Coursework Report

Erik Sanne
40276539@live.napier.ac.uk
Edinburgh Napier University – Software Engineering (SET09102)

1 Introduction

Euston Leisure requires a system (Euston Leisure Messaging (ELM)) which will validate, sanitize and categorise incoming messages to Euston Leisure in the form of SMS text messages, emails and Tweets.

This report details the:

- Requirement specification
- Implementation
- Testing and test plan
- Version control
- Evolution

2 Requirements

Requirements are split into functional and non-functional requirements.

Functional requirements define what functionality the system should provide, how the system should process inputs and how the system should behave in situations.

Non-functional requirements define constraints on the system and how the system should behave. These affect the whole system rather than individual features or services.

2.1 Functional requirements

The system is required to validate, sanitize and categorise incoming messages. Messages can be SMS text, Emails or Tweets.

On a high level the Euston Leisure Messaging system must:

- Detect the message type
- Validate the message
- Sanitize the message
- Maintain a list of hashtags and the number of occurrences for each hashtag
- Maintain a list of Twitter ids mentioned in messages
- Maintain a list of quarantined URLs
- Maintain a list of Significant Incident Reports
- Output the message in JSON

The below is a detailed specification for validating, sanitizing and categorising the different types of messages.

The incoming messages have a Message Header and Body.

Message header:

The header should be validated and categorised into the correct message type.

- Validate:
  - Message ID is a Message Type (“S”, “E”, “T”) followed by 9 numeric characters.
    - i.e. “E1234568789”
- Categorize
  - “S” should be detected as an SMS message and processed as such
    - i.e. “S123456789”
  - “E” should be detected as an Email message and processed as such
    - i.e. “E123456789”
  - “T” should be detected as Tweet and processed as such
    - i.e. “T123456789”

Body:

The body should be processed for each of the message types as explained below.

SMS Messages:

Input	Output

Header:
S123456789
 	
Body:
+4412345678910
That’s funny LOL
	{
	"id": "S123456789",
	"sender": "+4412345678910",
	"messageText": "That’s funny LOL <Laughing out loud>"
	}

- Validate:
  - Sender is an international phone number
    - i.e. +4412345678910
  - Message text is no longer than 140 characters
- Sanitize:
  - Expand textspeak abbreviations
    - i.e. LOL <Laughing out loud>

Email Messages:

- **Standard Email Message**

Input	Output
Header:	{

E123456789 Body: john.smith@example.org Look at this This is a cool website http://www.cool.com	"id": "E123456789", "sender": "john.smith@example.org", "subject": "Look at this", "messageText": "This is a cool website <URL Quarantined>" }

    - Validate:
        - Sender is a standard email address
            - i.e. john.smith@example.org
    - Subject is no longer than 20 characters
    - Message text is no longer than 1028 characters
- Sanitize
    - URLs will be written to a quarantine list and replaced by "<URL Quarantined>"
        - i.e. "This a cool website http://www.cool.com" becomes "This a cool website <URL Quarantined>"
- Categorize
    - If subject is in the form of "SIR dd/mm/yy", the message should be detected as a Significant Incident Report and processed as such
- **Significant Incident Report:**

Input	Output

Header:
E123456789
 	
Body:
john.smith@example.org
SIR 25/11/19
Sport Centre Code: 66-666-66
Nature of Incident: Raid
See here http://www.araid.com
	{
	"id": "E123456789",
	"sender": "john.smith@example.org",
	"subject": "SIR 25/11/19",
	"messageText": "See here <URL Quarantined>",
	"sportCentreCode": "66-666-66",
	"natureOfIncident": "Raid",
	}

Same as standard email plus the below.

