



INTRODUCTION TO SOFTWARE ENGINEERING

Morse Code Translator



Team Members:

Joel Bitri



Morse Code Translator Requirements Specification



Table of Contents

1. EXECUTIVE SUMMARY 3

1.1 PROJECT OVERVIEW 3 1.2 PURPOSE AND SCOPE OF THIS SPECIFICATION 3

2. PRODUCT/SERVICE DESCRIPTION 3

2.1 PRODUCT CONTEXT 3 2.2 USER CHARACTERISTICS 3 2.3 ASSUMPTIONS 3 2.4 CONSTRAINTS 3 2.5 DEPENDENCIES 4

3. REQUIREMENTS 4

3.1 FUNCTIONAL REQUIREMENTS 5 3.2 NON-FUNCTIONAL REQUIREMENTS 5 3.2.1 *User Interface Requirements* 5 3.2.2 *Usability* 5 3.2.3 *Performance* 6 3.2.4 *Manageability/Maintainability* 6 3.2.5 *Security* 8 3.2.6 *Standards Compliance* 8 3.2.7 *Other Non-Functional Requirements* 9 3.3 DOMAIN REQUIREMENTS 9

4. **DESIGN THINKING METHODOLOGIES** 10 4.1 Negotiation 11 4.2 Empathy 11 4.3 Noticing 11 4.4 GUI 11

5. **SOFTWARE DESIGN** 12 5.1 Use Case 12 5.2 State Diagram 12 5.3 Class Diagram 12

APPENDIX 13

APPENDIX A. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 13 APPENDIX B. REFERENCES 13 APPENDIX C. ORGANIZING THE REQUIREMENTS 13



1. Executive Summary

1.1 Project Overview

The project is a Python program that can translate Morse code to English text and vice versa. The intended audience for this project could be anyone who is interested in learning Morse code or needs to translate Morse code messages into English text or vice versa.

This program could be useful for hobbyists, students, or professionals who work in fields that require Morse code proficiency, such as radio operators, pilots, or military personnel. It could also be helpful for people who are studying or researching Morse code, as it provides a practical way to practice translating messages.

1.2 Purpose and Scope of this Specification

The purpose of these specifications is to define the requirements and functionalities of the Python program that can translate Morse code to English text and vice versa.

The scope of these specifications includes:

1. Creating a Morse code dictionary that maps Morse code characters to English alphabet letters and vice versa.
2. Building a Python program that prompts the user to input a message to be translated.
3. Designing the program to translate the input message, which could be in the form of Morse code or English text, into the corresponding output message, which could be in the form of English text or Morse code.
4. Implementing the program to handle different types of Morse code input, including dots, dashes, and spaces, and to output the corresponding English text.
5. Implementing the program to handle different types of English text input, including upper and lower case letters, and to output the corresponding Morse code.

The following are outside the scope of these specifications:

1. Designing a graphical user interface (GUI) for the program.
2. Integrating the program with other systems or applications.
3. Providing error handling for unexpected or invalid input.
4. Implementing advanced features, such as sound generation for Morse code output or real-time translation for audio signals.

2. Product/Service Description

In this section, describe the general factors that affect the product/service and its requirements. This section should contain background information, not state specific requirements (provide the reasons why certain specific requirements are later specified).

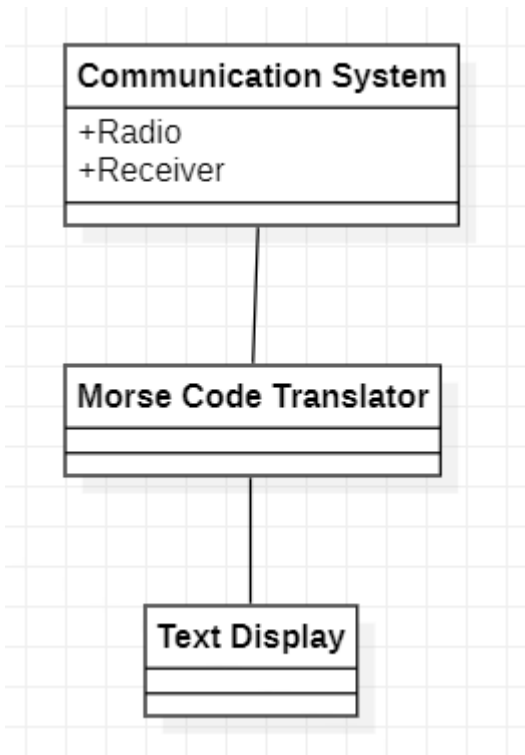


2.1 Product Context

Based on the specifications provided, this Morse code translator program appears to be an independent and self-contained product. It does not interface with any external systems or applications, and its functionality is limited to translating Morse code to English text and vice versa.

However, it is possible that this program could be integrated with other systems or applications in the future. For example, it could be incorporated into a larger communication system used by pilots, military personnel, or radio operators to enhance their ability to communicate using Morse code.

If integrated with a larger communication system, the Morse code translator program would likely interface with other components through a common communication protocol. For example, the translator could receive Morse code signals from a radio receiver component within the communication system, translate them into English text, and then send the text output to a display or another component for further processing.



2.2 User Characteristics

1. Students

- Students of any age who are learning Morse code as part of a course or extracurricular activity.
- May have limited experience with Morse code, but are familiar with basic computer skills and programming concepts.
- May prefer a user-friendly interface and clear instructions to guide them through the translation process.



