enigma functions, the JUnit test was able to include all of the prepared test cases that don't involve a plugboard, uploaded files, or non-alphanumeric input (see the table below for test cases included). The expected results for these tests were created by the use of an Enigma simulator available online, and verified against two other Enigma simulators (because it is unlikely that three different online Enigma simulators are wrong in the same way). The JUnit test uses a loop to create instances of the Rotors class that have differing test configurations. A second loop then feeds each Rotors instance with its corresponding test input and then compares the actual results with the expected results. The test class was designed in such a way to make it easy to add new test cases as they are deemed necessary.

<table>
<thead>
<tr>
<th>Ring</th>
<th>Rotor</th>
<th>Plugboard</th>
<th>Reflect or</th>
<th>Rotor Order</th>
<th>Input</th>
<th>Expected output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>B</td>
<td>123</td>
<td>AAAA</td>
<td>BDZGO</td>
</tr>
<tr>
<td>BBB</td>
<td>AAA</td>
<td>(none)</td>
<td>B</td>
<td>123</td>
<td>AAAA</td>
<td>EWTYX</td>
</tr>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>B</td>
<td>321</td>
<td>ELLEN</td>
<td>VONDB</td>
</tr>
<tr>
<td>AAA</td>
<td>AAB</td>
<td>(none)</td>
<td>B</td>
<td>123</td>
<td>AAAA</td>
<td>DZGOW</td>
</tr>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>B</td>
<td>135</td>
<td>AAAAJESSY</td>
<td>LFLCC OXZIS</td>
</tr>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>B</td>
<td>435</td>
<td>AAAAAWALTER</td>
<td>OGUYV EVJWX K</td>
</tr>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>C</td>
<td>123</td>
<td>AAAAADOLPH</td>
<td>PBZUQ QKFHO D</td>
</tr>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>B</td>
<td>678</td>
<td>AAAAIKLEY</td>
<td>GJUBB PFPRS</td>
</tr>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(none)</td>
<td>C</td>
<td>678</td>
<td>AAAAMONTANEZ</td>
<td>MWMJL CFGQE EQF</td>
</tr>
<tr>
<td>AAA</td>
<td>AAV</td>
<td>(none)</td>
<td>B</td>
<td>123</td>
<td>AAAAALYN</td>
<td>UQOFX MBD</td>
</tr>
<tr>
<td>AAA</td>
<td>AEA</td>
<td>(none)</td>
<td>B</td>
<td>123</td>
<td>AAAAMATTHEW</td>
<td>FJBWZ CCJFN OS</td>
</tr>
<tr>
<td>AAAA</td>
<td>AAA</td>
<td>(none)</td>
<td>B thin</td>
<td>B123</td>
<td>AAAAOHLMACHE R</td>
<td>BDZGO KTGOT WOLM</td>
</tr>
<tr>
<td>AAAA</td>
<td>AAAA</td>
<td>(none)</td>
<td>B thin</td>
<td>G876</td>
<td>AAAAWINSTEAD</td>
<td>VFITY YAYYW CZO</td>
</tr>
<tr>
<td>AAAA</td>
<td>AAAA</td>
<td>(none)</td>
<td>C thin</td>
<td>G867</td>
<td>AAAAGENEG</td>
<td>DCTVG JVFUL</td>
</tr>
</tbody>
</table>

Testing revealed that the Rotors class was unable to properly encrypt messages that were intended to cause all three rotors to turn at once. After this problem was reported, a logic error was uncovered in the Rotor class that was quickly corrected, after which all tests cases passed.

The EnigmaMachine class includes the Plugboard and adds the ability to encrypt on a String level. As with the Rotors class, JUnit was designed to easily add more test cases. In the
original JUnit construction, a logic error was included in the method for testing the encryptChar function that yielded incorrect results. The problem was quickly identified as an issue with the test, not the EnigmaMachine itself, and corrected. In addition to the test cases included above, the following additional test case was added:

<table>
<thead>
<tr>
<th>Ring</th>
<th>Rotor</th>
<th>Plugboard</th>
<th>Reflect</th>
<th>Rotor Order</th>
<th>Input</th>
<th>Expected output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>AAA</td>
<td>(RW)</td>
<td>(OA)</td>
<td>(SL)</td>
<td>123</td>
<td>ROSANA</td>
</tr>
</tbody>
</table>

### 6.19.7 Summary of Results

After the rotor stepping bug (discussed in section 6) was corrected, all JUnit tests passed, verifying the operation of the EnigmaMachine’s back end.

The following table contains the overall results for each class.

<table>
<thead>
<tr>
<th>Class</th>
<th>Tester</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plugboard</td>
<td>JLI</td>
<td>Pass</td>
</tr>
<tr>
<td>Rotor</td>
<td>JLI</td>
<td>Pass</td>
</tr>
<tr>
<td>Rotors</td>
<td>JLI</td>
<td>Pass</td>
</tr>
<tr>
<td>EnigmaMachine</td>
<td>JLI</td>
<td>Pass</td>
</tr>
</tbody>
</table>

The following table contains the results of the Plugboard unit test.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>matchChar(char c)</td>
<td>Pass</td>
</tr>
</tbody>
</table>

The following table contains the results of the Rotor unit test.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNotchPosition()</td>
<td>Pass</td>
</tr>
<tr>
<td>getPosition()</td>
<td>Pass</td>
</tr>
<tr>
<td>cycleRotor()</td>
<td>Pass</td>
</tr>
<tr>
<td>forwardEncrypt(char)</td>
<td>Pass</td>
</tr>
<tr>
<td>reverseEncrypt(char)</td>
<td>Pass</td>
</tr>
</tbody>
</table>
The following table contains the results of the Rotors unit test.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypt(char)</td>
<td>Pass</td>
</tr>
</tbody>
</table>

The following table contains the results of the EnigmaMachine unit test.

<table>
<thead>
<tr>
<th>Method</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>encryptChar(char)</td>
<td>Pass</td>
</tr>
<tr>
<td>encryptString(String)</td>
<td>Pass</td>
</tr>
</tbody>
</table>

**6.19.8 Evaluation**

As has already been discussed, many of the classes do not have special handling for invalid input. Some modifications have been added to prevent runtime errors that will crash the program, but other types of invalid input are simply accepted and then produce strange results. This is, for the moment, considered acceptable behavior, and will be addressed during the GUI’s validation phase. Current work concentrates on preventing a system shutdown when the program receives invalid inputs.

**6.19.9 Summary of Activities**

The Test Lead wrote most of the JUnit tests over the course of half a week, largely during the evenings. The Project Lead, Requirements Lead, and Test Lead were involved in most of the discussion over invalid input handling. The Code Lead corrected the stepping bug that testing uncovered, and both the Code and Requirements Lead aided the Test Lead in identifying the logic error in the testing of the encryptChar method of the EnigmaMachine class. No hours were charged by anyone because this is not a paid project.

**6.19.10 Approvals**

<table>
<thead>
<tr>
<th>Approving Authority</th>
<th>Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing Lead (Jessica Ikley)</td>
<td>JLI</td>
</tr>
<tr>
<td>Project Planning Lead (Ellen Ohlmacher)</td>
<td>EEO</td>
</tr>
</tbody>
</table>

**6.20 Phase 2 System Test Report [TE-STPR0.8]**

**6.20.1 Summary**

This report contains the results of the Phase 2 System Test Report for the Enigma
Machine simulator being created by CMSC 495’s Team Enigma. Because the Enigma Machine simulator is not yet complete, it is expected that some sections of the test will fail. Consequently, this test also serves to document which functions have been completed and which have yet to be accomplished. This report should be viewed in conjunction with the Enigma System Test Procedures, and all testing steps referenced are from that document.

6.20.2 Test Items
This procedure tests the following operations:

1. Stand-alone operation
2. Web deployment operation
3. Expected error messages of the GUI
4. Encryption from the text input box
5. Encryption via uploaded file
6. Cryptanalysis GUI
7. Cryptanalysis functions
8. Website links

6.20.3 Environment
The system test requires a computer running JRE 7.0 and a browser, plus the .jar file necessary for stand-alone tests.

6.20.4 References
See section 6.18.1.

6.20.5 Variances
Because not all GUI functions have been completed, sometimes a test needs to be run through alternate means. In particular, the Cryptanalysis test currently has to be conducted using the Cryptanalysis test GUI and not the GUI that will be incorporated into the final project. Also, the web applet has not yet been deployed, meaning that all tests had to be conducted using the desktop Jar instead.

6.20.6 Comprehensiveness Assessment
No assessment was found to be necessary for the scope of this project.

6.20.7 Summary of Results
The following results apply to the Phase II version of the GUI, and not the final submission.

6.20.7.1 Stand Alone Application Operation
The following table contains the results of section 6.1 of the Enigma Machine System Test Procedure (EMSTP).

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.2</td>
<td>Verify that its appearance matches the screenshot.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Verify that “ILBDA” appears in the Output Text Box.</td>
<td>Pass</td>
</tr>
</tbody>
</table>
6.20.7.2 Web Deployment Operation

The following table contains the results of section 6.2 of the EMSTP. The applet has not yet been uploaded to the Team’s website (http://enigmamachine.info/), so the failure of all steps here is expected.

