Software Engineering Coursework

40283288| SET09102 | 23/11/2018
Lecturer: Xiaodong Liu

Introduction

This report contains the documentation for the SET09102 coursework, the Napier Bank Message Filtering Service. The coursework involved the development of an application prototype, which I have decided to implement into a WPF application using C#. The application uses various filters to correctly process the data into their corresponding message type.

Requirement Analysis

The client has requested the development of a service that will validate, sanitise and categorise incoming messages to Napier Bank. All messages have a message header comprising of a Message ID (Message-type “S”, “E”, or “T” followed by 9 numeric characters, e.g. “T123456789”) followed by the body of the message

The system must deal with the following three message types based on the message body:

- SMS Messages
  The SMS message body contains the Sender in the form of an international telephone number and the Message Text that is a maximum of 140 characters. The text message also may contain “textspeak abbreviations”, which are supplied in a CSV file. The abbreviations must be identified and expanded to their full form enclosed in a “<>” tag.
- Email Messages
  Emails are of two types: Standard email messages and Significant Incident Reports. Both types have a Sender in the form of an email address.
  Standard email messages have a subject that is maximum 20 characters long, followed by a maximum of 1028 characters message body.
  Significant Incident Reports have the subject in the form “SIR dd/mm/yy” and will start the message text with a six digit sort code on the first line, followed by the “Nature of incident” on the next line. Nature of incident can be one of the following:

Theft
Staff Attack
Raid
Customer Attack
Staff Abuse
Bomb Threat
Terrorism
Suspicious Incident
Intelligence
Cash Loss

In addition to these rules every email can contain URLs (in the form of "http://" or "https://") that need to be removed and written to a quarantine list and replaced by "<URL Quarantined>" in the message body.

- Tweet Message
  The tweet message body contains a Sender in the form of a Twitter ID that starts with "@" followed by a maximum of 15 characters. The tweet text is a max of 140 characters and contains the same textspeak abbreviations as SMS. In addition to this, they might also contain hashtags (e.g. #software) or other Twitter IDs in the form of "mentions". Hashtags will be added to a hashtag list that count the number of uses to produce a trending list. The mentions will be added to their own mention list.

Design

WPF User Interface

While designing the application's user interface, the Visual Studio WPF editor was used as the starting point to give the developer and idea about the scope and structure of the application.

Main Window

The Main Window [Figure 1 Main Window] of the application is where the user can view the full list of submitted messages and their various properties. Not all fields are used by all the messages, but a simple unified list helps the user by not overwhelming the screen with unwanted options. The main window is also where the user can view the trending and mention lists as well as go and submit new messages.

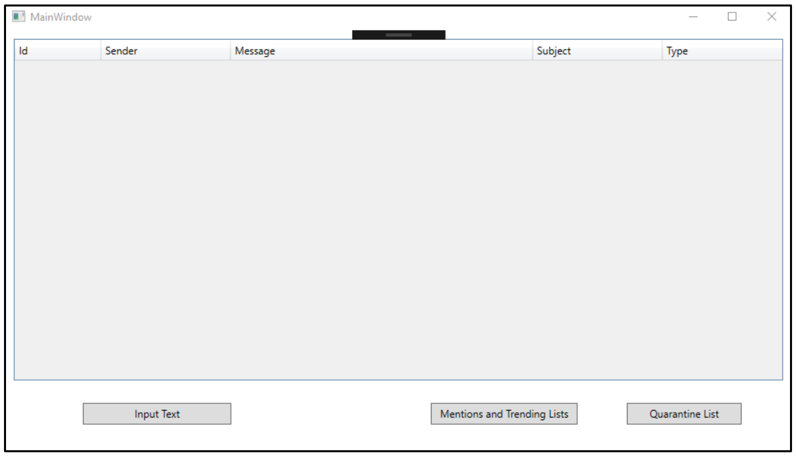


Figure 1 Main Window

Message Input

The Message Input Window [Figure 2] is where the user can input their various messages. The user can choose between the three message types, depending on the type different options might appear [Figure 3].

