Coursework Report

40313507@live.napier.ac.uk

Edinburgh Napier University – Software Engineering (SET09102)

1. Introduction

The purpose of this coursework is to develop a prototype for the Napier Bank Message Filtering Service (NBMFS) that validates the data, sanitizes and categorizes the incoming messages in the form of an SMS text message, Standard and Significant Incident Report email and tweet message. The report presents:

- requirement analysis of the NBM
- a class diagram illustrating the classes
- prototype testing
- plan to use version control to develop the system in an agile approach
- evaluation strategy including the maintainability of the system

2. Requirement Specification

The definition of requirements is that they are "a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system." [1]. The main purpose of gathering requirements (also known as requirement elicitation) is the process of generating a loss of the requirements from various stakeholders. If the software gets started without properly understanding the user's needs, there will be misalignment at the end.

In an agile approach, the team developing the application is always in contact with the product owner to define what user stories will be developed in the next iteration.

The software system requirements in this prototype include functional requirements (describes the behavior of a system), requirements (description of a property that a system must show) and interface specification.

2.1. Functional Requirements

Functional Requirements are the main drivers of the application architecture of a system. They define what the system will do or accomplish. They are features that allow the system to function as it was intended.

The functional requirements for the NBMFS system are shown in **Fig 1.**

These requirements are used in the user stories [see **Fig 2** for examples] and use cases as user stories are about the needs of the user to do his day-to-day job and the use cases are about the behaviour the software needs to meet.

2.2. Non-Functional requirements

The non-functional requirements support the functional requirements. They are used for the operational aspects of the architecture. These requirements may affect the overall architecture of a system rather than an individual component. Each non-functional requirement may generate several related functional requirements.

There are three types of non-functional requirements:

- Product requirements
- Organizational requirements
- External requirements

The **Product Requirements** include all functions, features and behaviors that the NBM system must have which include:

- **Usability** requirements where the interface should be easy to use and understand. The interface of the system is easy to understand and use without further training.
- **The efficiency** of use where the goal is easy to accomplish quickly and with few or no user errors. It should not take up a lot of space as employees must be able to use it while running other programs. This includes performance and space requirements.
- **Security:** The program should be secure for use as it will contain data which is sensitive to the company and must not allow anyone to access it that is not authorized.
- **Dependability:** Users should be able to depend on the program to do what it is created to do rather than fear it would crash and would have to temporarily replace it with another program.
- **Robustness:** The program must respond to errors where the program crashes and should not take long to restart. With that in mind users can save their work frequently just in case the program does crash so they do not lose progress.
- **Scalability** – the system must be able to handle changes in size or volume to meet user requirements. The system must be able to handle a large amount of data without crashing when loading the messages from file input and process each message accordingly. It must also be able to move to a new operating system if needed.

The **Organizational requirements** are environmental, operational and development that is of importance to organizational policies and procedures. They do not affect the program directly but must be implemented. This may include how will a user be authorized to use the system. For example, if every user has their separate machine and account, they could be authorized on logging into the operating system and the NBM software automatically checks their credentials. Another example would be how is the program distributed to users. It could be requested by a user that needs it, so they can do their job activities and are either approved or rejected to receive the software. The software may be assigned to a user or the machine itself for distribution.

The **External Requirements** arise from influences which are exterior to the system and its development. They include regulatory, ethical, legislative (account requirements and safety/security requirements). A data privacy policy should be implemented in the system for the country it is being used in.

3. System Design

3.1. Use case diagram

The use case diagram is designed to represent how the user can interact with the system and show how the system creates output for the user so they can view it, delete a message or download it [See **Fig. 3**] A use case for the input is also presented to show the different kinds of input the user can do. [See **Fig. 4**]

3.2. Class Diagram

The class diagram shown in **Fig. 5** describes the structure of the system by showing the system's classes, their attributes, operations and relationships among objects. Validation methods are displayed as they are used in class and in the program they are in the Validation class. Further splitting can be done for every type of message a single validation class so the program does not stop working when there is an error in the validation class.

3.3. Activity Diagram

The activity diagram in **Fig. 6** describes the dynamic aspects of the whole system and **Fig. 7** shows the aspects of creating an SMS message and the process for Tweet, SIR and Email is similar.

4. Implementation

4.1. Agile

Version control systems enable different team members like developers and designers to work together on the same project and edit and add their ideas in their branch. The team then reviews, and it can be committed to the master repository.

If we were to develop this in an agile way, every bit of the program would be split up on smaller goals so the team develops the project to be sustainable and aim for continuous deployment so they can get fast feedback from users.

We would use a Scrum framework that enables iterative and incremental development process where the project is divided into several phases, each of which results in a ready-to-use product. Customer feedback will help reveal possible issues or change in the development plan if needed.

In the scrum model, we would have to have a product owner that will ensure that the end user's interest is taken care of. There is also a scrum master who coordinates the development process and has the responsibility to make sure the scrum team meets regularly, and the Scrum framework is used properly. Lastly, there is the Scrum team who develops the product.

Backlog Creation

The first phase is to gather requirements and understand what the stakeholders need for the product to meet users' expectations. Every requirement from the stakeholders will be created as a User Story, prioritized put on an ordered Backlog. [See Fig. for example]. Tasks will be separated into columns where they can be in the "Planned", "To Do", "In Progress"," Tested" and "Complete". Ideas will be spread out on a backlog where the development team can pick up a task when the work in progress limit isn't reached.

Sprint Planning and Sprint Backlog Creation

The team should decide how long should a sprint's duration be as a short print would allow to release a working version and receive feedback more frequently as bugs and errors will be identified earlier in the process. Every sprint will have a goal the team would need to meet to be successful and every sprint will be filled with prioritized user stories.

If the requirement is too big the team can split it up into smaller pieces and turn them into a series of tasks. With all of the users' stories, the Sprint Backlog is created.

