EECS 4313 - Assignment 1

Bugs and Bug Reports

Ameer Bacchus	215572845
Chandler Cabrera	214407571
Anas Ishtiaque	215571342
Seran Sathivel	216986309

Lassonde, York University

EECS 4313 - Software Engineering Testing

Dr. Zhen Ming Jiang

October 12th, 2021

Task 1 - Reporting Bugs or Proposing Enhancements

Bug 1

Bug Report Title	Holiday Search Issue
Reported by	Ameer Bacchus
Date reported	2021-10-11
Program/Component of concern	Search Tool
Configuration	Operating System: Windows 10 Build #:1.9.04
Report Type	Coding Error
Severity	Minor
Problem Summary	Unable to search holidays using the search tool
Reproducibility	Yes
New or old bug	New

Problem Description

When trying to search holidays using the search tool, an error message is shown indicating no holidays were found. To reproduce:

1. Click on the search icon (binoculars). Make sure “Holiday” is the only filter selected.
2. Enter a holiday (ex. Halloween, Christmas, etc).
3. Click the Search button.

The expected outcome would be the date of the holiday in question; instead, it displays an error saying “The search returned no results”. Additionally, this also does not work when trying to create and search a new holiday. To reproduce:

1. Double click on any day in the calendar to open the Appointment editor.
2. Create a test appointment and make sure the Holiday field under “Properties” is selected.
3. Hit “Save & Close” at the bottom of the window.
4. Following the same steps above, try searching for this newly created holiday and the same error message should be displayed.

Bug 2

Bug Report Title	Recurrence Frequency Limitation Error
Reported by	Seran Sathivel
Date reported	Oct 12, 2021
Program/Component of concern	Booking recurring appointments
Configuration	Operating System: Windows 8 Build #: 1.9.04
Report Type	Coding Error
Severity	Minor
Problem Summary	Recurrence frequency fails to display all recurring appointments for the specified period of time.
Reproducibility	Yes
New or old bug	New

Problem Description

When booking a recurring appointment for an extended period of time (a couple of years), it does not book the appointment for the entire specified length.

To reproduce this error:

1. Start by adding a new appointment on a date in the year of 2021 (right click on a date in Month View, Week View or Day View and press the 'Add New' option).
2. Give the New APPT any title and select any time.
3. In the Recurrence section in the bottom left, select the Frequency to be 'daily'.
4. Select the 'Until' option. In the 'Until' option, input a date that is at least 3 years greater (past 2024) than the date of the appointment that is being created.
5. Click 'Save & Close'.

Now you will notice the recurring appointment in the calendar. However, if you go to any date starting in 2024, you will notice recurring appointments are not scheduled.

Feature Enhancement

Feature Enhancement Title	Quick Multi-Day Event Selection
Reported by	Anas Ishtiaque
Date reported	October 15, 2021
Program/Component of Concern	Quick Event Selection
Configuration	Operating System: Windows 11 Build #: 1.9.04
Report Type	Feature Enhancement
Feature Enhancement Summary	Quick multi-day event selection will allow the user to drag and be able to select multiple dates and schedule all of those days.

Feature Enhancement Description

Currently, a user can only operate on one day at a time (e.g. a user has to click and select one day, then select a frame of time for a specific type of property). This feature enhancement suggests being able to quickly select multiple days - like through click-and-drag - and add an event for all of them. This would make creating shorter, multi-day events more intuitive, and requires less steps than setting up a recurring event using the current method.

Creating a recurring event requires at least 4 clicks:

1. Click to open the frequency drop-down
2. Select the frequency
3. Select the end parameter (Times or Until)
4. Set the parameter

By using a click-and-drag method, a single click-and-drag can determine many factors at once:

- The start date
- The in-between dates
- The end date

Task 2 - Existing Bug Reports in Open Source Systems

Bug Reports in Mozilla Firefox

This bug report has a small number of concise steps to reproduce it. Its tone is also matter-of-fact, showing no malicious intent towards the developers.

However, there are some areas of improvement. Both the title and its description have a typo ('reciepient' instead of 'recipient'). The description could also include more details, like a comparison between the intended result and the actual result: "instead of a list of multiple recipients, the first recipient is listed multiple times". There is also no indication of build, configuration, environment, or platform.

Issue Reports in HBase

To get insight on the health of the project, one could analyze the bug reports generated to get an indication of how the project is progressing. One indicator to look out for could be the ratio of open bug reports to closed ones - for example, a high percentage of closed bug reports would be indicative of an efficient team.

To analyze the reports, I wrote three python scripts. Each one parsed through each file in the folder, looking for different xml tags to filter results. To find the proportion of reports that were bugs, I indexed through each report looking for the tag 'type', and getting its attribute. I tabulated the number of each type in a python dictionary. Among the database of 17257 reports, there were 8614 bugs, accounting for approximately half of the reports. The results can be seen in *Figure 1* with the implementation details in *Appendix 1*.

