Software Testing Report

<Project Name>

Student Names

Table of Contents

[1.0 Unit Tests 3](#_Toc49779837)

[2.0 Coverage Report 4](#_Toc49779838)

[3.0 Requirements Acceptance Testing 5](#_Toc49779839)

# Unit Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| **1** | **Reporting the information of all penalty cases function – show\_View\_Case\_Penalty()** | | |
| 1.1 | Load data from a valid file path | Data is successfully loaded. | Data is successfully loaded |
| 1.2 | Load data from an invalid file path. | Error message indicating invalid file path. | Error message indicating invalid file path. |
| 1.3 | Display penalty case information with a valid date range. | The grid or display panel should show all penalty cases for that period. | List of penalty cases within the selected period is returned. |
| **2** | **Producing a chart to show the distribution of cases in each offense code function – generate\_trend()** | | |
| 2.1 | Load data from a valid file path | Data is successfully loaded. | Data is successfully loaded |
| 2.2 | Load data from an invalid file path. | Error message indicating invalid file path. | Error message indicating invalid file path. |
| 2.3 | Generate distribution chart for a valid date range. | A chart should be displayed showing the distribution of cases in each offense code. |  |
| **3** | **Retrieving all cases captured by radar or camera based on offense description function – show\_radar\_camera\_cases()** | | |
| 3.1 | Load data from a valid file path | Data is successfully loaded. | Data is successfully loaded |
| 3.2 | Load data from an invalid file path. | Error message indicating invalid file path. | Error message indicating invalid file path. |
| 3.3 | Retrieve radar camera cases for a valid date range. | The grid or display panel should show all cases captured by radar or camera for that period. | List of cases captured by radar or camera within the selected period is returned. |
| **4** | **Analysing the cases caused by mobile phone usage function – generate\_mobile\_phone\_trend()** | | |
| 4.1 | Load data from a valid file path | Data is successfully loaded. | Data is successfully loaded |
| 4.2 | Load data from an invalid file path. | Error message indicating invalid file path. | Error message indicating invalid file path. |
| 4.3 | Analyse trend of mobile phone usage cases for a valid date range | Trend of mobile phone usage cases within the selected period is displayed. | Trend of mobile phone usage cases within the selected period are displayed. |
| **5** | **Analysing the cases caused by misuse seatbelt function – generate\_seatbelt\_trend()** | | |
| 5.1 | Load data from a valid file path | Data is successfully loaded. | Data is successfully loaded |
| 5.2 | Load data from an invalid file path. | Error message indicating invalid file path. | Error message indicating invalid file path. |
| 5.3 | Analyse trend of misuse seatbelt for a valid date range. | Trend of misuse seatbelt cases within the selected period is displayed. | Trend of misuse seatbelt cases within the selected period is displayed. |
| **6** | **Navigating function – switch\_to\_page()** | | |
| 6.1 | Navigate through the application. | The specified page or section is displayed correctly | The specified page or section is displayed correctly |

