Mailtools

Version: *1.3.0-stable*

Product Summary and Description

Authors

Christina Sowah

Enock Gansou

George Petterson

Harrison Linowes

Jack de La Beaujardiere

Matthew Burns

Nathaniel Lao

Shawn Verma

Table of Contents

1. Introduction **3**

1.1. Problem Statement 3

1.2 Target Audience 3

2. Solution Overview **4**

2.1 Features 4

2.2 Technologies for Implementation 4

2.3 Information Flow Architecture 7

3. Application Use **10**

3.1 Integration and Installation 10

3.2 Syntax 10

3.2.1 Script Syntax 10

3.2.2 Command Line Syntax 12

4. Testing **15**

4.1 Unit Testing 15

4.2 Integration Testing 15

4.3 Acceptance Testing 15

4.4 Performance Testing 16

4.5 Assessing User Satisfaction 17

5. Cost Estimation **18**

5.1 Internal Logic Files 18

5.2 External Interface Files 18

5.3 External Inputs 18

5.4 External Outputs 18

5.5 Function Point Totals and Task Hour Costs 18

6. Schedule **19**

7. Product Advertisement **20**

7.1 Mission Statement 20

7.2 Target Customer 20

7.3 Product Features 20

Appendix 1: **21**

Appendix 2: **31**

1. Introduction

The following report will provide an updated overview of the final version of *Mailtools* and outline tasks that were completed throughout the process. The purpose of this section is to explain the current problems interacting with emails via the command line, how our product addresses those problems and describe the target audience for the product.

1.1. Problem Statement

Email has been around since the 1960s and continues to remain a vital means of communication for both professional and personal use. There are several popular email clients such as *Outlook*, *Gmail*, *Yahoo*, and *Thunderbird* that have developed numerous features to benefit their users. Some of these features include searching for emails, smart reply, and customization of mailboxes. However, these powerful features are only limited to consumers using those services. Besides accessing email through an email client, consumers also have the ability to access their emails via the command line. Mailboxes can be stored as a *mbox* file, which contains all the emails of a mailbox concatenated. When accessing emails from the command line, we lose a lot of the powerful functionality email clients provide us. There are a few notable commands such as *grep* and *grepmail* that allow for searching mail via command line. However, these tools are often slow and functionality is limited. Many email clients can take specified actions on incoming mail or archived mail, such as organizing mail from certain senders. There is currently not a tool for *command line interface* (*CLI*) that allows for fast searching and allows actions to be taken on the search results. We have developed a suite of powerful tools that provide fast searching and allow for *If-This-Then-That* (*IFTTT*) mechanisms to be used on mail archives via *CLI*.

1.2 Target Audience

Our target audience is comprised of users who utilizes *CLI* are their primary way to interface with mail archives. Additionally, users may also run their mail server or utilize open source solutions, such as *Dovecot*. These users may be using tools such as *grep* and *grepmail* to search through their mail archives. They also may have the need to organize large batches of mail efficiently and swiftly

2. Solution Overview

This section will define the different features offered by *Mailtools* and how these features offer a solution to customer's needs. This section also discusses the various technologies used to implement this product. Finally, this section explains the system architecture for *Mailtools*.

2.1 Features

The *Mailtools* package addresses the customer's need for an *CLI* tool for manipulating mail. *Mailtools* directly integrates and pulls data from local directories containing raw mail files. The primary feature of *Mailtools* is *Smart Search*. Using this feature, the customer can search through archives of mail through a query-based language. Examples include searching for email sent by a certain user, email with a specific type of attachment name or type and email with a certain keyword in the body. Search queries can use regular expressions and can either be combined conjunctively or disjunctively to provide the user more capabilities with searching.

In addition to responding to queries defined by the user, *Mailtools* can be configured to execute various organizational and practical operations on mail and mail searches. Implemented alongside *Smart Search* is *IFTTT*. *IFTTT* allows users to create simple scripts to run tasks such as search, forward, and send emails. *IFTTT* scripts support being run as cron jobs allowing for timed and reoccurring scripts to be created.

2.2 Technologies for Implementation

Mailtools is comprised of multiple technologies encapsulated to implement the functionalities of the program. Detail of each technology used in *Mailtools* is explained below:

MBOX

Mbox is a generic term for a family of related file formats used for holding collections of email messages. All messages in a *mbox* file are concatenated and stored as plain text. Each message starts with the four characters "From" followed by a space ("From_ line") and the sender's email address. The format of mail archives *Mailtools* will search through are *mbox* files. The *Python* module used to handle *mbox* files is *Mailbox*.

SMTP

Simple Mail Transfer Protocol (*SMTP*) is an Internet standard for electronic (mail) transmission. First defined by *RFC 821* in 1982, it was updated in 2008 with Extended *SMTP* additions by *RFC 5321*, which is the protocol in widespread use today.

Although electronic mail servers and other mail transfer agents use *SMTP* to send and receive mail messages, user-level client mail applications typically use *SMTP* only for sending messages to a mail server for relaying.

