Pigeon - E-mail Client Application

Test Plan

by
Andreas Kaag Thomsen
Asger Song Høøck Poulsen
Buster Salomon Rasmussen
Niels Viggo Stark Madsen

A Software Engineering
Report

November 18, 2022

1 Preface

This report is intended for the client of the Pigeon e-mail system, more specifically Associate Professor, Lukas Ersterle. The report is also composed for the sake of the team behind in order to review the e-mail client later on. The report was revised last on December 20, 2022.

2 Introduction

The objective of this report is to ensure the e-mail client conforms to functional and non-functional requirements described in the *Requirements and Work Plan* document. As the development of the application continues from a minimal viable product into a minimum marketable product, documented testing ensures that the expectations of the end product, agreed between the company and costumer, is met. This report consists of three parts: Testing Strategy, Test Plan and Result processing.

3 Testing Strategy

Since the early development stages of the e-mail client, we have almost exclusively had a Test-Driven development (TDD) approach to the project. The code development and testing are interleaved such that the code is developed incrementally and tested before we continue to the next step of the development which goes hand in hand with our agile development methods.

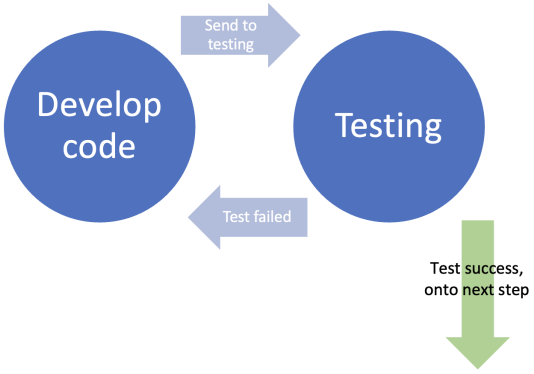

Figure 1: Development cycle

As we are working within agile methods the strategic testing approach for the development of the e-mail client will rely on the *Agile Testing Quadrants* for test inspiration, and as an extension of this *White Box Testing* and *Black Box Testing*.

One of the many advantages of the Agile method is that it provides continuous feedback to the development team especially such that bugs and issued can be resolved faster, compared to if testing was scheduled after all the different code components were implemented.

3.1 Agile Testing Quadrants

As a brief overview of the Agile Testing Quadrants depicted in Figure 2:

- **Q1:** Unit and component tests are performed throughout the application's development that provide feedback to the developers on the quality of their code on an ongoing basis, usually through repeated, automated processes.
- **Q2:** With a combination of manual and automated tests, these 'business-facing' tests are more customer-focused, but they support your application build as well. These include functional tests, which ensure the product does what it is meant to do.
- **Q3:** Involves manual testing and end-user testing. The aim here is to gain feedback and improve the quality of the product, ensuring it is fit for the designed purpose.
- **Q4:** These are technology-facing performance tests, like load testing and checking the data security of the application.

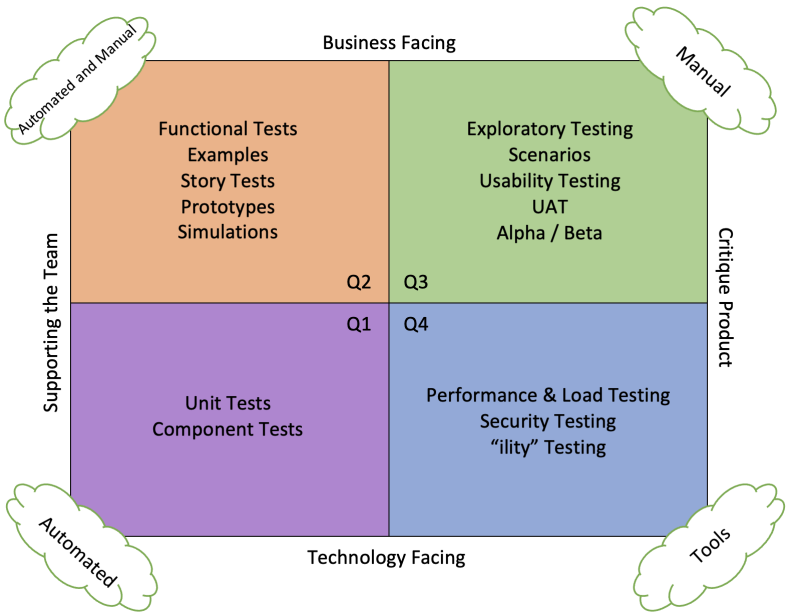

Figure 2: Agile Testing Quadrants

This is further specified project in subsection 4.1.

