SE 3XA3: Test Report
MAC Schedule Importer

Team 12, Team 0C
Cassandra Nicolak, nicolace
Michelle Leung, leungm16
Winnie Liang, liangw15

December 6, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
12/02/18	1.0	Revision 0
12/05/18	1.1	Revision 1

This document presents the testing results from the test cases, methods and tools described in the Test Plan document. The test report will give a summary of the function and non-functional requirement evaluations. This document will also list the outcomes of the unit testing for each of the modules that are described in the Module Guide document changes made from testing.

1 Functional Requirements Evaluation

Description of Tests: The purpose of these tests is to ensure that the user is able to use the software according to the requirements in the Specifications Documentation. These tests will include: mostly manual testing. The correctness will be verified through unit testing.

Test Name: TFR01

Results: The user is able to access the correct location to download their schedule from a button in the user interface. The user clicks 'Open Timetable' under 'File' in the menu of the application.

Test Name: TFR02

Results: The user is able to verify that the information being inserted into the calendar is correct. The application provides the user with this information through the use of a text box in the user interface.

Test Name: TFR03

Results: The user is able to make changes to their McMaster timetable and re-import this new schedule. The application accounts for multiple uses from a user by having individual calendars imported.

Test Name: TFR04

Results: The user is able to navigate the user interface as it has a simple GUI. This was tested through User tests and feedback surveys.

Test Name: TFR05

Results: The user is able to use the application easily and quickly by using the 'Help' menu and accessing a User Manual. There is also a shortcut menu for 'How to use' the application.

Test Name: TFR06

Results: The user is able to confirm their schedule information prior to importing as the fetch, login and import steps are separate commands (buttons) and there are text boxes in the user interface to update the user on the current progress.

Test Name: TFR07

Results: The user is informed of processing private information. After a user clicks the 'Login' button, they are brought to a page in their browser that requests permission to access the user's personal information in their Mosaic (Open Timetable) and Google account (Open Calendar and Import).

Test Name: TFR08

Results: The user is able to exit the application by either clicking the 'X' in the top-right corner of window or by clicking 'Exit' under 'File' in the menu.

Test Name: TFR09

Results: The user is able to confirm they are ready to retrieve their Mosaic schedule prior to executing the function. The user is also notified that they can only fetch their Mosaic calendar once per session.

Test Name: TFR10

Results: The user is informed on the progress status of the import step. The application updates a text box in the user interface with a message indicating whether the import was successful or unsuccessful.

Test Name: TFR11

Results: The user is informed on the progress status of the login step. The application updates a text box in the user interface with a message indicating that the application has successfully accessed the user's Google account.

2 Nonfunctional Requirements Evaluation

2.1 Usability

2.1.1 GUI Testing

Description of Tests: Usability of the Graphical User Interface (GUI) was tested by a several McMaster students, most of whom are not in a technical-related field of study. This allows for the assumption that our intended users have basic knowledge of how to use computers. Participants were given the application and were told to follow the displayed instructions. No further help, hints, cues, etc were given. Afterwards, the participants filled out a feedback form that evaluated the system on its usability.

Test Name: general_ui
Results: The overall user interface is simple to use based on the majority of the feedback. It's intuitive with minimal buttons, in the English language and does not use colour to convey meaning. Users were able to operate the application on a desktop and laptop.

Test Name: app_exe
Results: Users were able to operate the application on a desktop and laptop.

Test Name: browse_button
Results: The Browse button successfully allows a user to select the file from their computer and update the text box field.

Test Name: fetch_button
Results: The Fetch Schedule button successfully prompts the user with a pop-up asking if they have selected their schedule. After the schedule has been parsed, the application properly leaves an error message for the user if they click the button again.

Test Name: fetch_popup
Results: If the user selected 'Yes', the application successfully parses the user's information and displays this information in the text box. If the user selects 'No', the application closes the pop-up and doesn't execute any functions.

Test Name: login_button
Results: The Login button successfully opens the Google login authorization in a user's browser. The text box updates with the correct status response.

Test Name: import_button
Results: The Import button successfully imports a user's calendar. The text box updates with the correct status response.

Test Name: exit_app
Results: The application gracefully closes with the 'Exit' button or the 'X' in the top-right corner.