The Email input is the most complex one, and it can take the twi different types of emails: SIR and Standard. If the user chooses SIR the subject must be entered in the dd/mm/yy form. Another selection can be made for the nature of the incident and the user can type in a sort code as well. If standard email is selected the user can only input in the sender(in the form of an email address), the Subject (in any form they wish) and the message body. Regardless of email type, the system will then filter and quarantine the entered URLs.

If the user selects the message type as either SMS or Tweet, only a sender (int their respective format) and message body can be inputted in. The system will then expand the abbreviations for both types and count the hashtags and mentions for tweets.

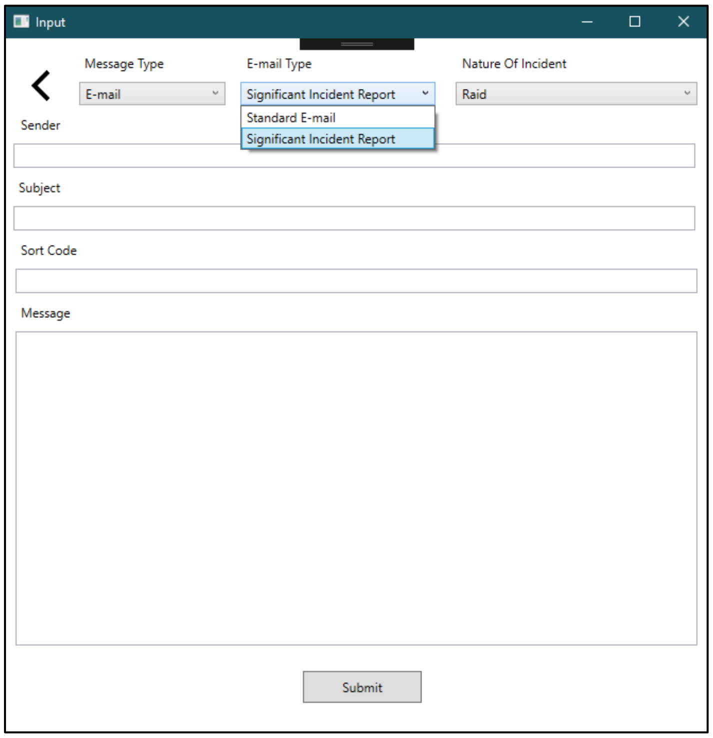


Figure 2 Input Window, No Selection made