3.2 White Box Testing

Is a structural testing strategy which considers the internal structure of a system or component. It focuses on the unit, integration and system levels of the software. The point of White Box Testing is to test with specified inputs to exercise paths through the code and determine the expected outputs.

3.3 Black Box Testing

Is a behavioral testing strategy which focuses on how a system acts rather than the mechanism behind its functions. It focuses on workflows, configurations, performance, and all elements of the

user journey. The point of Black Box Testing is to test from an end-user's perspective. Activity diagrams are good at providing inspirations for test cases for this type of testing.

4 Test Plan

In this section we are going to describe what types of testing we wish to perform, and what requirements we wish to test. All of the tests are performed manually.

4.1 Type of testing

Cf. the *Agile Testing Quadrants*, the tests can be partitioned into four quadrants. With the quadrants as the underlying basis, it will be discussed which types of testing there are relevant for the Pigeon email client below.

Q1

Ideally the tests of the first quadrant *unit-tests and component-tests*, should be a part of our testing strategy. This should in fact be a very important part of the testing strategy in every software product, as the units are the basis for any component, and so on. However, unit-testing is very comprehensive, as it requires a lot of tests. Thus, after talking with our client, we have decided not to include unit-testing, nor component-testing as a part of our testing strategy.
In general white-box testing is not a part of our testing strategy.

Q2

From Q2 we will focus on *Functional Tests and Story Tests*. All of these goes under the category of *black-box testing*, and they are very much *business driven* and done in close-cooperation with the customer.
Functional tests will be performed explicitly when we test all the mandatory functional requirements, that were specified in report one. Precisely how these will be tested, is described in subsubsection 4.4.1.
Story tests will not be performed explicitly, due to the fact that the functional requirements are derived from the user stories. Thus, correct story test results are ensured, if we get correct results from the functional requirements tests.

Q3

From Q3 we will focus on *Usability-, Alpha/Beta-, and User acceptance testing*. Concurrently with the development so far, we have done **Alpha testing**, as our customer have tested a prototype of the email client. In the future we will ask our customer to test the email client again. This is also a part of the User acceptance testing.

Q4

In general we will not focus on this quadrant as we lack the resources and security is not a focus of this project, as per costumer's request.

4.2 Boundary value analysis

Our test cases highly reflect that we perform boundary value analysis. That is, for most of the functionalities we test success cases as well as failure cases. It is critical for us that our program will not crash with invalid inputs but rather provide some warning/error message instead. For instance, the universal convention for e-mail addresses is 64 characters before the '@' and 255 characters after (all in all 321). This is a clear upper boundary. The lower boundary is an e-mail address that is not on the form 'local@domain.com'. This will also be tested.

4.3 Critical functionalities

It is obvious that some functionalities are more critical than others. For instance, if the user is not able to login, none of the other functionalities are relevant. Also we prioritize the basic e-mail operations such as sending and receiving e-mails. This is clear in our test-cases below since these functionalities are tested with more test cases than other, less critical functionalities, are.

4.4 Scenario testing

Scope defines the functional or non-functional requirements of the software that will be tested, and out of scope defines and discusses what will not be tested.

4.4.1 In scope

The following requirements which were derived in Report 1 are tested:

- UFM.(1) 5 test cases.
  We have derived the number of test-cases from a decision table, which can be seen at Appendix A (Figure 5). Please note, that we actually have five test-cases, as we are testing for tree different types of invalid emails: non-existing emails, empty emails (hence, also non-existing) and email addresses that are too long (and therefore also invalid).

  Test case 1

  - **Original state:** An empty login screen.
  - **Action:** Fill username with: 'sandkassesandbox@gmail.com' and password with our personal access token. Press the login button afterwards.
  - **Expected result:** The login form should be hidden, and the mailbox form should appear.

