

SET09102 – SOFTWARE ENGINEERING

Napier Bank Message Filtering System

MATRIC NUMBER: 40218056

26/11/2018

Contents

1. Requirements analysis	3
1.1. System Summary	3
1.2. Requirement Specification	3
1.2.1. Use Cases	4
1.2.2. Non-functional Requirements	4
1.3. List of requirements	5
1.3.1. High-level Functional Requirements	5
1.3.2. Message Type-specific Requirements	6
2. System Design	8
2.1. Object-Oriented Model	8
3. Testing	14
3.1. Testing Strategy	14
3.2. Testing Plan	14
3.2.1. Test items	14
3.2.2. Objectives	14
3.2.3. Testing levels and scope	14
3.2.4. Tasks and deliverables	17
3.2.5. Test environment needs	17
3.2.6. Tools	17
3.2.7. Test schedule	17
3.3. Test cases	18
3.3.1. Unit test cases	18
3.3.2. Validation test cases	21
4. Version Control Plan	22
4.1. Collaboration between developers	22
4.2. Collaboration between the developing and the QA teams	23
5. Evolution Strategy	24
5.1. Evolution and maintenance predictions	25
5.2. System maintainability and predicted costs	25
Appendix I – Results of Unit Test Cases	26
Appendix II – Results of Validation Test Cases	27

1. Requirements analysis

1.1. System Summary

The system being developed is a message filtering and processing system for the Napier Bank. Its primary function is to process and categorize incoming messages according to the message type. Messages can be SMS, Tweets, Emails and Significant Incident Reports (SIR), which are a special kind of email. The details of how each message type must be processed are discussed below. The system must allow users to enter messages through a Graphical User Interface (GUI) directly, or by loading in messages from a text file. When a message is entered and processed successfully, it must be re-displayed on screen in its processed form and saved to an output file in JSON format. At the end of an input session, a user must be able to view all processed messages, the list of mentions, the trending list and the list of SIRs. Additionally, the system must validate user input and inform the user when invalid input is entered by displaying an error message. The system must be easy to use, and it must have high performance.

1.2. Requirement Specification

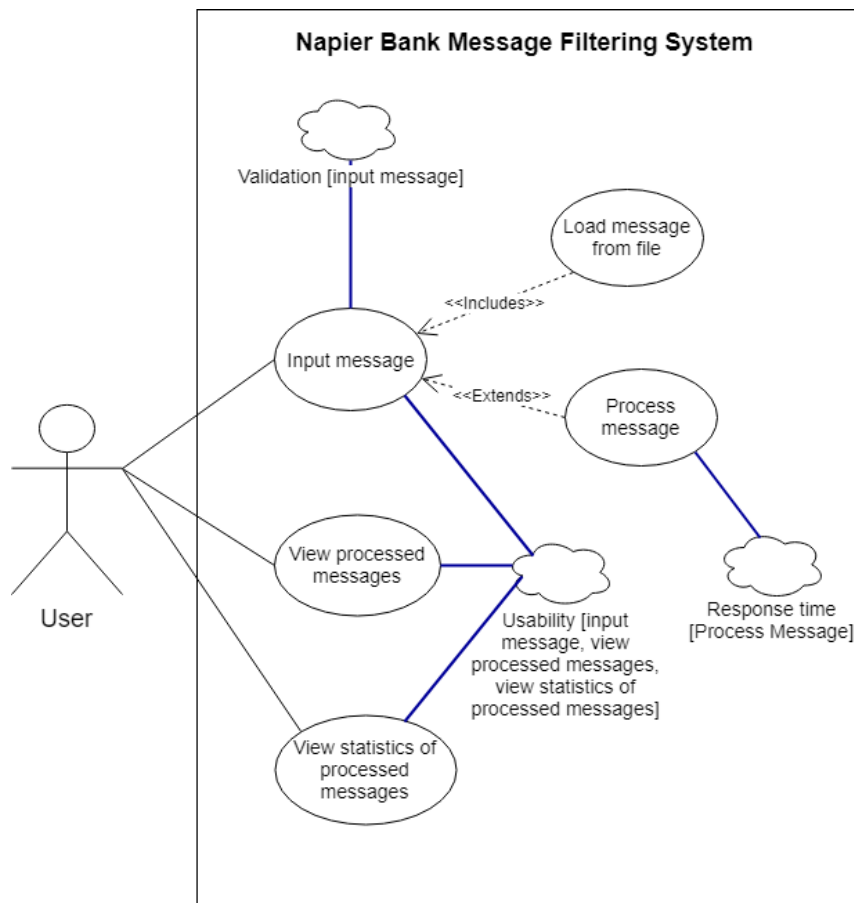


Figure 1. High level Use Case - NFR diagram for the Napier Bank Message Filtering System.

The Use Case Diagram above (figure 1), aims to provide a general understanding of the main functionality of the system. To keep the diagram simple and easy to understand, low level details and functions are not included. However, the diagram includes the most important functional and non-functional requirements.

1.2.1. Use Cases

Input Message

A user must be able to input messages directly using the system's user interface (UI). Input messages are entered in the form of a message header (containing the message id) and the message body (containing different information depending on the message type) into appropriate text boxes in the UI. The format of the message body is dependent on the message type.

Load message from file

A user shall have the option to load in messages from a file. The UI must allow the user to browse the file system to provide the path of the file containing the messages to load. The system must read in the messages from the file and allow the user to see and process loaded messages one by one.

Process message

When a valid message is entered, the system must be able to automatically detect the message type and process it accordingly (details of how each type of message is processed are discussed below). The system must be capable of processing SMS messages, tweets, emails and significant incident reports. Successfully processed messages are immediately re-displayed on screen in their processed form and saved to an output file in JSON format.

View processed messages

At the end of an input session, a user must be able to view all the messages processed during the session. The system must allow the user to search for a message by providing the message id. Additionally, the system must provide message filtering functionality so the user can filter messages by type.

View statistics of processed messages

At the end of an input session, a user must be able to view the statistics from the messages processed during the session. The statistics available for viewing are:

- A **list of mentions** containing all the twitter ids mentioned in the processed tweets.
- A **trending list** containing the hashtags extracted from the processed tweets ordered by the number of times each hashtag was used.
- A **list of SIRs** containing the sort code and nature of incident of all processed significant incident reports.

1.2.2. Non-functional Requirements

Validation

The system must validate input messages according to the message type. If the input is invalid, the system shall display an error message and prompt the user to re-enter the message in a valid format.

Usability

The system must be easy to use and easy to learn. With this goal in mind, the user interface layout must be simple and structured in a way that more important elements draw the user's attention. Common UI elements must be used so they feel familiar to the user. The interface must be consistent, e.g. text boxes for user input must look the same across the whole application. Additionally, the system must clearly communicate what is happening, informing users when an operation is completed, or an error occurs.

Response time

Processing messages must be a quick operation. When a user enters a message for processing, they must see an error message if the processing fails, or a confirmation that the processing was successful otherwise. A user mustn't be able to notice a significant delay between the time they enter the message and the time the message is processed.

1.3. List of requirements

This section lists all the requirements for the NBMFS system. The requirements are numbered so they can be easily referenced throughout the document if needed.

1.3.1. High-level Functional Requirements

1. Users must be able to input messages into the system directly through the Graphical User Interface (GUI).
2. Users must be able to load in messages from an input text file.
3. Messages can be entered in the form of a message header and message body into appropriate text boxes.
4. When a message is entered, the system must identify the message type automatically and process it accordingly.
5. The system must be able to process 4 types of messages: SMS messages, Tweets, Emails and Significant Incident Reports (these are a special type of email).
6. Successfully processed messages shall be re-displayed in their processed form in the same window.
7. Successfully processed messages must be saved to an output file in JSON format.
8. After an input session, a user shall be able to view the messages processed during the session in their processed form.
9. A user must be able to search a processed message by entering the message id.
10. A user must be able to filter processed messages by type, so that only messages of a specific type are displayed.
11. A user must be able to view a list of mentions which consists of a list with all the tweeter ids mentioned in the processed tweets.
12. A user must be able to view a trending list of all the hashtags extracted from the processed tweets ordered by the number of times each hashtag was used.
13. A user must be able to view a list of Significant Incidents containing the sort code and nature of incident of all processed significant incident reports.