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2</td>
<td>Verify that the page and applet load.</td>
<td>Fail</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Verify that the appearance of the applet matches the screen shot.</td>
<td>Fail</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Verify that “ILBDA” appears in the Output Text Box.</td>
<td>Fail</td>
</tr>
</tbody>
</table>

6.20.7.3 Error Message Verification

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.2</td>
<td>Verify that an error message, “You cannot reuse rotor choices” pop ups and that the second rotor changed returns to its original state.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Verify that the Rotor 4 Ring Setting automatically reverts to the blank option.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.7</td>
<td>Verify that a popup warning that says, “With the fourth rotor inactive, you can only choose reflector B or C”.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.9</td>
<td>Verify that the Rotor 4 Ring and Rotor settings switch to “A” on their own.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.10</td>
<td>Verify that the Reflector automatically switches to “REFLECTOR B THIN”.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.12</td>
<td>Verify that the Rotor 4 Ring Setting drop down selection automatically reverts to the previous setting.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.14</td>
<td>Verify that a popup warning appears that says, “With the fourth rotor active, you can only choose reflector B thin or C thin” appears.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.16</td>
<td>Verify that the Reflector drop down selection automatically reverts to the previous setting.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.18</td>
<td>Verify that the Rotor 4 Ring and Rotor settings return to the blank option on their own.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.19</td>
<td>Verify that the Reflector switches to “REFLECTOR B” on its own.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.21</td>
<td>Verify that the Rotor 4 Ring and Rotor settings switch to “A” on their own.</td>
<td>Pass</td>
</tr>
<tr>
<td>6.3.22</td>
<td>Verify that the Reflector automatically switches to “REFLECTOR B THIN”.</td>
<td>Pass</td>
</tr>
</tbody>
</table>
### 6.7.20.4 Encryption from Manual Input

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction or Input Text</th>
<th>Expected Output</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.3</td>
<td>As you type each letter, verify that the expected letter is displayed in the Output Text Box and the corresponding Lightboard letter lights up.</td>
<td></td>
<td>Pass</td>
</tr>
<tr>
<td>6.4.4</td>
<td>As you type each letter, verify the spinners representing the Rotor Positions advance.</td>
<td></td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>AAAAA</td>
<td>BDZGO</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>AAAAA</td>
<td>EWTYX</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>ELLEN</td>
<td>VONDB</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>ROSANA</td>
<td>KDBMW K</td>
<td>VIPGC W</td>
</tr>
<tr>
<td></td>
<td>AAAAA</td>
<td>DZGOW</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>AAAAAAJESSY</td>
<td>LFLCC OXZIS</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>AAAAAAWALTER</td>
<td>OGUYV EVJWX K</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>AAAAAADOLPH</td>
<td>PJBUZ QKFHOD</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>AAAAAAIKLEY</td>
<td>GJUBB PFPRS</td>
<td>Pass</td>
</tr>
</tbody>
</table>

### 6.20.7.5 Encryption from the Bulk Encryption field

Still does not properly adjust Rotor Positions after a bulk encryption, leading to mismatch between the GUI display and the underlying Enigma.

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction or Input Text</th>
<th>Expected Output</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.5</td>
<td>Verify that the Rotor Positions read “AAM”.</td>
<td>AAM</td>
<td>AAA</td>
</tr>
<tr>
<td></td>
<td>AAAAAAMONTANEZ</td>
<td>MWMJL CFGQYE QF</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>aaaaalyn</td>
<td>UQOFX MBD</td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td>aaaaamatthew</td>
<td>FJBWZ CCJFN OS</td>
<td>Pass</td>
</tr>
</tbody>
</table>
### 6.7.20.6 File Upload Encryption

This section verifies the operation of the File Upload Encryption feature. The validation check that prevents the user from uploading a non-.txt file has not yet been implemented, so that particular failure was expected and will be addressed in a future version of the software. The remainder of the failures can be attributed to problems already discovered and discussed in the previous two sections. The next version of the test procedures will be updated to ensure that the Rotor Position indicators on the GUI match the Rotor Positions in the Enigma back-end after a file upload.

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction or Input Text</th>
<th>Expected Output</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.1</td>
<td>Using the “Browse…” button, attempt to upload a file that does not end in “.txt”. Verify that you are unable to do so.</td>
<td>N/A</td>
<td>Fail</td>
</tr>
<tr>
<td></td>
<td><strong>TestCaseCryptextl_II_III_CBU_SteckerBI_CBU.txt</strong></td>
<td>TESTI NPUTI ONEII TEOII ITHR ESTEC KERBI OUSTA RTPOS CBU</td>
<td><strong>TESTBNPO TBUNMBBY EUBBBTVR EEESTECKE RXBUOSTARNPUSCIP</strong></td>
</tr>
<tr>
<td></td>
<td>TestCaseImageAsTxt.txt</td>
<td><em>Any, as long as the program doesn’t crash</em></td>
<td>Pass</td>
</tr>
<tr>
<td></td>
<td><strong>TestCasePlaintextAllValidNoSpaces.txt</strong></td>
<td>APGDD IMRKL BTELJ RNML VDLQJ YFKSN YFCKP WCIOU NFSMK HMDOZ ZXUBG DFG</td>
<td>OPGNDXC WATBMBNJ WEZRSVUL ZJFFFFLXMU CJRRCIANT SLMVWBWHH ZGFYBPDC</td>
</tr>
<tr>
<td>Q</td>
<td>PVGMF TVVMU IVYQS SWWYX QSDFJ NEJLM GISGY IVSID D OUHOLEZE YAERSFKW XRPQWBD ZIPWSAYPUI RBJEDBAU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TestCasePlaintextWithCasesAndSpaces.txt</td>
<td>ZPJJS VSPGB WUGZC FGBBC LLWST WIJKH TFTCM SCIGV NBOKK NAHUM XRWNM RNDPW C OPGNDXCR WOMKBNSJ UTYDBABLC JRSBOCODI WBKVJSICA TIOTWHHZ WZJNCLOE UIU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TestCasePlaintextWithCasesSpacesNumbersPunctuation.txt</td>
<td>ZCNXM CFWJB EGNDN FTPFV KQGFN GKHI TLFRXHJLO NORABDRZ VCMBWVFJ YPPOCFZX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TestCasePlaintextWithCasesSpacesSpecialPunctuation.txt</td>
<td>HZQQW BVYWL OTJBQ CFRRX HTQBF DQCHI HRUGI VRZGR CJXMV QTARS JKFFW YIPKT RWAYP FTSMM VFCZB PFHUX XTZSK EBGZQ VJBOB SJCL QHDEH EECBU FRLAN NOJLB KYZWH LBOQN GFBKC GXGVM ITAOS JDUCE ROOXT HRFMY KKNP MOUTH SZOXR DXNTR OIXCH ZLHWK YUQEF AFWEE SGBV GLYIQFQKG EGUBUCXT WMCHAIQV CNBUBYLFY OOYTIIIRKV PIRYTLQCC HGPOXJRP PQKKLUFF UVAGAGWD JYKIEJLIDQI YAUGVIGGX TXGBQDGBI DUHDIACKP CFVHEWDJ DMJEEMBV FCFFVQVBY XRHJMFOG NVVCXYVJY TENCJTBAI PJSOUMUI WYQZXRZF TPFOSXWHI POMCQENG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.20.7.7 Cryptanalysis Verification

The Cryptanalysis phase of the test has only partially been run because a full run that includes all rotor choices, reflector choices, plugboard options, etc., can currently take two or more hours on the tester’s computer. The test cases, because they were designed to test the ability of the QuadBomb (the name given the Cryptanalysis algorithm being used), often include the full array of possible code complications, meaning they run on the “worst case” end of the time spectrum. Both the code and test leads have been testing individual Cryptanalysis aspects and found them to be reliable (for example, limiting the rotors to the first three while testing the plugboard, or excluding the plugboard while testing the full range of rotors). Full tests will be conducted over the course of the week, although the code lead will also be working to speed up execution time. The length of time a Cryptanalysis attempt takes is not unexpected, given the high number of potential Enigma keys (approximately $10^{23}$).

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction or Input Text</th>
<th>Expected Output</th>
<th>Resulting Output</th>
<th>Predicted Setting s</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7. 1</td>
<td>Verify that the Cryptanalysis view matches the screen shot</td>
<td>N/A</td>
<td>Fail</td>
<td></td>
</tr>
<tr>
<td>6.6. 4</td>
<td>Verify that the progress bar advances as the decryption process occurs.</td>
<td>N/A</td>
<td>Fail - Pending Retest</td>
<td></td>
</tr>
</tbody>
</table>
| 1A | ZTZBLEVIBXEIHBRTNPHXLYF XXFCTSIWIIXYSYDLCMDSL KBYCHJHKIMNNZZCKRJHHDMR BUYBPCBXPXOPHEWVSMWS XJLLBFIZBNLMNIVSNUC | CARRYOUTARANDOME ATOFKindness wTHNExpectatI onofRewardsAf EINtheKnowleDg EthatoneDaysome OeneMightDote SameForyou | Correct | Rt: 314
| | | | | Rf: B
| | | | | Rng: AZN
| | | | | Ind: MZN |
| 1B | EVHSWBEHJVTHQUMAIIGUUU BQOWUAYJPXDDBFBFWXVW TSGPAEMJCKVARZSRGYSQF QRGCRNIRNMIGSXMHELI IJLBKXSBYNZWTZEEK | ILOOKALOTBUSIERT HANIAMSIMAActual LyarthersSpora Dicrandomperso NandillplayaFew GigsandthendiSa Ppearforawhile | Correct | Pass |
| 2A | QPSQRJHQJJGVGZGCXQXDI NZTJCFZSZGXTSDBVBWVU XHPJIRLTSYPKXNVTIJCVRZ ZIEVEMKYCDQXLMJJFAFTA PKQHOKFPXNOomboHKTBYLMKINNNPPHEJUEVB | IFYOUSChooSeA ChdaycanbeFILL EdwiTHEVENmore JOYTHANTHEONEBEFOREIFYOUSoCH OoseeeventHemos TseemeinglyRando Meventscanwor | LRCCKYVVSC BRXPCXACHDA YCANBEFille DwiTHEVENm OREJOYTHAN THEONEBEFO REIFYOUSoC | Rt: 235 |
| | | | | Rf: C
| | | | | Rng: ASD |
| | | | | Ind: BJL |