  Test case 2

  - **Original state:** An empty login screen.
  - **Action:** Fill username with: 'sandkassesandbox@gmail.com' and wrong password. Press the login button afterwards.
  - **Expected result:** A messagebox should appear saying that the e-mail/password combination is unknown.

  Test case 3

  - **Original state:** An empty login screen.
  - **Action:** Fill username with: 'SOMEINVALIDUSERNAME@gmail.com' and password with our personal access token. Press the login button afterwards.

- **Expected result:** A messagebox should appear saying that the e-mail/password combination is unkwown.

Test case 4

- **Original state:** An empty login screen.
- **Action:** None.
- **Expected result:** Login button should be disabled.

Test case 5

- **Original state:** An empty login screen.
- **Action:** Fill username field with at least 321 characters (including '@' and '.com'). Fill password.
- **Expected result:** Login button should be disabled.

- UFM.(2) 2 test cases
As there is only one variable, i.e. is the checkbox checked or unchecked, and hence only two test-cases.
Test case 1

  - **Original state:** An empty login screen.
  - **Action:** Fill username with: 'sandkassesandbox@gmail.com' and password with our personal access token. Check the 'remember me' checkbox. Press the login button afterwards. Wait for the logging in to finish and close the program. Open the program again.
  - **Expected result:** The username and password text boxes should be filled, and we should be able to simply press login and the Mailbox should open.

  Test case 2

  - **Original state:** An empty login screen.
  - **Action:** Fill username with: 'sandkassesandbox@gmail.com' and password with our personal access token. Uncheck the 'remember me' checkbox. Press the login button afterwards. Wait for the logging in to finish and close the program. Open the program again.
  - **Expected result:** The username and password text boxes should be empty.

- UFM.(3) 1 test case
Test case 1

  - **Original state:** The user has entered valid credentials.
  - **Action:** Press the login button.
  - **Expected result:** The mailbox opens and the folder names become visible.

- UFM.(4)
Test case 1

- **Original state:** The login procedure has executed flawlessly and the Mailbox is open with folder names visible. Some non-empty folder has been clicked and the message summaries are visible.
- **Action:** Double-click some message.
- **Expected result:** The Mailbox-form should stay open and a new form should appear in front of it displaying the body of the message, subject, sender, recipients, date and possibly attachments. If the message was not previously read, it is now indicated that it is read.

- UFM.(5)
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open with folder names visible.
  - **Action:** Press the 'new-email' button.
  - **Expected result:** The Mailbox-form should stay open and a new form should appear in front of it displaying several active textboxes labeled 'recipient', 'Cc', 'subject', etc. It should be indicated how to add multiple recipients and how these should be separated.

- UFM.(6)
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open. The user has pressed the 'new e-mail' button and and the new e-mail form has opened. The user has filled at least the recipient field with a valid recipient.
  - **Action:** Press the 'send' button.
  - **Expected result:** The current form should close and the mailbox should be visible. The status log should indicate that the message has been sent.

- UFM.(7)
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open. The user has pressed the 'new e-mail' button and and the new e-mail form has opened.
  - **Action:** Write 'NOTVALID' in the recipient field.
  - **Expected result:** The text in the recipient should turn red, and the send button should be grey/disabled.

  Remark: We are not able to validate an e-mail further than simply checking it is on the form 'text@text.text'. However, GMail provides a notification e-mail if the message could not be delivered.

- UFM.(8)
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open. The user has clicked some non-empty folder and the message summaries are visible. The user has clicked some message and the message is opened.

- **Action:** Press the 'forward' button.
- **Expected result:** A new form similar to the new e-mail form should appear with subject and body field auto-filled with the contents of the message that was just read. If an attachment was included in the orignal e-mail a window should pop up asking the user if he/she wishes to include this in the forwarded e-mail.

- UFM.(9) 2 test cases
  As the e-mail client should be able to update the inbox regardless of internal or external actions, there are 2 test cases:
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open. The user has clicked on some folder that can be either empty or non-empty.
  - **Action:** Send an mail to the logged in user and press the 'refresh' button.
  - **Expected result:** The e-mail summary is now visible above the rest of the messages in the folder. The folder count next to the folder name has incremented by 1.

  Test case 2
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open. The user has clicked on some folder that can be either empty or non-empty.
  - **Action:** Login to the same user's account from another device/e-mail client and delete an e-mail. Press the 'refresh' button.
  - **Expected result:** The folder count of the currently deleted message's folder is decremented by one and the message summary is no longer visible.