|  |  |  |
| --- | --- | --- |
| **No** | **Test Case** | **Unit Test Code** |
| **1** | **Reporting the information of all penalty cases function – show\_View\_Case\_Penalty()** | |
| 1.1 | Load data from a valid file path – test\_file\_load.py | import csv def load\_data\_from\_csv(file\_path: str) -> list:  with open(file\_path, 'r') as file:  reader = csv.reader(file)  data = list(reader)  return data  def test\_load\_data\_from\_csv():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert data is not None # Ensure data is loaded  assert isinstance(data, list) # Ensure data is a list of rows from cvs  assert len(data) > 0 # Ensure data is not empty  def test\_load\_data\_from\_csv\_failed():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert len(data) == 0, f"Expected an empty list, but got a list of length {len(data)}."  A screenshot of a computer program  Description automatically generated |
| 1.2 | Load data from an invalid file path – test\_invalid\_file\_load.py | import wx import csv from unittest.mock import patch def load\_data\_from\_file(file\_path: str) -> list:  try:  with open(file\_path, "r") as file:  reader = csv.reader(file)  data = list(reader)  return data  except FileNotFoundError:  wx.MessageBox(f"The file '{file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  return [] def test\_load\_data\_invalid\_csv():  invalid\_file\_path = "/invalid/path.csv" # Example of an invalid file path  with patch('wx.MessageBox') as mocked\_msgbox:  data = load\_data\_from\_file(invalid\_file\_path)  # Check if wx.MessageBox was called with the expected arguments  mocked\_msgbox.assert\_called\_once\_with(  f"The file '{invalid\_file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  # Ensure data is an empty list when given an invalid path  assert data == []  A screenshot of a computer  Description automatically generated |
| 1.3 | View penalty case information with a valid date range – test\_display\_penalty\_case.py | import csv import wx def load\_penalty\_data(file\_path: str) -> list:  try:  with open(file\_path, "r") as file:  reader = csv.reader(file)  next(reader)  months\_years = sorted(list(set(f"{date.split('/')[1]}/{date.split('/')[2]}" for date in [row[1] for row in reader] if len(date.split('/')) == 3)), key=lambda x: (int(x.split('/')[1]), int(x.split('/')[0])))  return months\_years  except FileNotFoundError:  wx.MessageBox(f"The file '{file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  return []  except Exception as e:  wx.MessageBox(f"An error occurred: {e}", "Error", wx.OK | wx.ICON\_ERROR)  return []  def test\_load\_penalty\_data():  file\_path = "penalty\_data\_set\_2.csv"  data = load\_penalty\_data(file\_path)  assert data is not None, "Data is None"  assert isinstance(data, list), "Data is not a list"  assert len(data) > 0, "Data list is empty"   # Check if the dates are sorted correctly  sorted\_data = sorted(data, key=lambda x: (int(x.split('/')[1]), int(x.split('/')[0])))  assert data == sorted\_data, "Data is not sorted correctly" |
| **2** | **Producing a chart to show the distribution of cases in each offense code function – generate\_trend()** | |
| 2.1 | Load data from a valid file path – test\_file\_load.py | import csv def load\_data\_from\_csv(file\_path: str) -> list:  with open(file\_path, 'r') as file:  reader = csv.reader(file)  data = list(reader)  return data  def test\_load\_data\_from\_csv():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert data is not None # Ensure data is loaded  assert isinstance(data, list) # Ensure data is a list of rows from cvs  assert len(data) > 0 # Ensure data is not empty  def test\_load\_data\_from\_csv\_failed():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert len(data) == 0, f"Expected an empty list, but got a list of length {len(data)}."  A screenshot of a computer program  Description automatically generated |
| 2.2 | Load data from an invalid file path – test\_invalid\_file\_load.py | import wx import csv from unittest.mock import patch def load\_data\_from\_file(file\_path: str) -> list:  try:  with open(file\_path, "r") as file:  reader = csv.reader(file)  data = list(reader)  return data  except FileNotFoundError:  wx.MessageBox(f"The file '{file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  return [] def test\_load\_data\_invalid\_csv():  invalid\_file\_path = "/invalid/path.csv" # Example of an invalid file path  with patch('wx.MessageBox') as mocked\_msgbox:  data = load\_data\_from\_file(invalid\_file\_path)  # Check if wx.MessageBox was called with the expected arguments  mocked\_msgbox.assert\_called\_once\_with(  f"The file '{invalid\_file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  # Ensure data is an empty list when given an invalid path  assert data == []  A screenshot of a computer  Description automatically generated |