SMTP communication between mail servers uses *TCP* port 25. Mail clients on the other hand, often submit the outgoing emails to a mail server on port 587 or 465. The latter port was previously deprecated, but this changed with *RFC 8314* and its use is now recommended to ensure security.

Although proprietary systems (such as *Microsoft Exchange* and *IBM Notes*) and webmail systems (such as *Outlook.com*, *Gmail* and *Yahoo! Mail*) use their own non-standard protocols to access mailbox accounts on their own mail servers, all use *SMTP* when sending or receiving email from outside their own systems.

MongoDB

Emails are aggregated, parsed and stored in a *MongoDB* database. *MongoDB* is a free and open-source cross-platform document-oriented database program. Classified as a *NoSQL* database program, *MongoDB* uses a *JSON*-like file schema. *MongoDB* is developed by *MongoDB Inc.*, and is published under a combination of the *Server Side Public License* and the *Apache License*.

MongoDB is widely used nowadays and provides several features relevant to allow *Mailtools* to function properly and efficiently. An important benefit of using *MongoDB* is its ability to handle large write/inserts volumes into a database/table. Since we process large mail archives into our database, we must ensure insertion is as fast as possible. Also, *MongoDB*, using *JSON*-like documents with schemata, can be useful for future tasking, for example, the use of *JavaScript* in later versions of *Mailtools*.

Additionally, research has shown that *MongoDB* is highly scalable allowing us to easily maintain new and changing data. *MongoDB* works well for *Mailtools* since it does not require the use of fixed fields, standard used by many *SQL* databases namely *MySQL*. In fact, each mail is different in the parts it contains. For instance, some emails contain attachment(s) while others do not.

Bash

The main frontend of *Mailtools* is a Bash command line interface. Users can call the command mailtools in the server in order to use *Mailtools*. By default, the command mailtools starts a new instance of the *Mailtools* interpreter. This *Mailtools* interpreter will allow the user to issue quick commands to the *Mailtools* software without the need to create an extraneous *.ms* script file. All commands in the *Mailtools* language can be issued here, similar to the interpreter of the script language such as *Python.* The mailtools command also accepts *.ms* script files in its argument to run without the interface. We utilized *Bash* scripting to create a command line program which will allow users to seamlessly interface with *Mailtools*. The syntax allows users to link their

email server, query and organize their emails, and much more. See section 3.2.1 for specific syntax definition.

Python

Mailtools will implement most of the email management and logistics with *Python* version 3.x. *(Note that all references in this document to Python refers to this version.) Python* allows for lightweight scripts while making a multitude of libraries available to help streamline the process.

- *PyMongo*
- *Email*
- *smtplib*
- *Threading*
- *Mailbox*

Cron

The unix *cron* utility will be used to implement recurring scheduled actions as specified by the user. An example of a schedule action includes performing a regular mailbox cleanup. *Mailtools* will interface with the *cron* utility and will store a lookup table of all associated *cronjobs* in a local database.

2.3 Information Flow Architecture

Mailtools is comprised of multiple software components which encapsulate the multiple features available through the application. Figure 2.3.1 details how information is transferred throughout the *Mailtools* application.