86 | Enigma Machine
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>KINYOURFAVOR</th>
<th>HOOSEEVENT HEMOSTSEEMINGLYRANDOM EVENTSCANWORKKINYOURFAVOR</th>
<th>88% correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B</td>
<td>EOXLBIZKMTGHXKSOJWKHJ QAPFTYKNRZULCMFJCGIDAV QJNOPCFWOJFMKMRUMPPTH PHRAIQWMWQFGYRPAPSRU SKCXMWQFIFXKTRSMJUHCP MEVESYCMHBMRXFCCTXMGYOZFJKPEZPE</td>
<td>BIGCOMPANIESARE LIKEMARCHINGBAND SEVENIFHALFTHEBANDISPLAYING RANDOMNOTESITSTILLSOUNDSKINDOFLIKE MUSICTHECONCEALMENTOFFAUTUREISBUILDINTOTHEM</td>
<td>KLOUPBFEER XXDDELIKEMARCHINGBANDS EVENIFHALFTHEBANDISPLAYING RANDOMNOTESITSTILLSOUNDSKINDOFLIKE MUSICTHECONCEALMENTOFFAUTUREISBUILDINTOTHEM</td>
<td>89% correct</td>
</tr>
<tr>
<td>3A</td>
<td>ZDQWEPHPBPVMVOKPHYKSR HVXBPVMWVHUPSMELGJOKMI BBAABOGVQVQNVUPDCUND RMROMGCBYPYXQNVLYRZ SLMOAHGQVRJUFQZWDNT XBEWCQOKZ</td>
<td>OMANYASHAFTATRANDOMSENTFINDDMAKETHARCHELITTLE MEANANDMANYAWORDATRANDOMS POKENMAYSOOTHE ORWOUNDAHEART THATSBROKEN</td>
<td>QNANYASHAFT AEUENDOMSENFTLUMBMARK THYARCINPILT TIEMEANXMQI MANYAWORDX CEANDOMSPOKENQYSOHT HEORWOFOD AHEARTTHAW TXROKEN</td>
<td>Rt: 146 Rf: B Rng: AIN Ind: AFN Plug: IL 72%</td>
</tr>
<tr>
<td>3B</td>
<td>FCJNBIGLLJNTABLQZZAQLBU FXEG ABXSCFYAYWFLNDRYWRBSCIFEUHAF GDYKSCPXMDBPASHKDH RHKMFMOM GCNENITPHXSDOBBMCMBO OSTTOXX DMJCMXMAZNTBICDNJUMR PHNXK XELBYB</td>
<td>KNOWWHATYOU'RETRYINGTODOBEFORE REYOUTDOITTURNINGGNKOBSATRANDOM ISN'TENLIGHTENINGANYMORETHANTHROWINGPAINTATAWAL BLINDFOLDEDWILL LEYOUUPAINTANCIPERATURE</td>
<td>OAOWWHATYOU'REXYBYINGTG ODIEIREYO UDOITUEHN GNKOBSATRKIRSMISNTENLIE BVENINGANYM OUPZTHANTHR OWINFBAINTA TAWALLOEKN DFOLDEDWIS HETYOUPAIN TNMLCEPICTU</td>
<td>Rt: 146 Rf: B Rng: AIN Ind: AFN</td>
</tr>
<tr>
<td>4A</td>
<td>PCTEBITGGLQMEWBLJUHPD TLHK RHRMDYHTGJZXCVTACOMB JZBU TIOUAXYNFMCIVQGBW LXMQ XBZBGNGONLEGARACYZHI HPDV HJJXKDOFIZPDCTCMQYJGR ZKY AFVUDBEKPFWJUWPAEQ KYGXX MUHUJREATKNFZQOZNSRD OSPF IHGZTMMTXINTDSVFBBFXLJR GDJOUWQCSSQFQYFWVMXC PGDWE UUTDQUSFYYAVBOKSSILB WR</td>
<td>RE</td>
<td>IDONTBELIEVETHATIFYOUODOGOODGOODTHINGSWILLHAPPEN EVERYTHINGISCOMPLETELYACCIDENTALANDRANDOMSOMETIMESBADTHINGSAPPENTOVERGOODPEOPLEANDSOMETIMESGOODTHINGSAPPENTTOBADPEOPLEBUTATLEASTIFYOUTHINTOGOODTHINGSTHENYOURESPENDINGYOURTIMEDOINGSMETHINGSWORTHWHILE</td>
<td>Rt: 258 Rf: B Rng: AON Ind: PYY looks good!</td>
</tr>
<tr>
<td>4B</td>
<td>UZPPYGECGBMYIFQAWVCLS KHZKF LVBFIYAGDZXOZBNJLYZJHCPF AE TBOJNWUPDLUYBWADCZIF AYKKG IQBZDNQCNXVSFXJEKPSAM DPTQ WAQJPHFKZCJCUZYNHGRECR PEKXW IOGCHXYBTYXGMVSWVBEDGCYFJR EHRGTXFUKPSNABDQCWXMERYMB VTHIHPGWRDZCVRKEKBNJYLUNTNFVJYYWKQMNFXVGEJDNEZMZBCRFYXK</td>
<td>RE</td>
<td>THETRUTHISTHATKILLINGINNOCENTPEOPLEISALWAYSWRONG ANDNOARGUMENTOREXCUSENATUREHOWDEEPLYBELIEVDECANEVERMAKEITRIGHTNORELIGIONONEARTH ANDTHERANDOMKILLINGOFYOURBROTHERSANDSISTERSISONTHEISARTH</td>
<td>Rt: 258 Rf: B Rng: AJN Ind: PTY - note how the ring and ind. are offset by five spaces from 4A...</td>
</tr>
<tr>
<td>5A</td>
<td>FADADUBVDEJUMKTPZQADTPGJKJYDGHBFDXUYDGJUOKOARHUBLXAUFBCWIVOYNDFFGGLBYBQMDPSKDFUERGRSGWKEWJOJSZBMXYDSPFVBSYSHPKXEMMDFEQDBPOVWVKYVWGVEQBWIMBFPHGGLMXBAOEWQYTQINMAVQIYBBAAUHGVRLCKMBLLRWWKWHRULVIAHYADFQWMUFKRMFXFWEBQAPQGHRLQFIRWATEMF</td>
<td>ATHERISKOFSONDINGPEDESTRIANILBECOMpletelyYNONESTTHEFIRSTTHINGSNIGIOINTHEMORNINGSNGCHECKGOOGLENEWSPARTIALLYBECausesITSEEKSSORTOFRANDOMANDUNBIASEDANDPARTIALYYBECAUSEITENDTOSTAINTHELSTHATDONTNECESSARILYHAVETHETHETESTINTERNETCONNECTIONS</td>
<td>ATHERISKOFSONDINGPEDESTRIANILBECOMpletelyYNONESTTHEFIRSTTHINGSNIGIOINTHEMORNINGSNGCHECKGOOGLENEWSPARTIALLYBECausesITSEEKSSORTOFRANDOMANDUNBIASEDANDPARTIALYYBECAUSEITENDTOSTAINTHELSTHATDONTNECESSARILYHAVETHETHETESTINTERNETCONNECTIONS</td>
<td>Rt: 365Rf: BRng: AOTInd: FOT</td>
</tr>
</tbody>
</table>

| 5B | KSPJUMHTCygWZGLNHWDBWGNJXKARUZJUCXMEHOSZYHXUHFRZVOYQDUNNAQSMKDPCKNALEFSDTDRDZSYOXOBSTBVUOYQFRZKFAEMVKWXWUWZNXIQQHPJDLDUXGYTOBYGZKFFPAHUHJNAONPNLTSWUXTYBYDZRATZCJPYSIFAZTOUCPFLDCCMDXFWFLWXBHKBLUKGDAAWPEGPBYOFXLLVIA | IVEBEENASKINGMYSELFWHYPUTTOGETHERTHESETHINGSALBUMSTHEANSWERICAMEUPWITHISWELLTSOMETIMESTSARTISTICALLYVIALETSONOTJURNALOMCOLLECTIONOFSONGSSOMETIMESTHESONGSHELVEACOMMONTHREADEVENIFITSONTOBIVIOUSOREVENCONSCIOUSONTHEARTISTSPART | IVEBEENASKINGMYSELFWHYPUTTOGETHERTHESETHINGSALBUMSTHEANSWERICAMEUPWITHISWELLTSOMETIMESTSARTISTICALLYVIALETSONOTJURNALOMCOLLECTIONOFSONGSSOMETIMESTHESONGSHELVEACOMMONTHREADEVENIFITSONTOBIVIOUSOREVENCONSCIOUSONTHEARTISTSPART | Rt: 365Rf: BRng: ACTInd: FCT |

| 6A | JZQOLJGHPUPLGMQSWSNYZDLPX | ALOTOFPEOPLEDOTALKABOUTTHEDEE | ALOTOFPEOPLEDOTALKABOUTTHEDEE | Rt: 473 |
IREMEMBERGOING WITHMYMOMTOA GARAGESALE ASDAKANDTHINKING OLEWORLDWASAN LYTOTRANSITIONASHATAGROSSPLACE THATREALLYIS

IREMEMBERGOING WITHMYMOMTOARANDOMGARAGESALE ASDAKANDTHINKING OLEWORLDWASAN LYTOTRANSITIONASHATAGROSSPLACE THATREALLYIS

ANOTHERRANDOMTHINGIDOISTHESEARCHFOREXTRATESTRIALINTELLEGENEORTSETIANDYOU MAYBEFAMILIARWITHTHEMOVIECONTACTWHICHSORTOFPO PUARIZEDTHATITURRENSOUTHEREAREREALPEOPLEWHO GOOUTANDSEARCHFOREXTRASTERESCALINGSCIENTIFICWAY