| 2.3 | Generate distribution chart for a valid date range – test\_chart\_generating.py | import csv def get\_trend\_data(offence\_code: str, start\_month\_year: str, end\_month\_year: str) -> (list, list):  with open("penalty\_data\_set\_2.csv", "r") as file:  reader = csv.reader(file)  headers = next(reader)  start\_month\_year\_num = int(start\_month\_year.split('/')[1] + start\_month\_year.split('/')[0])  end\_month\_year\_num = int(end\_month\_year.split('/')[1] + end\_month\_year.split('/')[0])  filtered\_data = [row for row in reader if row[2] == offence\_code and start\_month\_year\_num <= int(  row[1].split('/')[-1] + row[1].split('/')[1]) <= end\_month\_year\_num]   dates = [row[1] for row in filtered\_data]  values = [int(row[24]) for row in filtered\_data] # Assuming 24th column has the trend values   # Sum values by date (month/year)  datewise\_values = {}  for date, value in zip(dates, values):  if date in datewise\_values:  datewise\_values[date] += value  else:  datewise\_values[date] = value   sorted\_dates = sorted(datewise\_values.keys(), key=lambda x: (int(x.split('/')[-1]), int(x.split('/')[1])))  sorted\_values = [datewise\_values[date] for date in sorted\_dates]   return sorted\_dates, sorted\_values   def test\_get\_trend\_data():  offence\_code = "74703" # Replace with an actual valid offense code  start\_month\_year = "01/2012"  end\_month\_year = "12/2017"   dates, values = get\_trend\_data(offence\_code, start\_month\_year, end\_month\_year)   # Check if dates and values are non-empty lists  assert dates, "Dates list is empty"  assert values, "Values list is empty"   # Ensure dates and values have the same length  assert len(dates) == len(values), "Length of dates and values lists differ"   # Ensure dates are sorted  assert dates == sorted(dates, key=lambda x: (  int(x.split('/')[-1]), int(x.split('/')[1]))), "Dates are not sorted correctly"   def test\_get\_trend\_data\_failure():  # Intentionally using an offense code that we know has data  offence\_code = "74703" # Replace with an actual valid offense code  start\_month\_year = "01/2012"  end\_month\_year = "12/2017"   dates, values = get\_trend\_data(offence\_code, start\_month\_year, end\_month\_year)   # Intentionally asserting the opposite of our expectations to force a failure  assert not dates, "Expected no dates, but got some."  assert not values, "Expected no values, but got some." |
| **3** | **Retrieving all cases captured by radar or camera based on offense description function – show\_radar\_camera\_cases()** | |
| 3.1 | Load data from a valid file path – test\_file\_load.py | import csv def load\_data\_from\_csv(file\_path: str) -> list:  with open(file\_path, 'r') as file:  reader = csv.reader(file)  data = list(reader)  return data  def test\_load\_data\_from\_csv():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert data is not None # Ensure data is loaded  assert isinstance(data, list) # Ensure data is a list of rows from cvs  assert len(data) > 0 # Ensure data is not empty  def test\_load\_data\_from\_csv\_failed():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert len(data) == 0, f"Expected an empty list, but got a list of length {len(data)}."  A screenshot of a computer program  Description automatically generated |
| 3.2 | Load data from an invalid file path – test\_invalid\_file\_load.py | import wx import csv from unittest.mock import patch def load\_data\_from\_file(file\_path: str) -> list:  try:  with open(file\_path, "r") as file:  reader = csv.reader(file)  data = list(reader)  return data  except FileNotFoundError:  wx.MessageBox(f"The file '{file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  return [] def test\_load\_data\_invalid\_csv():  invalid\_file\_path = "/invalid/path.csv" # Example of an invalid file path  with patch('wx.MessageBox') as mocked\_msgbox:  data = load\_data\_from\_file(invalid\_file\_path)  # Check if wx.MessageBox was called with the expected arguments  mocked\_msgbox.assert\_called\_once\_with(  f"The file '{invalid\_file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  # Ensure data is an empty list when given an invalid path  assert data == []  A screenshot of a computer  Description automatically generated |