    - Validate
        - Subject is formatted as "SIR dd/mm/yy"
            - i.e. "SIR 25/11/19"
        - Sport Centre Code is in valid format
            - i.e. "Sport Centre Code: 66-666-66"
        - Nature of Incident is a valid incident type
            - i.e. "Nature of Incident: Raid"
    - Categorize
        - Centre code and Nature of Incident will be written to SIR list

Tweets

Input	Output

Header:
T123456789
 	
Body:
@JohnSmith
My first #Tweet @Twitter
	{
	"id": "T123456789",
	"sender": "@JohnSmith",
	"messageText": "My first #Tweet @Twitter",
	}

- Validate
    - Sender is a valid Twitter ID
        - "@" followed by a maximum of 15 characters
        - i.e. "@JohnSmith"
    - Tweet text is no longer than 140 characters
- Sanitize
    - Expand textspeak abbreviations
        - i.e. LOL <Laughing out loud>
- Categorize
    - Hashtags (Words preceded by a "#") will be written to a hashtag list that will also count the the number of uses for each hashtag. The count will be used to produce a trending list.
        - i.e. #BBCClick
    - Mentions (Twitter Ids) will be written to a mentions list
        - i.e. "@JohnSmith"

2.2 NON-FUNCTIONAL REQUIREMENTS

The main non-functional requirements for ELM are:

- Usability
- Reliability
- Scalability
- Efficiency
- Capacity
- Security

These requirements affect the whole system rather than single components. If these requirements are not met the system might not be able to be deployed or fail after being deployed.

Product requirements

Usability: The system shall be easy to understand, and user-friendly as non-technical staff must be able to use to the system. Any errors or issues shall be displayed clearly with an easy to understand the description.

Reliability: The system shall always be available and should recover from any issues/errors, crashes or other downtimes (if necessary, by restarting) quickly.

Scalability: The system must be able to scale with changing user needs such as a growing customer base of Euston Leisure.

Efficiency: The system shall be efficient both in time and space complexity, so it doesn't require excessive resources.

Capacity: Enough resources should be allocated to support the entire userbase as well as a buffer.

Security: The system and its data shall only be accessible to authorised staff.

Organizational requirements

The system will only be used by staff, this means authentication must be available using the staff member's login/id card. The development and system itself always have to follow any applicable internal organizational procedures.

External requirements

The system must follow all applicable regulatory, ethical, legislative and other legal requirements. As the system stores data such as phone numbers, emails and twitter id's a privacy policy relevant to its location as well as any additional data security and safety requirements shall be implemented.

2.3 USE CASE DIAGRAM

The use case diagram is Figure 1 in the appendix.

3 IMPLEMENTATION

The system is going to be divided into three layers, the data layer, business layer and presentation or API layer.

The data layer has a data provider which will allow the system to access a list of textspeak abbreviations and their expanded meaning and to store add or increment items for the trending, mentions, quarantine and SIR lists.

The business layer will define the message in the form of classes and includes the message processor. The message processor takes the message in a raw format, detects its type and processes it accordingly. This class uses the data provider from the data layer and facilitates any non-static message processing. Static processing happens when the relevant message class is initialized, this ensures messages are formatted correctly regardless of textspeak words, URL quarantines or other sanitization that requires the data provider.

Finally, the presentation layer provides an interface for the user or other systems to access ELM. In the prototype, this is implemented as a WPF frontend which takes the input in text boxes or as a file and uses the message processor to process the messages. The interface

also uses the data provider to allow access to the trending, mention, quarantine and SIR lists, i.e. in the WPF prototype, these are displayed in the window. Another implementation of this interface would be an API for other systems to connect to this system and process messages as required.

3.1 CLASS DIAGRAM

The class diagram for this implementation is Figure 2 in the appendix.

4 TESTING

Testing is important in any system to detect errors and ensure all requirements are implemented. There are various types of testing that should be undertaken during the development of ELM:

Unit testing
Unit testing tests each modules' individual functionality but not the interaction of the module with other modules. It is used to find bugs in the logic, data and algorithm of individual modules.

In ELM this will be used to verify that modules correctly validate, sanitize and categorize the different messages.

Integration testing
Integration testing tests the interaction between modules and finds bugs in interfaces between modules.

ELM integrates modules such as the data provider or message processor as sub-systems. It is important that these modules integrate correctly and interface with the rest of the system without any issues. To ensure this ELM will use integration tests.

Validation testing
Validation testing ensures that the system meets all functional and performance requirements. This should be completed before presenting the system to any stakeholder.