Test Name: internal_menu
Results: The following internal menu options work as intended: Obtaining your schedule, Fetching your schedule, Logging into your Google account, Importing your schedule, Exit, and About... (which contains information about Gitlab and the developers).

Test Name: external_menu
Results: The following external menu options work as intended: Open Timetable, Open Calendar, and Full User Manual.

2.2 Performance

2.2.1 Installation

Description of Tests: The following test tests the performance of the user's ability to install the program.

Test Name: install_perf
Results: All participants were able to successfully complete the task of installing the program in under 10 minutes. This includes download and extraction time for slow internet connections.

2.2.2 Parsing

Description of Tests: The following tests test the performance of the parsing component of the application.

Test Name: parse_perf
Results: The application parses a user's schedule in under 0.5 seconds.

2.2.3 Google Connection

Description of Tests: The following tests test the performance of the Google API component of the application.

Test Name: login_perf
Results: A user is able to authenticate their account in under 3 steps. This

takes overall less than 10 seconds to do. The Chrome window opens in less than 0.5 seconds.

Test Name: import_perf
Results: The application takes up to 30 seconds depending on a user's connection speed. The user is notified of this through a text box once the user reaches this step.

3 Comparison to Existing Implementation

The existing implementation is a Google Chrome extension developed using JavaScript, HTML, CSS and the Google API (for web). The re-implementation is a Python desktop application developed using Python, Scrapy web crawler, A Tkinter wrapper library and the Google API (for Python). Both implementations parse an HTML document and import this information into a student's Google calendar. The existing implementation uses regular expressions to parse and select the HTML DOM elements. The re-implementation uses XPath selectors.

4 Unit Testing

4.1 GUI Testing

Test Name	test_convert_url
Initial State	-
Input	user_input
Expected Output	test_url
Actual Output	The requested action passed assertEqual test.

Table 2: Test for convert_url()

Test Name	test_set_list
Initial State	_fetched_list := [ ]
Input	fetch_list
Expected Output	-
Actual Output	The requested action passed assertEqual test.

Table 3: Test for set_list()

Test Name	test_print_sched
Initial State	-
Input	fetch_list
Expected Output	sched_str
Actual Output	The requested action passed assertEqual test.

Table 4: Test for print_sched()

Test Name	test_print_sched_err
Initial State	_google_conn := None
Input	Int
Expected Output	-
Actual Output	The requested action successfully raised a TypeError if an incorrect type is passed through.

Table 5: Test for print_sched_err()

Test Name	test_push_schedule_err
Initial State	_google_conn := None
Input	-
Expected Output	bool
Actual Output	The requested action successfully raised an AttributeError if called before a connection is created.

Table 6: Test for push_schedule_err()

Test Name	test_fetch
Initial State	_fetch_flg := False
Input	test_url
Expected Output	sched_str
Actual Output	The requested action passed assertEqual tests.

Table 7: Test for fetch()

Test Name	test_login_err
Initial State	_google_conn := None
Input	-
Expected Output	bool
Actual Output	The requested action successfully raised an AttributeError if called before a connection is created.

Table 8: Test for login_err()

Test Name	test_logout
Initial State	_google_conn := None
Input	-
Expected Output	False
Actual Output	The requested action passed assertEqual test.

Table 9: Test for logout()

4.2 Parse Testing

Test Name	test_parse_output
Initial State	ret := [ ]
Input	file_url
Expected Output	output_list
Actual Output	The requested action passed assertEqual test.

Table 10: Test for parse_output()

4.3 Connector Testing

The unit tests for the connector are to be done manually. The tester is to run testConnector.py. The tester proceeds through each case by pressing ”enter” to setup the test and is to follow the instructions printed on the console, listed under Input in the tables below.