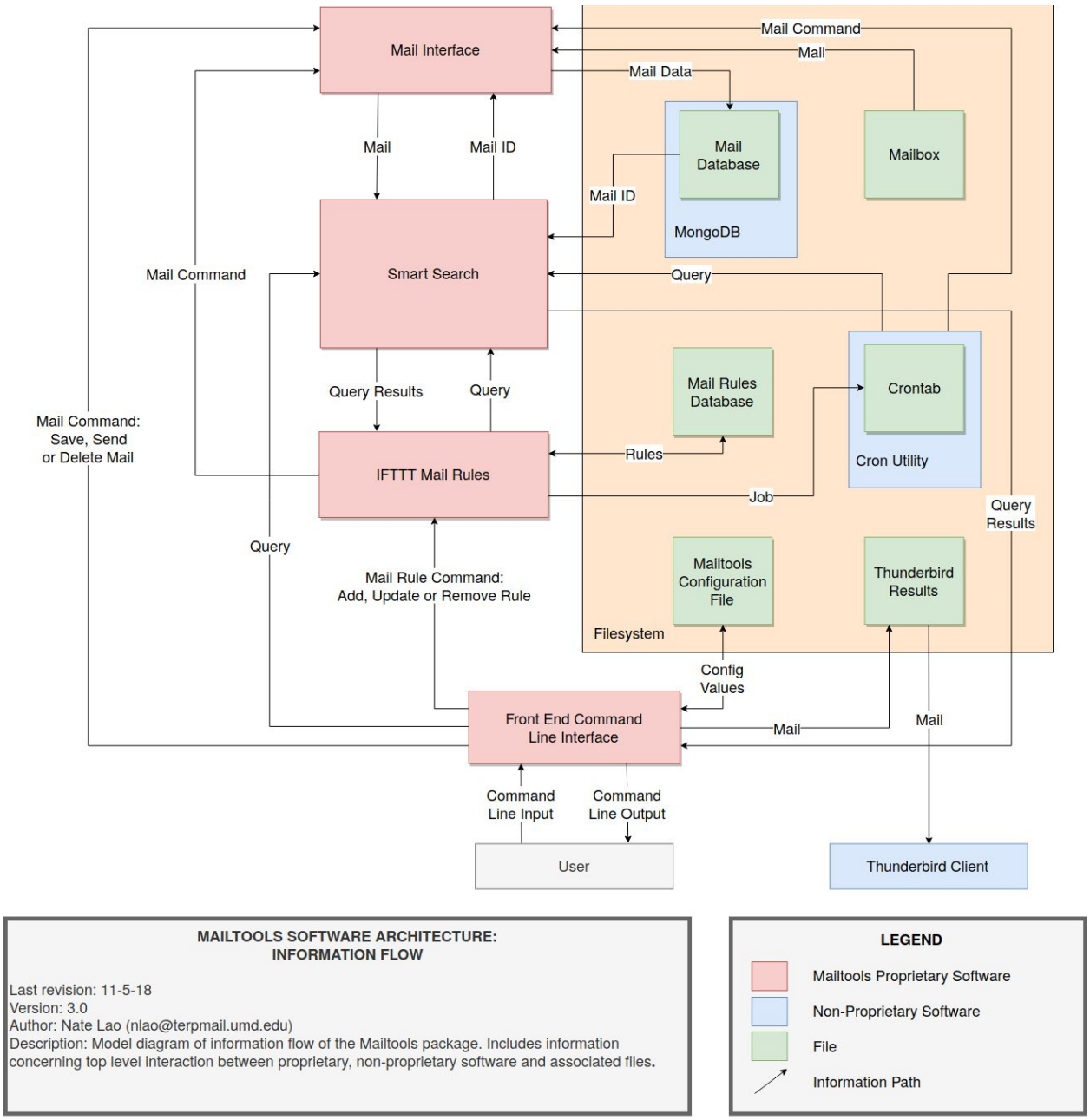

Figure 2.3.1: Information Flow Diagram

Mail Interface

Mailtools, during installation, gathers, parses, and organizes client's emails. *Mailtools* does this by interfacing with a directory on the client machine, which contains archives of emails. The *Mail Interface* (*MI*) then gathers this information, parses each email, and inserts it into the database (DB). *MI* is the only *Mailtools* component that will directly interface with raw mail files.

Front End Command Line Interface

Mailtools includes a command line interface (*CLI*) as a front end. The *CLI* parses and executes user-defined commands to *Mailtools.* See section 3.2.1 for more detail.

IFTTT Mail Rules

Manages the *If-This-Then-Than* (IFTTT) rules for the Mailtools package. IFTTT rules are defined as an event driven action that occur when either the user issues a command to Mailtools or during a preset scheduled time (either periodically or as a future one-shot event.)

Mailbox

The location of the user-defined mail file. The format of this mail file is *mbox*. The location of this file can be anywhere within the filesystem of the user's machine. The user must provide the *mbox* file path in order to process it.

Mailtools Database

The *Mailtools Database* utilizes *MongoDB*, a non-relational database which easily manages scalable data, for implementation. For more details on *MongoDB* see Section 2.2. The following are fields that will be stored for each email:

- *MessageID*: an ID we give each email in the mbox file. The first email will have id 0, and IDs are assigned sequentially
- *From*: from address
- *To*: to address
- *Reply-To*: reply-to address
- *CC*: cc address
- *BCC:* bcc address
- *Timestamp*: date and time
- *Subject:* the mail subject
- *Body*: the body as a text
- *AttachStatus*: 1 if the email contains attachments, 0 otherwise.
- *Attachments*: A list containing all attachments with their name (AttachName), their format (AttachFormat), and a directory to a file containing those attachments saved under the content (AttachContent).

Query results for mail will be returned in JSON format which are easily read using python's JSON parsing library. See figure 2.3.2 for an example of DB query response.

```
{'_id': ObjectId('5c0022bb84bfbd13e0b9d870'),
 'MessageID': 9,
 'From': 'Mailtools CMSC435 <mailtools435@gmail.com>',
 'To': 'Mailtools CMSC435 <mailtools435@gmail.com>',
 'Date': 'Thu, 25 Oct 2018 01:43:35 -0700',
 'Subject': 'More attachments testing',
 'Body': "b'Attachments\\r\\n'",
 'AttachStatus': 1,
 'Attachments': [{'AttachName': 'test',
   'AttachFormat': '.txt',
   'AttachContent': 'Attachments/Inbox/9/'},
  {'AttachName': 'test',
   'AttachFormat': '.docx',
   'AttachContent': 'Attachments/Inbox/9/'},
  {'AttachName': 'test',
   'AttachFormat': '.rtf',
   'AttachContent': 'Attachments/Inbox/9/'}]}
```


Figure 2.3.2: Example Database JSON query output

Crontab

Contains the scripted commands for the unix *cron* utility. *Mailtools* interfaces with this tool to provide scheduled task management to the user.

Mailtools Configuration

Contains the global configuration variables for the *Mailtools* package.