ANOTHERRANDOMTHINGIDOISTHESEARCHFOREXTRATESTRIALINTELLEGENEORTSETIANDYOU MAYBEFAMILIARWITHTHEMOVIECONTACTWHICHSORTOFPUARIZEDTHATITURRENSOUTHEREAREREALPEOPLEWHO GOOUTANDSEARCHFOREXTRASTERESCALINGSCIENTIFICWAY
<p>| 7B | VKJMANOHXHTKCQMYUDGMRERTUBBBNAHEVDNZIDHVTGMGUQTMYZPQXQTIPCHAOOEZWYHNZBXIPFXBFXMWYZPNLYLAFJVWPOEDBUVMYFNLUYCRCFXSXIIHBSUOIRNXSAHWPVPTXFMJARCNHCCHFVFLINYFXLABIVSQXCVNGGGYAYBCFCMMLBLFZQATATXCRXXBKDKTZIPFVUQVEATUAHNNOAPPWDBCQBXVLDVZFCUGJYECFQGUYQIT | HUMANWELLBEINGSNOTORANDOMPHEMENONITDEPDSONMANYFACTORSRANGINGFROMGENETICSANDNEUROBIOLOGYTOSOCIOLOGYANDECONOMICSBUTCLEARLYTHEREARESCIENTIFICSTRUTHSTOBKNOWABOUTHOWWEFLOURISHINTHISWORLDWHEREVERWECANHAVEANIMPACTONTHEWELLBEINGOFOOTHERSQUESTIONSOFMORALITYAPPLY | HUMANWELLBEINGSNOTORANDOMPHEMENONITDEPDSONMANYFACTORSRANGINGFROMGENETICSANDNEUROBIOLOGYTOSOCIOLOGYANDECONOMICSBUTCLEARLYTHEREARESCIENTIFICSTRUTHSTOBKNOWABOUTHOWWEFLOURISHINTHISWORLDWHEREVERWECANHAVEANIMPACTONTHEWELLBEINGOFOOTHERSQUESTIONSOFMORALITYAPPLY | Rt: 581 Rf: B Rng: ATG Ind: CTG |
| 8B | KJMHPQFJGHLCUASUYSWHTUREBIMYKCMGPUNBEAYSTF | DONTCRYBECAUSETSEVERSMILEBECASEITHAPPENED | OFRAMEYOPECBHRASEMSEEDORRU | Rt: 573 Rf: B Rng: |</p>
<table>
<thead>
<tr>
<th>COO stoi OWNG</th>
<th>Ind: NQB</th>
<th>Plug: GL QZ FV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>9A</strong></td>
<td>ZVPFWWZWIAUMZOJWTNQOUQQDWSQVYEKUPINCULBBAYEIXOHZXBOIFXHNSPICFPJIPGISQUFHVQKFST</td>
<td>BEWHOYOUAREANDSAYWHATYOUFEELBECAUSETHESEWOHHOMINDDONTMATTERANDTHOSEWHO MATTERDONTMIND</td>
</tr>
<tr>
<td><strong>9B</strong></td>
<td>ZVQLWUXSEZALXIGYCFVRKKELGIXOCJIZPS</td>
<td>BEYOURSELFEVERYONEELSEISALREADYTAKEN</td>
</tr>
<tr>
<td><strong>10A</strong></td>
<td>GSCBZITNMDQBTNWyBHIZAGBCTPSZCMSAMQKJAFHTXAPXCUFFYCPCOMCDNZZWJWNBWHPDRLJCBWNHNXCFNAXCFMFIEJVTXQCMNOVCHLDUGOBRRAREMZZUKWA</td>
<td>YOUVEGOTADANCERYLIKETHERESNOBODYDYWATCHINGLOVELIKEYOUULLNEVERBEHURTSLIKETHESNOBODYLISTENINGANDLIVELIKEITSHAVERNONEARTH</td>
</tr>
<tr>
<td><strong>10B</strong></td>
<td>GSCJCKHWNHRHTNWOREEDDAYGXI DHIDYVKUFJORXTNHW</td>
<td>YOUONLYLIVEONCEBUTIFYOUDOITRIIGHTONCEISENOUGH</td>
</tr>
<tr>
<td>Page</td>
<td>Text</td>
<td>QX</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>----</td>
</tr>
<tr>
<td>11A</td>
<td>AOQDZJRBFBRHOEFOEVEYWJ JRLBXR ZHCRMSVZRQSGFANXYJCYT CTAZPW WCMPOVCUDTBEZSORYWK ZGP</td>
<td>TWO THINGS ARE INFINITE THE UNIVERSE AND HUMAN STUPIDITY AND IM NOT SURE ABOUT THE UNIVERSE QMLAGREEAT ASDSLADONIF YCBPENSNYD RAWISEUNGD BEGRIESSWIS FESEBRUWSI ARANTHINEES KSU</td>
</tr>
<tr>
<td>11B</td>
<td>TKQUOKBNGYSDZTADQEYV DLRENGCR MJORMWNLXNXK</td>
<td>A ROOM WITHOUT BODY IS LIKE A SOUL DYAKEQOFK WNSITELLOR KTOASSTO PSESTOBSE</td>
</tr>
<tr>
<td>12A</td>
<td>DHBXTOUNVIXBTLXQUIXILV ZRIZFJ HGVHRLKTQBYFNKUIWBDG MYDMLPJ ZUELQGDXOOPCYWYCEEC QFESIRXF AWCSKI</td>
<td>FRIENDSHIP IS BORN AT THAT MOMENT WHEN ONE PERSON SAYS STO ANOTHER WHAT YOUOOOTHUGHT THAT NO ONE BUT MY SELF ACCURRAWKY SHALTUFESK TEOSWGEV OATSTAFDL AWINKGRINLO NCKISHMATO EEDIGERANS UAHCELDUPLT YCTHFL</td>
</tr>
<tr>
<td>12B</td>
<td>FWTOHQXGWJEUDIYAASWL LYCVAXA UJVAYPCBNQALYFQDZWQTN YIMMJO TUJDFPCVZXOVQXSMQCBBA FQNYMJ AAXDMPR</td>
<td>DONT WALK BEHIND ME IM NOT LEADING DONT WALK IN FRONT OF ME IM NOT FOLLOWING JUST WALK BEHIND ME AND BE MY FRIEND VVCERYATS WRQNIQUHUA OLYEMMFENG OHIDYRECSE UGEPAROASPUS RYHOPATEAT SADAYISRFTR MYNAUSUITEW IOZYS</td>
</tr>
<tr>
<td>13A</td>
<td>LUMMZMIKBHYGVPOZJFDA TIZNRFH XFKENVQGHUZDUQPEVNPUC MHLMLXWF QJYSLOBTPYMAWUCNWWY WAEXVFQFFN</td>
<td>IFYOUUXWANTXTOX KNOWXWHATXAXMA NSXLIKEEXTAKEAXG OOXLOOKXATXHO WXHEXTREATSXHIS XINFERIORSNOTX NACEALFECAN BWDEEHHRU BTAMPIIZDRI PMOFOXSLKP EONDISHSAT EGAPLVQUM</td>
</tr>
</tbody>
</table>
6.20.7.8 Website Links
The website links are currently operational, although there’s little of interest on the associated pages. Future versions of the test will be modified to include checking the website page content.

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.1</td>
<td>Verify that all links on the website are operational.</td>
<td>Pass</td>
</tr>
</tbody>
</table>

6.20.8 Evaluation
The current version of the GUI cannot yet be considered “stable” due to its problems with non-default configuration encryptions. Although the current problems prevent correct encryption, the GUI itself remains functional. Input is now stripped of invalid characters as it is supposed to, and the GUI prevents the selection of invalid configurations such as using the same rotor twice, error checking features that were not available last week.

6.20.9 Summary of Activities
The holiday interfered with testing activities somewhat because the Test Lead has a family and they occasionally like to be reminded that she’s not dead. Despite that, in the first half of the week the System Test Procedures were finalized, and thorough testing was conducted in the latter half of the week on all functions except the Cryptanalysis tests, which are still in progress.

6.20.10 Approvals
6.21 Final Testing Results

The full GUI and website have now been deemed acceptable for submission, as detailed below.

By final submission, all components of the primary GUI pass. This includes new stretch features added, such as the Machine Version dropdown, reset and text clear buttons, and output space options. The plugboard is now fully operational, and all known problems with it have been addressed. Ring position indicators update as expected whether using the manual input box, bulk encryption, or file upload. There are two known special cases that can cause deviation from the Enigma’s expected behavior:

1. When using the Enigma I machine version, the user is restricted to using the Rotors in the order of I, II, III. Enigma I only has three rotors available, and attempting to change one of the rotors (say, making the far right rotor into Rotor I) triggers the duplication prevention error condition, which then resets the rotor. This is a minor issue, and the GUI is considered shippable.

2. When using the file upload function, certain special characters (notably, black ascii heart) get translated into symbol-letter combinations that can throw off encryption, such as “e&”. In this case, the “e” would be encrypted, which would then result in future letters being encrypted “one letter off”. Experimentation with multiple character sets while reading the files could not correct this problem. It was eventually concluded that, due to the fact that the Enigma was never expected to encrypt characters such as “black ascii heart,” it is acceptable to ship the program with the notation that non-keyboard characters may occasionally produce unexpected results.

The Cryptanalysis portion of the GUI does not produce an accurate decryption every time, nor was it ever expected to. It has shown itself to be highly reliable in all unsteckered (non-plugboard) test cases, and reasonably reliable in low-steckered cases when given starting constraints for the other key components. Adding constraints (simulating the capture of a partial key) reduces the time of analysis. All features of the Cryptanalysis GUI work as expected, and the GUI has been deemed acceptable.