Test Name	test_check_perms_1
Initial State	self.service = None. calendar.dat file does not exist.
Input	No particular input. User presses enter to start test.
Expected Output	False
Actual Output	Console printed False - Pass

Test Name	test_login_1
Initial State	self.service = None
Input	User presses ”allow” on the webpage that appears.
Expected Output	Browser displays: The authentication flow has completed. Console prints True. calendar.dat file exists.
Actual Output	Browser displayed the image. Console printed True - Pass

Test Name	test_check_perms_2
Initial State	self.service = Not None. calendar.dat file exists.
Input	No particular input. User presses enter to start test.
Expected Output	true
Actual Output	True - Pass

Test Name	test_create_cal_1
Initial State	-
Input	No particular input. User presses enter to start test.
Expected Output	Console prints True, a schedule named Mac Schedule appears in Google calendars. There should be no event in the schedule.
Actual Output	True and blank calendar appears. - Pass

Test Name	test_insert_events_1
Initial State	-
Input	No particular input. User presses enter to start test.
Expected Output	Console prints True, a schedule named Mac Schedule appears in Google calendars. There should be no event in the schedule.
Actual Output	Console prints True and blank calendar appears. - Pass

Test Name	test_get_num_events_1
Initial State	Calendar made with create_cal() during the session has been filled with events.
Input	No particular input. User presses enter to start test.
Expected Output	Console prints a number
Actual Output	Console prints 3 - Pass

Test Name	test_check_insertion_1
Initial State	Calendar made with create_cal() during the session has been filled with events using insert_events().
Input	No particular input. User presses enter to start test.
Expected Output	Console prints True
Actual Output	Console prints True - Pass

Test Name	test_remove_new_cal_1
Initial State	Calendar made with create_cal() exists and its corresponding identifier is in self.bodies of the connector object.
Input	No particular input. User presses enter to start test.
Expected Output	Console prints True. The created schedule has been removed from Google Calendars.
Actual Output	Console prints True and Mac Schedule is no longer on Google Calendars - Pass

Test Name	test_push_to_schedule_1
Initial State	-
Input	No particular input. User presses enter to start test.
Expected Output	New Mac Schedule is created and filled with events.
Actual Output	Schedule created with events. Events match the inputs given. Pass

Test Name	test_logout_1
Initial State	self.service = Not None. calendar.dat file exists.
Input	No particular input. User presses enter to start test.
Expected Output	True, self.service = None, calendar.dat no longer exists.
Actual Output	True - Pass

4.4 Converter Testing

4.4.1 Function Tests: offset_date

Test Name	offset_date_1
Initial State	-

Input	date = datetime.date(2018, 12, 18)
	Rfc.offset_date(date, [0])

Expected Output	[”2018”, ”12”, ”24”]
Actual Output	[”2018”, ”12”, ”24”] - Pass

Test Name	test_offset_date_2
Initial State	-

Input	date = datetime.date(2018, 12, 18)
	Rfc.offset_date(date, [2])

Expected Output	[”2018”, ”12”, ”19”]
Actual Output	[”2018”, ”12”, ”19”] - Pass

Test Name	test_offset_date_3
Initial State	-

Input	date = datetime.date(2018, 12, 18)
	Rfc.offset_date(date, [3, 0, 2])

Expected Output	[”2018”, ”12”, ”19”]
Actual Output	[”2018”, ”12”, ”19”] - Pass

Table 11: Tests for Rfc.offset_date()

4.4.2 Function Tests: extract_date

Test Name	extract_date_1
Initial State	-

Input	date_str = ”2018/09/04 - 2018/12/05”
	Rfc.extract_date(date_str, [0])

Expected Output	(’2018-9-10’, ’20181205’)
Actual Output	(’2018-9-10’, ’20181205’) - Pass

Test Name	extract_date_2
Initial State	-

Input	date_str = ”04/09/2018 - 05/12/2018”
	Rfc.extract_date(date_str, [0])

Expected Output	(’2018-9-10’, ’20181205’)
Actual Output	(’2018-9-10’, ’20181205’) - Pass

Table 12: Tests for Rfc.extract_date()

4.4.3 Function Tests: to_military

Test Name	to_military_1
Initial State	-
Input	Rfc.to_military(”2:00AM”)
Expected Output	”2:00”
Actual Output	”2:00” - Pass

Test Name	to_military_2
Initial State	-
Input	Rfc.to_military(”2:00PM”)
Expected Output	”14:00”
Actual Output	”14:00” - Pass

Test Name	to_military_3
Initial State	-
Input	Rfc.to_military(”12:00AM”)
Expected Output	”00:00”
Actual Output	”00:00” - Pass

Test Name	to_military_4
Initial State	-
Input	Rfc.to_military(”12:00PM”)
Expected Output	”12:00”
Actual Output	”12:00” - Pass

Table 13: Tests for Rfc.to_military()