1.3.2. Message Type-specific Requirements

SMS messages

14. SMS message header must comprise the message id starting with “S” followed by 9 digits, e.g. “S273982911”.
15. SMS message body contains the sender (1st line) followed by the message text (subsequent lines).
16. The message sender is an international phone number starting with “+” followed by the phone number, e.g. “+447881234567”.
17. The message text can have a maximum of 140 characters.
18. Textspeak abbreviations contained in SMS messages being processed will be expanded to their full form enclosed in “<>” (E.g. “ROFL” becomes “ROFL <Rolls on the floor laughing>”).

Tweets

19. Tweet message header must comprise the message id starting with “T” followed by 9 digits, e.g. “T273982911”.
20. Tweet message body contains the sender (1st line) followed by the tweet text (subsequent lines).
21. The tweet sender must have a maximum of 16 characters and must be a valid tweeter id starting with “@”.
22. The tweet text can have a maximum of 140 characters.
23. Textspeak abbreviations contained in Tweets being processed will be expanded to their full form enclosed in “<>” (E.g. “ROFL” becomes “ROFL <Rolls on the floor laughing>”).
24. Twitters ids embedded into tweets are added to a list of mentions.
25. Hashtags embedded into tweets are added to a trending list.

Emails

26. Email message header must comprise the message id starting with “E” followed by 9 digits, e.g. “E273982911”.
27. Email message body contains the sender (1st line), the subject (2nd line) followed by the email text (in the subsequent lines).
28. The email sender must be a standard email address e.g. “example@example.com”.
29. The email subject must have a maximum of 20 characters.
30. The email text must have a maximum of 1024 characters.
31. Hyperlinks contained in email messages shall be added to a quarantine list and replaced by “<URL Quarantined>”.

Significant incident reports

32. Significant incident reports are a special type of email where the subject has the incident data in the form "SIR dd/mm/yy", e.g. "SIR 03/08/2018".
33. SIR message body contains the sender (1st line), the subject (2nd line), sort code (3rd line) and nature of incident (4th line), followed by the email text (in the subsequent lines).
34. The sort code which is comprised of 6 digits in the form "XX-XX-XX" (E.g. 88-32-50).
35. The nature of incident can be one of the following: Theft; Staff Attack; ATM Theft; Raid; Customer Attack; Staff Abuse; Bomb Threat; Terrorism; Suspicious Incident; Intelligence; Cash Loss.
36. Sort code and nature of incident contained in processed SIRs are added to an SIR list.
37. As with standard emails, hyperlinks contained in SIRs shall be added to a quarantine list and replaced by "<URL Quarantined>".

2. System Design

The NBMFS system will be developed using an object-oriented approach. The program will consist of a WPF application written in .NET (C#) using Visual Studio 2017.

2.1. Object-Oriented Model

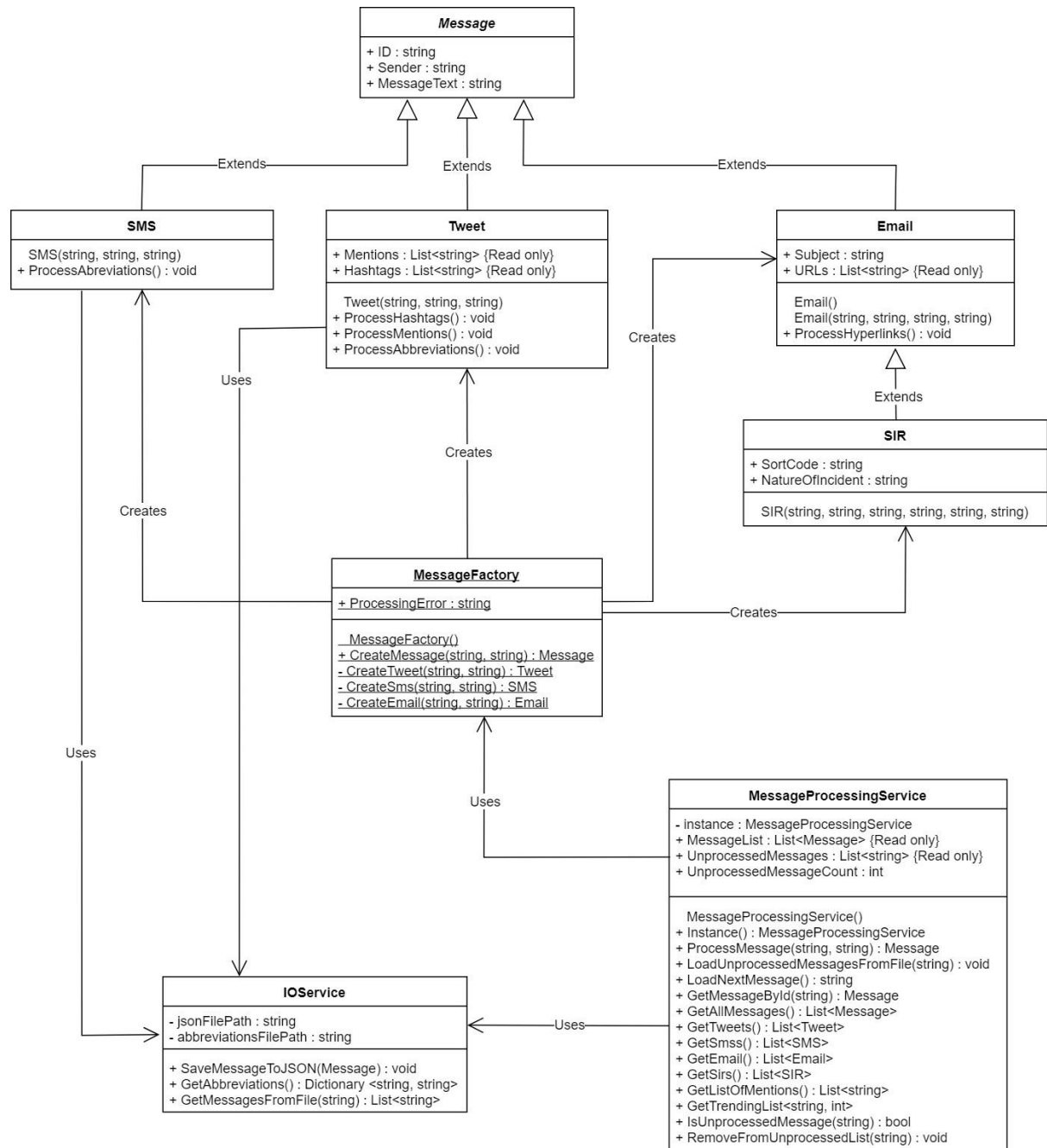


Figure 2. NBMFS system class diagram

The NBMFS class diagram above (figure2) describes the overall structure of the system, including all the classes, with their properties and method, and the relationships between classes. For simplicity reasons, the diagram only shows the logical classes and so, the UI classes are not included.

The system consists of 8 classes:

- Message
- SMS
- Tweet
- Email
- SIR
- MessageFactory
- IOService
- MessageProcessingService

Below is a description of the purpose and functionality for each class, its properties, and methods.

Message

The message class is an abstract class containing the properties that are common to all the different message types: id, sender and message text. Being an abstract class, this class is never instantiated. Its main function is to serve as a base class from which the classes for the different message types can inherit from.

Properties

ID	String to store the message id
Sender	String to store the message sender
MessageText	String to store the message text

SMS

The SMS class inherits from Message. Its main function is to model SMS messages. SMS objects can be created by passing in the message id, the message sender and the message text to the constructor.

Properties

ID	String to store the message id (inherited from Message)
Sender	String to store the message sender (inherited from Message)
MessageText	String to store the message text (inherited from Message)

Methods

ProcessAbbreviations	It converts the textspeak abbreviations contained in the SMS text to their full form.
----------------------	---

Tweet

The Tweet class inherits from Message. Its main function is to model Tweets. Tweet objects can be created by passing in the message id, the message sender and the message text to the constructor.

Properties

ID	String to store the message id (inherited from Message)
Sender	String to store the message sender (inherited from Message)
MessageText	String to store the message text (inherited from Message)
Mentions	List of strings to store the tweeter ids contained in the tweet text.
Hashtags	List of strings to store the hashtags contained in the tweet text.