Thunderbird Results File

This file stores query results for the *Thunderbird* client. This allows the user to see the queried mail through their *Thunderbird* graphical interface. The format of this file is *mbox* and the user can specify when to update this file during search.

3. Application Use

The purpose is this section is to explain how *Mailtools* integrates with client's machines, the syntax for the *MailScript* (*.ms*) files and the command line notation. This section also provides sample use cases for *Mailtools*.

3.1 Integration and Installation

Installation of *Mailtools* package is conducted within the mail server machine. The user must specify to *Mailtools* the location of the mailbox file to use. Otherwise by default, *Mailtools* will use the user's inbox location located in /usr/mail/username. This default action can be changed in the Mailtools configuration file located in home/username/.mailtools/config.yml. In this installation process, *Mailtools* parses and databases the information contained in the mail. If changes are made to the mbox file, the user must initiate a complete reingestion of data into the database with --use option.

3.2 Syntax

3.2.1 Script Syntax

Mailtools scripts are written in a language called *MailScript*. *MailScript* files are UTF-8 encoded text files with the *.ms* extension and are formatted in the following manner:

%%%

HEADER

%%%

FILTERS

~~~

FOOTER

~~~

The header contains script metadata including *MailScript* version and declaration of custom actions. The following is a simple header that uses version 1 of *MailScript* and declares the "reverse" custom action which uses the "reverse_msg" function defined in the footer:

%%%

(version 1)

(action reverse reverse_msg)

%%%

The footer consists of valid *Python* code and includes the definition of custom actions as well as any necessary machinery to make them function (imports, helper functions, class definitions, etc). The functions that serve as the bodies of custom actions must be top-level functions that take a single argument - the message struct. For example, this is the corresponding footer to the above header:

~~~

def reverse_msg(msg, env):

    env[msg][“Subject”][::-1]

    env[msg][“Body”][::-1]

~~~

The bulk of the script is made up of filters. Filters are of this basic form:

if (MATCH STATEMENT) {

    ACTIONS

    ...

}

else if (MATCH STATEMENT) {

    ACTIONS

    ...

}

    ...

else {

    ACTIONS

    ...

}

Additionally, filters may be nested.

if(MATCH STATEMENT) {

```
    if(MATCH STATEMENT) { ... }
}
```

For more specific information on *mailscript* please visit unixmailtools.com/mailscript.html.

3.2.2 Command Line Syntax

For every call to Mailtools, the user may provide the mbox file to use. By default, if the mbox file is not provided, the user's inbox file is used instead.

```
mailtools
```

or

```
mailools --use foo.mbox
```

This will allow the user to use the *mbox* file "foo.mbox" (in the current directory).

Search

Once the table has been created/updated on the machine, the user will search in the table specific entries meeting certain criteria as it is defined in the command line. Using the query results, we will store on the machine the emails of interest in the results file (*mbox* format). *Thunderbird* will also use the same results file to display emails. The command line for the search tools is defined as follows:

```
mailtools search [<TAG> <VALUE>]... [--negate] [--exact_string] [-o FIELD | --output FIELD] [--list_field FIELD] [--attachments ATTACH_DIR] [--reply] [--forward RECIPIENT [RECIPIENT ...]]
```

- <TAG> is the field being queried in the search. The following are possible queries for this parameter:
    - -s: search by Subject field.
    - -b: search by Body field.
    - -f: search by From field.
    - -t: search by To field.
    - -rt: search by Reply-To field.
    - -cc: search by CC field.
    - -bcc: search by BCC field.
    - -d: search by Date field.
    - -atn: search by Attachment Name field.
    - -atf: search by Attachment Format field.
    - -ats: search by Attachment Status field.
- --field: search by a user specified FIELD for a VALUE. This is the only <TAG> that accepts only 2 parameters.
  Syntax: mailtools search --field <FIELD> <VALUE>

- <VALUE> is the target of the search. By default, search conducts a partial match. Note that the <TAG> and <VALUE> query pairs are actual optional. If no <TAG> <VALUE> pair is provided, the search will result in the entirety of the input mailbox. Multiple pairs can be combined in a search. The search query will be a conjunctive form of these pairs. (i.e. AND operator between queries).

- --negate is an optional argument that returns the negation of the search query. This is performed after all matching of the original search query.

- --exact_string is an optional argument that returns the results of the exact strings of the input mailbox.

- -o or --output is an optional argument that outputs the results of the search query in a user-specified FILE.
  Syntax: mailtools search <TAG VALUE PAIR> -o FILE

- --list_field is an optional argument that prints to STDOUT the user-specified FIELD of the email results. Multiple --list_field FIELD pairs can be given. The output of this will be query in a comma-separated value matrix (CSV).
  Syntax: mailtools search <TAG VALUE PAIR> --list_field FIELD

- --attachments is an optional argument that outputs the attachments of the email results to a user-specified attachment directory.
  Syntax: mailtools search <TAG VALUE PAIR> --attachments ATTACH_DIR

- --reply is an optional argument that will prompt the user to send a mass email to all addresses in the *From* field in the email results.
  Syntax: mailtools search <TAG VALUE PAIR> --reply

- --forward is an optional argument that forwards the mail results to a list of one or more recipients.
  Syntax: mailtools search <TAG VALUE PAIR> --forward RECIPIENT [RECIPIENT ...]