2. Faculty
 - Teachers or instructors who teach Morse code or related subjects.
 - Likely have extensive experience with Morse code and related technologies, as well as programming experience.
 - May prefer advanced features and customization options to fit their specific teaching needs.
3. Staff
 - Individuals who work in fields that require Morse code knowledge, such as emergency responders, pilots, or military personnel.
 - May have varying levels of Morse code proficiency, but are likely to be technically savvy.
 - May require a simple, efficient, and reliable interface that allows them to quickly and accurately translate messages.
4. General public
 - Anyone who is interested in learning Morse code or has a personal interest in the language.
 - May have little or no experience with Morse code or programming.
 - May prefer a user-friendly and intuitive interface that simplifies the translation process and provides clear instructions.

2.3 Assumptions

1. Equipment availability: It is assumed that users have access to a computer or mobile device with a keyboard, speakers or headphones, and an internet connection. The program may need to be adjusted if certain equipment is not available or if the equipment varies significantly between users.
2. User expertise: The program assumes that users have a basic understanding of Morse code and its principles. The program may need to provide additional guidance or instructions if users have limited knowledge of Morse code.
3. Language proficiency: The program assumes that users are proficient in English and that the Morse code signals being translated are in English. The program may need to be adjusted for users who speak other languages or for messages that are not in English.
4. Morse code standards: The program assumes that Morse code signals follow standard Morse code conventions. The program may need to be adjusted if non-standard Morse code signals are used.
5. Operating system: The program assumes that users have access to a computer or mobile device with an operating system that supports Python. If the operating system does not support Python or has compatibility issues, the program may need to be adjusted or an alternative version of the program may need to be provided.
6. Input methods: The program assumes that users will input Morse code signals through a keyboard. The program may need to be adjusted if users require alternative input methods or if the program is to be integrated with other systems that use different input methods.



2.4 Constraints

1. Parallel operation with an old system: The program must be designed to integrate and operate alongside any existing Morse code systems or technologies that users may be using. Compatibility with older systems may limit design options and require additional testing and validation.
2. Audit functions: The program should include audit functions such as log files and an audit trail to track user activity and system events. These audit functions can help with system monitoring, troubleshooting, and compliance.
3. System resource constraints: The program must be designed to operate within system resource constraints, such as limits on disk space or memory. Design options may be limited by these resource constraints, and the program may need to be optimized for performance and efficiency.
4. Other design constraints: The program must comply with relevant design or programming standards, such as those set by the programming language or framework used to develop the program. These standards may constrain design options and require adherence to specific coding practices or conventions.

2.5 Dependencies

1. Morse code standards: The program's requirements will be influenced by the standards for Morse code. These standards will determine the length and format of Morse code signals and may influence the design of the program's user interface.
2. Third-party libraries and modules: The program may rely on third-party libraries and modules for specific functionalities, such as audio processing or networking capabilities. These dependencies will affect the requirements for the program, such as the need to install and configure these libraries or modules.
3. Operating system requirements: The program's requirements may be affected by the operating system it will run on, such as specific hardware or software requirements. For example, if the program requires a specific version of Python, this will affect the requirements for the program and any installation instructions.

3. Requirements

- Describe all system requirements in enough detail for designers to design a system satisfying the requirements and testers to verify that the system satisfies requirements.
- Organize these requirements in a way that works best for your project. See [Appendix D](#) [Appendix D. Organizing the Requirements](#) for different ways to organize these requirements.
- Describe every input into the system, every output from the system, and every function performed by the system in response to an input or in support of an output. (Specify what functions are to be performed on what data to produce what results at what location for whom.)
- Each requirement should be numbered (or uniquely identifiable) and prioritized. See the sample requirements in Functional Requirements, and System Interface/Integration, as well as these example priority definitions:



3.1 Functional Requirements

1. The program should support both Morse code to text and text to Morse code translation.
2. The program should allow users to input Morse code signals through a variety of methods, including audio input and keyboard input.
3. The program should support multiple languages for text output.
4. The program should have a user-friendly graphical user interface (GUI) that allows users to easily input Morse code and view the translated text.
5. The program should have a command-line interface (CLI) for advanced users.
6. The program should support file input and output, allowing users to translate entire files of Morse code signals or text.
7. The program should support real-time translation of Morse code signals received through an audio input source.
8. The program should be able to handle interruptions during Morse code signal input, such as pauses between letters and words.
9. The program should have an option to play audio output of translated Morse code signals.
10. The program should have a help menu that provides instructions on how to use the program.

Req#	Requirement	Comments	Priority	Date	Reviewed by / Approved by
BR_MC_01	The program shall accept a string of alphanumeric characters as input.		1	22/05/2023	Joel Bitri
BR_MC_02	The program shall translate the input string into Morse code.		1	22/05/2023	Joel Bitri
BR_MC_03	The program shall display the Morse code translation of the input string to the user.		1	22/05/2023	Joel Bitri
BR_MC_04	The program shall allow the user to input a Morse code string.		2	22/05/2023	Joel Bitri
BR_MC_05	The program shall translate the Morse code string into alphanumeric characters.		2	22/05/2023	Joel Bitri

Morse Code Translator Requirements Specification

BR_MC_06	The program shall display the alphanumeric translation of the Morse code string to the user.		2	22/05/2023	Joel Bitri
BR_MC_07	The program shall allow the user to save the translated output to a file.		3	22/05/2023	Joel Bitri
BR_MC_08	The program shall allow the user to load an input file containing alphanumeric or Morse code strings to be translated.		3	22/05/2023	Joel Bitri



3.2 Non-Functional Requirements

1. The program should be compatible with Windows, macOS, and Linux operating systems.
2. The program should have clear error messages and handling for unexpected inputs or errors.
3. The program should be secure, with appropriate access controls and protection against common security vulnerabilities.
4. The program should be maintainable and easily extensible with new features or input/output methods.
5. The program should be tested thoroughly to ensure that it meets all functional and non-functional requirements.
6. The program should follow industry standards for code documentation and commenting to facilitate future maintenance and development.