Working on the Sprint. Daily Scrum Meetings

Every sprint will have its own chosen user stories that would need to be completed. The tasks will be separated into: “Planned”, “To Do”, “In Progress”,” Tested” and “Complete”. The board will either be implemented online or on an actual board depending if the team is cooperating remotely or in person. Preferably it would be done both ways, so it is accessible to anyone at any time.

Daily Scrum meetings are important as it will show how is the project going so far and ensure that all team members are on the same page. During the meeting, team members will share what they have done for the Sprint Goal, what they will do next and what problems have they faced during work and if they were overcome. This way it ensures that team members do not work on the same task and can help if another member has run into problems. On every meeting, a burndown chart (used to display how many tasks remain uncompleted) will be updated so the team keeps track and help conclude the speed of work and depending on the conclusions the number of user stories can be changed for the next sprint.

As every sprint has to deliver a working product the testing process is important and to minimize the testing period, either the number user stories will be minimized, or Quality Assurance engineering will be included into the Scrum team.

Product Increment and Sprint Review

After every sprint, a working product should be demonstrated to the customer. After the demonstration, the team will then create a new version of the software with new or improved feathers. This will enable the stakeholders to express any concerns or changes so on the next sprint the development team can work on them.

Retrospective and Next Sprint Planning

On a retrospective, the team should conclude what went well and what can be done better during the next iteration and together with the last sprint review from the stokeholds they can plan the next sprint.

4.2. Version Control

Version Control is used for backup and restoring files. It also enables people to synchronize their work from different machines and mistakes can easily be undone by pooling the previous version of the code. Changes can also be tracked and see what is done and who has done it. It also enables to test the changes in an isolated area before merging to the main repository.

Version control will be done during the whole development part from the start to finish. Every task will have its branch so the team can review, test, edit and share their ideas and when it is done it will be merged. Using Version control it will help the team keep track of what is going on and help everyone see the source code and they can go back and compare the earlier version of the code so they can fix mistakes easily if the new version is not working they can just pull up an earlier version and work from there.

The type of version control used is:

- Distributed version control system: stores the entire history of the file/files on each machine locally and syncs the changes that are made to the server so that changes can be shared with the team which enables collaborative working environment.

Version Control is used during the process where the master copy will be stored in a repository and for teammates to see the differences every commit will be tagged with an appropriate name so teammates can know what it is about. Changes can be done easily as they can pull the version and comment on the changes provided or suggest improvements or speak to each other. Going to an earlier version is also easy as you can pull it from history.

When a sprint is finished the team should implement other features to the current version they have. Team members would be working on different user stories, collaboratively (paired programming) or alone but before making a commit to the current version they have to ensure everything is working well as a whole. This can be done by the branching tool which enables the developer to have a separate repository where the course code can be edited independently.

Optimistic locking will be used for conflict resolution where developers can edit any file but before committing the system will check If any changes have occurred since the last read. This enables developers to work on the same thing without waiting for the other person to finish with the file for the other person to start working on it. That would rarely occur though as everyone will be working on their own user story but if a conflict does occur the version will be rolled back. This way the team can sit and discuss the changes without being locked out until it is unflagged.

5. Testing

Testing the software is used to identify any issues in the program and provides information about the quality of the system. This gives developers enough data to fix errors and create a working version.

Software testing evaluates the software product if it meets the required conditions and needs test cases, test data and test result so it can compare the results with the excepted output whether it passes or not and that produces test reports.

5.1 Test Plan

Test-driven development was used for the project as it relies on software requirements being converted to test cases before the software is fully developed which improves the validation of the data as the software was developed against test cases, so the input data is correct. Some test methods were written after the program was developed to test how it validates the data some were written while the program was in development to ensure basic validation of user input. Using TDD the system ensures that many of the defects are removed early on as we come up with some input that may be entered and see how our system does. Tests were run regularly to make sure all of them were passing after writing a new validation for the system.

Testing was applied to the Business Layer where the messages are created and separated into three categories: SMS, Tweet, Email – email has Standard email and Significant Incident Report(SIR), so another class was created for SIR because it is the only one that has a subject. Each class is tested, and all validation methods are in class Validation to keep the code clean and easy to understand.

Types of Testing done to the system:

Stress testing (Volume Testing – Flood Testing)

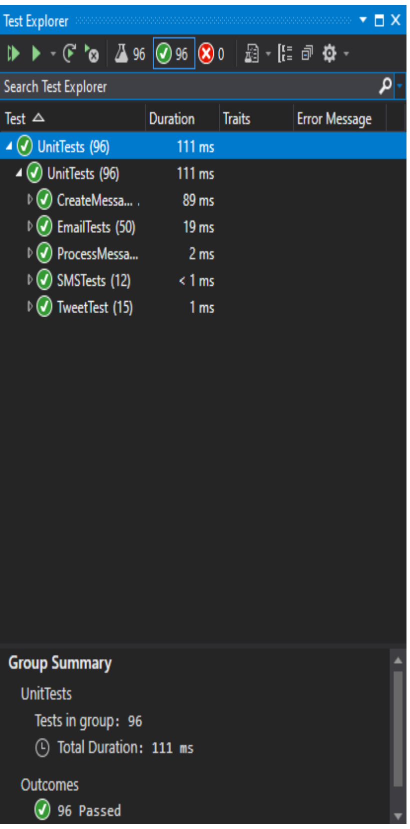
Stress testing was performed by inserting a text file with a lot of messages to see if the system will crash or slow its performance.

