Industrial Programming

Coursework 2

Data Analysis of a Document Tracker

F21SC

Boris Mocialov
(H00180016)

Heriot – Watt University, Edinburgh
December 2014
## Contents

1. Introduction ................................................................................................................. 4
   1.1. Purpose .................................................................................................................. 4
   1.2. Report structure ................................................................................................. 4
   1.3. Challenges .......................................................................................................... 5
   1.4. Assumptions ......................................................................................................... 5
   1.5. Environment ......................................................................................................... 6
2. Requirements .................................................................................................................. 7
3. Design Considerations ................................................................................................. 8
   3.1. UI design .............................................................................................................. 8
   3.2. Unit design ........................................................................................................... 8
   3.3. Choice of data structures ..................................................................................... 9
   3.4. Use of advanced language constructs ................................................................ 10
4. Guides .......................................................................................................................... 11
   4.1. User Guide .......................................................................................................... 11
      4.1.1. Start with parameters .................................................................................. 12
      4.1.2. Show views by continents ......................................................................... 13
      4.1.3. Changing parameters ................................................................................ 14
      4.1.4. Update presentation ...................................................................................... 15
   4.2. Developer Guide ................................................................................................... 15
      4.2.1. Start with parameters .................................................................................. 16
      4.2.2. Show views by continents ......................................................................... 16
      4.2.3. Show views by browsers .......................................................................... 17
      4.2.4. Show readership profile .......................................................................... 18
      4.2.5. Show related documents sorted by f(x) .................................................... 18
      4.2.6. Change of parameters ................................................................................ 20
      4.2.7. Update presentation .................................................................................... 20
      4.2.8. Data modification before rendering chart ................................................. 20
      4.2.9. Rendering of the actual chart object .......................................................... 20
5. Testing & Proof ............................................................................................................ 22
6. Future Work ................................................................................................................. 24
7. Discussion ..................................................................................................................... 25
8. Conclusion .................................................................................................................... 27
Tables

Table 1 Requirements checklist ................................................................. 7
Table 2 Used libraries/modules ............................................................... 10
Table 3 Testing results ............................................................................... 23

Figures

Figure 1 Use-case diagram ..................................................................... 11
Figure 2 Starting application with parameters ......................................... 12
Figure 3 Views by continents ................................................................. 13
Figure 4 Changing parameters .............................................................. 14
Figure 5 Updated presentation ............................................................... 15

Code snippets

Code snippet 1 Triggering main class .................................................. 16
Code snippet 2 Extracting and parsing command line parameters .......... 16
Code snippet 3 Extract data from JSON file ......................................... 16
Code snippet 4 Create in-memory supporting data structure for views by continent task ... 17
Code snippet 5 Pass extracted data to chart-rendering unit .................... 17
Code snippet 6 Create in-memory supporting data structure for views by browsers task .... 17
Code snippet 7 Create in-memory supporting data structure for readership profile task..... 18
Code snippet 8 Create in-memory supporting data structure for the related documents tasks, sorted by a function f(x) ................................................................. 19
Code snippet 9 Extract related data elements and sort by the sorting function ........ 19
Code snippet 10 Changing parameters .................................................. 20
Code snippet 11 Update chart view ........................................................ 20
Code snippet 12 Data re-transformation prior creation of a chart .......... 20
Code snippet 13 Rendering of a chart ...................................................... 21
Code snippet 14 JSON test data ............................................................. 30
1. Introduction

1.1. Purpose
The purpose of this report is to present the development of an application that performs basic data analysis of a document tracker issuu.com. The developed application can process documents in JSON format, extracted from the mentioned document tracker website and perform such tasks as: present views by continent, present views by browser, present reader profiles and list ‘also likes’ related entries.