3.2.1 User Interface Requirements

User Input

- User input should be accepted through a text box
- Input validation should be performed to ensure only valid characters are entered
- The text box should be easily accessible and visible on the screen
- The input text should be automatically cleared after each translation

User Output

- Translated text should be displayed in a separate text box
- The output text should be easily distinguishable from the input text
- The output text should be selectable and copyable
- The output text should be formatted in a readable way

Error Messages

- Error messages should be displayed in a separate window or dialog box
- Error messages should clearly describe the problem and provide possible solutions
- Error messages should be easily dismissible

User Interface

- The user interface should be intuitive and easy to navigate
- The layout of the interface should be organized logically
- Menus should be easily accessible and clearly labeled
- Functionality should be easily discoverable and accessible
- Function keys should be labeled and have a clear purpose



3.2.2 Usability

Learnability:

- The program should be designed with a simple and intuitive user interface, making it easy for users to learn how to use the program quickly.

Efficiency:

- The program should be responsive and quick to execute commands, with minimal lag time or delays.

Error Prevention:

- The program should have clear error messages and prompts that provide users with feedback and guidance on how to correct errors.
- The program should prevent users from accidentally deleting or modifying important data.

3.2.3 Performance

Static numerical requirements:

- The system should support at least one input method (e.g. text input).
- The system should support at least one output method (e.g. sound output).
- The system should support at least one language (e.g. English).
- The system should be able to handle messages up to a certain length (e.g. 100 characters).
- The system should be able to handle input in a certain format (e.g. ASCII characters).



3.2.4 Manageability/Maintainability

3.2.1.1 Monitoring

1. In case of a system failure, the system shall automatically generate a log file that includes the error details, timestamp, and user details for troubleshooting purposes.
2. The system shall detect and prevent any data corruption, loss, or unauthorized access by implementing a secure database backup and restore mechanism.
3. The system shall provide a user-friendly error message that explains the error and suggests possible corrective actions.
4. The system shall be designed to handle unexpected user inputs and invalid data by providing appropriate error handling and validation mechanisms.
5. The system shall have a user-friendly interface for error correction that allows the user to easily correct mistakes and retry failed operations.
6. The system shall implement a periodic system maintenance schedule for detecting and fixing any potential issues before they cause problems.

3.2.1.2 Maintenance

1. Documentation: The system should be well-documented, with clear and concise documentation for each module, function, and interface. This will help maintenance personnel to understand how the system works and how to troubleshoot problems.
2. Error handling: The system should have robust error handling and logging capabilities. All errors should be logged in a way that is easy to understand and analyze, and error messages should provide useful information to maintenance personnel.
3. Testability: The system should be designed with testability in mind, so that maintenance personnel can easily test individual components and verify that they are working correctly.
4. Interface design: The system should have well-designed interfaces that are easy to understand and use. This will help to minimize the learning curve for maintenance personnel and reduce the risk of errors or mistakes.

3.2.1.3 Operations

Normal operations:

- Periods of interactive operations: The user will interact with the program during the translation process. The duration of these periods will depend on the length of the input text and the speed of the user.



Data processing support functions:

- The program should be able to handle various types of input data, such as text, audio, or image files.
- The program should be able to translate Morse code to text and vice versa accurately.

Safety considerations and requirements:

- The program should not cause any harm to the user's device or data.
- The program should not transmit any user data or information to external parties without the user's consent.

3.2.5 Security

3.2.5.1 Protection

Authentication: The system should require users to authenticate before accessing any sensitive functionality. This can be achieved through the use of usernames and passwords or other forms of authentication such as biometric authentication.

- **Encryption:** Any sensitive data that is stored or transmitted by the system should be encrypted to prevent unauthorized access or disclosure. This can be achieved through the use of industry-standard encryption algorithms such as AES.

- **Access control:** The system should implement access control mechanisms that restrict users' access to data and functionality based on their roles and privileges. This can be achieved through the use of role-based access control (RBAC) or other access control models.

- **Logging:** The system should log all user activity, including failed login attempts, access to sensitive data, and changes to system configuration. This will enable administrators to identify potential security breaches and take appropriate action.

- **Data integrity checks:** The system should implement mechanisms to ensure data integrity, such as checksums or digital signatures. This will prevent unauthorized modifications to data and ensure that the system is operating as intended.



3.2.5.2 Authorization and Authentication

Authentication:

- The system shall require users to enter a valid username and password to access the system.
- The system shall use password hashing to securely store user passwords.
- The system shall provide a mechanism for users to reset their passwords if forgotten or compromised.
- The system shall automatically log out users after a certain period of inactivity.
- The system shall prevent brute force attacks by locking out user accounts after a specified number of failed login attempts.

Authorization:

- The system shall use role-based access control to determine what actions a user can perform within the system.
- The system shall provide a mechanism for creating and managing user roles.
- The system shall prevent unauthorized access to sensitive data or functions by limiting user access based on their roles.
- The system shall provide an audit trail of all user actions within the system to track any unauthorized access or modifications.
- The system shall provide a mechanism for revoking access to the system for users who no longer require it.

3.2.6 Standards Compliance

For the morse code translator program, there are no specific requirements derived from existing standards, policies, regulations, or laws. However, it is important to ensure that the program does not violate any intellectual property rights related to the Morse code, such as using it for commercial purposes without proper licensing or authorization.