| 3.3 | Retrieve radar camera cases for a valid date range. | import datetime import csv  def get\_radar\_camera\_cases(start\_month\_year: str, end\_month\_year: str) -> list:  start\_month, start\_year = start\_month\_year.split('/')  end\_month, end\_year = end\_month\_year.split('/')   start\_date = datetime.date(int(start\_year), int(start\_month), 1)  end\_date = datetime.date(int(end\_year), int(end\_month), 1)   with open("penalty\_data\_set\_2.csv", "r") as file:  reader = csv.reader(file)  headers = next(reader)  filtered\_data = [row for row in reader if  ("Camera" in row[3] or "Radar" in row[3]) and start\_date <= datetime.date(  int(row[1].split('/')[2]), int(row[1].split('/')[1]),  int(row[1].split('/')[0])) <= end\_date]   # Sort the filtered data by date (oldest to latest)  filtered\_data.sort(  key=lambda x: datetime.date(int(x[1].split('/')[2]), int(x[1].split('/')[1]), int(x[1].split('/')[0])))   return filtered\_data   def test\_get\_radar\_camera\_cases():  start\_month\_year = "01/2012"  end\_month\_year = "12/2017"   start\_month, start\_year = start\_month\_year.split('/')  end\_month, end\_year = end\_month\_year.split('/')   start\_date = datetime.date(int(start\_year), int(start\_month), 1)  end\_date = datetime.date(int(end\_year), int(end\_month), 1)   data = get\_radar\_camera\_cases(start\_month\_year, end\_month\_year)   # Ensure that data is non-empty  assert data, "Data list is empty"   # Ensure that each row in data contains either "Camera" or "Radar" in the offense description (assuming it's in column index 2)  for row in data:  assert "Camera" in row[3] or "Radar" in row[3], f"Unexpected offense description in row: {row}"   # Ensure that each row's date falls within the specified range  for row in data:  date = datetime.date(int(row[1].split('/')[2]), int(row[1].split('/')[1]), int(row[1].split('/')[0]))  assert start\_date <= date <= end\_date, f"Date out of range in row: {row}" |
| **4** | **Analysing the cases caused by mobile phone usage function – generate\_mobile\_phone\_trend()** | |
| 4.1 | Load data from a valid file path – test\_file\_load.py | import csv def load\_data\_from\_csv(file\_path: str) -> list:  with open(file\_path, 'r') as file:  reader = csv.reader(file)  data = list(reader)  return data  def test\_load\_data\_from\_csv():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert data is not None # Ensure data is loaded  assert isinstance(data, list) # Ensure data is a list of rows from cvs  assert len(data) > 0 # Ensure data is not empty  def test\_load\_data\_from\_csv\_failed():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert len(data) == 0, f"Expected an empty list, but got a list of length {len(data)}."  A screenshot of a computer program  Description automatically generated |
| 4.2 | Load data from an invalid file path – test\_invalid\_file\_load.py | import wx import csv from unittest.mock import patch def load\_data\_from\_file(file\_path: str) -> list:  try:  with open(file\_path, "r") as file:  reader = csv.reader(file)  data = list(reader)  return data  except FileNotFoundError:  wx.MessageBox(f"The file '{file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  return [] def test\_load\_data\_invalid\_csv():  invalid\_file\_path = "/invalid/path.csv" # Example of an invalid file path  with patch('wx.MessageBox') as mocked\_msgbox:  data = load\_data\_from\_file(invalid\_file\_path)  # Check if wx.MessageBox was called with the expected arguments  mocked\_msgbox.assert\_called\_once\_with(  f"The file '{invalid\_file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  # Ensure data is an empty list when given an invalid path  assert data == []  A screenshot of a computer  Description automatically generated |
| 4.3 | Analyse trend of mobile phone usage cases for a valid date range |  |
| **5** | **Analysing the cases caused by misuse seatbelt function – generate\_seatbelt\_trend()** | |
| 5.1 | Load data from a valid file path – test\_file\_load.py | import csv def load\_data\_from\_csv(file\_path: str) -> list:  with open(file\_path, 'r') as file:  reader = csv.reader(file)  data = list(reader)  return data  def test\_load\_data\_from\_csv():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert data is not None # Ensure data is loaded  assert isinstance(data, list) # Ensure data is a list of rows from cvs  assert len(data) > 0 # Ensure data is not empty  def test\_load\_data\_from\_csv\_failed():  file\_path = "penalty\_data\_set\_2.csv" # Example valid file path  data = load\_data\_from\_csv(file\_path)  assert len(data) == 0, f"Expected an empty list, but got a list of length {len(data)}."  A screenshot of a computer program  Description automatically generated |