Methods

ProcessAbbreviations	It converts the textspeak abbreviations contained in the tweet text to their full form.
ProcessHashtags	Extracts hashtags contained in the tweet text and add them to the list of hashtags
ProcessMentions	Extracts twitter ids contained in the tweet text and add them to the list of mentions

Email

The Email class inherits from Message. Its main function is to model Emails. On top of the properties common to the other message types, emails have an additional property – the Subject. Email objects can be created by passing in the message Id, sender, subject and message text to the constructor.

Properties

ID	String to store the message id (inherited from Message)
Sender	String to store the message sender (inherited from Message)
MessageText	String to store the message text (inherited from Message)
Subject	String to store the message text
URLs	List of strings to store the hyperlinks contained in the email text.

Methods

ProcessHyperlinks	Removes hyperlinks from the email text and replaces them with "<URL Quarantined>". Removed hyperlinks are added to the list of URLs
-------------------	---

SIR

The SIR class models Significant Incident Report messages. Since SIRs are special kinds of emails, the SIR class inherits from the Email class. However, SIRs have 2 additional properties: SortCode and NatureOfIncident. SIR objects can be created by passing in message id, sender, subject, sort code, nature of incident and message text to the constructor.

Properties

ID	String to store the message id (inherited from Message)
Sender	String to store the message sender (inherited from Message)
MessageText	String to store the message text (inherited from Message)
Subject	String to store the message text (inherited from Email)
URLs	List of strings to store the hyperlinks contained in the email text (inherited from Email)
SortCode	String to store the SIR sort code
NatureOfIncident	String to store the SIR nature of incident

Methods

ProcessHyperlinks	Removes hyperlinks from the email text and replaces them with "<URL Quarantined>". Removed hyperlinks are added to the list of URLs (method inherited from Email)
-------------------	---

MessageFactory

The MessageFactory class is a static class that implements the Factory design pattern and its main function is to manage the creation of message objects of different types. This functionality is provided by the method CreateMessage().

Properties

ProcessingError	String to store the error message when input format is invalid
-----------------	--

Methods

CreateMessage	This method takes in the message header and message body as input and returns a message object of the respective type depending on the message id passed in the message header, i.e. if the message id is "S987654321", it creates and return a SMS object. If the format of the input does not match the format of any of the different message types, the method returns null instead. Additionally, when the message validation fails, the validation error is stored in the ProcessingError property.
CreateSms	Private method used by the CreateMessage method to create SMS objects
CreateTweet	Private method used by the CreateMessage method to create Tweet objects
CreateEmail	Private method used by the CreateMessage method to create Email and SIR objects

IOService

The IOService class is a helper class which main function is to handle IO operations.

Properties

JsonFilePath	String to store the file path of the JSON output file
AbbreviationsFilePath	String to store the file path of the csv file containing the textspeak abbreviations

Methods

SaveMessageToJSON	Takes in a message object, converts it to JSON and appends it to an output file
GetAbbreviations	Returns a Dictionary with the textspeak abbreviation by reading them in from a csv file into a Dictionary where the key is the abbreviation and the value is the same abbreviation in its expanded form.
GetMessagesFromFile	Takes in the file path of an input text file and returns a list of strings where each string in the list corresponds to a message in the file.

MessageProcessingService

The MessageProcessingService class provides a sort of interface between the UI classes and the rest of the system. This class implements the Singleton design pattern to ensure that only one instance is ever created and that it remains in memory until the user exits the application.

Properties

Instance	A reference to the Singleton MessageProcessingService object
MessageList	List of all the messages processed in the current session
UnprocessedMessages	A list that stores the messages loaded from an input file.
UnprocessedMessageCounter	An integer that keeps track of index of the unprocessed message displayed in the UI.

Methods

Instance	Returns an existing Singleton MessageProcessingService object if one has been created or a new MessageProcessingService object otherwise.
ProcessMessage	Takes in the message header and the message body as input and returns a processed Message in the form of a Message object of type SMS, Tweet, Email or SIR depending on input. If the processing fails, the method returns null. This method used the MessageFactory class.
LoadUnprocessedMessageFromFile	Takes in the file path of an input text file and returns a list of strings where each string in the list corresponds to a message in the file.
LoadNextMessage	Returns message from UnprocessedMessages list at index UnprocessedMessageCounter.
GetMessageById	Takes a message id as input and returns the message object with that id from the list of processed messages (MessageList). If there isn't a message with that id in the list, returns null.

GetAllMessages	Return the list of processed messages (MessageList)
GetTweets	Return all Tweets in the MessageList
GetSmss	Return all SMSs in the MessageList
GetEmails	Return all Emails in the MessageList
GetSirs	Return all Significant Incident Reports in the MessageList
GetListOfMentions	Returns a list with all the tweeter ids mentioned in the processed tweets for the current session
GetTrendingList	Returns a dictionary containing all hashtags extracted from the tweets processed in the current session where the dictionary key is the hashtag and the value is the number of times the hashtag was found.
IsUnprocessedMessage	Takes in a message id and checks if a message with that id exists in the UnprocessedMessages list. Returns true if it does or false otherwise.
RemoveFromUnprocessedList	Removes a message with the passed id from the UnprocessedMessages list.

3. Testing

3.1. Testing Strategy

The testing will be carried out using a continuous testing strategy to guarantee that testing is done early, fast and often. Continuous testing allows the development and QA teams to cooperate very closely because individual modules are tested immediately after development. This means that defects can be identified and addressed early in the development process. To ensure a high testing coverage of the system, both unit testing and validation testing will be performed. However, because exhaustive testing is not possible, testing will focus mainly on the most important modules, i.e. modules that address the most requirements, with higher complexity and that are directly involved in the message processing functionality. Test cases will be identified and designed based on user requirements to guarantee requirements are met. The parameters passed in to the test cases will test edge cases to increase the chance of finding defects.

3.2. Testing Plan

3.2.1. Test items

The item being tested is the Napier Bank Message Filtering System (NBMFS) version 1.0.

3.2.2. Objectives

The objectives of the testing are: ensuring that the requirements defined in the requirements analysis are met and that the delivered product has a high quality.

3.2.3. Testing levels and scope

Unit Testing

Purpose: Test individual methods with the aim of identifying defects early in the development process and make sure the functionality provided by these methods behaves as specified in the requirements.

Scope:

Unit tests in scope (table below) include the methods directly involved in message processing. Other less important methods are not within the testing scope but might be indirectly tested when they are called by the tested methods.

ID	Tested Method	Class	Test acceptance criteria
U1	ProcessAbbreviations	SMS	Textspeak abbreviations in SMS messages are correctly expanded to their full form
U2	ProcessAbbreviations	Tweet	Textspeak abbreviations in Tweets are correctly expanded to their full form
U3	ProcessMentions	Tweet	Embedded twitted ids in tweets correctly identified and added to the mentions list
U4	ProcessHashtags	Tweet	Embedded hashtags in tweets correctly identified and added to the hashtags list
U5	ProcessHyperlinks	Email	Hyperlinks contained in email messages shall be added to a quarantine list and replaced by "<URL Quarantined>".
U6	CreateMessage	MessageFactory	Type of input message is correctly identified, and input is validated according to each type. If validation of input fails, an appropriate error message is generated.
U7	ProcessMessage	MessageProcessingService	Message type correctly identified, and input validated correctly. Entered message processed according to its type and saved to output file in JSON format.

Validation Testing

Purpose: The purpose of the validation testing is to test the NBMFS system as a whole, to ensure it meets the requirements specification, in particular, the high-level requirements and the non-functional requirements outlined in the use case diagram above (session 1.2 of this document). The validation tests will consist of tasks (scenarios) that will be completed using the application's UI.

Scope:

The scenarios for the validation tests were designed specifically to test the use cases outlined in the use case diagram, including the non-functional requirements (except usability). The decision of excluding Usability testing was based on the fact that in order to properly test the system's Usability, it would be necessary to have a group of users actually using the application, which is not feasible in this case.