3.2.7 Other Non-Functional Requirements

1. Performance:
 - The system shall respond to user requests within 2 seconds.
 - The system shall be able to handle a minimum of 1000 concurrent users.
 - The system shall be able to process a minimum of 10,000 records per minute.
2. Availability:
 - The system shall be available for use 24 hours a day, 7 days a week, with no more than 2 hours of downtime per month.
 - The system shall have a failover mechanism that ensures availability in case of system failure.
3. Security:
 - The system shall use HTTPS encryption to protect sensitive data.
 - The system shall enforce password policies, including minimum length and complexity.
 - The system shall have access controls that restrict user access to only the data and functions necessary for their role.
5. Compatibility:
 - The system shall be compatible with commonly used web browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge.
 - The system shall be compatible with Windows, macOS, and Linux operating systems.
 - The system shall be compatible with databases such as MySQL, Oracle, and Microsoft SQL Server.

3.3 Domain Requirements

- The software must comply with the International Morse Code standards
- The software must support a wide range of Morse code transmission rates
- The software must be able to handle noise and interference in the transmission signal
- The software must be able to differentiate between different types of Morse code signals (e.g., tone, frequency, duration)
- The software must provide a user-friendly interface for Morse code enthusiasts with varying levels of experience and skill levels
- The software must be able to handle Morse code transmissions in various languages and character sets
- The software must be able to decode Morse code transmissions sent using different types of modulation (e.g., amplitude, frequency, phase)



4. Design thinking methodologies

4.1 Negotiation

In the case of this one-member project, negotiation involves the individual negotiating with themselves or with stakeholders external to the project, such as customers or end-users. The negotiation process involves the following steps:

1. **Preparation:** The individual should prepare themselves with a clear understanding of the project goals, constraints, and requirements. They should also be aware of the available resources, timelines, and budget.
2. **Discussion:** The individual should initiate a discussion with stakeholders to identify their needs, expectations, and requirements. They should listen actively and ask questions to clarify any doubts or uncertainties.
3. **Evaluation:** The stakeholders should evaluate the proposed solution against their needs, expectations, and requirements. They may provide feedback or suggest changes to the proposal.
4. **Agreement:** Once the stakeholders have evaluated the proposal and provided feedback, the individual should negotiate to find a mutually acceptable solution. They should agree on the scope, requirements, timelines, and budget of the project.

4.2 Empathy

Empathy would involve putting oneself in the shoes of the user and understanding their experience with the product. This might involve conducting user research to gather feedback on the current design, observing how users interact with the product, and engaging in dialogue with users to better understand their needs and desires.

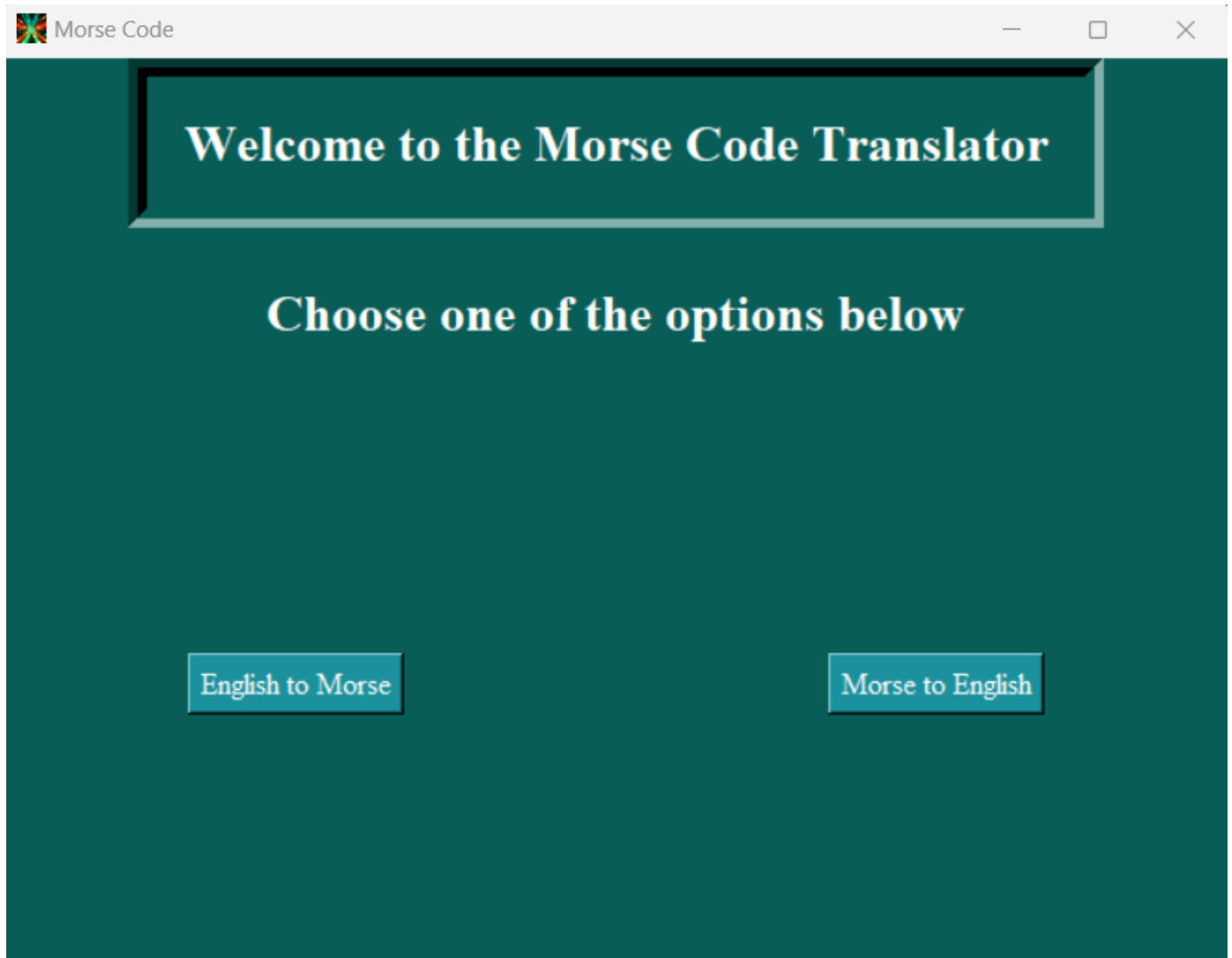
Once the designer has developed a deep understanding of the user's experience, they can use that knowledge to inform the design of the product. For example, they might design a user interface that is intuitive and easy to use, or they might incorporate features that address common pain points or frustrations that users have reported. By empathizing with the user, the designer can create a product that is more effective, efficient, and satisfying to use.