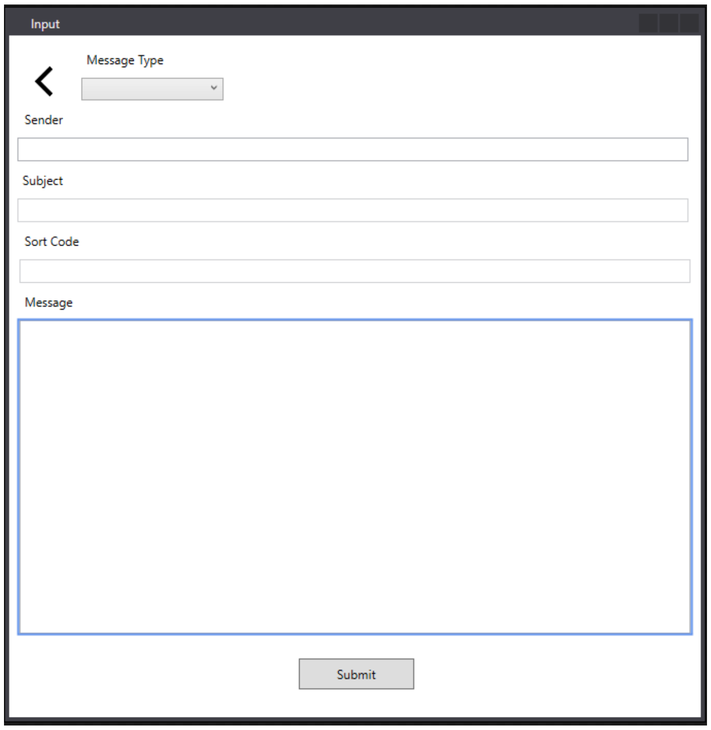


Figure 3 Choosing Email Options

Use Case Diagram

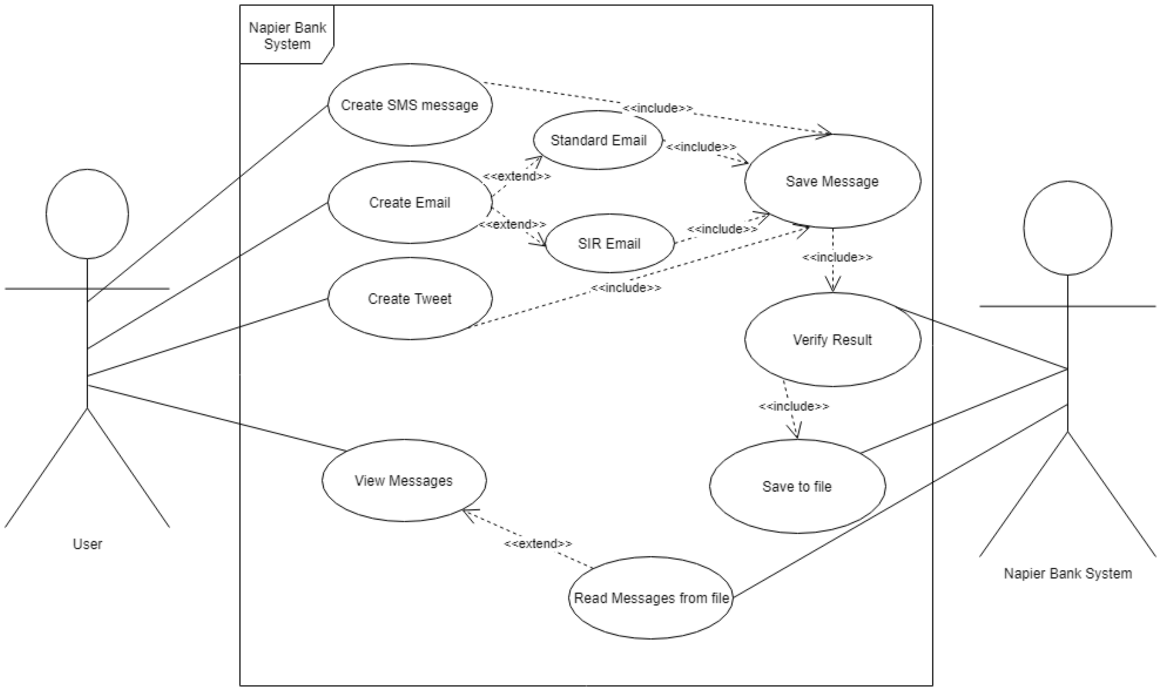


Figure 4 The Use Case Diagram

The Use Case Diagram [Figure 4] was created as part of the specification requirements. The diagram involves two actors: the user and the application system. The user can create the three types of messages and view them afterwards. The system manages the validity of the data entered and saves the messages to a file if they are valid. When the user wants to view the list of messages, the system reads them from the file and displays them as appropriate.

Class Diagram

The Class Diagram [Figure 5] shows the Classes that were built for the app. The WPF window classes were not included in the diagram. Below the diagram you can see a brief explanation of the purpose of the classes.