Acceptance testing
Acceptance testing validates that the system meets all customer requirements and extends on the existing validation testing. This should be completed after validation testing is successful.

Regression testing
After the initial version of the system has been completed, further changes or new features may be implemented. To ensure that these changes do not break any functionality and that the system still meets its requirements the system shall provide the relevant regression tests.

Sample unit test cases for

a SMS message:

Test case	Expected result	Actual Result	Notes
Id = “S123456789”	Success	✅ Success	
Id = “s123456789”	Success	✅ Success	Assuming lowercase type is allowed
Id = “”	Error	✅ Error	Can’t be empty
Id = “S12345678”	Error	✅ Error	Should be 9 digits
Id = “S1234567890”	Error	✅ Error	Should be 9 digits
Id = “S123ABC789”	Error	✅ Error	Should be followed only by digits, no letters allowed
Id = “A123456789”	Error	✅ Error	Invalid message type
Sender = “+44123456789”	Success	✅ Success	Standard number
Sender = “”	Error	Error	Can’t be empty
Sender = “+441234”	Success	✅ Success	Minimum allowed length
Sender = “+44412345678910”	Success	✅ Success	Maximum allowed length
Sender = “44412345678910”	Error	✅ Error	Country code (prefixed with a “+”) required
Sender = “+44123”	Error	✅ Error	Too short
Sender = “+4412345678910111”	Error	✅ Error	Too long
Message text = “This is a text”	“This is a text”	✅ “This is a text”	Standard text
Message text = “This is a text LOL”	“This is a text LOL <Laughing out loud>”	✅ “This is a text LOL <Laughing out loud>”	Standard text with textspeak
Message text = “”	Error	✅ Error	Can’t be empty
Message text = “3exoOfP vKEjmgKKCk56lBf29vrN5 HC7j khu4QpcC aiialbD1 Y2FcFWzd0dyjsOjwZ0H0 GRZ SbD4h1bh 2ECzxvLe PQ0xjF3StHrnavsZDmBZ Q0I2q4HKKf7SbtSzQtpP e7PbuLoLfReNcc”	Error	✅ Error	Text too long (141 characters), maximum 140 characters allowed