The GUI has been uploaded to the web as an applet, and is verified to work as expected there. Because the applet is self-signed, Java’s built-in security features require the user to grant permission. The applet, like the app, requires Java 7.0. It has been tested with Chrome, Firefox, and IE, and has been checked across multiple computers. Due to licensing issues between Oracle and Google, it does not work on Android devices, and does not work on Apple mobile devices because Apple lives in its own little world. It has not been verified to work on Apple desktop environments, and is unlikely to work on any Mac running Chrome due to the previously mentioned licensing conflicts.

The links in the remainder of the website have been tested and verified to work. The Users Guide has been embedded in one of the website pages and is readable by anyone with a PDF plug-in for their browser. Downloads of both the Users Guide and the full stand-alone App are available on the resources page.

6.21.1 Approvals
6.22 Glossary

See section 1.1.5.

7 Development Phase I Progress Report

7.1 Phase I Milestone Status

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and Submit Phase 1 Documentation</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Demonstrable Rotor</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Rotor Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Rotor Sign-Off Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Demonstrable Reflector</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Reflector Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Reflector Sign-Off Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Demonstrable Stepping Mechanism</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Stepping Mechanism Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Stepping Mechanism Sign-Off Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Block-out UI for current code</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Block-out UI Verified</td>
<td>Complete</td>
</tr>
</tbody>
</table>
Review: Block-out UI Sign-Off Complete  Complete
Design: UI Wireframes Complete  Complete
Design: UI Art Direction in Progress  Complete

7.2 Progress Updates

Included in the code submitted this week is the completed back-end of the Enigma Machine. The components have undergone unit testing as detailed in the Enigma Unit Test Report. The GUI is only partially functional at this point. The user can adjust the settings as desired, but full input validation is not yet completed. Text can be entered in the input box, and will be encrypted according to the chosen settings when the “Encrypt” key is selected, but the lightboard and letter-by-letter processing is not yet available. Although the “Upload File” button allows the user to choose a file, due to a known bug the GUI will only encrypt the last word of any text file that contains spaces. Then JUnit tests used in verifying the component classes are being included in the submitted code, as is an executable Jar file of the current GUI (due to the fact that in-browser operation has not yet been added).

1. Over the course of the project thus far, the team has accomplished the following tasks, including but not limited to the initial planned milestones:
2. Completion of Rotor, Reflector, Stepping Mechanism, Ring Setting, Rotor Position, and Plugboard back-end code.
3. Completion of block-out UI for testing purposes.
   3.1. Four rotor selection drop-downs implemented, with preventative code that will not allow the user to set a fourth ring setting or rotor position if the fourth rotor is not chosen.
   3.2. Four ring setting drop-downs implemented.
   3.3. Four rotor position settings implemented.
   3.4. Reflector drop-down implemented.
   3.5. Twenty-six plugboard drop-downs implemented, with preventative code which will enforce 1:1 selections, blocking the user from sending invalid input.
   3.6. Text input and output boxes implemented.
   3.7. File input selector implemented, with known bugs. See section 7.3. Problems Encountered.
   3.8. Encryption button implemented, with both four and three rotor options accepted.
4. Exploration into UI art functionality using JLayeredPanes to get away from the traditional outdated look of Java Swing.
5. Exploration of cryptanalysis stretch-goal techniques.
   5.1. Quadgram statistical analyzer (wheel order, ring and indicator settings, currently) in progress.
   5.2. Received permission from James Lyon, owner of practicalcryptography.com, to use his Quadgram algorithm ideas in our project.

The team has also set up several tools for project management, including:
1. Trello boards for task and resource tracking.
2. GitHub for code collaboration.
3. Drone.io for automated build deployment.
5. Google Mailing List for announcements/broadcasts.
7. GoDaddy SQL Database for DB experimentation.
8. Google Shared Calendar for schedule coordination.

7. 3 Problems Encountered
The team encountered the following problems:

1. Additional work was discovered during the planning and design phase which increased
the complexity of the project. Much of this complexity related to error checking and end
user experience. Those items which could be added as stretch goals were added. Those
which were essential for Quality Assurance were added to the project plan. These items
included.
   1.1. Plug Board selection parity - In order to prevent user frustration if the same character
       is mapped two different characters on the plugboard by mistake, we realized we
       must both enforce this in the GUI, and add an exception on the back end.
   1.2. Java Swing “ugliness” - Giving our GUI a more contemporary feel proved to be a
       major challenge using Java Swing. The solution we found was to use
       JLayeredPanels to add a background image and “animated” icons to the GUI, to fake
       the appearance of a more elegant GUI solution.
   1.3. Historical rotor selection enforcement - The team has been discussing forced rotor
       setting configurations to prevent the end user from selecting rotor combinations that
       would not be possible on the original machine. The devised solution is a drop-down
       menu which will automatically set the machine to match the configuration of a
       particular machine version. This has been added as a stretch goal.
2. During a team meeting, it was discovered that we were not all on the same page
regarding the functionality of the ring setting. Documentation we were using for reference
was spotty, and this led to confusion. During the meeting we explained the functionality
and shared references which were more informative. Design documents were updated
to reflect this discovery.
3. During Unit Testing, the Test Lead uncovered problems with the system’s stepping
mechanism and rotor set-up. Configurations intended to check the machine’s ability to
step three rotors at once yielded incorrect results. The bug was reported and quickly
addressed by the Coding Lead.
4. During Unit Testing, the Test Lead uncovered a problem with single-character encryption
vs String encryption. String encryption yielded correct results, but character encryption
did not. This proved to be a logic error in the test itself and was quickly addressed.
5. During Systems testing for the GUI an issue was discovered with txt file encryption. Only
the final word in a txt file with spaces was encrypted. This, and other input error checking
including capitalization, number to letter parity, and invalid characters will be investigated in Phase 2.

### 7.4 Decision Reevaluations

The team has made the following changes to original design and project plan/schedule:

1. Cryptanalysis stretch goals were pulled forward when it was discovered that the basic machine back-end could be completed fairly quickly.
2. Stretch goals were added to the GUI plan, including:
   2.1. Additional error checking. The GUI will prevent the user from selecting invalid machine options, instead of allowing the passage of bad data to the back-end.
   2.2. Machine Version (Historical version) drop-down. The GUI will have a drop-down which can be used to set the machine options to those available in a specific machine model.
   2.3. Settings reset button. This will allow the user to entirely reset the machine to default settings.
   2.4. Spacing/Text Block Drop-Down. This drop-down will be used to specify the location of spaces in the encrypted output.

### 7.5 Documents Revised from the Beginning of the Project

The following documents have been revised:

1. Enigma Machine Software Design Document
2. Enigma Mechanical Technical Specification
3. Enigma Machine Project Plan
4. Enigma Machine Users Guide
5. Enigma machine Testing Plan

### 7.6 Appendices

Links, attached documents, other information

References for quadgram stat. analysis:


### 8 Development Phase II Progress
## 8.1 Phase II Milestone Status

### Deliverable Schedule

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and submit a draft Project Plan. Language, IDE, Collaboration, and primary design objectives defined.</td>
<td>November 3, 2013</td>
</tr>
<tr>
<td>Compile and submit draft Design documentation.</td>
<td>November 17, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 1 Documentation. Demonstrable Rotor, Reflector, and Stepping Mechanism code complete, reviewed, and tested. UI Wireframes and Block-out UI in progress. Ring Setting and Plugboard code complete.</td>
<td>November 24, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 3 Documentation. Demonstrable decryption/encryption, Cryptanalysis, and machine versioning code complete reviewed, and tested. UI Art complete and reviewed. Stretch goal progress evaluated.</td>
<td>December 8, 2013</td>
</tr>
<tr>
<td>Compile and submit final Project code and Documentation. Stretch goals implemented only if reviewed and tested. Successful and unsuccessful investigations documented.</td>
<td>December 15, 2013</td>
</tr>
</tbody>
</table>

### Phase II Goal Status

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and Submit Phase 2 Documentation</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Demonstrable Ring Settings</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Ring Settings Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Ring Setting Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Demonstrable Plug Board</td>
<td>Complete</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Testing: Plug Board Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Plug Board Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Demonstrable Light Board</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Light Board Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Light Board Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Text File Entry</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Text File Entry Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Text File Entry Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Text Entry Error Checking</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Error Checking Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Error Checking Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Block Out (Original) GUI</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Block Out GUI Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Block Out GUI Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Design: Final GUI Style Selected</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Final GUI Style In Progress</td>
<td>On Track</td>
</tr>
<tr>
<td>Code: Cryptanalysis Code In Progress</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Cryptanalysis GUI In Progress</td>
<td>On Track</td>
</tr>
<tr>
<td>Testing: Cryptanalysis In Testing</td>
<td>On Track</td>
</tr>
</tbody>
</table>

### 8.2 Progress Updates

#### 8.2.1 Introduction and Summary

At this point in the project a first draft (block-out) UI has been created which can be used to test all existing machine features. A final UI design is in progress, and is in a comparable state to the block-out design. The components of this new design currently reside in a separate “New GUI” directory for verification. Cryptanalysis code is 90% complete, with small features and bug-fixing remaining. Cryptanalysis GUI is also in-progress, and a design has been decided upon. The Cryptanalysis GUI will use the same window as the machine GUI, and components
will swap or gray out depending on which UI is chosen by the user.

### 8.2.2 Accomplishments

#### 8.2.2.1 Enigma Machine
1. Basic functionality is implemented for the Enigma machine rotor settings, ring settings, rotor positions, reflector settings, plug board, light board, text box input, character input, and file input.