4.3 Noticing

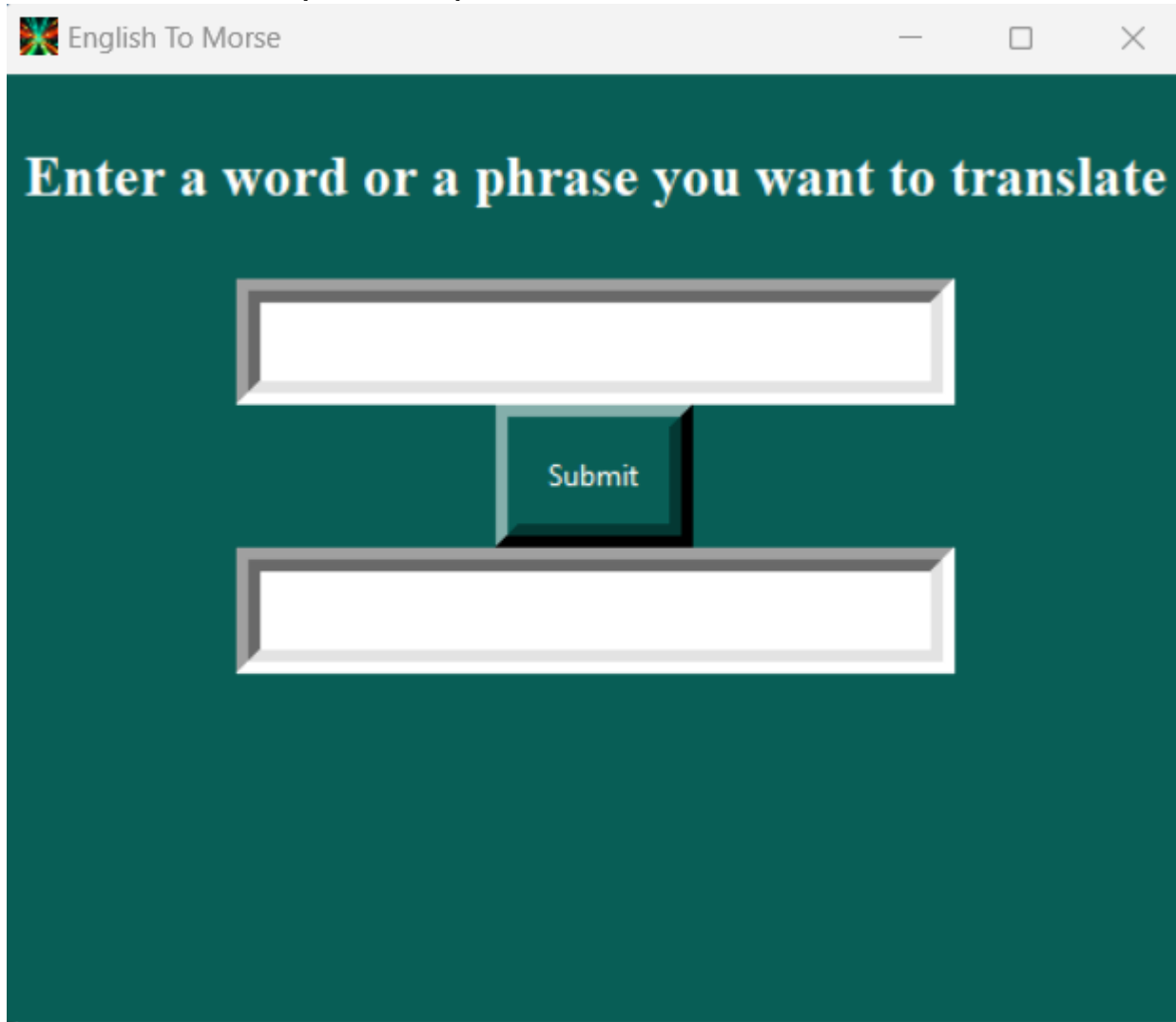
During the noticing phase, the designer might observe that some users may not be familiar with the Morse code system and may struggle to understand the translations provided by the software. As a result, the designer might decide to include a user-friendly interface that provides information and guidance on Morse code. The designer might also notice that some users may need to translate large amounts of text at once and require an efficient and streamlined process. To address this, the designer might incorporate a batch processing feature that can quickly translate entire documents or sections of text.