As a test the program was inputted with a text file with more than 500 lines of messages, each line being a message of its own. The program was then tested with 7673 lines of messages. The program did not crash when inserting the message nor when processing it. It listed all the messages correctly into the system. Another file with different kinds of messages was added to the program with 10125 lines of messages and no issues were found [See **Fig. 9**]. Messages were saved into a JSON file successfully and uploaded as a JSON file successfully without any issues on any end. Messages were processed successfully. Further Stress Testing can be done with more messages. If there is an issue with any of the messages it will show a dialogue message what the issue is but the system and tell which message has an error and it does not go further so when the message is corrected in can read the rest of the file. [See **Fig. 8**]

5.2. Test Methods

Unit Testing

Unit tests were done to see how well the program accepts input from the user and if it crashes. It is the key in this program for the test-driven development as thanks to it we can keep track of what input is tested and fixed and see how well validation on the data is done. Unit tests were done while still in development where the unit test was written and is it fails; code was written so it gets the expected output.



```
Test Explorer
96  96  0
Search Test Explorer
Test                 Duration  Traits  Error Message
UnitTests (96)         111 ms
  UnitTests (96)       111 ms
    CreateMessa...      89 ms
    EmailTests (50)     19 ms
    ProcessMessa...      2 ms
    SMSTests (12)      < 1 ms
    TweetTest (15)       1 ms

Group Summary
  UnitTests
    Tests in group: 96
    Total Duration: 111 ms
  Outcomes
    96 Passed
```

Some examples are shown below in a table and a couple of examples:

Example:

Tweet.messageSender = “@” was accepted as a valid input because the system only checked if the string was empty or above a certain amount of characters.

messageID = “s123456789” was expected to pass as it is a valid id with the correct pattern but the system accepts upper cases so we cast id.ToUpper()

There are 96 Unit Test cases performed to make sure no user input would crash the program and the messages are correctly processed and recognized properly so they are later displayed in the appropriate lists.

I. Option 1 input

For testing purposes there are two options for input, one is a simple input with help that identify what the message is and let you know what input the program expects. This input also has a separate Text Box for ID, Sender, Subject and body. It is a simpler version to the second option where mistakes may happen, or the user might not know what exactly to input so it can be treated as training input.

II. **Option 2 input**

Here there are two choices for input – one being the message header where the ID should be put and the second textbox which is the Message Body should contain its sender, subject [if appropriate] and message body. The system successfully recognizes the inputs and creates new SMS/Email/SIR/Tweet messages. The Message Processing for Message Body is tested in ProcessMessages where the user in a WPF format would have his input and a preview to the output when Process has been clicked or they could just go and add the message straight away with the Add Message button.

Integration Test

Integration Tests are done to confirm that two or more app components work together to produce the expected result.

Incremental testing with a sandwich approach has been used in some of the test cases where we start only with a few values and check if they are valid and increment the components. This has been done to ensure nothing has been missed for testing and everything passes criteria. Both top-level and low-level components were tested to validate and evaluate the performance and process of the program.

They can be viewed in the ProcessMessageTests and CreateMessageTests as input is declared and it runs through the whole program right to the output which in this case is what message does it return before it is added to the specific list.

ProcessMessageTests use the White Box testing mechanism which needs an understanding of the structure of the system where it is used for verification and focuses on how the output is achieved wherein CreateMessageTests we are using Black Box output where we validate the already processed data and see what the output is.

Acceptance Testing

Acceptance testing can be external and internal. A beta testing was done by a few people who have tried the program to see if it was easy to use and understand. A View Message Button is added so the user can easily select a message from all messages and see the whole message rather than just in a list. Manual alpha-beta testing was done to ensure all buttons were working correctly.

5.3. Test Cases

Test cases were written to ensure the data meets expected results where they consist of assigned input data and a result value to check if the data values pass the test or not.

The system has two test approaches: one is to check if the returned Boolean result is the same as the expected Boolean, ant the other one is to expect an exception if the data returns false so the program does not crash but can show to the user what the issue is so the user can fix input issues and continue without relaunching the program.

5.4. Test Outputs

In the table below, some of the tests are presented to show what was done if the expected output was not met. The "Expected" column shows if the test has passed or failed and the Passed column shows whether the expected output was the same as the returned output. The rows highlighed in red show if the returned output is not as the expected, the issue and what has been done so it meets the expected result.

Input	Expected	Passed	Issues	Resolution
SMS				
messageID = ''s123456789"	Pass	no	ID is expected to be upper case	return new SMS(id.ToUpper()…)
messageID = ""	Fail	YES		
messageID = 'S1234567890"	Fail	YES		
SMS.messageSender = "+447378320249"	Pass	YES		
SMS.messageSender = "+44737832024956665"	Fail	no	Has to be no more than 12 character (numberic values)	number = number.Substring(1); if (number.Length < 13)

SMS.messageSender = "447378320249"	Pass	no	SMS sender has to be an international phone number	Add + in front of numbers with function addChar(number) and replace it in messageBody
Tweet				
Tweet.messageSender = "@venetsia"	Pass	YES		
Tweet.messageSender = “@’	Fail	No	Validation was only done for more than 15 characters	`IndexOutOfRangeException` will be thrown if sender is more than 15 characters or is equal to 0
Tweet.messageSender = "v@venetsia"	Fail	no	Tweet ID has to start with @	tweetSender[0] == '@'
Tweet.messageSender = "@venetsiahghghghghhghghghghhghghghgh"	Fail	no	Tweet sender has to be no more than 15 characters	if (tweetSender.Length > 15)
Email				
Email.messageSender ="venetsia@icloud.com"	Pass	yes		
Email.messageSender ="venetsia@@com"	Fail	no	Has to be valid email	if(!new EmailAddressAttribute().IsValid(emailSender))
Email.messageSender =""	Fail	YES		
Email.url = "google.com"	Pass	yes		
Email.url = "google"	Fail	no	Since (Uri.TryCreate(s, UriKind.Absolute, out resultURI) tries every string with https:// infront it return true for everything	Check if string contains a ‘.’ And then try URI
SIR				
SIR.messageBody = "Sort Code: 95-95-95\r\nNature of Incident: Theft"	Pass	no	Message body for SIR has to start with Sort Code and contain Nature of Incident	var modifiedString = messageBody.Replace("\r\n", " ");
SIR.messageBody = "Sort Code: 95-95-95\r\nNature of Incident: Nothing"	Fail	no	Nature of incident must be from specific choices	Loop through array of incidents and see if message contains it
SIR.messageBody = "sort code: 95-95-95 nature of incident: Theft"	Pass	no	Message Body Must start with Sort Code and contain Nature of Incident	Change lower case where needed to upper case
SIR.messageBody = " sort code: 95-95-95 nature of incident: Theft"	Pass	no	Message Body should start with Sort Code - no space before it	Replace " sort"/" Sort" with "Sort"
SIR.messageBody=""Sort Code: 95-95-955 Nature of Incident: Theft"	Fail	no	Sort Code should be vali (??-??-?? - digit number where there is ?)	Loop through each pair of number removing '-' and check if number and length is 2