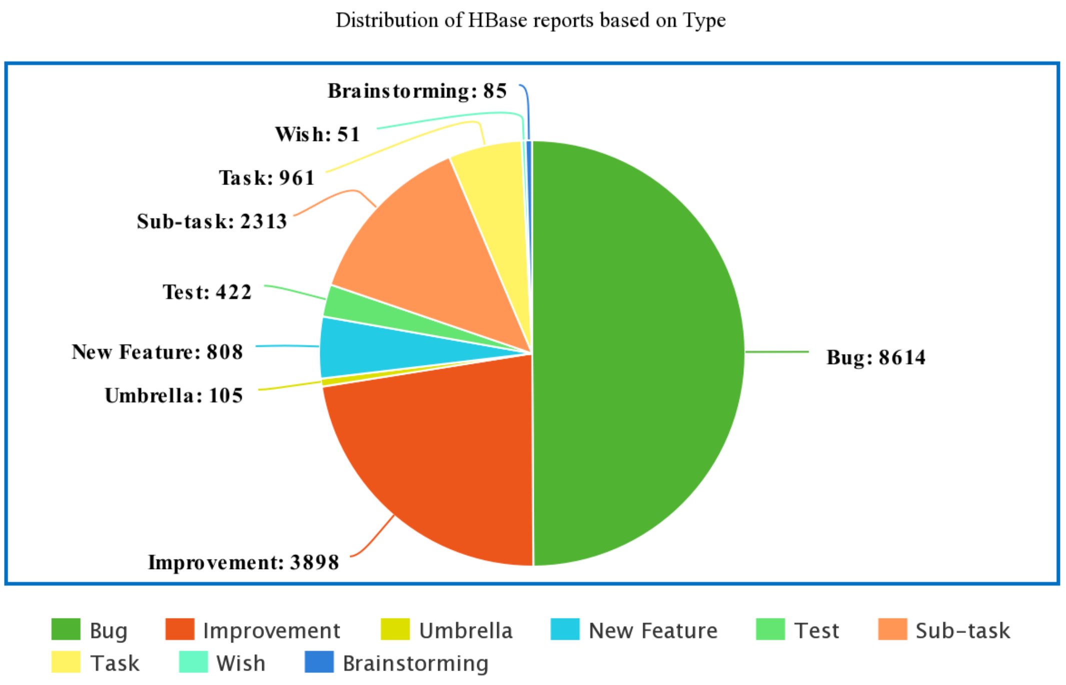

Figure 1: Distribution of HBase reports based on Type.

Analyzing the set of reports that were bugs, I used a similar method to quantify the distribution of each report's current state. By filtering results using the report's 'status' tag, I found that there were 6 different statuses possible for bug reports: Open, In Progress, Closed, Resolved, Patch Available, and Reopened (implementation details found in *Appendix 2*). The proportions of each category are seen below (*Figure 2*).