Process

The process function executes a *mailscript* file.

mailtools process --file FILEPATH [--cron TIME]

- Mailtools runs the *mailscript* defined in FILEPATH.
- --file is REQUIRED as an argument tag.

- FILEPATH is REQUIRED as an argument. This is the relative or absolute path to the *Mailscript* file.
- --cron is an optional argument. The provided TIME code is in *cron* format. Mailtools will save the provided *mailscript* file under the /home/username/.mailtools/mailscript_jobs directory. In order to stop *cronjobs*, the associated command has to be deleted in the crontab and the saved *mailscript* can be optionally deleted in /home/username/.mailtools/mailscript_jobs.
- Examples:
    - mailtools process --file foo.ms
      Run the 'foo.ms' *mailscript* file on the user's inbox.
    - mailtools --use baz.mbox process --file bar.ms --cron '* * * * *'
      Run the 'bar.ms' *mailscript* file on the 'baz.mbox' and set a *cronjob* on this command.
- Optional Arguments:
    - --info Printout Mailtools configuration information.
    - --quiet Run Mailtools without terminal output (quietly).
    - --verbose Run Mailtools with debug information.
    - --reset Reset the Mailtools database.
    - --reset_config Reset the Mailtools configuration to the default configuration at setup.

For more information on *Mailtools* syntax and detailed examples please visit http://unixmailtools.com/guide.html.

4. Testing

There are primarily two components to our acceptance testing. The first component we ensured that the overall system is functionally correct and usable for the customer, Dr. Jim Purtilo. The second component we focused on the user satisfaction of the program. Determining satisfaction of the program will involve subjective evaluation criteria.

4.1 Unit Testing

To ensure basic functionality of *Mailtools* we conducted unit testing on various portions of the system architecture. Unit testing consisted of isolating various components of *Mailtools* and running customized scripts to test their functionality. The following areas within *Mailtools* were examined to ensure usability.

1. Mail Interface
2. *IFTTT* Mail Rules
3. Front End *Command Line Interface*
4. *Smart Search*

All of the sections above successfully passed unit testing proving their basic functionality.

4.2 Integration Testing

We used integration testing to determine if components that are connected and interact in the desired way. We tested every connection defined in the architecture diagram (figure 2.3.1) and ensured that the information flow from component to component is correct. For example, we validated that calling the search command from the command line interface returns the correct emails through the *Smart Search*.

4.3 Acceptance Testing

More in depth than unit or integration testing, acceptance testing is used to ensure that *Mailtools* can work as intended when the user issues various commands and actions. These tests determine whether *Mailtools* can be accepted as the working product its advertised as. Below are the criteria tested for acceptance testing.

1. Can the user display search results in a results file?
2. Can the user execute a search based on negation?
3. Can the user match to searches with exact strings?
4. Can the user redirect results to a specified file?
5. Can the user list a specified field of the search results?
6. Can the user save attachments of emails matching the search in a directory?

7. Can the user reply to emails matching the search results?
8. Can the user forward search results?
9. Can the user execute a *Mailscript* expression defined in the CLI?
10. Can the user execute a given *Mailscript* file?
11. Can the user execute a given *Mailscript* expression or file at a specified time?
12. Can the user can ingest a specified file for mailtools to use on initial start up?
13. Can the user can reingest the same data with additional emails or ingest another file for mailtools to use after already ingesting a file?
14. Can the user can display a help message listing possible actions?
15. Can the user can stop mailtools from printing after commands?
16. Can the user run *Mailtools* in verbose mode?
17. Can the version number and configurations be displayed?
18. Can the *Mailtools* database be reset?
19. Can the default configurations be displayed?

All acceptance testing for *Mailtools* succeeded proving its usability. See the appendix 1 for more detailed results of the acceptance tests.

4.4 Performance Testing

Since the ability to search large amounts of email quickly is central to the product's purpose, we will be testing performance with large datasets to ensure that the code is efficient enough. Performance testing will be conducted by generating a large test data set, then performing searches on it and timing the searches. The average search times will be compared to *grepmail*, which we are using as a performance benchmark that we want to exceed. The execution time will be calculated using the *Python time* module.

Mailtools aims to be more verbose and efficient with searching than *grepmail*. The following tests were performed on a 2GB mbox file:

Action	Mailtools	Mailgrep
Ingest mbox into database	480 seconds = 8 minutes	NA
Search database	22 seconds	38 seconds

Mailtools searching can be done quickly by utilizing the efficiency of MongoDB. Compared to *Mailgrep, Mailtools* is able to more precisely search emails based on full and partial search on various portions of the email such as subject and body. However, the downside to its quick and thorough searching is the data ingestion phase. The majority of time *Mailtools* searches are conducted, their speed is constricted by read and write speeds of the mbox file. Since *Mailtools*

uses mbox files to load data and show output, this can cause some minimal lag when comparing it to Mailgrep. However, *Mailtools* provides much more verbose searching and scripting capabilities rendering its small inefficiencies nominal.