SIR.messageBody = " Sort code:95-95-95 nature of incident: Theft"	Pass	no	Validator does not recognise the sort code number because there is no space	check if sort code number contains "Code:" - yes -> var code = sortCode.Trim('C','o','d','e',':','-');
SIR.messageBody = " Sort code:9b-95-95 nature of incident: Theft"	Fail	yes		
SIR.messageSubject = "SIR 12/02/21"	Pass	yes		
SIR.messageSubject = "SIR1 12/02/21"	Fail	no	Subject should be "SIR dd/mm/yy"	Loop through and check every word if it is in correct format
SIR.messageSubject = "SIR 112/02/21"	Fail	no	DateTime returns and date.Lenght = 2, month.Lenght = 2, year.Lenght = 2 or 4	Loop though if It passes DateTime and check length
Abbreviations				
SMS.MessageBody = "Testing AAP"	Pass	no	Message was not returning replacement	Loop through dictionary with Appreviations<Short,Long> and if Short (string) was found replace with "<" +result ">"
SMS.MessageBody = "Testing aap"	Pass	no	Not detecting lower case	transform word into Upper case and if found in dictionary replace

6. Evolution

The importance of evolution is critical to businesses as the assets must be changed over time or updated with new features of for security issues.

For example, the NBM system will change after the prototype has been tested by the customers and remove unneeded features if any (like two options for manual input – testing purposes) or add new features such as adding a database like an SQL server.

Software change is inevitable and can have a few reasons for change and changes could have a different priority. Software change includes new requirements, business environment changes, error, equipment change or improving the performance of the system. All of these reasons are important for one organization.

<u>**New Requirements**</u>

New requirements can be easily added to the program as it is separated in layers and everything is in its class and can be easily changed. New requirements can be non-functional(design changes such as removing lists that are not needed for customers to view which were added as part of the testing and layout of the system so customers can see how effective the program is) or functional (such as adding a view button so that user can view inserted message – which the program already has so customers could decide if they need it or not and can be easily removed)

<u>**Business Environment Changes**</u>

Business Environment is crucial to the organization such as having the program running on multiple operating systems or having multiple version for different kind of users who can access it and view it with different access. A change in the business environment could be how the system is requested and distributed to users when multiple teams with different members start using it. Another change could be due to competitors, the system would have to improve drastically to compete in such a way that it has the functionalities the customers would need.

Errors

The system would need an emergency evolution if it is no longer usable as a major error that has affected multiple teams and could no longer be used. The developers need to fix the issue very quickly for users to continue to do their job well.

Equipment Change

Whenever there is new equipment such as new computers the software would have to change over time to be able to run smoothly as technology always changes so the program may become out of date if there is no evolution, so change is inevitable.

6.1. Maintenance

A software needs maintenance after it has been put into use and either fault are found, or adaptation to a new system is needed or the system needs new features to be useful. This does not involve major changes, the software should still be able to do the same things that were requested in the first place, otherwise, it would change into the new software.

Maintenance could sometime corrupt the already implemented components and that is why while doing maintenance everything should be checked again if it is working well together. In an agile way, nothing new would be added to the software if it is not compatible with the system and no longer works so further development would be needed for that to happen.

As the software ages, it could have high support costs due to it being implemented for an older system for example or no longer supported compiler/operating system.

Maintenance to repair software faults (corrective)

After the software is presented to the customers if anything does not meet the requirements requested the software will be changed accordingly so it meets its requirements.

Maintenance to adapt the software to a different operating environment (adaptive)

The software was not tested on a different operating system so extra testing on an Apple device would be required if users would like to use it there and adapting the software so it runs smoothly would be needed if any issues are found.

Maintenance to add to or modify the system's functionality (perfective)

New features would be added to the system as this is how it evolves and becomes better and still usable.

6.2. Factors to reduce maintenance cost

If the team that developed the software are involved with its maintenance for a while and they train a technology support staff to understand how the system was developed so no mistakes are made after they take over as the software may become unsupported for a while until the other team figures out how to fix issues or even add new features to it, commenting the code would be a benefit as well.

6.3. Maintenance prediction

For the maintenance of the system, developers should always consider:

- Expensive parts to maintain: the system is not expensive to maintain as it is a simple program so changing its structure and even process would not be difficult as every function is there for a reason and it is made clear what it does. Adding functions and removing or even changing should be simple and efficient.
- Lifetime maintenance cost: usually the cost for the program would be higher than the development as changing something might break something else so further maintenance and troubleshooting would be required. Translating the program to another language should not be as hard as the pattern and syntax are similar to other current languages right now.
- Maintaining the system for the next year: maintenance, in the beginning, would be difficult due to untrained staff of finding issues that were not found previously but once all of that is cleaned the program should be easy to maintain and evolve.
- Change Requests: upon defining requirements for the system, usually, clients are not very specific and developers could understand something else so it will be completely up to the clients if they would like to change anything in a design perspective or a software component aspect. A few extra non-functional and functional requirements were added so the clients can see if they like it and if not, it could be easily changed/removed.
- Part of the system that is most likely going to change:
  - Extra features added:
    - View all messages: This could be removed as users may not need to see all the messages and just be able to see the processed message
    - In option 2 for adding a message, there is a process button that when clicked it will show how the message would be processed which could be removed as it may be unnecessary for the user but it was developed so they users could see in advance how would the message be processed.
    - Adding input as JSON file – this was not required as a function but was added so clients can see the difference on how it does work, and which function would be easier for them to input.
    - Clearing all messages was not a feature but was added so the users could see how the program saves the messages in a JSON format and input them later again. – a convert function could be added between JSON and TXT if users would like to keep both.
  - (Input) A feature that could be added is connecting an email account/system, Twitter account or phone number to the program so that it can automatically receive messages and view them in a processed format in the lists rather than insert them manually.