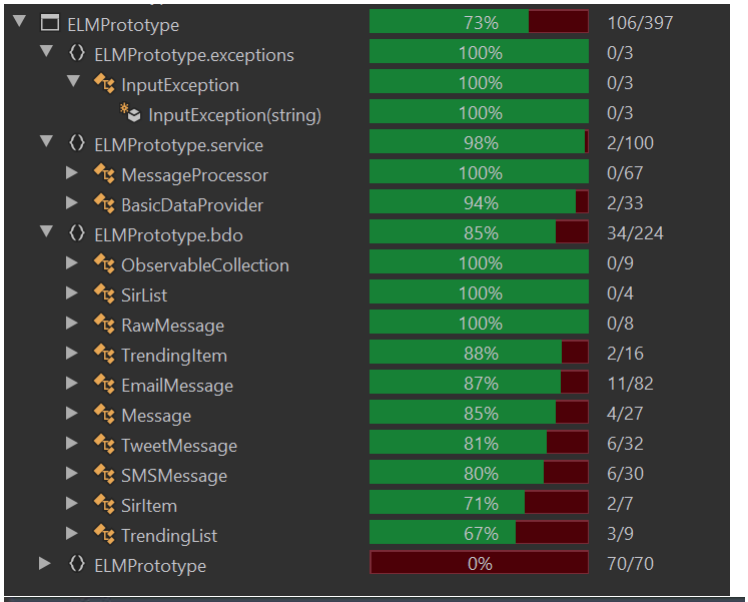
Test plan

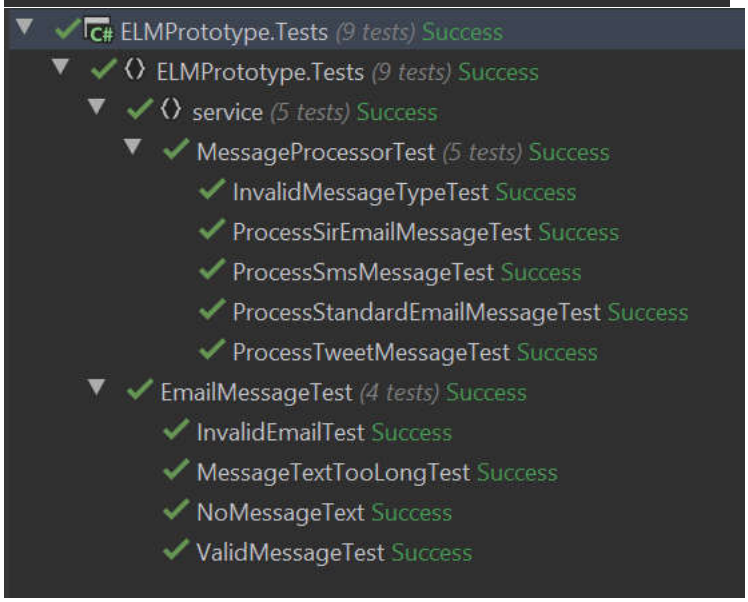
Developers shall develop unit tests for all modules they develop, if the module integrates with other modules, they shall also develop integration tests. As described in the version control plan, when pull requests get reviewed, the reviewer(s) must ensure that this is done correctly.

Once the system can be tested as a whole, testing such as validation and acceptance testing shall be conducted. Once these tests are successful, regression testing should be developed to ensure that any other future changes do not break any requirements.

Unit and integration tests will use the white box method as knowledge of the system's internal design is required for this type of testing. Validation, acceptance and regression tests will use black-box testing to verify the functionality as no knowledge of the system's internal design is required and may bias the test developer to miss crucial test cases.

Prototype test coverage (all tests passed)





5 VERSION CONTROL

A version control system is a place to store the software's source files, it maintains a history of changes made to the files, allows viewing the state at any point and provides tooling for a team of developers to work on the same files simultaneously.

There are various version control systems available however git is by far the most popular and is also freely available as it is open-source. It is easy to set up and developers will ideally not require any additional training to use it. As the system is to be developed with an agile approach, the project will be split into many small tasks which developers pick up and work on. It is important that a defined and though-out workflow is used to avoid collaboration issues such as erroneous code being shipped or hard to solve merging issues.

Git can be used in different ways depending on preference and requirements. Some options are a centralized workflow where all changes are committed to the master branch, feature branch workflow where a new branch is created for each feature and when completed the branch is merged to master.
Another option is the forking workflow where every developer forks their own copy of the repository, develops the feature in a feature branch in their own repository and when ready merges the branch into the main repository master by creating a pull request. This approach is very popular with open source projects and agile teams.

For the ELM system, both branching or forking are viable options however forking allows for better quality control as only the maintainer can push to the main repository. For this reason, ELM should use a forking workflow for the development.

Version control plan

Each developer shall have their own fork of the central repository. In their fork, the developer will create a branch for the task they are working on. When the task is completed the developer will open a pull request to merge the branch to the central repository's master branch. The pull request will be reviewed by at least 2 other developers and must pass all tests, it is then ready to be merged. The maintainer will also check the pull request and when if the master is ready for it the maintainer will merge the pull request in the master branch. The task is now complete, and the developer can pick up another task.

This plan ensured quality and integrity across the entire codebase.

6 SOFTWARE EVOLUTION

The system is likely to require further maintenance as well as further feature development as the needs of Euston Leisure change or the wider environment such as the specification of incoming messages changes. I.e. the system is developed with the assumption that tweets are at maximum 140 characters long, but this has recently changed to 280 characters.

Maintenance costs are likely to be 2x to 30x the cost of development, however, this can be reduced by proactively predicting changes and building the system so that it can easily be

maintained or expanded. Keeping the same staff, possibly in a reduced capacity, as during development will reduce costs as this staff will have acquired domain knowledge about the system and its environment. Developers should also keep evolution in mind during development and ensure their components are maintainable and expandable. Furthermore, the development of automated regression tests as described in the test plan will allow more safe modification of the system.