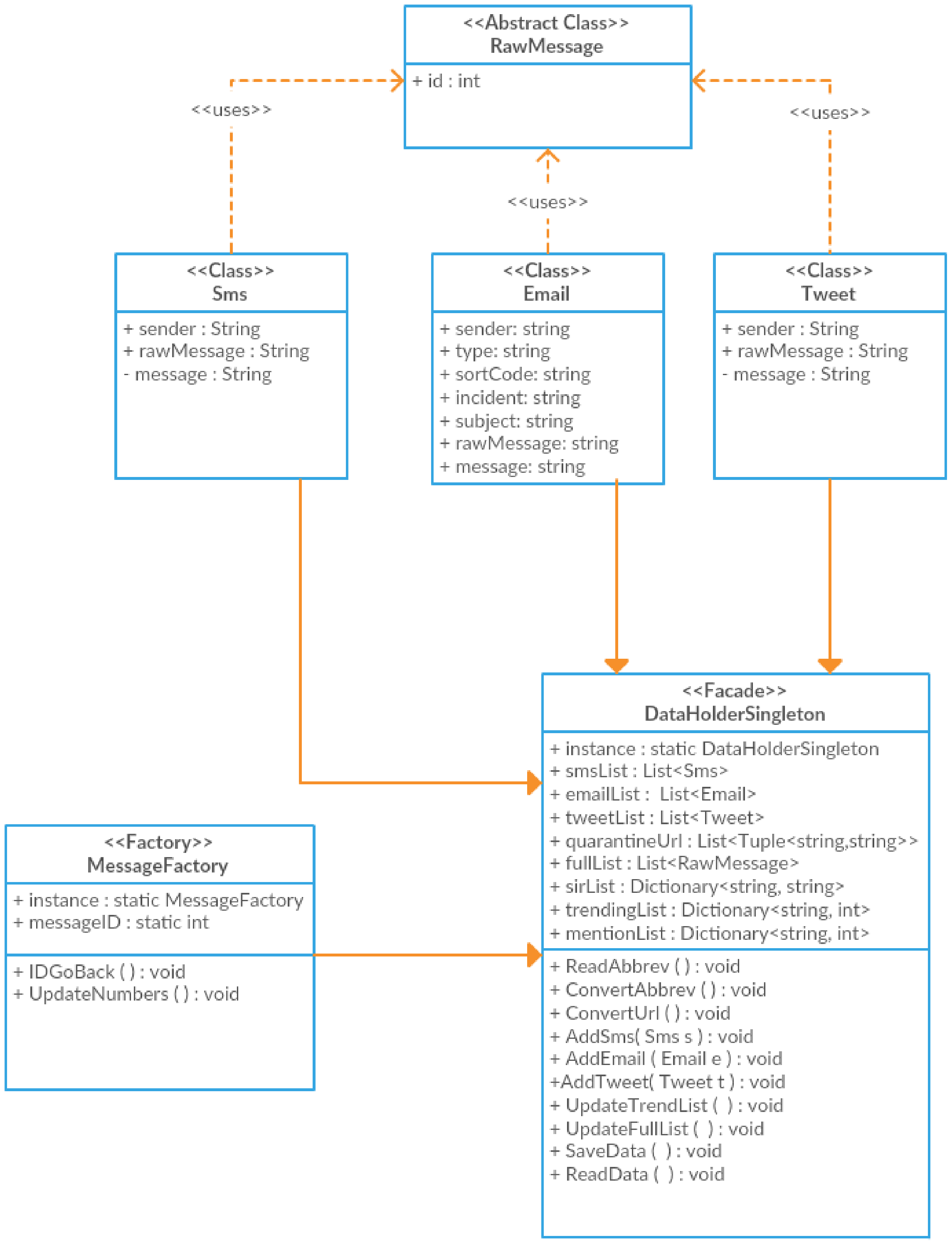


Figure 5 The Class Diagram

Class Name	Use
RawMessage	Abstract parent class of Tweet, Sms and Email
Sms	Contains the attributes of an SMS message and the validation checks
Email	Contains the attributes of an Email message and the validation checks
Tweet	Contains the attributes of a Tweet message and the validation checks
MessageFactory	Used to assign IDs for the messages
DataHolderSingleton	The interface that stores the various message lists and writes or reads data from file

Functional Requirements

Based on the scenario and requirements listed in the coursework, they have been developed into functional requirements. In order to consider the project successful, there requirements need to be implemented and pass functional testing.