6.4. Reengineering

After the system has been maintained a long time and the cost for it has increased over time, reengineering can take place to refactor the system to make it better. The cost of reengineering an existing system would be less than developing a new software so if it needed the program could be re-engineered in a continuous process of improvement.

It is up to the organization if they want to scrap the system or they could continue maintaining the existing one, so the system quality and value is taking into account when deciding this sort of stuff.

7. Reference

[1] Sommerville, Ian and Pete Sawyer. 1997. Requirements Engineering: A Good Practice Guide. Chichester, England: John Wiley & Sons Ltd.

8. Appendix

Fig. 1 Functional requirements

```
1. Read file input (.txt) containing various messages or manual input
    1.1 Validation and Identification of input
        1.1.1. SMS
        1.1.2. Tweet: contains mentions and trending list
        1.1.3. Email: contains Standard Email Messaging and Significant Incident Reports
    1.2 Message Processing
        1.2.1. SMS
            a. detect text abbreviations and replace by full definition
        1.2.2. Email
            a. Standard Email Messages : Quarantine URLs and add to list
            b. Significant Incident Repors (SIR)
                b.1 Quarantine URLs and add to list
                b.2 Detect Sort Code and Nature of Incident and add to SIR list
        1.2.3 Tweet
            a. detect text abbreviations and replace by full definition
            b. recognise hashtags and add to tending list
            c. recognise mentions and add to mention list
    1.3 Display lists
2. Output processed messages to a JSON file
```

Fig. 2 User Stories

ID	User Story
a-001	As a user , I want to be able to add three types of messages.
a-002	As a user, I want to be able to understand abbreviations clearly.
a-003	As a user, I want to be able to make an input to the system using a file.
a-004	As a user, I want to be able to download the messages on my machine.
a-005	As a user, I want to be able to see processed messages in appropriate lists.
a-006	As a user, I want to be able to input messages manually.
a-007	As a user , I want to be able to understand the system easily.