| 5.2 | Load data from an invalid file path – test\_invalid\_file\_load.py | import wx import csv from unittest.mock import patch def load\_data\_from\_file(file\_path: str) -> list:  try:  with open(file\_path, "r") as file:  reader = csv.reader(file)  data = list(reader)  return data  except FileNotFoundError:  wx.MessageBox(f"The file '{file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  return [] def test\_load\_data\_invalid\_csv():  invalid\_file\_path = "/invalid/path.csv" # Example of an invalid file path  with patch('wx.MessageBox') as mocked\_msgbox:  data = load\_data\_from\_file(invalid\_file\_path)  # Check if wx.MessageBox was called with the expected arguments  mocked\_msgbox.assert\_called\_once\_with(  f"The file '{invalid\_file\_path}' was not found.", "Error", wx.OK | wx.ICON\_ERROR)  # Ensure data is an empty list when given an invalid path  assert data == []  A screenshot of a computer  Description automatically generated |
| 5.3 | Analyse trend of misuse seatbelt for a valid date range. | import csv import datetime   # Replicating the provided generate\_trend\_for\_offense function def generate\_trend\_for\_offense(file\_path, offense\_description, start\_month\_year, end\_month\_year):  # Convert MM/YYYY to YYYYMM for comparison  start\_month\_year\_num = int(start\_month\_year.split('/')[1] + start\_month\_year.split('/')[0])  end\_month\_year\_num = int(end\_month\_year.split('/')[1] + end\_month\_year.split('/')[0])   # Read data and filter based on the offense description and selected date range  with open(file\_path, "r") as file:  reader = csv.reader(file)  headers = next(reader)  filtered\_data = [row for row in reader if offense\_description in row[3].lower() and start\_month\_year\_num <= int(  row[1].split('/')[-1] + row[1].split('/')[1]) <= end\_month\_year\_num]   # Accumulate (sum) the TOTAL\_VALUE for each unique date  datewise\_values = {}  for row in filtered\_data:  date = row[1]  value = int(row[24]) # Assuming 24th column has the trend values  if date in datewise\_values:  datewise\_values[date] += value  else:  datewise\_values[date] = value   # Sorting dates (month/year) for return  sorted\_dates = sorted(datewise\_values.keys(), key=lambda x: (datetime.datetime.strptime(x, "%d/%m/%Y").year,  datetime.datetime.strptime(x,  "%d/%m/%Y").month)) # Sorting by year first, then month  sorted\_values = [datewise\_values[date] for date in sorted\_dates]   return sorted\_dates, sorted\_values   # Test case def test\_generate\_mobile\_phone\_trend():  file\_path = "penalty\_data\_set\_2.csv" # Path to the sample data file  start\_month\_year = "01/2013"  end\_month\_year = "02/2015"   sorted\_dates, sorted\_values = generate\_trend\_for\_offense(file\_path, "seatbelt", start\_month\_year,  end\_month\_year)   # Assert that the function processes the data correctly  assert sorted\_dates and sorted\_values, "Failed to generate the trend" |
| **6** | **Navigating function – switch\_to\_page()** | |
| 6.1 | Navigate through the application. | import wx from homepage import MainPage  def test\_switch\_to\_page():  app = wx.App(False) # Create a new app, don't redirect stdout/stderr to a window  frame = MainPage(None) # Create a frame   # Test navigation to "View\_Case\_Penalty"  frame.switch\_to\_page("View\_Case\_Penalty")  assert hasattr(frame, 'active\_panel'), "Failed to navigate to View\_Case\_Penalty"   # Test navigation to "Offence\_code"  frame.switch\_to\_page("Offence\_code")  assert hasattr(frame, 'active\_panel'), "Failed to navigate to Offence\_code"   # Test navigation to "Radar/Camera Cases"  frame.switch\_to\_page("Radar/Camera Cases")  assert hasattr(frame, 'active\_panel'), "Failed to navigate to Radar/Camera Cases"   # Test navigation to "Mobile Phone Usage"  frame.switch\_to\_page("Mobile Phone Usage")  assert hasattr(frame, 'active\_panel'), "Failed to navigate to Mobile Phone Usage"   # Test navigation to "Seatbelt not fastened"  frame.switch\_to\_page("Seatbelt not fastened")  assert hasattr(frame, 'active\_panel'), "Failed to navigate to Seatbelt not fastened"   frame.Destroy() # Cleanup |