  Remark: Note that this requirement is now somewhat superfluous since the mail client refreshes automatically as described in UNM2 and UNM5. We have, however, chosen to keep this functionality to give the user a familiar interface.

- UFM.(10)
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open with folder names visible.
  - **Action:** Click all folders one by one.
  - **Expected result:** When each folder is clicked, the messages inside this folder should appear to the right. If there are no messages this should be displayed in a status field.

- UFM.(11) The user shall be able to reply an e-mail by same manner as described in UFM.(6)[1].
  Test case 1
  - **Original state:** The user is currently reading a message.
  - **Action:** Click the 'reply' button.
  - **Expected result:** A new form similar to the new e-mail form should appear with subject, recipient, and body field auto-filled with the contents of the message that was just read.

[1] This requirement was not described in Report 1 handed in and has been updated for the final report

- UFM.(12)
  Test case 1
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open with folder names visible.
  - **Action:** The user double clicks on a message marked "unread" and highlighted red.
  - **Expected result:** The e-mail will be opened and unmarked as "unread" as well as the color highlight being removed.

  Test case 2
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open with folder names visible.
  - **Action:** The User selects a "read" e-mail and clicks on "read/unread".
  - **Expected result:** The e-mail is marked "unread" in the list and highlighted in red.

  Test case 3
  - **Original state:** The login procedure has executed flawlessly and the Mailbox is open with folder names visible.
  - **Action:** The User selects an "unread" e-mail and clicks on "read/unread".
  - **Expected result:** The string 'unread' is removed and the mail is no longer marked red.

- UFO.(1)
  Test case 1
  - **Original state:** The user has logged in and the mailbox is open.
  - **Action:** Click the settings button. Click 'add another account' button. A form similar to the login form should appear and the user fills in the credentials of another e-mail account. Press save.
  - **Expected result:** The user should now be able to see folders from the newly added account and interact with these in a similar way as described in all the other test cases and requirements.

- UFO.(2)
  Test case 1
  - **Original state:** The user has logged in and the mailbox is open.
  - **Action:** Click the settings button. Click 'Create auto respond' button. A window will pop op asking the user if they want to use the default auto response or create a new. The user clicks the default auto response and the window closes. Send an e-mail to the currently logged in user.
  - **Expected result:** The user that sent the e-mail should receive an e-mail as an automatic response.

- UFO.(3) 3 test cases
  The three test cases describe: Adding an attachment, removing an attachment and sending an email with an attachment.
  Test case 1

- **Original state:** The user is currently writing an e-mail either as a response, forward or new e-mail.
- **Action:** Click the 'attach' button, which opens the file explorer. Select a file.
- **Expected result:** The filename of the newly selected attachment should now be visible in a listing as well as the total size of attachments in MB.

Test case 2

- **Original state:** The user has selected an attachment as described above.
- **Action:** Select some attachment from the listing and click the 'Remove attachment' button.
- **Expected result:** The filename of the deleted attachment is no longer visible in the listing. If it was the last attachment, the remove attachment button and listing becomes inactive.

Test case 3

- **Original state:** The user has selected an attachment as described above and filled at least the recipient field as well.
- **Action:** Press send.
- **Expected result:** The recipient, to which the e-mail was sent, should now be able to open the e-mail and download the attachment.

- UFO.(4)
Test case 1

- **Original state:** The mailbox is open and the messages are loaded.
- **Action:** Fill the search query text field with a specific sender, select the "sender" field that should be searched within. Press the search button or enter.
- **Expected result:** All e-mails matching the search query should be visible.

Test case 2

- **Original state:** The mailbox is open and the messages are loaded.
- **Action:** Fill the search query text field with a specific content, select the "content" field that should be searched within. Press the search button or enter.
- **Expected result:** All e-mails matching the search query should be visible.

Test case 3

- **Original state:** The mailbox is open and the messages are loaded.
- **Action:** Fill the search query text field with a specific subject, select the "Subject" field that should be searched within. Press the search button or enter.
- **Expected result:** All e-mails matching the search query should be visible.

- UFO.(5)
Test case 1

- **Original state:** The user has started the process of writing an e-mail.