#### 8.2.2.2 GUI
1. Final UI design has been decided upon, and is in progress. The new design will break the program up into separate files instead of the single monolithic file design used in the block-out class.
2. Both the Block-out and New GUI now have a working Light Board. Current functionality uses the last character from an encrypted string or character input to light up the appropriate light on the board.
3. The Block-out GUI now accepts and encrypts/decrypts a full text file properly. The earlier bug which encrypted only the final word in a file has been resolved.
4. A new plugboard design has been implemented and is in testing. This new design draws links between selected buttons, and is more aesthetically impressive than the original drop-down box design.
5. The new GUI adds per-character encryption in addition to the string text box.
6. The new and old GUI both verify that a user is selecting the proper reflector for the number of rotors chosen. They also verify that a user can not pick two of the same rotor.
7. Both the old and new GUI will update the position of the rotors in real-time during encryption.

#### 8.2.2.3 Cryptanalysis
1. An internal tool was created to aid in the generation of large numbers of messages all encrypted with the same key. These messages can be used in decryption tests.
2. A redesign was made to be able to make use of multiple threads in a thread pool, with user-selected sizes.
3. Full decryption testing using n-gram statistical analysis is implemented, with options for the user to select settings that affect the runtime.
4. Testing different statistical tests for best approach to each phase of the decryption.
5. A tool to generate a corpus file has been created, and the official corpus will be constructed.
6. Currently, the most accurate decryption algorithm consistently decrypts and predicts machine settings for English encrypted messages that do not make use of the plugboard and assumes 3-rotor Enigma machines, but do use any available rotor and reflector combination. Larger messages with fewer plugboard pairs can also be decrypted.

#### 8.2.2.4 Web Page
1. The web page has been purchased and a basic home page has been set up. The address is http://www.enigmamachine.org. The page can also be reached from http://www.enigmamachine.info.

#### 8.2.2.5 Documentation
1. The following sections in the Enigma Project Plan have been updated:
1.1. Enigma Project Plan is updated with a schedule that includes the Light Board in Phase II.
1.2. The goal lists in section 1.1 has been updated to include stretch goals that have been added.
1.3. The User Requirements Specification has been updated to reference Java 7.0 rather than 6.0.

2. The following updates have been made to the User’s Guide:
   2.1. The introduction has been updated to reference Java 7.0. The standalone Jar file is now referenced as an alternative to browser operation.
   2.2. The Procedures section has been modified to reflect the new GUI layout.

3. The Enigma Machine System Test Procedures and report have been written, and are to be used as internal documents to track progress. Summaries of the results will be included in the progress reports.

8.2.2.6 Tools Used
   The team has also set up several tools for project management, including:
   1. Trello boards for task and resource tracking.
   2. GitHub for code collaboration.
   3. Drone.io for automated build deployment.
   5. Google Mailing List for announcements/broadcasts.
   7. GoDaddy SQL Database for DB experimentation.
   8. Google Shared Calendar for schedule coordination.

8.3 Problems Encountered

8.3.1 Enigma Machine
   1. The original Enigma Machine back-end did not include a method for returning rotor positions to the GUI for live updating. This method has been added.

8.3.2 GUI
   1. The original GUI design was an extremely large file with all components and processes in a single file. The final design (in progress) will break the UI up into multiple classes to make the code easier to modify and maintain.
   2. The new design will not be able to use the JLayeredPanels in the same way as the original design, so the background has been removed and replaced with a plain black background.
   3. A work-around had to be implemented in order to retain the current Light Board functionality, which still uses JLayeredPanels.
   4. With such an integral portion of the code in-flux, team members often worked on the same section of code simultaneously. We communicated changes via Trello to avoid stepping on each other’s work.
   5. During initial testing of the new GUI, it was discovered that the GUI would only encrypt as
though the user had left the key at the default settings. This was a problem that had arisen during the GUI transfer, and was quickly ironed out.

6. After the “default settings” bug was addressed, it was discovered that the GUI would not properly encrypt text when the key included a plugboard. This problem has been corrected.

7. During initial testing of the GUI, the tester discovered that when the bulk encryption option is used, the GUI does not update the rotor positions to the actual machine settings. It still encrypts the message correctly, but subsequent encryptions are off if the user goes by the displayed rotor position without manually adjusting the rotor positions. The troubleshooting for this problem is still in progress.

8.3.3 Cryptanalysis

1. Cryptanalysis testing was found to require large numbers of encrypted messages, often encrypted using a shared key. To address this, a test tool was created that would run multiple encryptions at once on an uploaded file.

2. Messages with a high number of plugboard pairs are proving difficult to adequately decrypt, usually over six plugboard pairs with messages approximately 200 characters or more.

3. Full Cryptanalysis tests using test cases that include a variety of features (the full range of rotors, reflectors, etc) have been found to take an extremely long time on the Test Lead’s computer, although they usually yield accurate decryptions when complete. This is not shocking, giving the very large number of potential Enigma key combinations. This issue is being addressed in two ways:
   3.1. during the testing of a specific feature (say, plugboards), options for an alternative feature are often limited (for example, by limiting the rotors to I-III).
   3.2. The Code Lead has been running many of the tests to aid the Test Lead, because his computer works much faster (five minutes vs an hour or more).

4. Running decryption tests involving tests for four-rotor combinations has been shown to be prohibitively long, and may crash the JVM. Using rotor and reflector constraints has alleviated some of the time required, but can still take a long time, even on modern computers running multiple threads.

8.3.4 Web Page

1. No issues at this stage in the project.

8.3.5 Documentation

1. Although our documentation specifies Java 6.0, the team has been coding to a Java 7.0 standard. The team has chosen to update the documentation rather than rewrite the code, and changing references from “Java 6.0” to “Java 7.0” is in progress.

8.4 Decision Reevaluations

1. During a weekly meeting, it was determined that the GUI should be split into separate class files. Because of this, the jLayeredPane concept for a background has been abandoned.

2. The team originally intended to produce the project using Java 6.0 in order to make the results available to a wider audience. Due to the convenience of features available in
Java 7.0 but not in 6.0 and the ease in upgrading for the home user, the team eventually decided to use the most up to date Java version in their work instead.

8.5 Planned Accomplishments for Next Week

Next week, we will submit Phase III documentation, complete the final GUI, and include Cryptanalysis code with Encryption/Decryption. We will also implement a stretch goal which will add a machine version drop-down for historical machine selection.

8.6 Appendices

Please reference our technical documentation for a complete understanding of the Enigma Machine components.

References for Cryptanalysis:
http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/

doi:10.1080/0161-110091888745

Corpus texts - http://www.gutenberg.org/

9 Development Phase III Progress Report

9.1 Phase III Milestone Status

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and submit a draft Project Plan. Language, IDE, Collaboration, and primary design objectives defined.</td>
<td>November 3, 2013</td>
</tr>
<tr>
<td>Compile and submit draft Design documentation.</td>
<td>November 17, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 1 Documentation. Demonstrable Rotor, Reflector, and Stepping Mechanism code complete, reviewed, and tested. UI Wireframes and Block-out UI in progress. Ring</td>
<td>November 24, 2013</td>
</tr>
</tbody>
</table>
Setting and Plugboard code complete.

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and submit Phase 3 Documentation. Demonstrable decryption/encryption, Cryptanalysis, and machine versioning code complete, reviewed, and tested. UI Art complete and reviewed. Stretch goal progress evaluated.</td>
<td>December 8, 2013</td>
</tr>
<tr>
<td>Compile and submit final Project code and Documentation. Stretch goals implemented only if reviewed and tested. Successful and unsuccessful investigations documented.</td>
<td>December 15, 2013</td>
</tr>
</tbody>
</table>

**Phase III Goal Status**

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and Submit Phase 3 Documentation</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: New Machine GUI Functional</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: New Machine GUI verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: New Machine GUI sign-off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Cryptanalysis GUI Implemented</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Cryptanalysis GUI Verified</td>
<td>In progress / Not at risk</td>
</tr>
<tr>
<td>Review: Cryptanalysis GUI Sign-off</td>
<td>In progress / Not at risk</td>
</tr>
<tr>
<td>Code: Cryptanalysis Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Cryptanalysis Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Cryptanalysis Sign-off</td>
<td>In progress / Not at risk</td>
</tr>
<tr>
<td>Code: Stretch: Machine Type Implemented</td>
<td>Incomplete / Not at risk</td>
</tr>
<tr>
<td>Testing: Stretch: Machine Type Verified</td>
<td>Not started</td>
</tr>
<tr>
<td>Review:Stretch: Machine Type Sign-off</td>
<td>Not started</td>
</tr>
<tr>
<td>Code: Stretch: Spacing Selection Implemented</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Stretch: Spacing Selection Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Stretch: Spacing Selection Sign-off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Stretch: Reset Button Implemented</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Stretch: Reset Button Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Stretch: Reset Button Sign-Off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: GUI Art Uniform and complete</td>
<td>Complete</td>
</tr>
</tbody>
</table>

### Roles / Task Assignments

<table>
<thead>
<tr>
<th>Task</th>
<th>Team Member</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation Reorganization</td>
<td>Ellen Ohlmacher</td>
<td>Complete</td>
</tr>
<tr>
<td>Phase III Documentation</td>
<td>Team Enigma</td>
<td>Complete</td>
</tr>
<tr>
<td>Add Cryptanalysis GUI to Main GUI</td>
<td>Bryan Winstead</td>
<td>Complete</td>
</tr>
<tr>
<td>Adjust Artistic Feel of Cryptanalysis GUI</td>
<td>Ellen Ohlmacher</td>
<td>Complete</td>
</tr>
<tr>
<td>Hook up back-end CA code to CA GUI</td>
<td>Walter Adolph</td>
<td>Complete</td>
</tr>
<tr>
<td>Complete Cryptanalysis Code</td>
<td>Walter Adolph</td>
<td>Complete</td>
</tr>
<tr>
<td>Implement Machine Type Selection</td>
<td>Rosana Montanez / Bryan Winstead</td>
<td>Incomplete</td>
</tr>
<tr>
<td>Implement Reset Buttons</td>
<td>Jessica Ikley</td>
<td>Complete</td>
</tr>
<tr>
<td>Implement Spacing Options</td>
<td>Jessica Ikley</td>
<td>Complete</td>
</tr>
<tr>
<td>Retire Block-out GUI</td>
<td>Ellen Ohlmacher</td>
<td>Complete</td>
</tr>
<tr>
<td>Unit &amp; Systems Testing</td>
<td>Team Enigma</td>
<td>Complete</td>
</tr>
<tr>
<td>Bug Fixing &amp; Remediation</td>
<td>Team Enigma</td>
<td>Complete</td>
</tr>
</tbody>
</table>
9.2 Progress Updates

9.2.1 Introduction and Summary
This week’s objective was to wrap-up feature implementation, begin initial implementation of stretch goals, and prepare all documentation for final submission. The team will have all major implementation complete by the end of the week, in order to use the final week for bug fixing and polish.