Fig. 3 Use Case Diagram

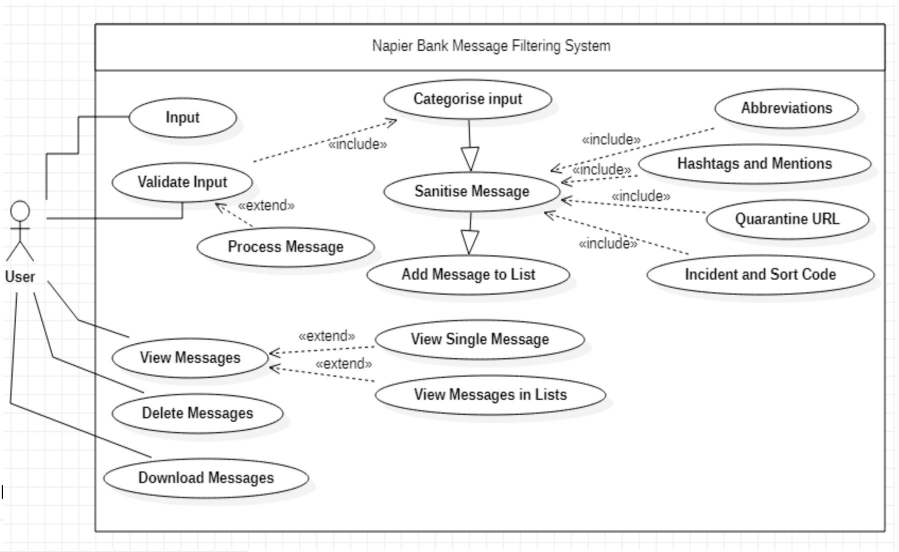


Fig. 4 Use Case Diagram fo

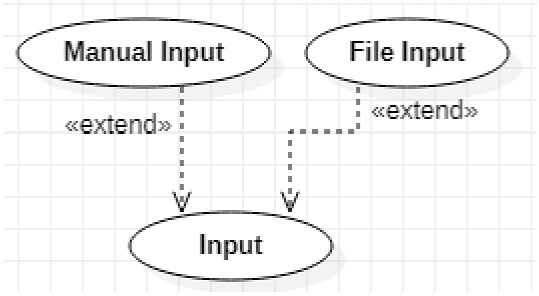


Fig. 6 Activity Diagram of System

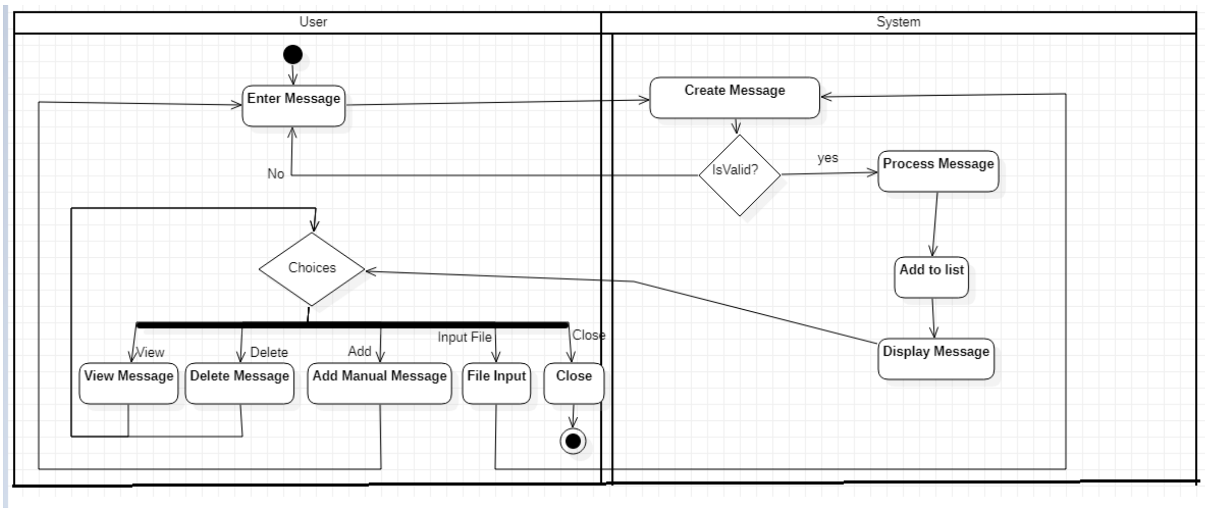


Fig. 7 Activity Diagram of Creating an SMS Message

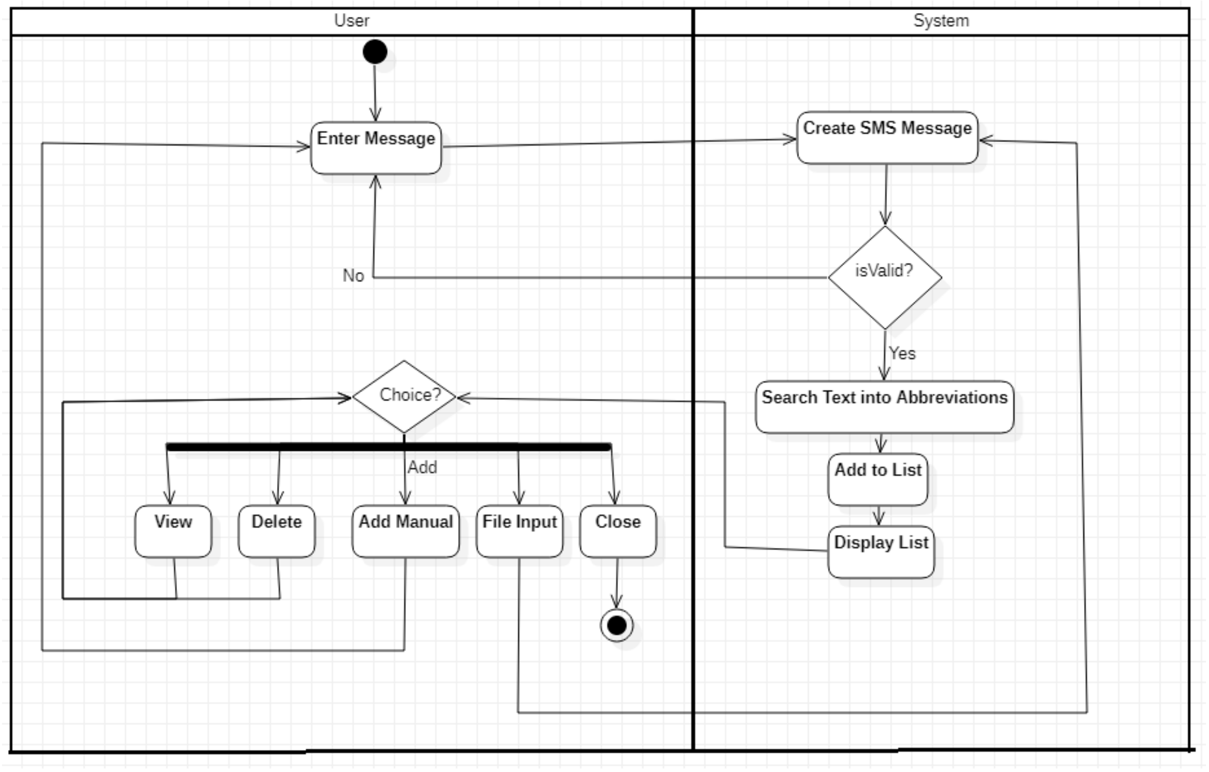
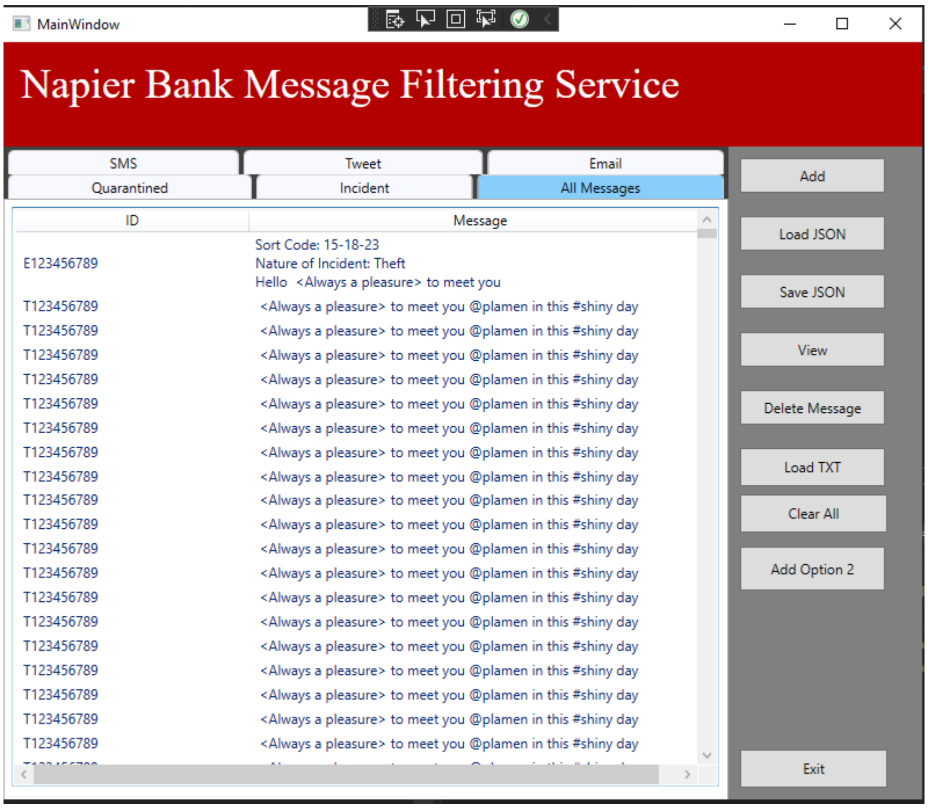
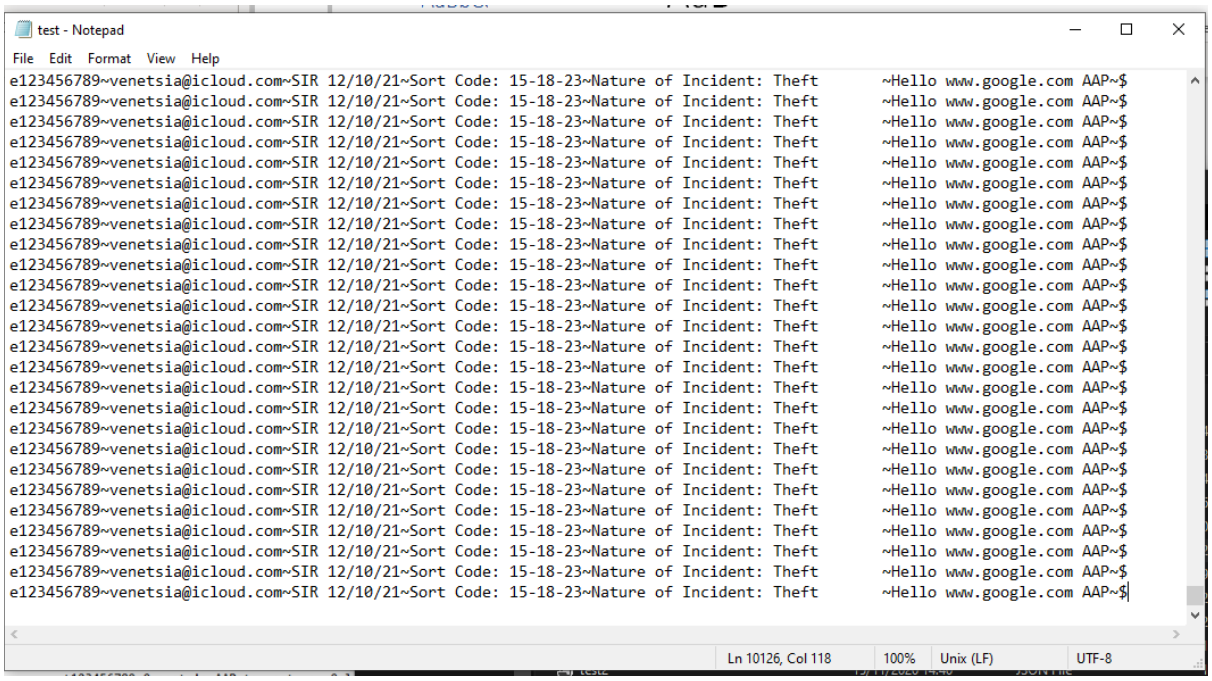