ID	Tested Use-Case	Test scenario	Test acceptance criteria	
V1	Input message	Use the appropriate textboxes in the system's UI to input a message in the form of a message header and message body.	It is possible to input a message using the system's UI into appropriate textboxes in the form of message header and message body.	FUNCTIONAL
V2	Process message	Process a message by clicking the 'Process message' button after inputting a message into the appropriate text boxes.	The message is correctly processed and redisplayed in the UI in its processed form. The message is saved to an output file in JSON format.	
V3	Load message from file	Load in a message from an input file into the appropriate text boxes in the UI.	Message contained in the input file is loaded into the appropriate textboxes in the UI successfully.	
V4	View messages	User inputs and processes a small number of messages of different types and then clicks the "view messages" button to view the processed messages. User clicks the filtering buttons to filter messages by type.	All the messages processed by the user in the current session are displayed correctly on the window. The system filters the processed messages correctly, e.g. when the "view SMS messages" button is clicked, only the processed SMS messages are displayed.	
V5	View statistics	User inputs and processes a small number of messages of different types and then clicks the "view statistics" button to view the list of mentions, the trending list and the list of incident reports.	Displayed statistics are correct for the processed messages displayed during the current input session. The trending list must display the hashtags contained in the processed tweets and how many times each hashtag was used. The list of mentions must display all twitter ids mentioned in the tweets processed. The List of Significant incident reports must show the sort code and nature of incident for all the processed SIRs.	
V6	Validation [input message]	User inputs a message with invalid format and clicks the "Process" button.	Message is not processed, and an error message is displayed in the UI informing the user that an error has occurred.	NON-FUNCTIONAL
V7	Response time [Process message]	User inputs a message with invalid format and clicks the "Process" button.	The entered message must be validated, processed, redisplayed on screen in its processed form and saved to an output file in JSON format very quickly. This operation must seem instantaneous from the user perspective.	

3.2.4. Tasks and deliverables

Tasks

- Produce a test plan
- Identify test cases
- Write unit tests
- Run unit tests
- Carry out validation tests
- Analyse and evaluate test results

Deliverables

- Test plan (session 3.2 of this document)
- Unit tests (included in the NBMFS application code)
- Test cases and test results (session 3.3 of this document)

3.2.5. Test environment needs

Tests were performed in Microsoft Windows 10 Education, version 10.0.17134 build 17134. Unit tests developed and run in Microsoft Visual Studio Community 2017 version 15.9.2. Validation tests carried out on the NBMFS version 1.0 (version submitted).

Although not tested, it should be possible to run the same tests on any major versions of Windows or macOS as long as they are compatible with Microsoft Visual Studio 2017.

3.2.6. Tools

The testing facilities of visual studio were used to run the unit tests:

- Microsoft.VisualStudio.TestPlatform.TestFramework version 14.0.0.0
- Microsoft.VisualStudio.TestPlatform.TestFramework.Extensions version 14.0.0.0

3.2.7. Test schedule

Unit tests

As soon as the module being tested, and its sub-modules have been developed.

Validation tests

At the end of the development process to verify if the system is ready for release.

3.3. Test cases

3.3.1. Unit test cases

The table below contains the test cases used for the unit tests. **Test results in Appendix I – figure 5.**

Test ID	Test case	Input	Expected output	Actual output	Pass
U1	U1-1	LOL	LOL<Laughing out loud>	As expected	YES
	U1-2	LOL testU1_2 AKA testU1_2 B4N.	LOL<Laughing out loud> test U1_2 AKA<Also known as> test U1_2 B4N<Bye for now>.	LOL<Laughing out loud> test U1_2 AKA<Also known as> test U1_2 B4<Before>N.	NO
	U1-3	PLZPLZPLZ	PLZ<Please>PLZ<Please>PLZ<Please>	As expected	YES
U2	U2-1	LOL	LOL<Laughing out loud>	As expected	YES
	U2-2	LOL testU2_2 AKA testU2_2 B4N.	LOL<Laughing out loud> test U2_2 AKA<Also known as> test U2_2 B4N<Bye for now>.	LOL<Laughing out loud> test U2_2 AKA<Also known as> test U2_2 B4<Before>N.	NO
	U2-3	PLZPLZPLZ	PLZ<Please>PLZ<Please>PLZ<Please>	As expected	YES
U3	U3-1	Hi @test, how are you?	1 mention in list of mentions: [@test]	As expected	YES
	U3-2	Hi, how are you?	List of mentions empty []	As expected	YES
	U3-3	Hi @test1 @test2 @test3 @test4 @test5, how are you?	5 mentions in list of mentions: [@test1 @test2 @test3 @test4 @test5]	As expected	YES
U4	U4-1	Hi, just arrived in Mexico. #travel #thegoodlife	2 hashtags in list of mentions: [#travel #thegoodlife]	As expected	YES
	U4-2	Hi, just arrived in Mexico.	List of hashtags empty []	As expected	YES
	U4-3	Hi, just arrived in Mexico. #travel#thegoodlife	2 hashtags in list of mentions: [#travel #thegoodlife]	1 hashtag in list of mentions: [#travel#thegoodlife]	NO
U5	U5-1	Hi, Please visit www.tesco.com/discount!	Hi, Please visit <URL Quarantined>! 1 hyperlink in list of quarantined URLs: [www.tesco.com/discount]	As expected	YES
	U5-2	Hi, Please visit https://tesco.com	Hi, Please visit <URL Quarantined> 1 hyperlink in list of quarantined URLs: [https://tesco.com]	As expected	YES

Mendes, Pedro
NAPIER BANK MESSAGE FILTERING SYSTEM

	U5-3	Hi, please visit http://tesco.com/discount and www.Tesco.com/bargains!!!	Hi, please visit <URL Quarantined> and <URL Quarantined>!!! 2 hyperlinks in list of quarantined URLs: [http://tesco.com/discount www.Tesco.com/bargains]	As expected	YES
U6	U6-1	Header: S000000001 Message body: +44829918273\nHi Bob, we have a great sale on at the moment, 50 % off!	Input message correctly identified as SMS. SMS object returned. Processing error null.	As expected	YES
	U6-2	Header: T000000001 Body: @BestCompany\nCongratul ations to @RobJones in accounting for winning our #NFL football pool!	Input message correctly identified as Tweet. Tweet object returned. Processing error null.	As expected	YES
	U6-3	Header: E000000001 Body: noreply@topshop.com\nwelc ome\nThank you for signing up to Topshop emails.\nYou'll now be the first to know about our latest collections.	Input message correctly identified as Email. Email object returned. Processing error null.	As expected	YES
	U6-4	Header: E000000002 Body: police@metro.uk\nSIR 12/05/18\n22-72- 87\nCustomer attack\nA customer has just been attacked in our branch in Edinburgh.	Input message correctly identified as significant incident report. SIR object returned. Processing error null.	As expected	YES
	U6-5	Header: 012345 Body: +44829918273\nHi Bob, we have a great sale on at the moment, 50 % off!!	Message validation fails and method returns null. Processing error: Invalid message id. Please enter a valid message id.	As expected	YES
U7	U7-1	Header: S000000001 Body: +44829918273\nHi HRU Bob, we have a great sale on at the moment, 50 % off! Visit www.topshop.com	Input message correctly processed as SMS message. The textspeak abbreviation HRU gets converted to its full form but the hyperlink www.topshot.com is not converted to <URL Quarantined>. SMS object returned. No error message occurs.	As expected	YES

Mendes, Pedro
NAPIER BANK MESSAGE FILTERING SYSTEM

	U7-2	Header: E000000001 Body: noreply@topshop.com\nwelcome\nHRU? Thank you for signing up to Topshop emails.\nVisit us at http://topshop.com	Input message correctly processed as Email. Testspeak abbreviation HRU is not converted to its full form (should only happen for SMS and tweets). The hyperlink http://topshop.com is converted to <URL Quarantined>. The method returns an Email object. No error message occurs.	As expected	YES
	U7-3	Header: E000000001 Body: +44829918273\nwelcome\nHRU? Thank you for signing up to Topshop emails.\nVisit us at http://topshop.com	Input message validation fails because the sender of email messages must be a valid email address and an international phone number was input instead. The message processing fails and a null object is returned. Processing error occurs: Invalid email sender. The email sender must be a valid email address.	As expected	YES