4.4.4 Function Tests: extract_weekdays

Test Name	extract_weekdays_1
Initial State	-
Input	Rfc.extract_weekdays(”MoWeFr”)
Expected Output	(”MO,WE,FR”, [0, 2, 4])
Actual Output	(”MO,WE,FR”, [0, 2, 4]) - Pass

Test Name	extract_weekdays_2
Initial State	-
Input	Rfc.extract_weekdays(”Tu”)
Expected Output	(”TU”, [1])
Actual Output	(”TU”, [1]) - Pass

Test Name	extract_weekdays_3
Initial State	-
Input	Rfc.extract_weekdays(”Th”)
Expected Output	(”TH”, [3])
Actual Output	(”TH”, [3]) - Pass

Test Name	extract_weekdays_4
Initial State	-
Input	Rfc.extract_weekdays("ThMo")
Expected Output	("MO,TH", [0, 3])
Actual Output	("MO,TH", [0, 3]) - Pass

Table 14: Tests for Rfc.extract_weekdays()

4.4.5 Function Tests: rfc_output

Test Name	rfc_output_1
Initial State	-
Input	Rfc.rfc_output("2019/01/07 - 2019/04/09", "We 2:30 - 3:20PM")
Expected Output	('2019-1-9T2:30:00', '2019-1-9T15:20:00', 'RRULE:FREQ=WEEKLY;UNTIL=20190409T045959Z;BYDAY=WE')
Actual Output	('2019-1-9T2:30:00', '2019-1-9T15:20:00', 'RRULE:FREQ=WEEKLY;UNTIL=20190409T045959Z;BYDAY=WE') - Pass

Table 15: Tests for Rfc.rfc_output()

4.4.6 Function Tests: convert

The inputs and outputs for this section are defined variables in the setUp function.

Test Name	convert_1
Initial State	-
Input	Converter.convert(self.input_1)
Expected Output	self.output_1
Actual Output	self.output_1 - Pass

Test Name	convert_2
Initial State	-
Input	Converter.convert(self.input_2)
Expected Output	self.output_2
Actual Output	self.output_2 - Pass

Test Name	convert_3
Initial State	-
Input	Converter.convert(self.input_3)
Expected Output	self.output_3
Actual Output	self.output_3 - Pass

Test Name	convert_4
Initial State	-
Input	Converter.convert(self.input_4)
Expected Output	self.output_4
Actual Output	self.output_4 - Pass

Table 16: Test for Converter.convert()

4.5 Setup Testing

Test Name	CPE-01
Initial State	Application is installed and ready to be opened
Input	Mouse click on application
Expected Output	The application opens to the main display
Actual Output	Application opened - Pass

Table 17: Test for CPE-01

Test Name	CPE-02
Initial State	Application is opened
Input	Mouse click on browse button
Expected Output	File explorer window is displayed
Actual Output	File explorer window opened - Pass

Table 18: Test for CPE-02

Test Name	CPE-03
Initial State	Application is opened and html file selected
Input	Mouse click on fetch button
Expected Output	The schedule information will be displayed in the text box in the application.
Actual Output	Schedule information displayed in text box - Pass

Table 19: Test for CPE-03

Test Name	CPE-04
Initial State	Application is opened
Input	Mouse click on login button
Expected Output	A browser will be opened to Googles sign in page where the user can enter their credentials and give permission for the application to access their account.
Actual Output	Browser opened to Google sign in webpage - Pass

Table 20: Test for CPE-04

Test Name	CPE-05
Initial State	Application is opened
Input	Mouse click on import button
Expected Output	Import successful will be displayed and the schedule is imported to the users Google Calendar.
Actual Output	Google calendar created and import successful displayed - Pass

Table 21: Test for CPE-05

Test Name	CPE-06
Initial State	Application is opened
Input	Mouse click on the help option
Expected Output	A list of the user manual and the additional information option is displayed.
Actual Output	User manual and additional information list displayed - Pass

Table 22: Test for CPE-06

Test Name	CPE-07
Initial State	Application is opened
Input	Exit button is pressed
Expected Output	Application is closed and the user is logged out of their Google account.
Actual Output	Application closed and user signed out of Google - Pass

Table 23: Test for CPE-07

5 Changes Due to Testing

5.1 GUI Testing

After conducting usability test on the GUI, it was decided that some methods were to become more modular. Click events now only call a method for their intended function.