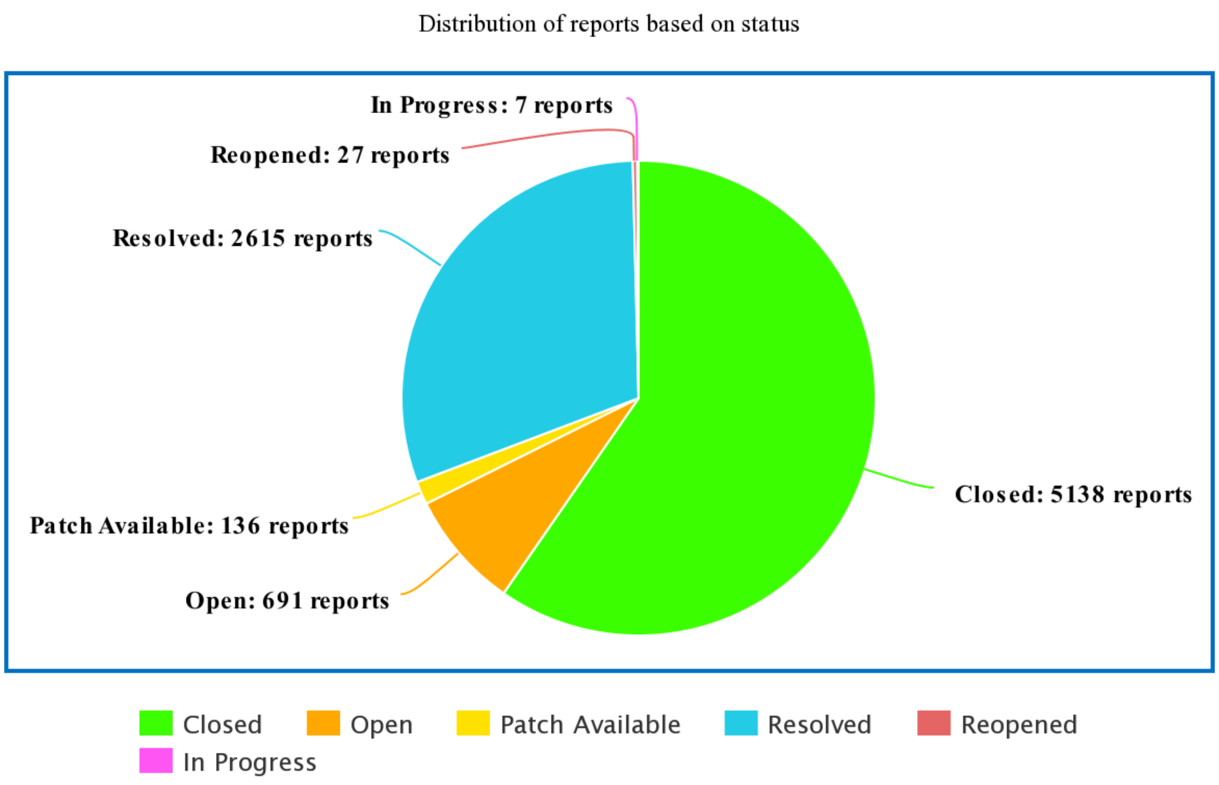

Figure 2: Distribution of bug reports based on status

Finally, using the same set of reports that were specifically bugs, I analyzed those that also had a status that was *fixed* or *closed*. To calculate the bug's lifespan, I looked for the 'created' and 'resolved' tags that contained the date (string format), converted it into python datetime objects, and calculated the difference between the two dates. I was then able to use this to find the minimum, maximum, mean, and median resolution times (implementation details in *Appendix 3*). The results were as follows.

Resolution Time Analysis	**Lifespan of bug report**	**Misc. Notes**
Minimum	0h:00m:14s	From bugReport_1396
Maximum	2960 days, 15h:33m:39s	From bugReport_24
Mean	102 days, 23h:40m:50s	n = 2615
Median	3 days, 22h:26m:19s	Sorted resolution times, then returned middle value

APPENDIX 1 - Python script to analyze categories of reports

```
import xml.etree.ElementTree as ET
import os

if __name__ == "__main__":

    directory = 'C:\\Users\\Chandler\\Downloads\\hbaseBugReport\\hbaseBugReport'
    types_of_reports = {}
    bug_list = []

    for filename in os.listdir(directory):
        if filename.endswith(".xml"):

            tree = ET.parse(os.path.join(filename))

            for node in tree.iter():
                if (node.tag == 'item'):
                    report_type = node.find('type').text

                    if report_type  in types_of_reports:
                        types_of_reports[report_type] += 1  # increment value of report_type
                    else:
                        types_of_reports[report_type] = 1   #initialize key/value pair

    print(types_of_reports)
```

APPENDIX 2 - Python script to analyze status of bug reports

```
import xml.etree.ElementTree as ET
import os
from datetime import datetime, timedelta

if __name__ == "__main__":

    directory = 'C:\\Users\\Chandler\\Downloads\\hbaseBugReport\\hbaseBugReport'
    bugs_by_status = {}
    total = 0

    for filename in os.listdir(directory):
        if filename.endswith(".xml"):

            tree = ET.parse(os.path.join(filename))

            for node in tree.iter():

                if (node.tag == 'item' and node.find('type').text == 'Bug'):

                    report_status = node.find('status').text

                    if report_status in bugs_by_status:
                        bugs_by_status[report_status] += 1
                    else:
                        bugs_by_status[report_status] = 1

                    total += 1

    print(bugs_by_status)
```

APPENDIX3 - Python script to calculate min, max, mean, and median

```
import xml.etree.ElementTree as ET
import os
from datetime import datetime, timedelta

def date_calculator(start_date, end_date):
    # parses strings and converts them to datetime objects. returns the difference between the dates (a timedelta object)
    start = datetime.strptime(start_date[0:-6], '%a, %d %b %Y %X')
    end = datetime.strptime(end_date[0:-6], '%a, %d %b %Y %X')
    diff = end - start
    return diff

if __name__ == "__main__":

    directory = 'C:\\Users\\Chandler\\Downloads\\hbaseBugReport\\hbaseBugReport'
    collection = {}
    res_times = []
    total_res_time = timedelta(seconds=0)

    for filename in os.listdir(directory):
        if filename.endswith(".xml"):

            tree = ET.parse(os.path.join(directory + "\\" + filename))

            bug_type  =  ''
            bug_status = ''
            creation_date_string = ''
            resolution_date_string = ''

            for node in tree.iter():
                if (node.tag == 'type' ):
                    bug_type = node.text
                elif (node.tag == 'created'):
                    creation_date_string = node.text
                elif (node.tag == 'resolved'):
                    resolution_date_string = node.text
                elif (node.tag == 'status'):
                    bug_status = node.text

            if (bug_type == 'Bug' and (bug_status == 'Closed' or bug_status == 'Resolved')):
                lifespan = date_calculator(creation_date_string, resolution_date_string)
                res_times.append(lifespan)
                total_res_time += lifespan

    res_times.sort()    # to find the median

    minimum = min(res_times)
    maximum = max(res_times)
    mean = total_res_time / len(res_times)
    median = res_times[    int(len(res_times)/2) ]

    print('Min resolution time:      ', minimum)
    print('Max resolution time:      ', maximum)
    print('Mean resolution time:    ' , mean)
    print('Median resolution time: ', median)
```