4.5 Assessing User Satisfaction

To gain knowledge about the quality and performance of our project, we will send surveys to our customers. Surveys will allow us to easily determine user satisfaction and find flaws in our application. Two surveys are sent out, one to gain insight on installation and one for general application use.

The first survey is sent to the customer once installation is complete. Its basic purpose is to ensure that the user was able to follow the installation and that our instructions are easy to understand. Questions asked in this survey are:

- Did you find the installation process simple and straightforward?
- What parts of the install were troublesome?
- What needs to be clarified further?

The second survey is sent out after sufficient use of the application. This survey is used to assess user satisfaction and gain valuable feedback. This survey includes the following questions:

- Would you recommend this application to a coworker/friend?
- How often do you use this application?
- Is the functionality everything you expected?
- Do you use the command line or visual interface more?
- How useful did you find the query function?
- How easy was it to learn how to use the application?

Please see appendix 2 for a response from one customer.

5. Cost Estimation

Using *Function Point (FP) Analysis* we were able to predict a cost estimation for *Mailtools*. The following are the predicted *FP* for each task which were determined in the proposal.

5.1 Internal Logic Files

- *Mail Database*: *FP* count of 7
- *Mail Interface* (parser): *FP* count of 7
- Parsed/internal searches: *FP* count of 15
- *Mail Rules* triggers from search results: *FP* count of 7

5.2 External Interface Files

- Raw emails: *FP* count of 5

5.3 External Inputs

- *CLI* Search inputs: *FP* count of 6
- Rules: *FP* count of 6

5.4 External Outputs

- Search results to user: *FP* count of 4

5.5 *Function Point* Totals and Task Hour Costs

The cost estimation calculated in the proposal determined it would take 57 function points to complete all tasks. However, this did not anticipate testing of *Mailtools*, documentation, meetings/strategy sessions or building of our website. In addition, an estimation was determined based on each function point taking around 1.5 tasks hours for a total cost estimation of 85.5 task hours. From recorded tickets we can determine the development of *Mailtools* took 104.75 task hours. With this information we can see the correct estimation of function point analysis was $104.75/1.5 \approx 69.83$ Function Points.

6. Schedule

The following table (figure 7.1.1) displays the amount of hours it took to complete each task related to the development of *Mailtools*.

Task	Cost
Database	15 hours
Rules	25 hours
Creating Raw emails	4.25 hours
Mail parser	13 hours
CLI/Search Inputs	33 hours
Parsed/Internal search	7 hours
Search results to user	7.5 hours
Unit testing	13 hours
Performance testing	3.5 hours
Proposal drafting	32 hours
Presentation drafting	7 hours
Meetings/Group Planning	120 hours
Documentation	26 hours

Figure 7.1.1: Hours Logged For Each Task

7. Product Advertisement

For enterprise level organizations to simple home servers, *Mailtools* offers a robust product solution to mail server management. We aim to provide a powerful tool to search through mail archives in a fast and efficient way while offering users a simple solution to mail management and automation. All instructions on how to download and use *Mailtools* are accessible via our website *unixmailtools.com*.

7.1 Mission Statement

To provide users a simple-to-use and powerful tool for mail management.

7.2 Target Customer

We aim to provide this product to customers in small to high level enterprises. Implementation is not limited in this application; home users can use this product in their own mail servers. Technical knowledge of *Linux* architectures and the terminal is required to use this product.

7.3 Product Features

Mailtools is designed with the following features:

- *Smart Search* - *Mailtools* can efficiently search through mail files using user defined queries based on mail content.
- *If-This-Then-That (IFTTT) Rules* - *Mailtools* can execute commands based on user-defined conditions on mail.
- *Scheduled Mail Tasks* - *Mailtools* can schedule automated tasks periodically or at one time as defined by the user.

Appendix 1:

Rule: Can the user display search results in a results file?

ID	Given	When	Then
1	Email exists matching the from address in the search query	User enters the following command: '$ mailtools search -f someaddress'	The results file will contain all the emails where the from address matched someaddress
2	No email exists matching the from address in the search query	User enters the following command: '$ mailtools search -f someaddress'	The results file will be empty
3	Email exists with a subject containing search terms in the search query	User enters the following command: '$ mailtools search -s somewords'	The results file will contain all emails with somewords contained in the subject
4	No email exists with a subject containing search terms in the search query	User enters the following command: '$ mailtools search -s somewords'	The results file will be empty
5	Email exists with a date stamp matching the date stamp given in the search query	User enters the following command: '$ mailtools search -d somedate'	The results file will contain all emails with somedate as the date stamp
6	No email exists with a date stamp matching the date stamp given in the search query	User enters the following command: '$ mailtools search -d somedate'	The results file will be empty
7	Email exists with a body containing search terms in the	User enters the following command: '$ mailtools search -b	The results file will contain all emails with somewords