3.3.2. Validation test cases

Test ID	Test case	Input	Test result	Tested on	Pass
V1	V1-1	A valid (correct format) unprocessed message	System UI allows easy input of messages using 2 textboxes. One textbox to input the message header (id) and another to input the message body. See Appendix II – figure 6	23/11/2018	YES
V2	V2-1	A valid (correct format) unprocessed message	A message is displayed to inform the user the entered message has been processed successfully. The message is correctly categorized as a sms message and processed accordingly. The textspeak abbreviation (HRU) is correctly converted to its full form (HRU<How are you?>). The message is redisplayed on screen in its processed form and saved to an output file in JSON format. See Appendix II – figure 7	23/11/2018	YES
V3	V3-1	Input is a text file containing one unprocessed message.	Message in the input file provided by the user is loaded into the appropriate text boxes in the UI. See Appendix II – figure 8	23/11/2018	YES
V4	V4-1	10 unprocessed messages of different types: <ul style="list-style-type: none"> • 3 SMS messages • 3 Tweets • 3 Emails • 1 Significant incident report See Appendix II – figure 9	The processed messages can be viewed in their processed form after an input session, and the filtering of messages by type works as expected. The message searching by id functionality also working as intended. See Appendix II – figure 10	23/11/2018	YES
V5	V5-1	Same as test V4-1 See Appendix II – figure 9	System displays the trending list, the list of mentions and the list of significant incidents matching the messages processed for the session. See Appendix II – figure 11	23/11/2018	YES
V6	V6-1	An invalid (incorrect format) unprocessed message	Message is not processed, and an error message is displayed on screen informing the user why the message processing failed. See Appendix II – figure 12	23/11/2018	YES
V7	V7-1	A valid (correct format) unprocessed message	No noticeable delay between the button “Process message” is clicked and the message is processed. The message processing is virtually instantaneously.	23/11/2018	YES

4. Version Control Plan

A version control system is a software package used for file storage that keeps track of all changes made to a file, allowing operations such as view the state of a file at a certain date in the past, compare different versions of the same document and see who was responsible for making the change. The figure 3 below shows a diagram of how a version control system can support the Agile development process of a software application.

For the Agile development of the NBMFS, GIT will be the chosen version control system. The main advantages of using GIT for this project are:

- Support the collaboration between the developers working on the project
- Facilitate the collaboration between the developing and the QA teams.

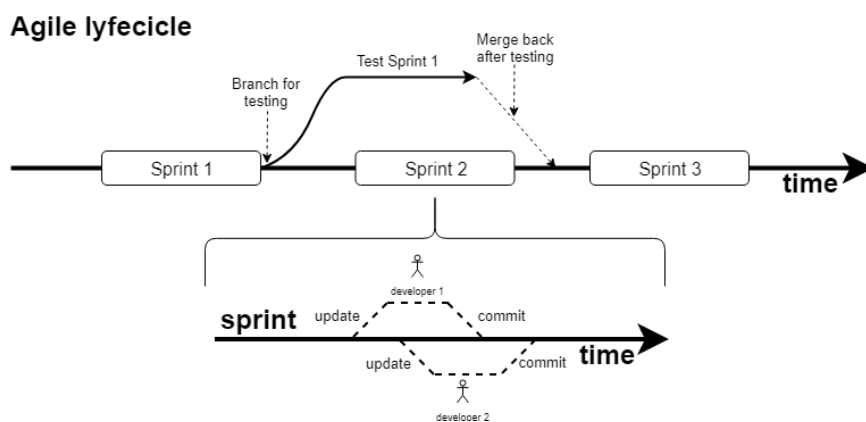


Figure 3. Diagram showing how a version control system can be used to support an Agile development.

4.1. Collaboration between developers

In Agile development, when a new sprint starts, developers in the team will be assigned tasks to implement the features defined in the sprint backlog. Using Git, developers working on the project will be able to check out the code from the central remote repository into their local repositories. Then, each developer can make the changes to the files in their local repositories to implement the features assigned to them. This means that they can work in parallel, even if they need to make changes to the exact same file. When a developer is finished making changes to a file, they can commit the changes to the central remote repository and every other developer will be able to see these changes once they update their local repositories i.e., once they check out the code in the remote repository to their local repository. The greatest advantage of using the version control system in this way is that developers do not have to wait for other developers to finish working on a file before they can work on the file, greatly increasing the productivity of the team. Additionally, any file can easily be reverted to a previous version if needed. This feature can be extremely useful when, for example, a bug is introduced in the system by mistake and an urgent fix is required.

4.2. Collaboration between the developing and the QA teams

An extremely important aspect of software project management is to manage resources efficiently so that no one has to sit idle waiting for someone else before they can do their job. In an Agile project with tight deadline and time constraints, it is not realistic to have the QA team waiting until the whole project is developed to start the testing. Instead, testing must be carried out during the development process so that defects are discovered early, and the software is delivered on time, according to requirements and with high quality.

Figure 3 shows how the developing and the QA teams can collaborate using a version control system. After a development sprint (sprint 1), the QA team creates a testing branch, where they can test sprint 1 on a stable build that is not changed by the developing team. If the QA team finds any defects after running the tests, these defects can be fixed in the testing branch and, when they are finished, the testing branch is merged back into the main branch so the fixes are incorporated into the main build. At the same time that testing of sprint 1 is taking place, the development team can start working on the next sprint in the main branch. The testing of branch 1 and development of branch 2 occurs at the same time, in parallel, but independently. This iteration process of branching out, testing and merging back in continues until the project is finished and the application is ready for release.

5. Evolution Strategy

Software evolution is the process of continuously maintaining or updating a software system to keep it useful throughout the system's lifetime. After the initial development and release of a system, the system evolution process is responsible for managing change. Changing is inevitable because bugs are discovered and needs fixing, new requirements emerge that need to be implemented and qualities such as reliability, performance or usability might need to be improved.

Figure 4 shows the evolution strategy to be implemented in the maintenance of the NBMFS.

Change request

Any new change is initiated by a changing request. A change request might be a new requirement to implement new functionality into the system or it might be a request to fix a new discovered bug.

Impact analysis

An impact analysis of any new change requested must be performed. This analysis must look at risks and benefits of making the required change by identifying what components of the system are affected by the change, how much time would it take, how much would it cost and what benefits would the change bring? After the impact analysis, the change request is either approved or rejected. If approved, the change request progresses to the next phase, the release planning.

Release planning

Release planning looks at what changes will be included in the next release. This will dependent on the importance and urgency of the change request.

Change implementation

This is the evolution phase that deals with the design, implementation and testing of new system changes. Change implementation requires that the developers involved in implementing the change understand the system structure and functionality, and how the new change might affect the system.

Change release

When a change is successfully implemented and passes all testing, it is ready to be released. A new release consists of a new version of the system that includes a series of changes i.e., new features and code fixes.

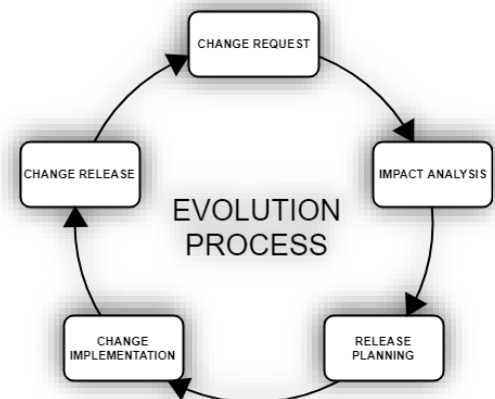


Figure 4. Diagram of the process used in the NBMFS evolution.

5.1. Evolution and maintenance predictions

Predicting the evolution and maintenance of a system is not always easy because new requirements change rapidly and often there are several stakeholders involved in requesting new changes. However, for the NBMFS there are a few changes that would likely be required in the near future. These are outlined below:

- Implementation of an access control system to manage access to the application in the form of a login system. Such a feature would be extremely important for security reasons.
- Implementation of a persistent data storage system in the form of a database so that data is not lost between sessions. At the moment a user can only view messages processed during the current session. To be really useful, the system must allow the user to view messages processed in previous sessions.
- New features, such as the ability to edit or delete a previously processed message will likely be required. Otherwise, correcting user input mistakes is not possible.
- Improve system UI to improve the system usability. This is probably not the most important and urgent change because the system is fairly usable in its current stage. However, with the implementation of new features in the system it is likely that the Usability will worsen becoming a problem that will need to be addressed.
- After the system is released and starts being used by real users, it is expected that new errors and bugs will be discovered. Depending on the severity and impact, these have to be fixed with more or less urgency.

5.2. System maintainability and predicted costs

In its current state, the system is fairly simple, the code quality is fairly high and it is well documented so it should be easily maintainable. However, it is known that systems tend to become more complex and less maintainable with time, hence, the maintenance costs are likely to increase. In the table below is the predicted time and cost estimates of maintaining the system to implement the changes outlined in the previous section of this document (section 5.1). The cost estimates assume an hourly rate of £20/hour.