- *Message Processing*: The user must be able to submit any of the three message types, Email, SMS or Tweet. The system must then categorise them as suited to make them accessible for viewing.
- *Textspeak abbreviations*: The system must be able to expand textspeak abbreviations.
- *Email Link Quarantine*: URLs need to be quarantined and removed from all emails, being replaced with "<<URL Quarantined>>".
- *Significant Incident Report Emails*: These types of emails must have additional rules from standard emails. A Significant Incident Report subject must be of the form "SIR dd/mm/yy", followed by a sort code and the nature of the incident.
- *Hashtags and mentions*: Any hashtags or mentions that are part of a tweet will be noted and counted to produce lists of how many times they have been used.
- *Ability to read and write data from specified JSON file*: The application will automatically read and write the specified JSON file from the path coded into the application. There are three JSON files used to store SMS lists, Email lists and Tweet lists respectively. The application will also read in the supplied CSV of abbreviations.

Non-Functional Requirements

Reliability

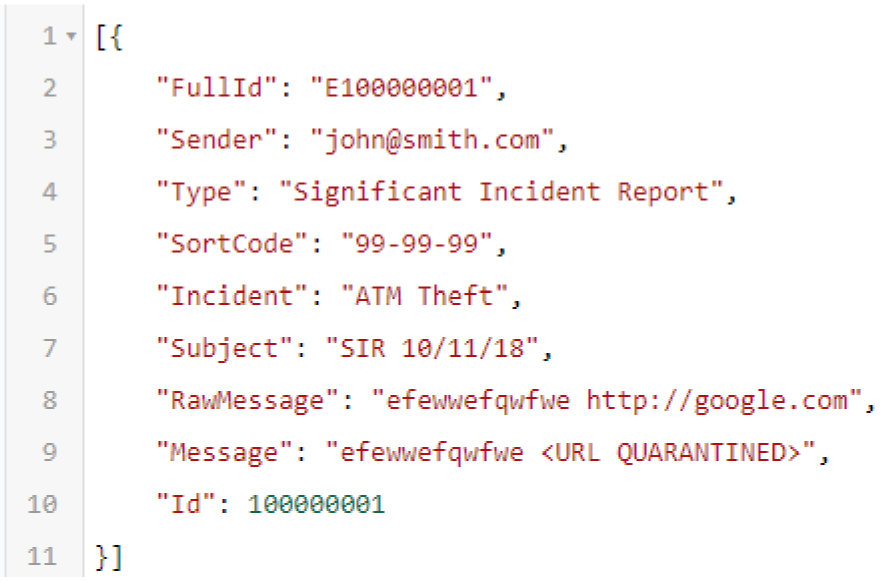
The software should be capable of handling various errors. To satisfy this, the software has various validation checks on the input fields to stop the submission of invalid messages. To verify the validity of international numbers, sort codes and URLs, Regex rules are used to check them, as well as try-catch methods throughout the submission stage to prevent the system from crashing.

Storage and Portability

The software should be usable on any Windows OS that is running Microsoft .Net 4.6.1 or higher. The application will create its own JSON file for storing the various lists if no file is present. The file must respect the JSON file scheme if a user would like to write their own file [Figure 6].

Documentation

The documentation is an important part of the requirements mentioned in the coursework brief. A Software Engineering report will be provided, that documents the design, development and testing of the system.



```json
[{
    "FullId": "E100000001",
    "Sender": "john@smith.com",
    "Type": "Significant Incident Report",
    "SortCode": "99-99-99",
    "Incident": "ATM Theft",
    "Subject": "SIR 10/11/18",
    "RawMessage": "efewwefqwfwe http://google.com",
    "Message": "efewwefqwfwe <URL QUARANTINED>",
    "Id": 100000001
}]
```

Figure 6 Example JSON Email List containing one element

User Hardware and Software Requirements

To use the software, the user must have the following system requirements:

- Device running Windows 7 or above
- .Net Framework 7.6.1 or above

Testing

Test Plan

The testing plan has two parts: Functional Testing and Test Cases. The test cases focus on a single window of the application at a given time, and tests the various options and combinations that can happen. If any of the tests fail, a fix will be applied and the test redone. If a fix cannot be found, the test case will be marked as a Fail.