- **Action:** The user closes the window and chooses 'yes' when asked if he/she wishes to save the e-mail as draft. Alternatively, the user clicks the "draft" button.
- **Expected result:** The email should be saved to the drafts folder and appear here.

- UFO.(6)
  Test case 1
  - **Original state:** The user has logged in and the mailbox is open.
  - **Action:** Click the settings button. Click 'Create auto respond' button. SA window will pop op asking the user if they want to use the default auto response or create a new. The user clicks the "create a new" button and the user should fill in the details and press save. Send an e-mail to the currently logged in user.
  - **Expected result:** The user that sent the e-mail should receive an e-mail similar to the one created.

- UFO.(7)
  Test case 1
  - **Original state:** The user have the mailbox windows open.
  - **Action:** User fills in the textbox in the bottom left corner and clicks the corresponding "Add folder" button.
  - **Expected result:** A folder is created with the corresponding name.

- UFO.(8)
  Test case 1
  - **Original state:** The user have instantiated a very long process, for example loading many emails from the server.
  - **Action:** User clicks cancel button on the loading indicator.
  - **Expected result:** The operation is canceled.
  - **Final result:** Test failed, not implimented.

- UFO.(9)
  Test case 1
  - **Original state:** The user has opened the settings window and the 'offline mode' checkbox is unchecked.
  - **Action:** User clicks 'offline mode' checkbox so that it is checked and restarts the application without internet connection.
  - **Expected result:** The mailbox should open and all functionality that needs server interaction are disabled.

  Test case 2
  - **Original state:** The user have opened the settings window and the 'offline mode' checkbox is checked.
  - **Action:** User clicks 'offline mode' checkbox so that it is unchecked and restarts the application without internet connection.

- **Expected result:** The mailbox should not open and it should be indicated that there is no internet connection.

- UFO.(10)
  Test case 1
  - **Original state:** The user have opened the settings window and the 'local storage' checkbox is unchecked.
  - **Action:** User clicks 'local storage' checkbox and restarts the application.
  - **Expected result:** All email are fetched and saved offline to a local file. This should increase the speed of program.

  Test case 2
  - **Original state:** The user have opened the settings window and the 'local storage' checkbox is checked.
  - **Action:** User clicks 'local storage' checkbox and closes the application.
  - **Expected result:** There is nothing stored locally and the speed of the program should now have decreased.

- UFO.(11)
  Test case 1
  - **Original state:** The user has created some custom folder as described in UFO.(7) and the mailbox is open.
  - **Action:** User marks the custom folder and clicks the 'delete folder' button.
  - **Expected result:** The concerned folder should disappear. If the folder contained messages the user should be notified and given the choice whether or not to proceed with the action.

- UFO.(12)
  Test case 1
  - **Original state:** The user have opened the setting window and the 'local storage' checkbox is checked.
  - **Action:** User clicks the 'clear cache' button.
  - **Expected result:** The e-mail client should restart and when the user logs back in the client should automatically fetch e-mails from the server.

- UFO.(13)
  Test case 1
  - **Original state:** The user has logged in and the mailbox is open. A new folder has been created by the user.
  - **Action:** User selects a message and selects the new folder in the dropdown menu in the top right corner and clicks 'move'.
  - **Expected result:** The message should appear in the selected folder and disappear from the previous folder it was moved from.

- UNM.(2)
  Test case 1
  - **Original state:** The User is on the main Mailbox window.
  - **Action:** An email has just been sent to the user. The user waits.
  - **Expected result:** The new email should automatically show in the inbox provided it was not spam.
- UNM.(3)
  Test case 1
  - **Original state:** The User is on the main Mailbox window.
  - **Action:** Cross reference every email with the Gmail server emails.
  - **Expected result:** No missing email.
- UNM.(4)
  Test case 1
  - **Original state:** The User is active on an arbitrary part of the client.
  - **Action:** Press the close button marked with an 'x' in the top right corner.
  - **Expected result:** The program closes all running processes and frees the used memory.
- UNM.(5)
  Test case 1
  - **Original state:** The User is in a arbritrary e-mail folder and has selected an e-mail.
  - **Action:** Press the "delete" button in the top right corner. A window will appear asking if you are sure you want to delete the selected e-mail. Press "Yes" and do nothing.
  - **Expected result:** The program should automatically update and the seleceted e-mail has been deleted.