Change Request	Estimated Time	Estimated Cost
Access control system	15 hours	£300
Database	20 hours	£400
New features	10 hours	£200
Usability improvement	15 hours	£300
TOTAL	60 hours	£1200

An estimation of the time and cost for bug fixing is not given because this will be totally dependent on the number and severity of bugs found so an accurate estimate is not possible.

Appendix I – Results of Unit Test Cases

Run All		Run...		Playlist : All Tests	▼
Napier Message Service (23 tests) 3 failed					
▲	✖	NBMFSTest (23)			210 ms
▲	✖	NBMFSTest (23)			210 ms
▲	✖	UnitTestU1 (3)			80 ms
	✔	U1_1			7 ms
	✖	U1_2			72 ms
	✔	U1_3			< 1 ms
▲	✖	UnitTestU2 (3)			1 ms
	✔	U2_1			< 1 ms
	✖	U2_2			1 ms
	✔	U2_3			< 1 ms
▲	✔	UnitTestU3 (3)			1 ms
	✔	U3_1			1 ms
	✔	U3_2			< 1 ms
	✔	U3_3			< 1 ms
▲	✖	UnitTestU4 (3)			1 ms
	✔	U4_1			< 1 ms
	✔	U4_2			< 1 ms
	✖	U4_3			< 1 ms
▲	✔	UnitTestU5 (3)			11 ms
	✔	U5_1			10 ms
	✔	U5_2			< 1 ms
	✔	U5_3			< 1 ms
▲	✔	UnitTestU6 (5)			4 ms
	✔	U6_1			< 1 ms
	✔	U6_2			< 1 ms
	✔	U6_3			1 ms
	✔	U6_4			2 ms
	✔	U6_5			< 1 ms
▲	✔	UnitTestU7 (3)			108 ms
	✔	U7_1			97 ms
	✔	U7_2			11 ms
	✔	U7_3			< 1 ms
Group Summary					Copy All
Grouped by Hierarchy: NBMFSTest					
Duration: 0:00:00.2100516					
✖ 3 Tests Failed					
✔ 20 Tests Passed					

Figure 5. Unit test results for the NBMFS testing.

Appendix II – Results of Validation Test Cases

Test case V1-1

The screenshot displays the NBMFS (Napier Bank Message Filtering System) web application. On the left is a dark sidebar with navigation links: "Input Messages", "View Messages", and "View Stats". The main content area has a dark background with the "NBMFS" logo and a green speech bubble icon in the top right. Two red rectangular boxes highlight the input fields: the first box encloses the "Message ID" label and a text input containing "E000000001"; the second box encloses the "Message body" label and a larger text input containing a sample email body. Below these inputs, a status message reads "There is no unprocessed messages." with a "Load next" button. Further down, there is a section for loading messages from a file, with a text input for the file path, a "Browse" button, and a "Load messages" button. At the bottom, a "Processed message" section contains a large, empty rectangular box.

Figure 6. Appropriate text boxes allow users to enter a message in the form of a message header (message id) and message body.

Test case V1-2

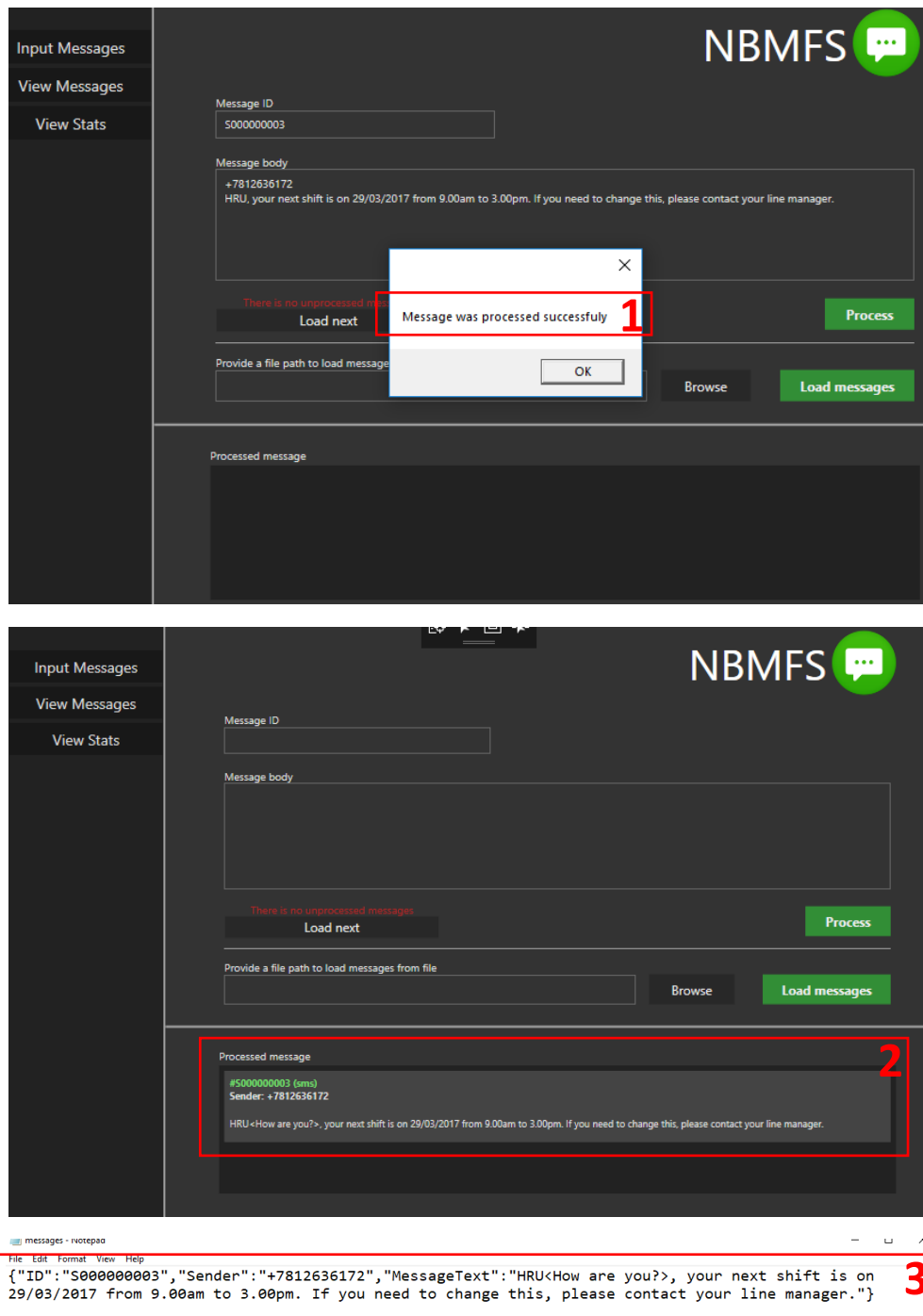


Figure 7. 1 - Message is displayed to inform the user the message has been processed successfully. 2 – The message is redisplayed on screen in its processed form (note the textspeak abbreviation has been converted to its full form). 3 -The processed message is saved to an output file in JSON format.