	search query	somewords’	contained in the body
8	No email exists with a body containing search terms in the search query	User enters the following command: ‘$ mailtools search -b somewords’	The results file will be empty
9	Email exists matching the to address in the search query	User enters the following command: ‘$ mailtools search -t someaddress’	The results file will contain all the emails where the to address matched someaddress
10	No email exists matching the to address in the search query	User enters the following command: ‘$ mailtools search -t someaddress’	The results file will be empty
11	Email exists matching the bcc address given in the search query	User enters the following command: ‘$ mailtools search -bc someaddress’	The results file will contain all the emails where the bcc address matched someaddress
12	No email exists matching the bcc address in the search query	User enters the following command: ‘$ mailtools search -bc someaddress’	The results file will be empty
13	Email exists matching the cc address given in the search query	User enters the following command: ‘$ mailtools search -c someaddress’	The results file will contain all the emails where the cc address matched someaddress
14	No email exists matching the cc address in the search query	User enters the following command: ‘$ mailtools search -c someaddress’	The results file will be empty
15	Email exists matching the reply-to address given in the search query	User enters the following command: ‘$ mailtools search -rt someaddress’	The results file will contain all the emails where the reply-to address matched someaddress
16	No email exists matching the reply-to address in the search	User enters the following command: ‘$ mailtools search -rt	The results file will be empty

	query	someaddress’	
17	Email exists with a priority matching the priority in the search query	User enters the following command: ‘$ mailtools search -p somepriority’	The results file will contain all the emails with priorities matching somepriority
18	No mail exists with a priority matching the priority in the search query	User enters the following command: ‘$ mailtools search -p somepriority’	The results file will be empty
19	Email exists with an attachment name matching the attachment name in the search query	User enters the following command: ‘$ mailtools search -atn somename’	The results file will contain all the emails with attachment name matching somename
20	No mail exists with an attachment name matching the attachment name in the search query	User enters the following command: ‘$ mailtools search -atn somename	The results file will be empty
21	Email exists with an attachment status matching the attachment status in the search query	User enters the following command: ‘$ mailtools search -ats 0/1’	The results file will contain all the emails with attachment status matching 0/1
22	No mail exists with an attachment name matching the attachment name in the search query	User enters the following command: ‘$ mailtools search -ats 0/1’	The results file will be empty
23	Email exists with an attachment format matching the attachment format in the search query	User enters the following command: ‘$ mailtools search -atf someformat’	The results file will contain all the emails with attachment format matching someformat
24	No mail exists with an attachment format matching the	User enters the following command: ‘$ mailtools search	The results file will be empty

	attachment format in the search query	-atf someformat’	
25	Email exists with value in specified field matching the search query	User enters the following command: ‘$ mailtools search <FIELD> <VALUE>’	The results file will contain all emails with value in FIELD matching VALUE
26	No email exists with value in specified field matching the search query	User enters the following command: ‘$ mailtools search <FIELD> <VALUE>’	The results file will be empty

Rule: Can the user execute a search based on negation?

ID	When	Given	Then
27	Emails exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> --negate’	The results file will contain all emails that do not match the search query
28	No email exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> --negate’	The results file will contain all the emails in the mbox file

Rule: Can the user match to searches with exact strings?

29	Emails exists matching the search query exactly	User gives the following command: ‘$ mailtools search <Tag Value Pair> --exact_string’	The results file will contain all emails that perfectly match the query
30	No email exists matching the search query or only partially matching	User gives the following command: ‘$ mailtools search <Tag Value Pair>	The results file will be empty

		--exact_string’	

Rule: Can the user redirect results to a specified file?

31	Emails exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> -o FILE’	FILE will contain all emails that perfectly match the query
32	No email exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> - o FILE’	FILE will be empty

Rule: Can the user list a specified field of the search results?

33	Emails exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> --list_filed FIELD’	The FIELD of the search results will be printed
34	No email exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> --list_field FIELD’	Nothing will be printed

Rule: Can the user save attachments of emails matching the search in a directory?

35	Emails exists matching the search query	User gives the following command: ‘$ mailtools search <Tag Value Pair> --attachments ATTACH_DIR’	ATTACH_DIR will contain the attachments of all emails match the query
36	No email exists	User gives the	ATTACH_DIR will

	matching the search query or none of the emails matching the search have attachments	following command: '$ mailtools search <Tag Value Pair> --attachments ATTACH_DIR'	be empty

Rule: Can the user reply to emails matching the search results?

37	Emails exists matching the search query	User gives the following command: '$ mailtools search <Tag Value Pair> --reply'	The user will reply to the emails matching the search query
38	No email exists matching the search query	User gives the following command: '$ mailtools search <Tag Value Pair> --reply'	The user will not be prompted to send a reply message

Rule: The user wishes to forward search results

39	Emails exists matching the search query	User gives the following command: '$ mailtools search <Tag Value Pair> --forward RECIPIENT [RECIPIENTS..]'	All emails matching the search query will be forwarded to RECIPIENT or a list of RECIPIENTS
40	No email exists matching the search query	User gives the following command: '$ mailtools search <Tag Value Pair> --forward RECIPIENT [RECIPIENTS..]'	No emails will be forwarded

41	Emails exists matching the search	User gives the following command:	The results file will contain all emails that

	query exactly	‘$ mailtools search <Tag Value Pair> exact_string’	perfectly match the query
42	No email exists matching the search query or only partially matching	User gives the following command: ‘$ mailtools search <Tag Value Pair> --exact_string’	The results file will be empty

Rule: Can the user execute a *Mailscript* expression defined in the CLI?