Three stretch goals were added to the project due to early completion of the basic machine functionality and cryptanalysis. These goals are:

1. A drop-down option to select a specific historical version of the Enigma Machine. This will restrict the user’s HUI options to only those which would be available on that version of the machine.
2. A drop-down option to set the location of spaces in the output text. Research indicated that operators typically added a space after every four characters, five characters, or no spaces at all. An additional option for original spacing is also under consideration, in spite of the security risks to leaving spaces unadjusted. (Easier decryption because word length is exposed.)
3. A series of buttons for resetting the main GUI options:
   3.1. "Default Config" - Acts like the reset button on the page linked. Clears everything to the default configuration.
   3.2. "Reset Indicators" - Sets the indicators to their initial configuration for the machine being used, clears out the text boxes. Useful for creating multiple messages using the same key and the same indicators.
   3.3. "Clear Text" - Leaves the machine alone, clears the text boxes in case they have become cluttered.

9.2.2 Accomplishments

9.2.2.1 Enigma Machine
1. The Enigma Machine back-end was completed before the start of this phase.

9.2.2.2 GUI
1. The team spent the early part of the week fixing existing bugs caused by GitHub growing pains. We restored functionality to the New GUI, and added Light Box and text error checking to all input types.
2. The Cryptanalysis GUI was attached to the New Machine GUI. The art style was adjusted to match our existing style, and the Cryptanalysis back-end was hooked up to it.
3. The Machine Type (Historical Version) selection feature has been implemented for the back-end, and is in progress for the GUI/front end.
4. Stretch goal features to reset the GUI selections and determine where spacing will be in the output are complete, tested, and bugs have been fixed.
5. The older block-out GUI has been retired for this version. Unused files have been stripped from the code base. All references to unused files have been cleaned up.
9.2.2.3 Cryptanalysis
1. Implemented options for the user to limit what settings to test for. If part of the machine settings are known, the user can direct the cryptanalysis engine to restrict decryption attempts to those settings provided, and try to figure out the rest. This has the added benefit of greatly speeding up the decryption process, assuming the constraints are correct.
2. An expanded word database was generated, with approximately 132 million n-grams and 8 million words are included, with frequency counts, pulled from classical English texts in open domain.
3. The cryptanalysis engine now has the capability to test for four-rotor configurations.
   a. Note: Most decryption attempts for four-rotor encryption takes an extremely long time, if not constrained.
4. A word parser was added to the end of the decryption engine to input spaces between discovered words. There was an issue with excessive spacing, so the feature is currently disabled pending a fix.
5. Testing with multiple statistics was conducted, with the decision to continue using quadgram frequency counts.

9.2.2.4 Web Page
1. SSH has been configured for automatic update deployment to the website.

9.2.2.5 Documentation
1. This week, all existing documentation for submission has been transferred to a single document per feedback from Dr. Duchon.
3. The user’s guide has been updated to include the new GUI features.
4. UML diagrams were added to the Software Design Document, however a final re-write of the document will be completed for final turn-in.
5. Java Docs have been added to the project, as well as automated UML diagrams.

9.2.2.6 Tools Used
The team has set up several tools for project management, including:
1. Trello boards for task and resource tracking.
2. GitHub for code collaboration.
3. Drone.io for automated build deployment.
5. Google Mailing List for announcements/broadcasts.
7. GoDaddy SQL Database for DB experimentation.
8. Google Shared Calendar for schedule coordination.
10. WinMerge for recovering functioning code when GitHub decides to smite us.
9.3 Problems Encountered

9.3.1 Enigma Machine
1. The original Enigma Machine was not properly encrypting lowercase letters with a plugboard when using the encryptChar method. This has been corrected.

9.3.2 GUI
1. Breaking the GUI components up into smaller more digestible files has been a challenge for the team, and maintaining this standard is difficult due to the nature of Swing. The team is working together and performing code reviews to maintain the standard.
2. The program was asserting if no input or invalid input is entered. This was bugged and fixed.
3. An issue was discovered where the reset buttons did not function properly with the fourth rotor. This was bugged and fixed.
4. The spacing drop-down option was initially not functioning with file input. New lines were added to the output text. This has been bugged and resolved.
5. The machine type drop down was implemented on the back-end, but was not integrated into the GUI front-end due to a series of unfortunate personal events. The team believes this stretch feature is not at risk for completion by the final week.

9.3.3 Cryptanalysis
1. Implementing the constraints settings necessitated a redesign of the cryptanalysis engine, as well as extensive testing to ensure proper operation.
2. An issue with the word detector was discovered, where excessive spacing occurred. The issue is with how the parser enters the spaces. A fix is identified, and will be in the final version.

9.3.4 Web Page
1. No web page problems at this time.

9.3.4 Documentation
1. Due to conversion between Microsoft Word and Google Docs, many of the existing documents had wildly different formatting and odd formatting errors. Maintaining clean and cohesive documents when the tools are automatically re-formatting content has been frustrating for the team.

9.3.5 Other Issues
1. There were growing pains as many members of the team were new to GitHub and unfamiliar with it’s quirks. Some changes were accidentally removed as team members struggled to resolve conflicts after refactoring. This was a learning experience, and all the bugs have been resolved. The team has made decisions as a group to maintain private copies of the code base for local work, and only merge after ensuring the
branched code is the latest version. WinMerge has been used to resolve initial issues.

9.4 Decision Reevaluations
1. The reset button stretch goal was re-evaluated after it was revealed that different team mates had different concepts of its functionality. The new design will have three options: resetting the entire program, only resetting the machine settings to initial user inputs, or only clearing the text.

9.5 Planned Accomplishments for Next Week
1. As next week is the final week, all code must be unit, system, and code standard tested and bugs must be fixed.
2. The program and User’s guide will be packaged and uploaded to the website for public consumption.
3. All documentation will be proofread, brought up-to-date, and prepared for submission.
4. Code will be cleaned of unused files and methods, and headers/comments will be proofread and prepared for submission.
5. Outstanding stretch goal completion must be wrapped up or abandoned before the end of the week. The team will make the call on any incomplete or unacceptable stretch goals.

9.6 Appendices
Please reference our technical documentation for a complete understanding of the Enigma Machine components.

References for Cryptanalysis:
http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/
doi:10.1080/0161-110091888745
Corpus texts - http://www.gutenberg.org/

10 Development Phase IV Progress Report
## 10.1 Phase IV Milestone Status

### Deliverable Schedule

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and submit a draft Project Plan. Language, IDE, Collaboration, and primary design objectives defined.</td>
<td>November 3, 2013</td>
</tr>
<tr>
<td>Compile and submit draft Design documentation.</td>
<td>November 17, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 1 Documentation. Demonstrable Rotor, Reflector, and Stepping Mechanism code complete, reviewed, and tested. UI Wireframes and Block-out UI in progress. Ring Setting and Plugboard code complete.</td>
<td>November 24, 2013</td>
</tr>
<tr>
<td>Compile and submit Phase 3 Documentation. Demonstrable decryption/encryption, Cryptanalysis, and machine versioning code complete, reviewed, and tested. UI Art complete and reviewed. Stretch goal progress evaluated.</td>
<td>December 8, 2013</td>
</tr>
<tr>
<td>Compile and submit final Project code and Documentation. Stretch goals implemented only if reviewed and tested. Successful and unsuccessful investigations documented.</td>
<td>December 15, 2013</td>
</tr>
</tbody>
</table>

### Phase IV Goal Status

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile and Submit Final Documentation</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Cryptanalysis Polish Complete</td>
<td>Complete</td>
</tr>
<tr>
<td>Testing: Cryptanalysis Polish Verified</td>
<td>Complete</td>
</tr>
<tr>
<td>Review: Cryptanalysis Polish Sign-off</td>
<td>Complete</td>
</tr>
<tr>
<td>Code: Stretch: Machine Type Implemented</td>
<td>Complete</td>
</tr>
<tr>
<td>Task</td>
<td>Team Member</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Testing: Stretch: Machine Type Verified</td>
<td></td>
</tr>
<tr>
<td>Review: Stretch: Machine Type Sign-off</td>
<td></td>
</tr>
<tr>
<td>Code: Java Doc Header/Comments</td>
<td></td>
</tr>
<tr>
<td>Testing: Header/Comments Proofread</td>
<td></td>
</tr>
<tr>
<td>Review: Header/Comments Sign-off</td>
<td></td>
</tr>
<tr>
<td>Code: Website Implemented</td>
<td></td>
</tr>
<tr>
<td>Testing: Website Verified</td>
<td></td>
</tr>
<tr>
<td>Review: Website Sign-Off</td>
<td></td>
</tr>
<tr>
<td>Testing: All Existing Code Tested</td>
<td></td>
</tr>
<tr>
<td>Review: All Known Bugs Fixed &amp; Closed</td>
<td></td>
</tr>
</tbody>
</table>