Test case V1-3

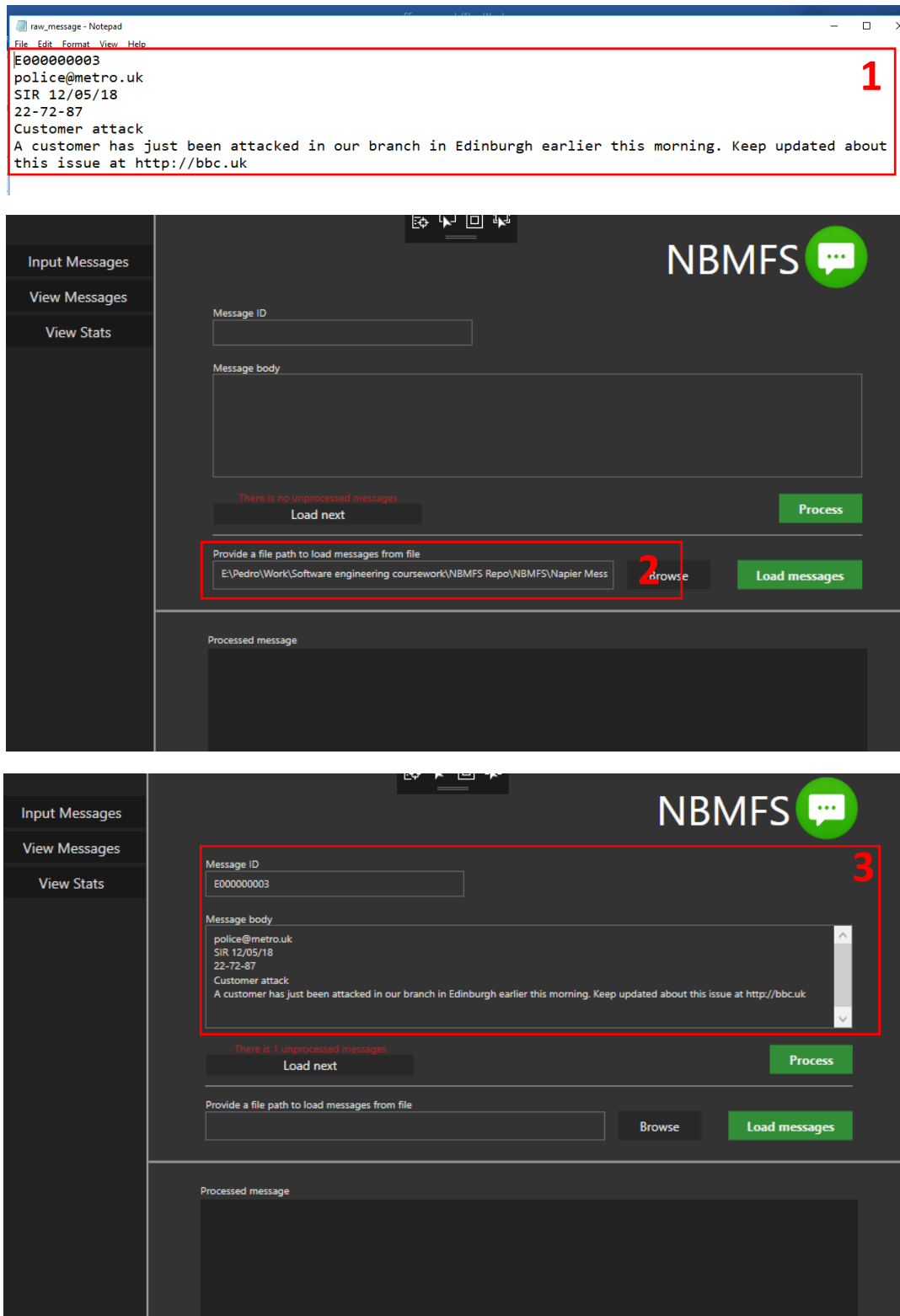
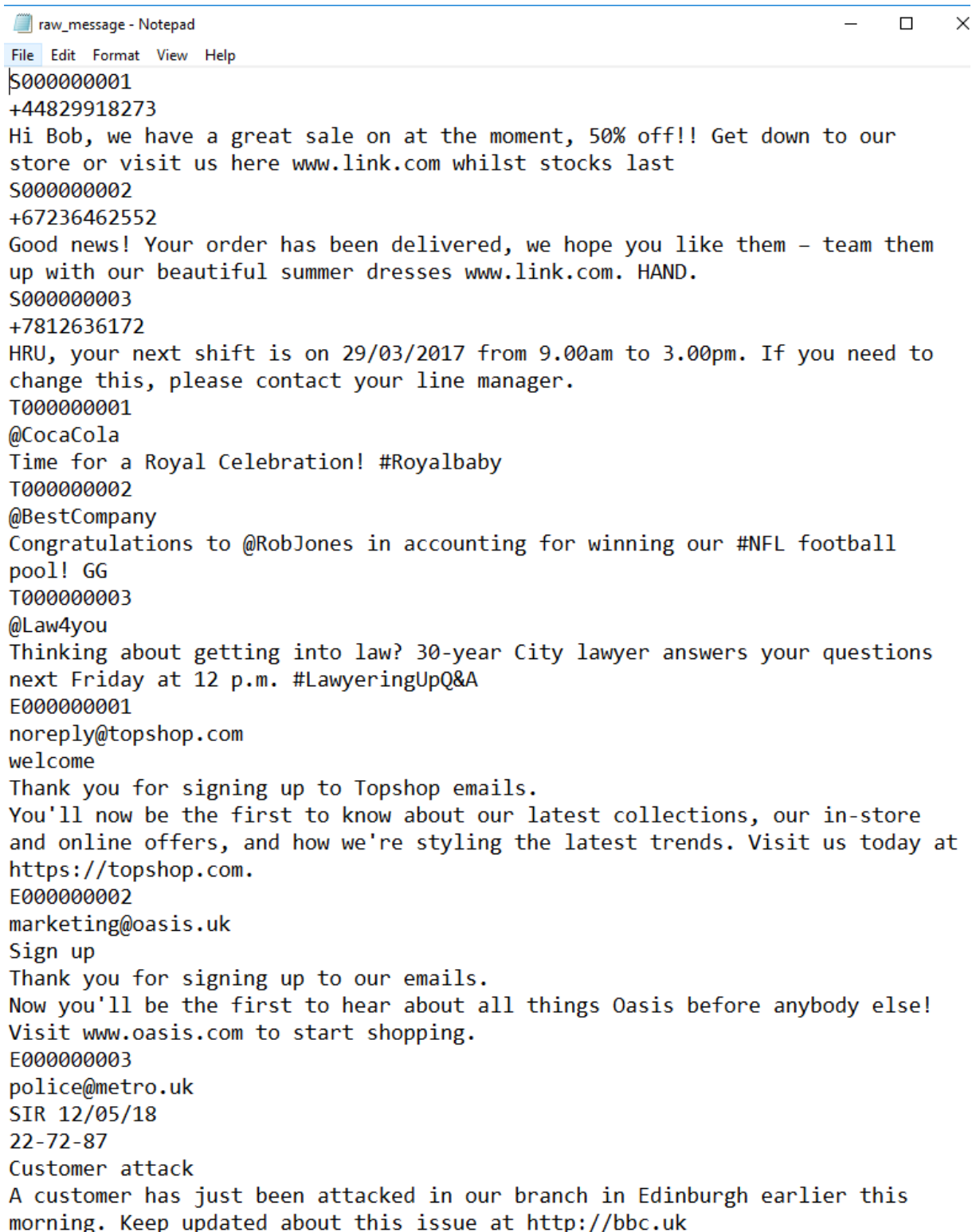


Figure 8. 1 - Message from input file. 2 - User provide the message file path and clicks load the "Load messages" button. 3 - The message is loaded into the UI textboxes and is ready to be processed.

Test case V1-4



raw_message - Notepad

File Edit Format View Help

S000000001
+44829918273
Hi Bob, we have a great sale on at the moment, 50% off!! Get down to our store or visit us here www.link.com whilst stocks last

S000000002
+67236462552
Good news! Your order has been delivered, we hope you like them - team them up with our beautiful summer dresses www.link.com. HAND.

S000000003
+7812636172
HRU, your next shift is on 29/03/2017 from 9.00am to 3.00pm. If you need to change this, please contact your line manager.

T000000001
@CocaCola
Time for a Royal Celebration! #Royalbaby

T000000002
@BestCompany
Congratulations to @RobJones in accounting for winning our #NFL football pool! GG

T000000003
@Law4you
Thinking about getting into law? 30-year City lawyer answers your questions next Friday at 12 p.m. #LawyeringUpQ&A

E000000001
noreply@topshop.com
welcome
Thank you for signing up to Topshop emails.
You'll now be the first to know about our latest collections, our in-store and online offers, and how we're styling the latest trends. Visit us today at <https://topshop.com>.

E000000002
marketing@oasis.uk
Sign up
Thank you for signing up to our emails.
Now you'll be the first to hear about all things Oasis before anybody else! Visit www.oasis.com to start shopping.