During development, the team should work closely with stakeholders to be aware of changing requirements and adapt as requirements change. Agile ceremonies allow the team to quickly learn about new or changed requirements and gives them the ability to quickly adapt their tasks and priorities.

7 APPENDIX

Figure 1: Use case diagram

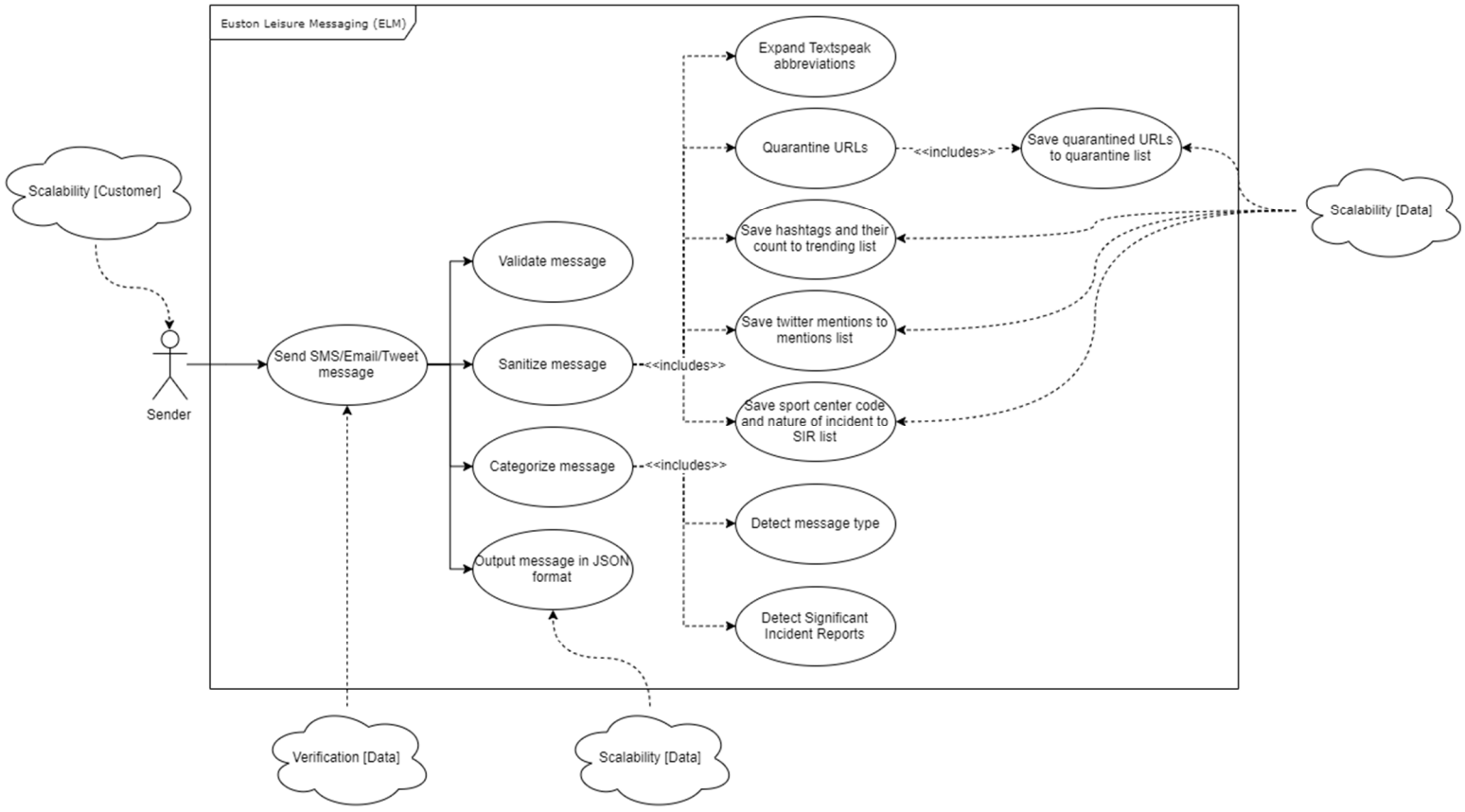


Figure 2: Class diagram