4.4 GUI (Screenshots)



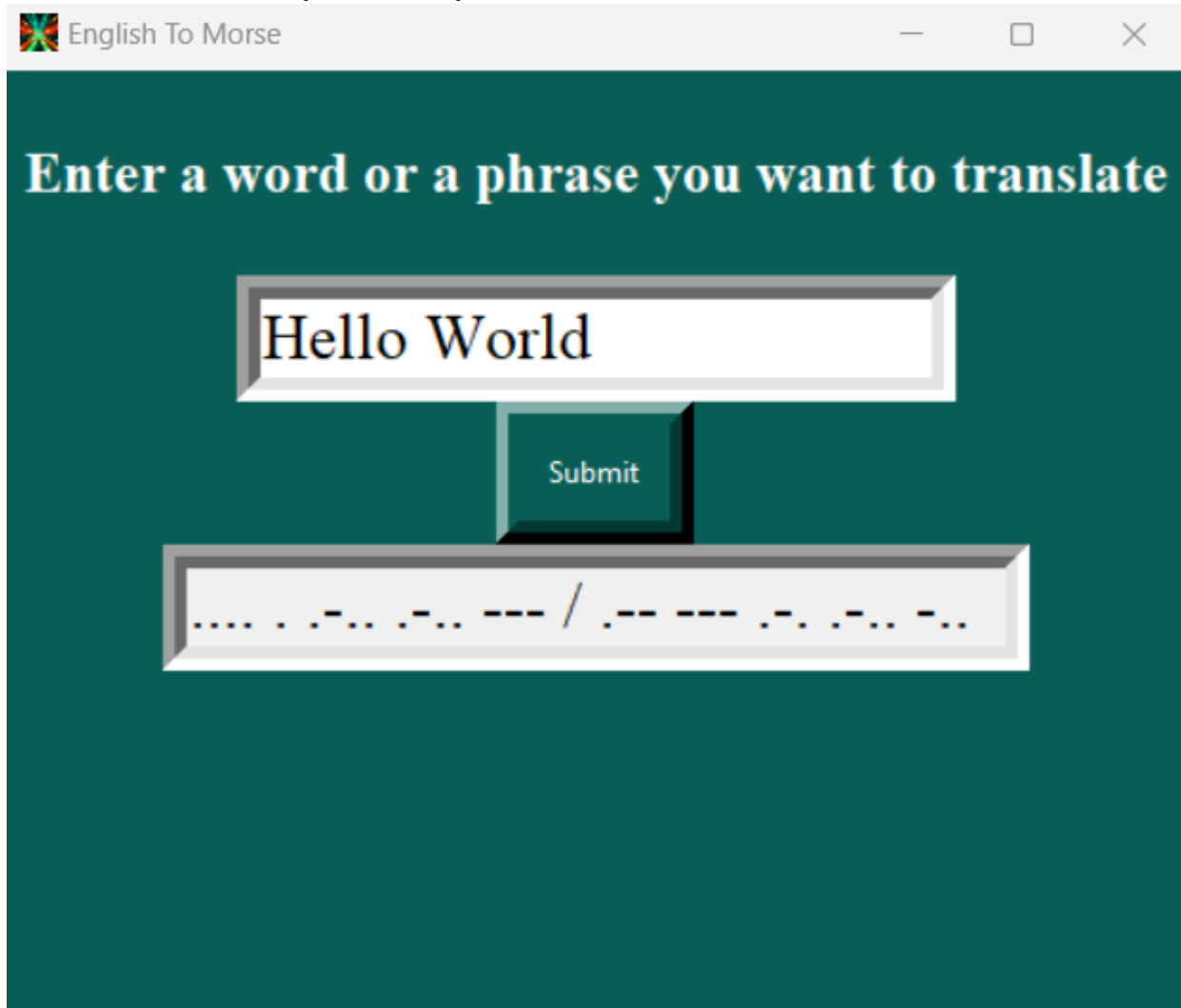
Morse Code Translator Requirements Specification



English To Morse

Enter a word or a phrase you want to translate

Submit



English To Morse

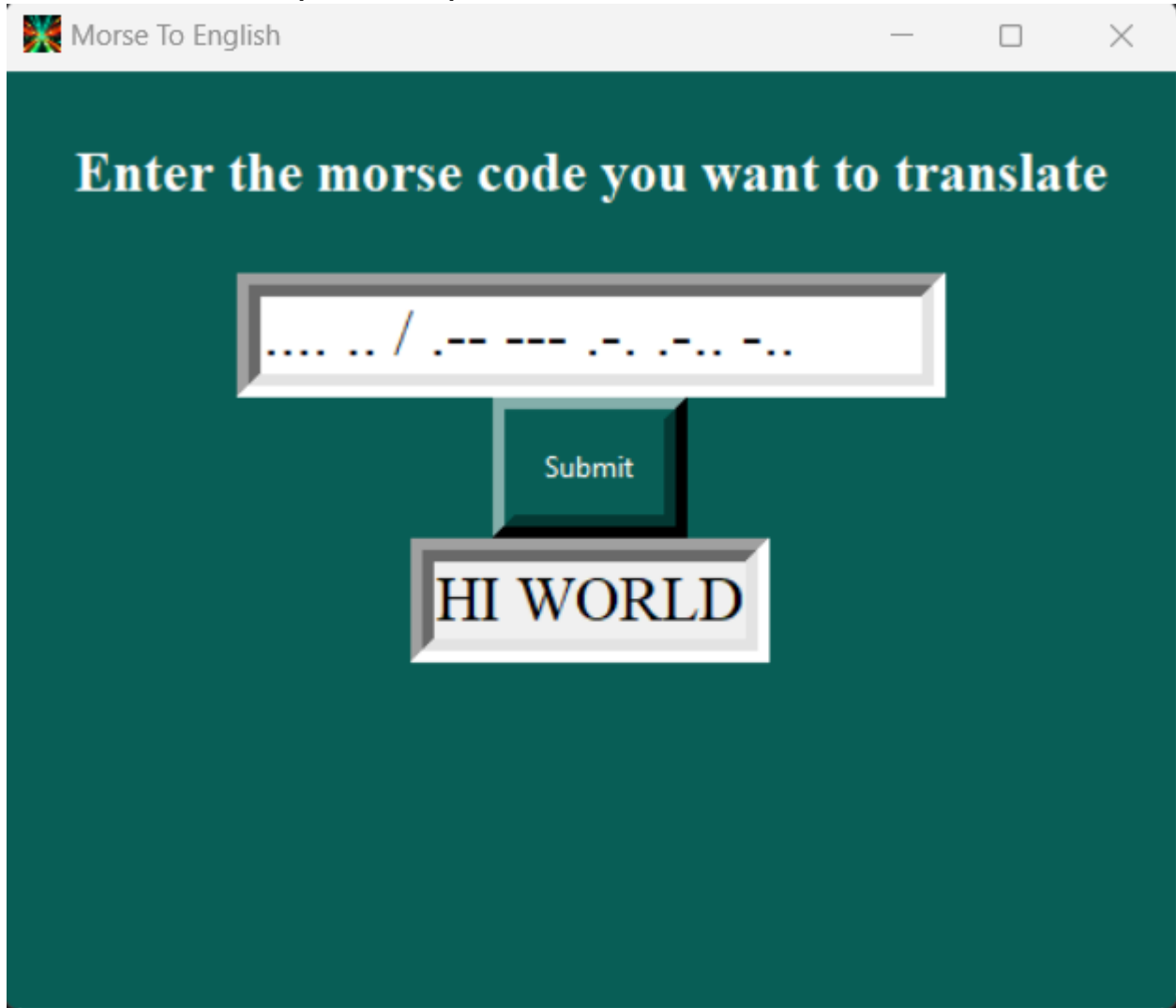
Enter a word or a phrase you want to translate