E000000003
police@metro.uk
SIR 12/05/18
22-72-87
Customer attack
A customer has just been attacked in our branch in Edinburgh earlier this morning. Keep updated about this issue at <http://bbc.uk>

Figure 9. Input messages for validation test V4-1

Mendes, Pedro
NAPIER BANK MESSAGE FILTERING SYSTEM

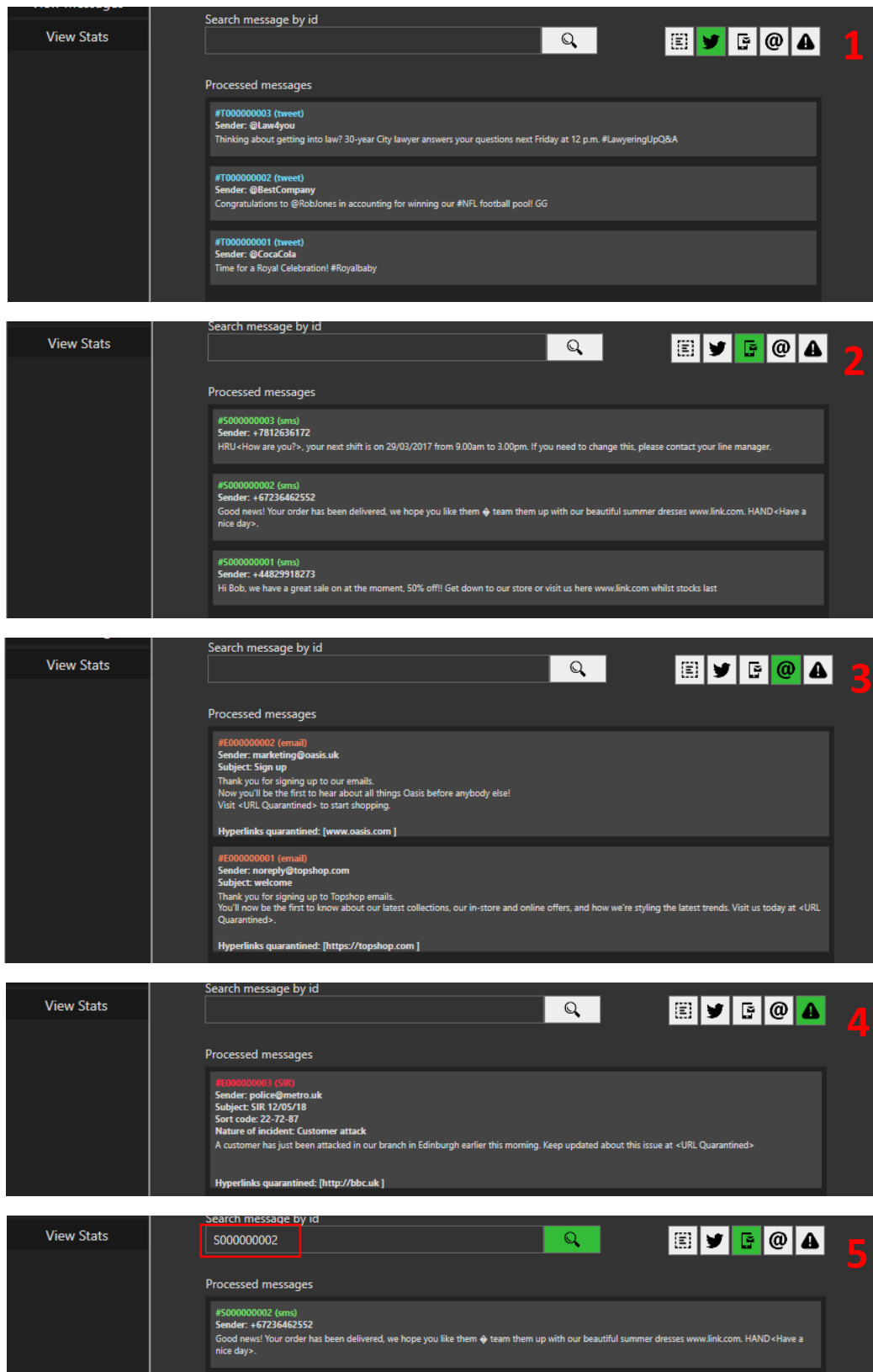


Figure 10. 1 - Filtering tweets only. 2 - Filtering SMS messages only. 3 - Filtering emails only. 4 - Filtering SIRs only. 5 - Searching message by id.

Test case V1-5

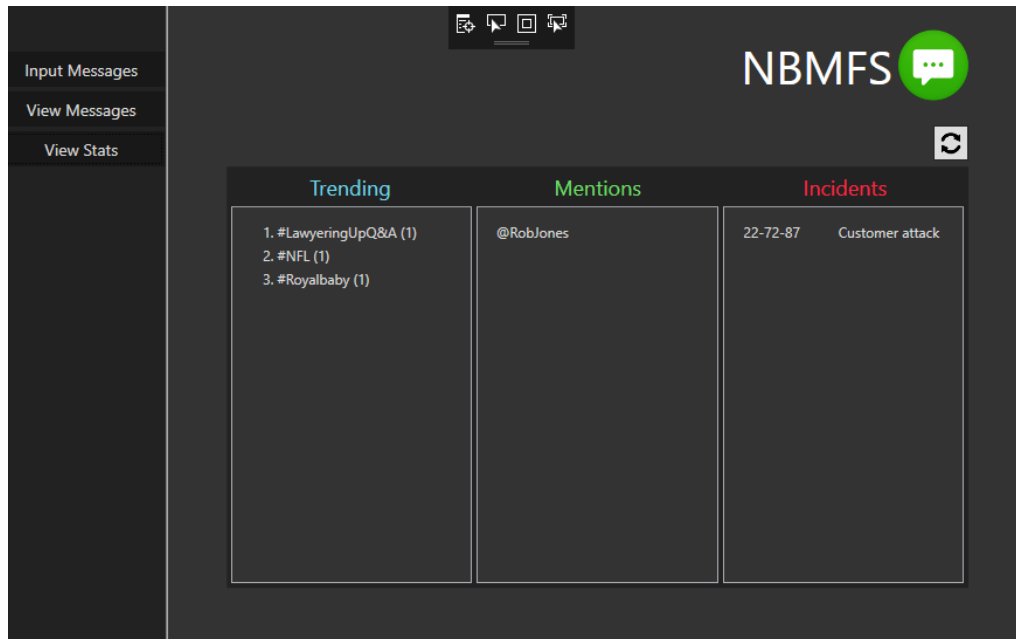


Figure 11. Trending list, list of mentions and list of incidents for the messages shown in figure 6.

Test case V1-6

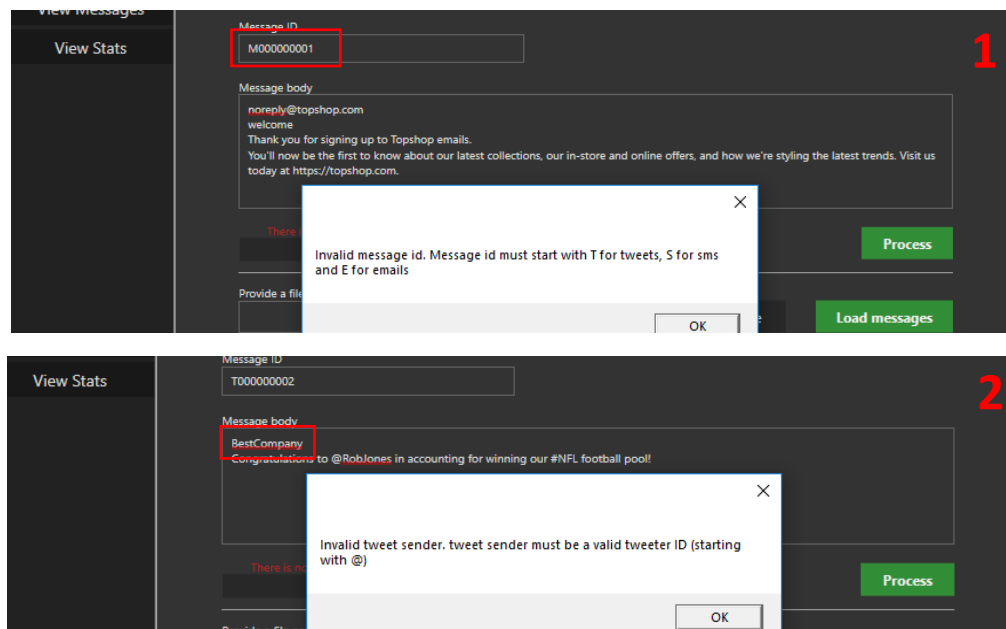


Figure 12. 1 – Message with an invalid message id is entered and an error message is displayed. 2 – A Tweet with an invalid sender is entered is an error message is displayed.