Software Engineering Coursework (SET09120)

Max McRuvie

School of Computing, Edinburgh Napier University, Edinburgh  
40496329@live.napier.ac.uk

**Abstract.** This paper contains information on of the design, creation, and testing of the Napier Bank Messaging System coursework for the Software Engineering module.

The NBM System’s purpose is to take in message inputs, sperate them based of their type (tweet, sms, or email), along with separating specified data depending on the data type (for example; for sms, abbreviation would be a specified data type) and display them on screen. The program must also be able to save files in JSON format to local storage.

Contents

[**Requirement Specs** 3](#_Toc58112376)

[**Use Case Diagram** 3](#_Toc58112377)

[**Class Diagram** 4](#_Toc58112378)

[**Functional Requirements** 4](#_Toc58112379)

[**Non-Functional Requirements** 5](#_Toc58112380)

[**Testing** 5](#_Toc58112381)

[**Version Control** 5](#_Toc58112382)

[**Evolution Strategy** 5](#_Toc58112383)

# **Requirement Specs**

## **Use Case Diagram**

To better grasp the requirement specification needed to create the NBM system I first had to create a use case diagram using StarUML. This process allowed me to visualize the requirements of the project.

Diagram

Description automatically generated

## **Class Diagram**

Diagram

Description automatically generatedAnother diagram I created to help visualise the requirements of the NBM System was a Class Diagram. This greatly helped when it came to building the system as it is essentially a blueprint for the required classes and the relationships, they will roughly have with each other.

## **Functional Requirements**

For functional requirements there is a lot to unpack. Firstly, the system must be able to take in a user’s input message (which will be a header, and a body) and verify that it meets the requirements specified.

For verifying the message header by ensuring start with the letter ‘S’, ‘E’ or ‘T’, this stands for the three different message types; Sms, Email, and Text. The header must then be followed by 9 numeric characters. The system must be able to check this to ensure that the message header is valid. After doing this it must then split the message into its specific data type based off the first letter in the message header.

After the message is split into its data type the class it is split into then verifies that it meets the requirements of that said data type. It does this by ensuring that the data is the correct length, contains the correct characters (for example emails must contain the @ char)

After the data types are verified the message must be checked for characteristics. Each message type has its own specific characteristics that the program must check for and then proceed to separate into either lists or dictionaries.

There may be two characteristics for tweets, mentions and hashtags. Mentions start with the ‘@’ char, and hashtags start with the ‘#’ char. The system must be able to recognise these and place them into their corresponding lists and dictionary.

For emails the characteristics are URL’s and SIR’s within the messages body. After these characvteristics hav been recognised they are added to aOIFMNOIEJF

## **Non-Functional Requirements**

# **Testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs | Expected Result | Result | Description | Solution |
| S123456789 | Fail | Fail | Exception is thrown saying the body cannot be empty | N/A |
| B123456789 | Fail | Fail | Exception thrown saying the header must start with ‘S’, ‘E’, or ‘T’ | N/A |
| +32132113213  SenderTest TextTest | Fail | Fail | Exception is thrown saying header must be length of 10 | N/A |
| S123  +32132113213  SenderTest TextTest | Fail | Fail | Exception is thrown saying header must be length of 10 | N/A |
| S123456789  +32132113213  SenderTest TextTest | Pass | Pass | All info inputted meets the message requirements | N/A |
| S123456789  +32132113213  SenderTest | Exception should be thrown saying text shouldn’t be empty | Exception is thrown saying text shouldn’t be empty | Exception is thrown saying text cannot be empty | N/A |
| S123456789  +32132113213 | Fail | Fail | Fails due to no sender | N/A |
| T123456789  @SenderText TextTest | Pass | Pass | The header is added to the message list, with the body being displayed below | N/A |
| T123456789  SenderText  TextTest | Fail | Fail | Exception Handle thrown saying that an ‘@’ must be present in sender | N/A |
| T123456789  SenderText |  |  |  |  |

# **Version Control**

For an agile approach I would highly suggest that GitHub for version control. GitHub allows for you to track every single change made in the form of commits, along with allowing you to create branches off from the main system. This allows you to design, build, and test potential functionality without compromising the main system. A merge can then be performed it is desired for the built branch to become the main system.

GitHub also allows for great cooperation amongst more than one developer as multiple people (if given access) can push commits to the system, all while everyone can see what everyone is doing and adjusting. Therefore, it’s a good platform for collaborating with teammates.

That’s why my proposed plan for version control would be to use GitHub.

# **Evolution Strategy**