  Test case 2
  - **Original state:** The User is in a arbritrary e-mail folder and has selected an e-mail.
  - **Action:** Press the "trash" button in the top right corner.
  - **Expected result:** The program should automatically update and the selected e-mail has been moved to the trash folder.

  Test case 3
  - **Original state:** The User is in an e-mail folder the user has created themself.
  - **Action:** Press the "delete folder" button in the bottom left corner.
  - **Expected result:** The program should automatically update and the selected folder has been deleted. If at least one e-mail is inside the folder a warning message will appear asking if the user is sure.

  Test case 4
  - **Original state:** The User is in an e-mail folder the user has not created themself.

- **Action:** Press the "delete folder" button in the bottom left corner.
- **Expected result:** An error message should appear saying that the user is not allowed to delete folders he/she has not created themselves.

- UNO.(1)
  Test case 1
  - **Original state:** The user has logged in and opened the 'blacklist' settings from the settings panel.
  - **Action:** Add some sender and some word to the blacklist. Send 1 e-mail from the blocked sender and 1 e-mail from a different sender containing the blacklisted word.
  - **Expected result:** The email ends up in the spam folder after automatically.

- UNO.(2)
  Test case 1
  - **Original state:** The user is logged in and and in the mailbox.
  - **Action:** Open an email containing CSS and HTML encoding.
  - **Expected result:** The email should be displayed with correct formatting.

4.4.2 Out of scope

- UNM.(1)
  Note that this would be tested by using some sort of timing library and implementing some testing code that times the responsiveness of every action and asserts that this is within 1 second. Since doing these types of tests is out of scope for this course, these kinds of tests will not be implemented. Instead as long as the user gets a feeling of instant responsiveness while using the e-mail client UNM.(1) will be seen as fulfilled.
  Test case 1
  - **Original state:** An arbitrary state in the client.
  - **Action:** Test all actions and time how long it takes to respond.
  - **Expected result:** All actions are responsive within 1 second.

4.5 Risks and Issues

We encounter some risks, due to the limitations of our testing capacity.

We do not have time to test the libraries we use to build the e-mail client. Thus we cannot guarantee that they are not malicious or faulty in some way. We have to trust other people who have vetted the libraries. The primary library used for the e-mail client is the MimeKit library[2]. Since MimeKit is an open source highly respected library we can safely assume it to be risk free, but we did not verify this ourselves, and thus cannot guarantee this.
Our e-mail client is dependent on Google to operate. This is something we have no control over and cannot test thoroughly to assure stable operation This could mean that the user is unable to use most of the functionality of our product if for example Google's servers are down.

[2]For more information: http://www.mimekit.net

Another aspect to this is that the way we interface with Google's email servers through the API. The API could be changed by google, making our product malfunction which would be out of our control.
The security of our e-mail client will not be tested since this is out of scope for this course. There will be some security risks when using the product, but the client is informed of this. Thus the functionality of the program can potentially be compromised by a malicious actor.

5 Result processing

All the bugs and issues found during testing will be logged in the backlog in Jira.