### Roles / Task Assignments

<table>
<thead>
<tr>
<th>Task</th>
<th>Team Member</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final Documentation Proofreading &amp; Revision</td>
<td>Ellen Ohlmacher</td>
<td>Complete</td>
</tr>
<tr>
<td>Addition of Error Messages to Code</td>
<td>Ellen Ohlmacher</td>
<td>Complete</td>
</tr>
<tr>
<td>Phase IV Documentation</td>
<td>Team Enigma</td>
<td>Complete</td>
</tr>
<tr>
<td>Complete Machine Type Selection</td>
<td>Bryan Winstead</td>
<td>Complete</td>
</tr>
<tr>
<td>Applet added to Website</td>
<td>Bryan Winstead</td>
<td>Complete</td>
</tr>
<tr>
<td>Website &amp; “About” Section Update</td>
<td>Ellen Ohlmacher</td>
<td>Complete</td>
</tr>
<tr>
<td>Cryptanalysis Polish Tasks</td>
<td>Walter Adolph</td>
<td>Complete</td>
</tr>
<tr>
<td>Java File Clean-up &amp; Refactoring</td>
<td>Jessica Ikley</td>
<td>Complete</td>
</tr>
<tr>
<td>Header/Comments Completed &amp; Proofread</td>
<td>Jessica Ikley</td>
<td>Complete</td>
</tr>
<tr>
<td>Unit &amp; Systems Testing</td>
<td>Team Enigma</td>
<td>Complete</td>
</tr>
<tr>
<td>Bug Fixing &amp; Remediation</td>
<td>Team Enigma</td>
<td>Complete</td>
</tr>
</tbody>
</table>
10.2 Progress Updates

10.2.1 Introduction and Summary

The final phase of the project was primarily a bug fixing and polish week. Some systems had small outstanding features to complete such as the machine version selection GUI and web site applet. Otherwise, the team put all energy into bug fixing, polish tasks, and hardening. The project paper and codebase were proof-read, checked for style and accuracy, and revised where necessary.

10.2.2 Accomplishments

10.2.2.1 Enigma Machine

1. The Enigma Machine software was not altered between Phase III and Phase IV.

10.2.2.2 GUI

1. Error codes found in the User’s guide were added to the GUI. Some codes in the guide were no longer relevant, as the software was designed to police itself in some areas. Additional error codes were added to the code base but not the guide. These are now located in both places.
2. The machine version selection drop-down menu was added to the machine GUI. This adds restrictions to the rotor and reflector sections based on the options which were available in the selected historical machine version.
3. An original graphic was added to the GUI to give it a more unique appearance.
4. The Plug Board was revised so that connecting lines are aligned exactly with radial buttons, instead of offset.
5. The GUI was system tested and all bugs were resolved.

10.2.2.3 Cryptanalysis

1. The cryptanalysis packages were refactored in order to fix bugs found in phase three. Some files were not building with the Jar in previous versions. This is resolved.
2. The cryptanalysis GUI and code were thoroughly systems and unit tested, and any found bugs were fixed.
3. Helper text was expanded on for clarity and comprehensiveness.

10.2.2.4 Web Page

1. The applet has been added to the web page for public viewing.
2. The about page has been updated, and the User’s guide has been added for user reference.

10.2.2.5 Documentation

1. The entire project document has been revised for submission. Redundant sections now reference each other instead of repeating information. Sections which contained similar
information on the same subject have been combined. Formatting is standardized.

2. The Software Design section has undergone some revisions to cover more of the project software architecture. Previous documents involving the Enigma Machine technical specifications have been moved into their own section.

3. Testing results have been added to the testing section.

4. Phase four and conclusion sections have been added to the final document.

10.2.2.6 Tools Used
The team has set up several tools for project management, including:

12. Trello boards for task and resource tracking.
13. GitHub for code collaboration.
15. Google Docs for shared documentation.
17. GoDaddy web hosting for live web app.
18. GoDaddy SQL Database for DB experimentation.
19. Google Shared Calendar for schedule coordination.
20. Google Hangouts for weekly live meetings.
21. WinMerge for recovering functioning code when GitHub decides to smite us.
22. Drone.io for automatic build and deployment.

10.3 Problems Encountered

10.3.1 Enigma Machine
1. The Enigma Machine itself encountered no issues in Phase IV, as it received the most hardening of any systems in earlier phases.

10.3.2 GUI
1. Some sections of the GUI remain “monolithic” in part, as the team did not have time to complete all of the desired refactoring.
2. Many bugs were tracked and resolved in gitHub during systems testing. These bugs were primarily edge-case errors when testing with invalid input. Core functionality remained solid.

10.3.3 Cryptanalysis
1. The statistical analyzer was expanded to include testing for four-rotor machine configurations, in addition to the existing tests for three-rotor configurations.
2. A panel for the user to enter partial settings was incorporated as a way to speed up decryption times. These settings act as constraints, thus only tests with those settings would be run.
3. The “Check Cribs” button (defined in Section 10.5) had to be removed after an evaluation of remaining tasks and time at the beginning of Phase IV. This button was an interesting
concept, but did not appear to be as effective as the setting constraints option. As a result, it fell lower in priority than several bug fixes and polish tasks that the team felt were required for project submission.

4. Additional algorithms were researched early on in the project, but only the quadgram natural language statistical analysis could be completed in time for submission.

10.3.4 Web Page

1. The team encountered some small delay due to web hosting customer support slowness around the holidays and weekends. These issues have been resolved.

2. The web-hosted applet is currently self-signed, requiring users to grant permission to it in order for it to run. The team decided against using a certificate from a certifying authority because they cost money.

10.3.4 Documentation

1. The team initially developed separate documents instead of one single document. When the documents were combined, many sections were redundant and/or contained different material on the same subject. This required a reorganization of the documents between Phase III and Phase IV.

2. The Software Design Document initially contained a great deal of information about the Enigma Machine itself, and less information about the project design. This section had to be separated out and re-written for the current incarnation of the project software.

10.3.5 Other Issues

There were no additional issues in Phase IV.

10.4 Decision Reevaluations

At the beginning of Phase IV, the team evaluated remaining stretch goals and tabled those which were deemed lower priority than existing bug fixing and polish tasks. Those tasks which were abandoned during this process are listed under Section 10.5.

10.5 Incomplete or Abandoned Tasks and Goals

The following Stretch goals were abandoned at the beginning of Week 4 because they were deemed to be lower priority than remaining polish and bug fixing tasks.

1. Check Cribs Button (In Development) – The check cribs button can be used after a partial decrypt. The user may recognize some words in the decrypted output, and may wish to adjust settings to further decrypt the partial result. The output can be manually changed, and a crib check can be run to see if the rest of the message can be decrypted.

2. Enigma Machine sounds (Researched, but not implemented) – The team had initially discussed the addition of accurate machine sounds to the software. This stretch goal was never started, as it was triaged against other work and found to be lower priority.

3. Cryptanalysis text file entry and input error checking – As the second tab was deemed to be a stretch goal for use by more advanced users, it was not given the same
“hand-holding” error prevention and ease-of-use as the machine tab. Only those errors which would break program functionality are restricted.

4. Alternate Cryptanalysis algorithms – Though the team researched several different types of cryptanalysis, only the quadgram statistical language analysis could be implemented in time for project submission.

10.6 Appendices

Please reference Section 2 for a complete understanding of the Enigma Machine components, Section 3 for requirements specifications, section 4 for project software design, Section 1.1.4 for reference materials, and Section 1.1.5 for a glossary of terms.

11 Closing

11.1 Closing Remarks

Over the course of this project, all of the members of Team Enigma maintained highly professional attitudes towards the project contents, process, and schedule. Each member was able to self-organize and focus on his or her own tasks without losing sight of the larger picture. Communication was excellent across the board, with total transparency and frequent open discussion. Both the code base and documentation were standardized, and every member was able to adhere to all standards and policy. Every one of the planned goals and several of the stretch goals were implemented, tested, and polished. The team was able to actively course correct when faced with setbacks or design re-evaluations, leaving no essential work at-risk. Overall, this project was an excellent simulation of a real-world software project on a smaller scale, and Team Enigma worked together as an ideal and highly effective team. The experience has been invaluable in preparing the team for future employment.

11.1.1 Lessons Learned

Each member of the team brought a different skill set to the table, and others learned from that member’s experience. Though there were few failures to learn from, the team pushed themselves to leave their comfort zones and learn new development methods, tools, and code/documentation styles. A large part of the team’s success is due to the sharing of personal experience and members taking the initiative to thoroughly research any and all unknowns. Though there were many creative ideas put forth at the beginning of the project, the team was able to prioritize work required for project completion over that which was deemed to be non-essential.

11.1.2 Design Strengths

The Enigma Machine Project design is built for user friendliness. It is feature rich and includes error messaging and prevention. Though the machine itself is complex due to the nature of the subject matter, the User’s Guide is thorough and informative. Due to exhaustive testing, the team is confident that the results generated by the machine are historically accurate under all conditions.

11.1.3 Design Limitations

Due to the cost of code certifications, the web applet has been temporarily self-certified.
Because of upcoming web browser restrictions on self-certified code, the project will not remain conveniently displayed on the the front page of the website unless the team is willing to spend significant money to keep it displayed.

The cryptanalysis functionality can be slow and inaccurate depending upon the complexity of the entered text and settings. This could be improved with the incorporation of different analysis algorithms which were not in the scope of this project.

11.1.4 Suggestions for Future Improvement

Completion of tasks defined in Section 10.5 of this document would significantly improve the software functionality and feel. However, these goals were not within the scope of this project.