Test Schedule

The following test schedule shows the pass or fail of the test cases created and the date they were tested:

Name	Description	Criteria	Pass/Fail	Date of Testing
Testcase 1	Main Window testing	Pass all tests	Pass	16/11/2018
Testcase 2	Input Window SMS testing	Pass all tests	Pass	16/11/2018
Testcase 3	Input Window Email testing	Pass all tests	Pass	19/11/2018
Testcase 4	Input Window Tweet testing	Pass all tests	Pass	19/11/2018

Test Cases

- Testcase 1, Main Window Testing

#	Action	Expected Result	Actual Result	Pass/Fail	Notes
1	Display Messages List	Displays messages	Displays messages	Pass	
2	Double clicking specific message	Displays all the information of the message	Displays all the information of the message	Pass	

3	Click “Input Text” button	Change window to the input window	Change window to the input window	Pass	
4	Click “Mentions and Trending Lists” button	Displays the mentions and trending lists in a new window	Displays the mentions and trending lists in a new window	Pass	
5	Click “Quarantined URLs” button	Displays the list of quarantined URLs in a new window	Displays the list of quarantined URLs in a new window	Pass	

- Testcase 2, Input Window SMS testing

#	**Action**	**Expected Result**	**Actual Result**	**Pass/Fail**	**Notes**
1	Leave all fields blank	Invalid action message	Invalid action message	Pass	
2	Input international number starting with “+44”	Record is saved	Record is saved	Pass	
3	Input international number starting with “0044”	Record is saved	Invalid action message	Fail	Regex rule changed to accept either “+” or “00” for international numbers
3b	Input international number starting with “0044”	Record is saved	Record is saved	Pass	Test 3 is now successful
4	Entering more than 140 characters in message body	Invalid action message	Invalid action message	Pass	

- Testcase 3, Input Window Tweet testing

#	**Action**	**Expected Result**	**Actual Result**	**Pass/Fail**	**Notes**
1	Leave all fields blank	Invalid action message	Invalid action message	Pass	

2	Enter sender without starting with "@" symbol	Invalid action message	Invalid action message	Pass	
3	Entering more than 140 characters in message body, or more than 18 characters in sender	Invalid action message	Invalid action message	Pass	

- Testcase 4, Input Window Email testing

#	Action	Expected Result	Actual Result	Pass/Fail	Notes
1	Leave all fields blank	Invalid action message	Invalid action message	Pass	
2	Enter incomplete email address in sender field	Invalid action message	Record is saved	Fail	New address check method using the built-in c# class EmailAddressAttribute
2b	Enter incomplete email address in sender field	Invalid action message	Invalid action message	Pass	
3	Leaving email type blank	Invalid action message	Invalid action message	Pass	
4	Entering more than 1028 characters in the message body, or more than 20 in Subject	Invalid action message	Invalid action message	Pass	
5	Not entering the correct format for subject in a SIR	Invalid action message	Invalid action message	Pass	
6	Not entering the correct format for sort code in a SIR	Invalid action message	Record is saved	Fail	New method is added that checks if the sort code contains exactly six digits, regardless if the user has introduced delimiters

6b	Not entering the correct format for sort code in a SIR	Invalid action message	Invalid action message	Pass	
7	Entering a sort code without dashes (e.g. 999999)	Record is saved	Record is saved	Pass	The system will convert the sort code into 99-99-99

Functional Testing

During the requirement analysis, there were several functional requirements as part of the brief that if implemented, meant the application had met the expectations set by the client's initial brief and its requirements. The results have shown that many of the features have been included and demonstrated in the prototype.