Hello World

Submit

.... . -... .-.. --- / .-- --- .-.. -...

Morse Code Translator Requirements Specification



The screenshot shows a web browser window with the title "Morse To English". The main content area has a dark green background. At the top, the text "Enter the morse code you want to translate" is displayed in a yellow, serif font. Below this text is a white rectangular input field with a 3D effect, containing the Morse code ". / . - - - . - . - . - .". A green "Submit" button is positioned directly below the input field. At the bottom, a white rectangular output box with a 3D effect displays the translated text "HI WORLD" in a black, serif font.



5. Software Design

5.1 Use Case

The Morse Code Translator project aims to provide a simple and easy-to-use tool for users to translate text into Morse code and vice versa. The following are the major functions that the product will perform:

1. Text to Morse Code Translation: Users can input text into the system and get its corresponding Morse code output.
2. Morse Code to Text Translation: Users can input Morse code into the system and get its corresponding text output.
3. Audio Playback: Users can listen to the Morse code audio of their input text or Morse code output.
4. User Input Validation: The system will check the validity of the user input and provide appropriate error messages in case of invalid input.
- 5.

Business Scenario: A small communication company has a need to communicate messages to their remote field staff, who may not have access to the internet or a phone signal. They want to use Morse code to transmit these messages but do not have staff who are proficient in Morse code. They require a tool that can quickly and easily translate text into Morse code and vice versa, allowing them to communicate with their field staff effectively.

Actors:

- Communication Company
- Remote Field Staff

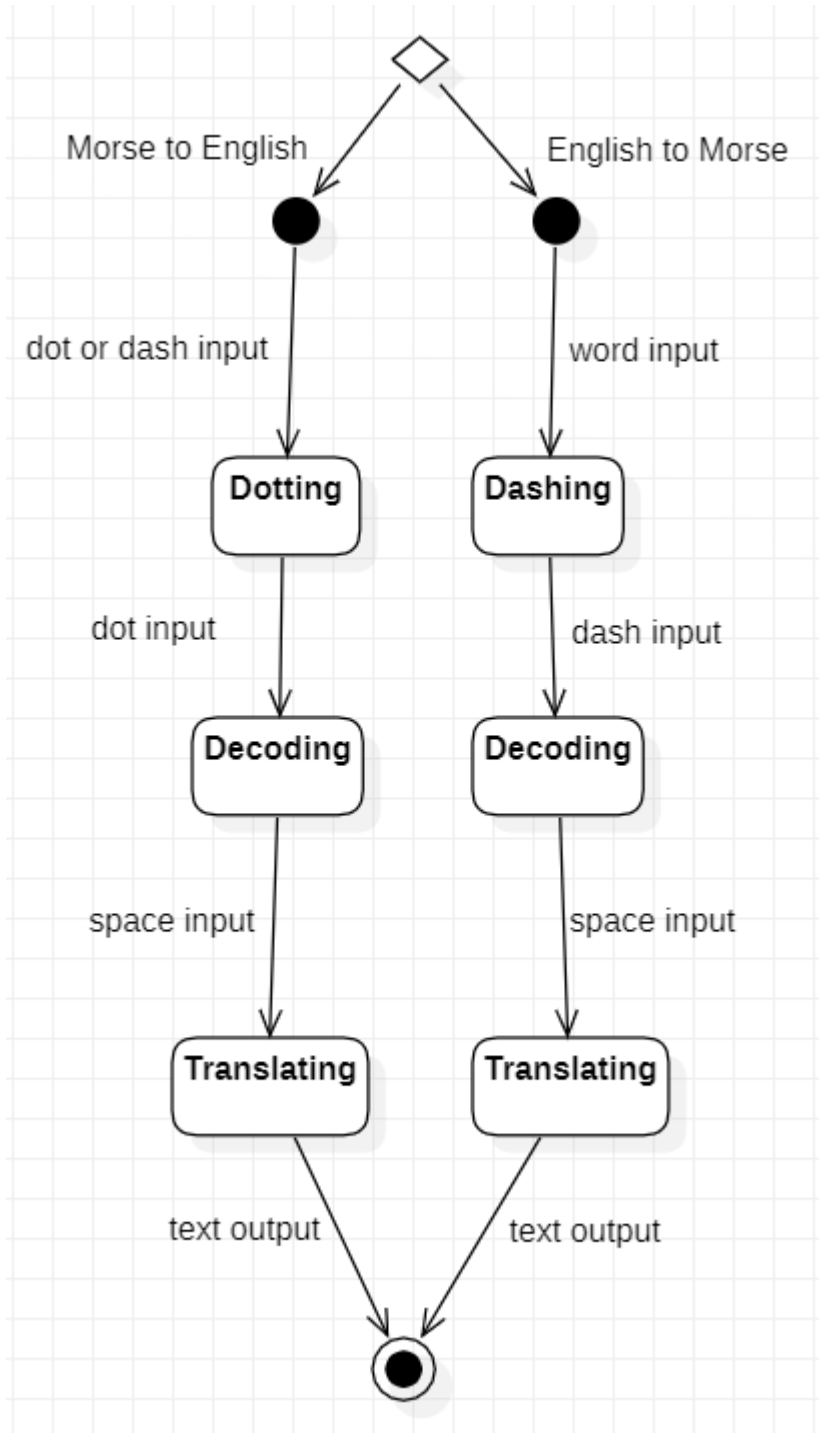
Objectives:

- Enable the communication company to send messages to their remote field staff via Morse code
- Provide a simple and easy-to-use tool for translating text to Morse code and vice versa
- Ensure the accuracy of the Morse code translation
- Customize the settings of the translator according to the communication company's preferences
- Improve communication between the communication company and their remote field staff

Metrics for Success:

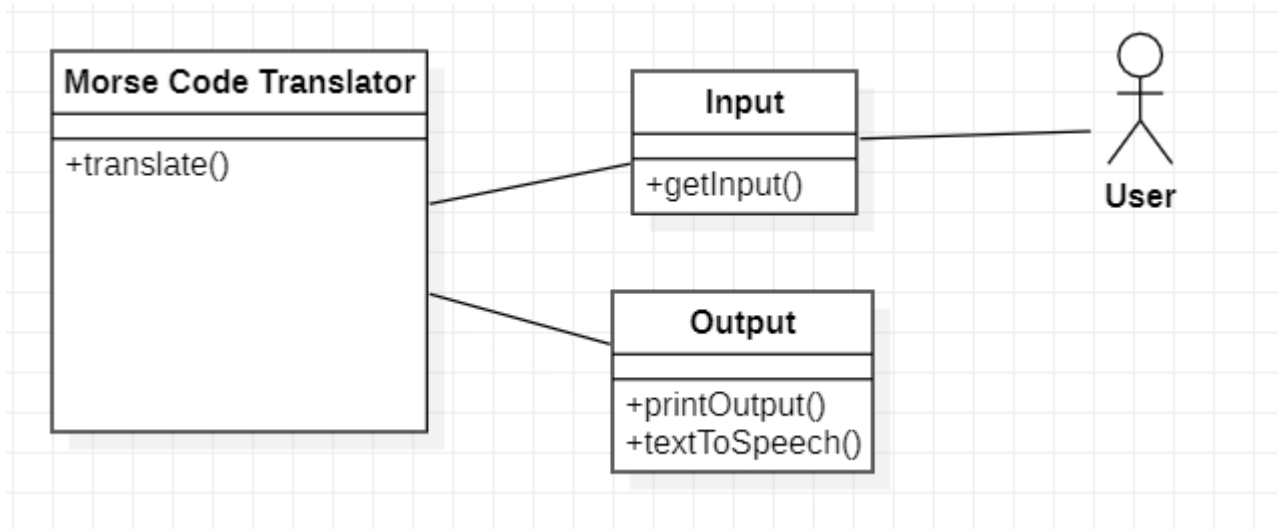
- Decrease in communication errors between the communication company and their remote field staff
- Increase in efficiency of communication with remote field staff
- High accuracy of Morse code translation
- High user satisfaction with the translator's customization options and ease of use

5.2 State Diagram





5.3 Class Diagram





APPENDIX

The appendixes are not always considered part of the actual Requirements Specification and are not always necessary. They may include

- Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;
- Supporting or background information that can help the readers of the Requirements Specification;
- A description of the problems to be solved by the system;
- Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When appendixes are included, the Requirements Specification should explicitly state whether or not the appendixes are to be considered part of the requirements.

Appendix A. Definitions, Acronyms, and Abbreviations

Define all terms, acronyms, and abbreviations used in this document.

Appendix B. References

List all the documents and other materials referenced in this document.

Appendix C. Organizing the Requirements

This section is for information only as an aid in preparing the requirements document.

Detailed requirements tend to be extensive. Give careful consideration to your organization scheme. Some examples of organization schemes are described below:

By System Mode

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency.

By User Class

Some systems provide different sets of functions to different classes of users. For example, an elevator control system presents different capabilities to passengers, maintenance workers, and fire fighters.

By Objects

Objects are real-world entities that have a counterpart within the system. For example, in a patient monitoring system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc. Associated with each object is a set of attributes (of that object) and functions (performed by that object). These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.

By Feature

A feature is an externally desired service by the system that may require a sequence of inputs to affect the desired result. For example, in a telephone system, features include local call, call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs, and may include validity checks on inputs, exact sequencing of operations, responses to abnormal situations, including error handling and recovery, effects of parameters, relationships of inputs to outputs, including input/output sequences and formulas for input to output.



By Stimulus

Some systems can be best organized by describing their functions in terms of stimuli. For example, the functions of an automatic aircraft landing system may be organized into sections for loss of power, wind shear, sudden change in roll, vertical velocity excessive, etc.

By Response

Some systems can be best organized by describing all the functions in support of the generation of a response. For example, the functions of a personnel system may be organized into sections corresponding to all functions associated with generating paychecks, all functions associated with generating a current list of employees, etc.

By Functional Hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be used to show the relationships between and among the functions and data.

Additional Comments

Whenever a new Requirements Specification is contemplated, more than one of the organizational techniques given above may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; and when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.