Fig. 8 All Messages – Input from File



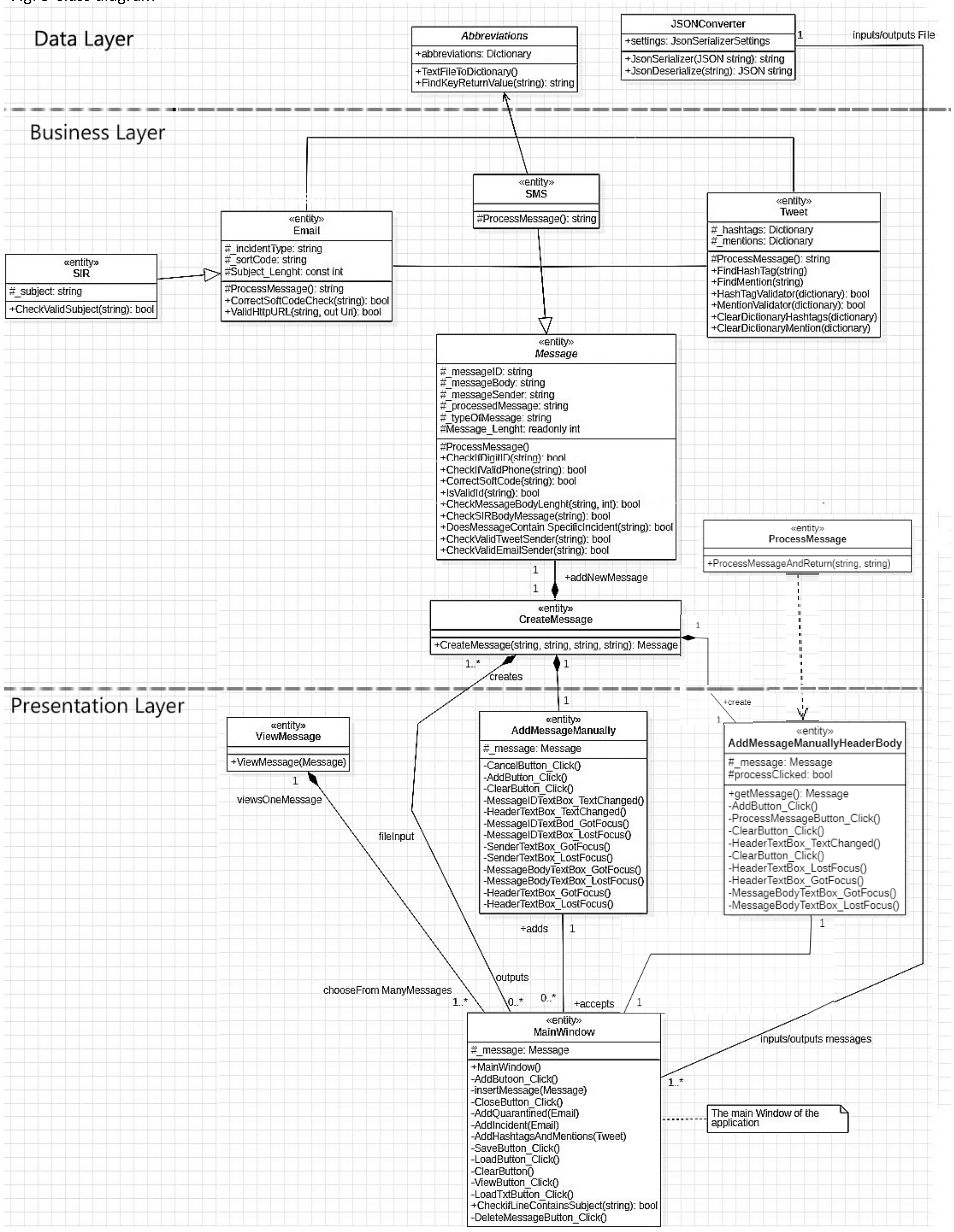
ID	Message
E123456789	Sort Code: 15-18-23 Nature of Incident: Theft Hello <Always a pleasure> to meet you
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day
T123456789	<Always a pleasure> to meet you @plamen in this #shiny day