# Coverage Report

A description of the coverage of your unit tests, including how you evaluated coverage (function, statement, branch, condition)

*A screenshot of a computer

Description automatically generated*

Figure 1: Coverage Report

**Coverage Report Summary**

The coverage report for the unit tests shows how much of the code was actually run during testing. A detailed image of the coverage metrics for each module is presented in the report, and a 95% total coverage percentage is stated. After further analysis of the metrics, it indicates that:

**Function Coverage**: This measure assesses the execution of each function or method in the codebase. The report does not provide explicit function coverage metrics. However, by correlating with statement coverage, we can infer that most functions were likely called, given the high statement coverage. Still, some functions or parts thereof in modules like test\_display\_penalty\_case.py (74% coverage) might not have been completely invoked.

**Statement Coverage**: This is the report's main performance indicator. In respect to the total number of statements in the codebase, it counts the executed statements. Files like homepage.py and test\_chart\_generating.py have coverage of 95% and 97%, respectively, indicating a majority of their code was executed. In contrast, test\_display\_penalty\_case.py has a lower coverage at 74%, suggesting certain statements were not executed.

**Branch Coverage**: This metric evaluates if each possible branch from each decision point (like if, else statements) has been executed. Given the provided report and the individual file reports, it's evident that many branches have been tested, especially in files with 100% coverage. However, files with less than 100% coverage indicate potential missed branches, which should be inspected further.

**Condition Coverage**: This involves examining the Boolean sub-expressions of a decision point, ensuring they evaluate both to true and false. Although the provided report doesn't offer explicit condition coverage, the statement and branch coverage metrics can provide indirect insights. For complete condition coverage, each condition should be tested in all possible true and false scenarios. Modules with less than 100% coverage might have conditions that were not fully tested.

# Requirements Acceptance Testing

(You will need to fill out the column on the left with the requirements listed in software design documents and the columns on the right with the results of your own testing)

| **Software  Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| --- | --- | --- | --- | --- |
| 1 | Accept multiple file names as arguments from the command line |  |  |  |
| 2 | Display the details of all valid files |  |  |  |
| 3 | Display an appropriate message if a file does not exist or if a file name is invalid |  |  |  |
| 4 | Display a message if an argument is a directory instead of a file |  |  |  |
| 5 | File name can be a simple file name or include the full path of the file with one or more levels |  |  |  |
| 6 | file names must start with an alphabetical character |  |  |  |
| 7 | Valid file name extensions must be 3 or 4 alphabetical characters preceded by a dot) |  |  |  |
| 8 | Directory/level names must start with an alphabetical character to be considered valid |  |  |  |
| 9 | The program should be able to accept as many levels for each file name as the user wants to input. This is limited only by the number of levels allowed in Windows (approximately 120) |  |  |  |