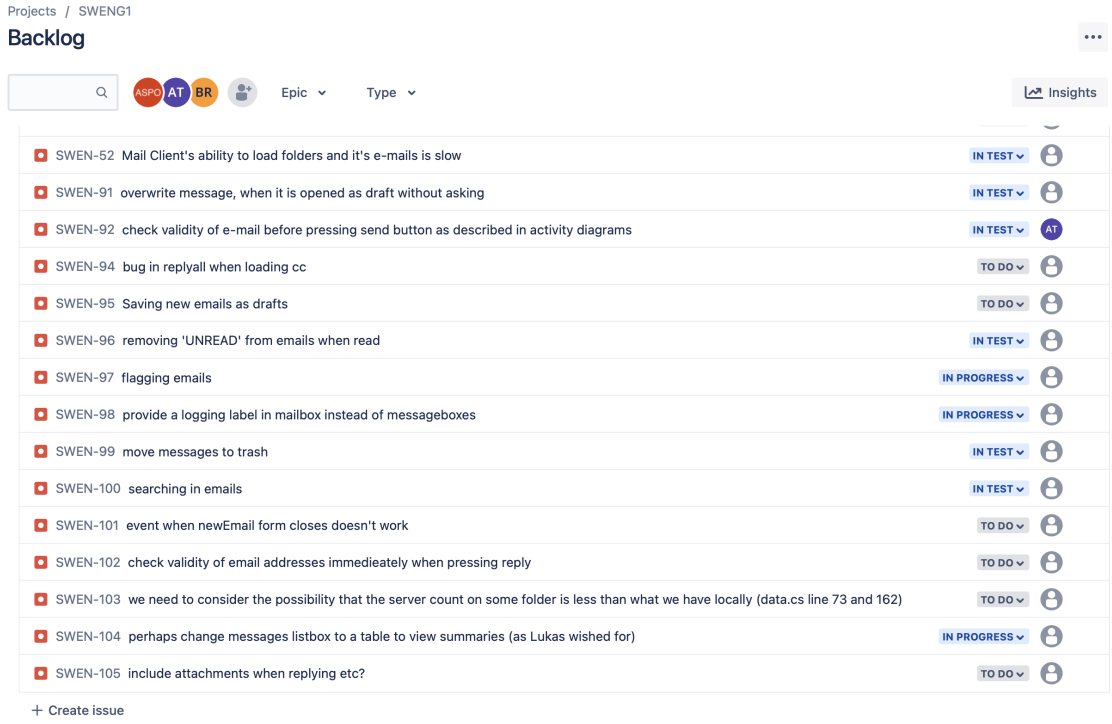

Figure 3: Backlog Jira November 18, 2022

This provides an overview of all development issues and their current process. Any useful information regarding the issue is attached to the bug such that any developer can tackle the issue without speaking with the team member who added the bug to the backlog.

A team member can assign himself to a bug such that the rest of the team knows that this specific bug is under progress. During every stand-up meeting specific bugs are selected and dragged into a sprint and assigned to a specific team member such that it is ensured that those bugs are fixed. The bugs' priority are discussed during the stand-up meeting. When a bug is fixed it is set "in review" and then a team member who did not collaborate on the fixing of the bug tests and confirms that the bug has been fixed and updates this in Jira.

This also applies for general testing and development of the application. Each requirement is included

in our backlog as a separate task. Since every test case is based on these requirements, the current state of this task indicates the state of the test case. Therefore when performing the test cases, the results of these test cases are found in Jira.

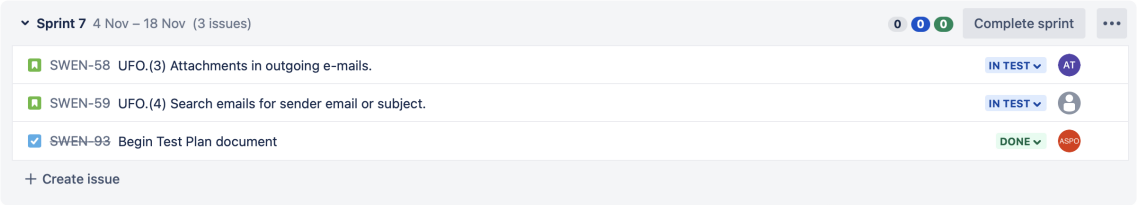

Figure 4: Sprint 7

By documenting the results of test cases in Jira, every team member has constant access to the progress of the project which requires less constant communication between members who may be working on different parts of the project. The documented test results also give the team grounds for which path in the Development cycle depicted in Figure 1 to take.

Appendix

Appendix A - Decision table UFM. (1)

		Rules			
		1	2	3	4
Conditions	Valid email	T	T	F	F
	Invalid email	F	F	T	T
	Valid password	T	F	F	T
	Invalid password	F	T	T	F
Actions	Login	X			
	Show error		X	X	X

Apply inverse rule

		Rules			
		1	2	3	4
Conditions	Valid email	T	T	F	F
	Valid password	T	F	F	T
Actions	Login	X			
	Show error		X	X	X

Using redundancy rule, and a bit of common sense

		Rules		
		1	2	3
Conditions	Valid email	T	T	F
	Valid password	T	F	-
Actions	Login	X		
	Show error		X	X

Figure 5: Decision table UFM.(1)