- SMS
  Users can enter the details into a form and the system will save their records to a file, as well as read them from a file. The application can expand the textspeak abbreviations that might be included in the message body. The system lets the user enter any international number that contains the valid prefix (+ or 00).

- Email
  Users can enter the details into a form and choose the suitable email type, standard or Significant Incident Report. The user can select the nature of their report and include the sort code. Any URLs entered in the email will be quarantined and can be viewed in the quarantine list. The system will save the records to a file, as well as read them from a file.
- Tweets
  Users can enter the details into a form. Their textspeak abbreviations will be expanded, and any hashtags or mentions will be counted and displayed in the hashtag and mention list respectively. The system will save the records to a file, as well as read them from a file.

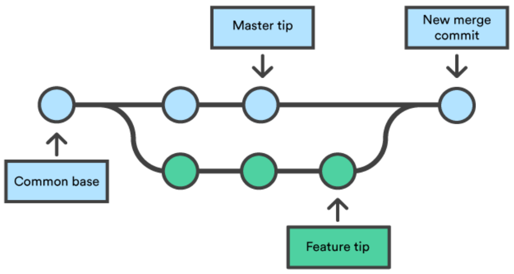
Result Analysis

The functional testing has shown that all the functional requirements set during the requirement analysis have been implemented in the application, and the application has passed all the test cases from the schedule.

Version Control

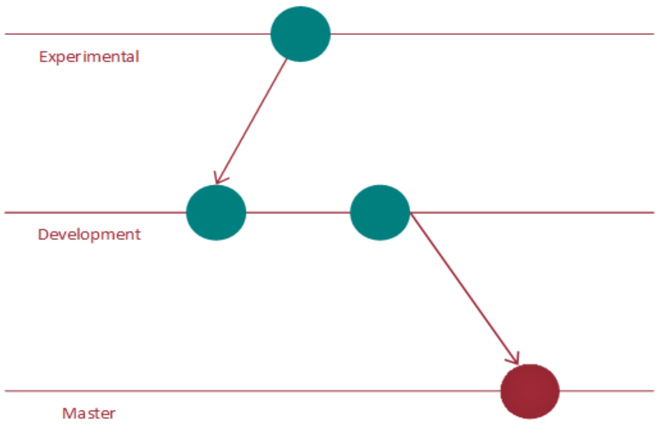
Github

Github is a web-based hosting service for version control that uses Git and it is commonly used to host open source software projects. Projects (or repositories) can be accessed using the standard Git command-line interface. This makes it very easy to use, as it is very intuitive to create and merge branches. Regarding the project, Github will be used for version control. As only one developer is working on the product, code review and branching was not as prominent. Therefore, a development branch was created that was occasionally merged into the master branch when a set of features was finished and implemented.



Proposed Development Plan

The proposed approach for future development would be a multi-branch system. This approach allows for more people to work on a project, as some would work on experimental features, separated from the main development branch. When a new feature is finalized, it can then be merged into the development branch, where it can then be pushed to the master branch. This approach works well with agile sprints, as a new feature could be added in each sprint to the master release.



Evolution Strategy

Predicted Maintainability and Cost

The maintenance cost mainly depends on new features being added to the .Net framework. As new features are added into WPF and c#, old methods or classes might be deprecated, and the application might stop working altogether on newer releases and be incompatible with the latest versions of the Windows OS. However, one developer should be enough to maintain the application, because the application has been built very robustly to minimise down-time and maintenance.

Future Development

If development continues, some features worth looking into could be:

- **Message deletion**
  The current system has no way of deleting messages. A feature that would allow the user to delete messages under certain circumstances, such as messages made by mistake, could be useful.
- Administrator Control Panel
  Another useful feature would be an administrator control panel on the front-end side of the app, where the data could be managed (editing and deleting) without tempering with the JSON files in the back-end.