Fig. 9 Text File Input



```
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
e123456789~venetsia@icloud.com~SIR 12/10/21~Sort Code: 15-18-23~Nature of Incident: Theft       ~Hello www.google.com AAP~$
```

Fig. 5 Class diagram


Data Layer
Abbreviations
+abbreviations: Dictionary
+TextFileToDictionary()
+FindKeyReturnValue(string): string
JSONConverter
+settings: JsonSerializerSettings
+JsonSerializer(JSON string): string
+JsonDeserialize(string): JSON string
1
inputs/outputs File
Business Layer
«entity»
SMS
#ProcessMessage(): string
«entity»
Email
#_incidentType: string
#_sortCode: string
#Subject_Lenght: const int
#ProcessMessage(): string
+CorrectSoftCodeCheck(string): bool
+ValidHttpURL(string, out Uri): bool
«entity»
SIR
#_subject: string
+CheckValidSubject(string): bool
«entity»
Tweet
#_hashtags: Dictionary
#_mentions: Dictionary
#ProcessMessage(): string
+FindHashTag(string)
+FindMention(string)
+HashTagValidator(dictionary): bool
+MentionValidator(dictionary): bool
+ClearDictionaryHashtags(dictionary)
+ClearDictionaryMention(dictionary)
«entity»
Message
#_messageID: string
#_messageBody: string
#_messageSender: string
#_processedMessage: string
#_typeOfMessage: string
#Message_Lenght: readonly int
#ProcessMessage()
+CheckIfDigitID(string): bool
+CheckIfValidPhone(string): bool
+CorrectSoftCode(string): bool
+IsValidId(string): bool
+CheckMessageBodyLenght(string, int): bool
+CheckSIRBodyMessage(string): bool
+DoesMessageContain SpecificIncident(string): bool
+CheckValidTweetSender(string): bool
+CheckValidEmailSender(string): bool
«entity»
ProcessMessage
+ProcessMessageAndReturn(string, string)
1
+addNewMessage
1
«entity»
CreateMessage
+CreateMessage(string, string, string, string): Message
1
1..*
creates
1
Presentation Layer
1
+create
1
«entity»
ViewMessage
+ViewMessage(Message)
1
viewsOneMessage
fileInput
«entity»
AddMessageManually
#_message: Message
-CancelButton_Click()
-AddButton_Click()
-ClearButton_Click()
-MessageIDTextBox_TextChanged()
-HeaderTextBox_TextChanged()
-MessageIDTextBod_GotFocus()
-MessageIDTextBox_LostFocus()
-SenderTextBox_GotFocus()
-SenderTextBox_LostFocus()
-MessageBodyTextBox_GotFocus()
-MessageBodyTextBox_LostFocus()
-HeaderTextBox_GotFocus()
-HeaderTextBox_LostFocus()
«entity»
AddMessageManuallyHeaderBody
#_message: Message
#processClicked: bool
+getMessage(): Message
-AddButton_Click()
-ProcessMessageButton_Click()
-ClearButton_Click()
-HeaderTextBox_TextChanged()
-ClearButton_Click()
-HeaderTextBox_LostFocus()
-HeaderTextBox_GotFocus()
-MessageBodyTextBox_GotFocus()
-MessageBodyTextBox_LostFocus()
+adds
1
1
outputs
chooseFrom ManyMessages
1..*
0..*
0..*
+accepts
1
inputs/outputs messages
«entity»
MainWindow
#_message: Message
+MainWindow()
-AddButoon_Click()
-insertMessage(Message)
-CloseButton_Click()
-AddQuarantined(Email)
-AddIncident(Email)
-AddHashtagsAndMentions(Tweet)
-SaveButton_Click()
-LoadButton_Click()
-ClearButton()
-ViewButton_Click()
-LoadTxtButton_Click()
+CheckifLineContainsSubject(string): bool
-DeleteMessageButton_Click()
1..*
The main Window of the application