ID	When	Given	Then
43	The user gives a valid mailscript expression that can run without exceptions or errors	User enters the following command: ‘$ mailtools process someexpression’	Someexpression will be executed
44	The user gives a valid mailscript expression that can run without exceptions or errors	User enters the following command: ‘$ mailtools process someexpression’	An error message is displayed indicating the problem associated with someexpression

Rule: Execute a given mailscript file

ID	When	Given	Then
45	The user gives a valid mailscript file that can run without exceptions or errors	User enters the following command: ‘$ mailtools process somefile’	Somefile will be executed
46	The user gives an invalid mailscript file that cannot run without exceptions or errors	User enters the following command: ‘$ mailtools process somefile’	An error message is displayed indicating the problem associated with somefile

Rule: Can the user execute a given *Mailscript* expression or file at a specified time?

ID	When	Given	Then
47	The user gives a valid mailscript file/expression that can run without exceptions or errors	User enters the following command: '$ mailtools process something TIME'	Something will be executed at TIME
48	The user gives an invalid mailscript file/expression that cannot run without exceptions or errors	User enters the following command: '$ mailtools process something TIME'	An error message is displayed indicating the problem associated with something

Rule: Can the user can ingest a specified file for mailtools to use on initial start up?

ID	When	Given	Then
49	The user specifies an existing mbox file	User enters the following command: '$ mailtools --use somefilepath'	The file at somefilepath will be ingested, displaying a success message and the time it took to ingest
50	The user specifies a non existent mbox file	User enters the following command: '$ mailtools --use somefilepath'	An error message will be displayed indicating the file is non existent

Rule: Can the user can reingest the same data with additional emails or ingest another file for mailtools to use after already ingesting a file?

ID	When	Given	Then
51	The user specifies an existing mbox file	User enters the following command: '$ mailtools --use	The file at somefilepath will be ingested, displaying a

		somefilepath’	success message and the time it took to ingest
52	The user specifies a non existent mbox file	User enters the following command: ‘$ mailtools --use somefilepath’	An error message will be displayed indicating the file is non existent

Rule: Can the user can display a help message listing possible actions?

ID	When	Given	Then
53	At any stage of use	User enters the following command: ‘$ mailtools --help’	A help message will be displayed

Rule: Can the user can stop mailtools from printing after commands?

ID	When	Given	Then
54	At any stage of use	User enters the following command: ‘$ mailtools --quiet somecommand’	Messages will not be printed by Mailtools while executing somecommand

Rule: Can the user run *Mailtools* in verbose mode?

ID	When	Given	Then
55	At any stage of use	User enters the following command: ‘$ mailtools --verbose	More details about execution will be printed
56	The user has already used --verbose and wishes to reverse its effects	User enters the following command: ‘$ mailtools --verbose	More details about execution will not be printed

Rule: Can the version number and configurations be displayed?

ID	When	Given	Then
57	At any stage of use	User enters the following command: ‘$ mailtools --info	A info message will be displayed

Rule: Can the *Mailtools* database be reset?

ID	When	Given	Then
58	At any stage of use	User enters the following command: ‘$ mailtools --reset’	The database specified in the config file will be reset

Rule: Can the default configurations be displayed?

ID	When	Given	Then
59	At any stage of use	User enters the following command: ‘$ mailtools --reset_config’	The configuration file will be reset to its default values.

Appendix 2:

Below is one response from a customer. For privacy purposes the name and affiliated company will remain anonymous.

Initial survey with questions and answers:

- Did you find the installation process simple and straightforward?
    - For the most part the installation guide on the website was very well documented and straightforward.
- What parts of the install were troublesome?
    - There was some confusion with errors that I was getting during installation process and I could not find a troubleshoot section.
- What needs to be clarified further?
    - One thing that needs to be clarified is that MongoDB is used and needs to be separately installed on the machine which uses Mailtools.

Secondary follow up survey with questions and answers:

- Would you recommend this application to a coworker/friend?
    - I would recommend this application to a coworker or friend. Its easy integration mail clients such as Thunderbird made it easy to use and its searching capabilities out performed grepmail.
- How often do you use this application?
    - I use this application when I attempt to search large archived mail, usually dated from at least a year ago. This does not occur too often for me but, I would say about three to four times a month.
- Is the functionality everything you expected?
    - Yes.
- Do you use the command line or visual interface more?
    - I use a combination of both. Primarily the CLI to create scripts and queries and Thunderbird to view the results.
- How useful did you find the query function?
    - This was the main attraction to the software for me, so very useful.
- How easy was it to learn how to use the application?
    - There was a little bit of a learning curve, but the --help option was implemented well facilitating my use of the product.