5.2 Parse Testing

There have been no changes to the methods of testing as a result of completed tests.

5.3 Connector Testing

There have been no changes to the methods of testing as a result of completed tests.

5.4 Converter Testing

There have been no changes to the methods of testing as a result of completed tests.

5.5 Setup Testing

There have been no changes to the methods of testing as a result of completed tests.

6 Automated Testing

6.1 GUI Testing

Description of tests: testGUI.py was the test suite used to test guiClient.py. Majority of the functions needed to be tested manually except for functions that referenced other modules.

6.2 Parse Testing

Description of tests: testParse.py was the test suite used to test parseMosaic.py. For automated testing of the output, one unit test was used.

6.3 Connector Testing

This module must be tested manually. However the compilation of unit tests for the tester is located in testConnector.py.

6.4 Converter Testing

Description of tests: testConverter.py was the test suit used to test converter.py. Each module was tested with at minimum, a case for each input domain.

7 Trace to Requirements

Test	Requirements
Functional Requirements Testing	
TFR01	FR01
TFR02	FR02, FR06, FR08
TFR03	FR03
TFR04	FR04
TFR05	FR05
TFR06	FR07
TFR07	FR09
TFR08	FR10
TFR09	FR11
TFR10	FR12
TFR11	FR13

Test	Requirements
Non-functional Requirements Testing	
general_ui	NF01, NF02, NF03, NF05, NF06
app_exe	NF08
browse_button	NF14
fetch_popup	NF07
login_button	NF07
import_button	NF07
exit_app	-
internal_menu	NF07
external_menu	NF07, NF10, NF12, NF13
install_perf	NF04
parse_perf	NF09
login_perf	NF09
import_perf	-
Automated Testing	
test_convert_url	NF14
test_set_list	FR02, FR06, FR08
test_print_sched	FR02, FR06, FR08
test_print_sched_err	FR02, FR06, FR08
test_push_schedule_err	FR07, NF07
test_fetch	FR02, FR06, FR08
test_login_err	FR09
test_logout	NF14
test_parse_output	FR02, FR06, FR08

Table 24: Trace Between Tests and Requirements

8 Trace to Modules

Test	Modules
Functional Requirements Testing	
TFR01	M2, M5
TFR02	M5, M5,
TFR03	M4, M5
TFR04	M5
TFR05	M5
TFR06	M5
TFR07	M4, M5
TFR08	M5
TFR09	M6
TFR10	M6
TFR11	M6
Non-functional Requirements Testing	
general_ui	M5
app_exe	M5
browse_button	M2, M3, M5
fetch_popup	M5
login_button	M5
import_button	M5
exit_app	-
internal_menu	M5
external_menu	M5
install_perf	M5
parse_perf	M2, M3, M4, M5
login_perf	M2, M3, M4, M5
import_perf	-

Table 25: Trace Between Tests and Modules

Test	Modules
Automated Testing	
test_convert_url	M2, M3, M5
test_set_list	M5
test_print_sched	M5
test_print_sched_err	M5
test_push_schedule_err	M5
test_fetch	M5
test_login_err	M4, M5
test_logout	M2, M3, M5
test_parse_output	M5

Table 26: Trace Between Tests and Modules

9 Code Coverage Metrics

The 0C team has managed to produce approximately 98 percent code coverage through our tests. This number is based off the fact that all of the modules have been covered in testing. Please refer to the trace to modules section to see how all of our modules have been covered. See the tables below for details on each test suite.

Name	**Stmts**	**Miss**	**Cover**
converter.py	92	2	98%
testConverter.py	56	0	100%
TOTAL	148	2	99%

Table 27: Code coverage for testConverter.py

Name	**Stmts**	**Miss**	**Cover**
parseMosaic.py	50	0	100%
testParse.py	18	1	94%
TOTAL	68	1	99%

Table 28: Code coverage for testParse.py

Name	**Stmts**	**Miss**	**Cover**
connector.py	88	71	19%
converter.py	92	83	10%
guiClient.py	114	67	41%
parseMosaic.py	50	0	100%
testGUI.py	42	1	98%
TOTAL	148	2	99%

Table 29: Code coverage for testGUI.py