1.2. Report structure
The report will guide its reader through the development process of the application by listing tools and libraries that were chosen for the implementation of the application following with a short discussion about possible alternatives that exist and were consider by the developer. The list of requirements will clearly state what was required for the final version of application, what was achieved and what was omitted in the final release. Design considerations chapter will give detailed insights of the application by listing artifacts within the structure of the application followed by a discussion on advantages/disadvantages of various choices and their impact on the overall design of the application. Later, a descriptive section is there to guide a user of the application towards some specific tasks that the application can perform, giving textual as well as visual representation of the states of the application. After, developer guide will go into the implementation details and give considerable amount of practical information to another developer who wishes to design a similar piece of software. Towards the end, test cases will be listed together with the testing results, which should give more confidence to the end user in using the application. Testing section is also a proof of the requirements section, which would allow reader to decide for himself/herself whether listed requirements were met. At the end, discussion section will reflect on the developed application and critically evaluate used elements within the application.

Since discussed developed application is a continuation of the previous work – “Developing Simple Web Browser”, the discussion section will compare and contrast development on two different platforms, namely, Windows and Linux. In addition, discussion section will
look at programming languages in general and compare compiled languages with scripting languages and consider advantages and disadvantages of both sides.

1.3. Challenges
In order to avoid additional burden of environment set-up, client-provided facilities were used for the development of this application. Facilities included installation of Python distribution as well as additional libraries that were proposed for the development of the application. Though it had been explicitly mentioned that the application should be Python3.x compatible, provided installation had major issues with regards to proposed additional libraries, such as Pandas (powerful data-analysis library), tkinter (UI library) and matplotlib (plotting library). Therefore the final application had been developed using Python2.x, which was more stable with respect to proposed additional libraries. The use of Pandas library had been omitted due to the fact that the environment did not provide developer with the proper installation of it.

Despite difficulties presented by the development environment, developer regarded mentioned shortcomings as advantageous for better understanding and use of basic tools and libraries provided by Python2.x distribution. In developers’ view, the use of additional libraries makes the job simpler, but at the same time moves the developer from the standard API of the original language to another, more abstract level, which might hide original languages’ structures. This can be perceived as a disadvantage due to the fact that at the end of the project the developer might not know the original programming language as it is, but to know it from a more abstract level, without actual understanding of how certain things are managed.

1.4. Assumptions
Due to the nature of the application and corresponding identified requirements, it is reasonable to make an assumption that the application should not exhibit any advanced UI rendering and relatively little attention should be paid to non-functional requirements, such as safety and usability. Despite the fact that requirements did not explicitly stated the need for non-functional requirements, application should be developed to demonstrate ability to withstand faults (fault tolerance), be efficient in data processing (operability/efficiency), be resilient to failures (fault tolerance/recoverability). In addition the application must be
developed with the though that at any time it may be extended. This thought puts additional constraints on the development, such as that the application must be scalable, extensible and maintainable.

1.5. Environment

The development of the application proceeded in Python2.x using Geany IDE (made available by the presented environment). Additional libraries, namely, matplotlib, collections, itertools, json, string, tkinter and getopt were used for the development of the application, the use of which will be justified later in this chapter. Some of the used libraries are provided by the Python distribution, while others, like matplotlib and tkinter are external libraries.

As for the Geany - development environment used extensively during development of the application, the appreciation of the autocomplete, in-place build&run facilities, overview of units structure, supported the development process and lead to smoother development. Not all the functionality of the IDE had been explored, but it will be recommended to the fellow developer, who would like to use any user-friendly IDE for the python development. Apart from any text-editing software, python development process can be supported by a wide array of commercial and non-commercial IDEs, such as listed at (IntegratedDevelopmentEnvironments, 2014). Some of the alternative IDEs have support for rapid prototyping, which gives a nice touch to the idea of using scripting languages for prototyping purposes.

All of the hacking had been done in Linux environment made available by the presented facilities, which had been different from the first work done in the course of the development activities for the customer. The reader is welcomed to read the discussions section, where some differences between the two environments are given and a general reflection on the languages used is presented.
2. Requirements

The following table lists all the requirements that were posed for the final version of the application. Green checkboxes symbolise the achievement of the requirement, while red checkboxes mean that the requirement was not met.

<table>
<thead>
<tr>
<th>Functional requirements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application should be able to:</strong></td>
<td></td>
</tr>
<tr>
<td>Read JSON file</td>
<td>✔</td>
</tr>
<tr>
<td>Process JSON file</td>
<td>✔</td>
</tr>
<tr>
<td>Interpret JSON file data</td>
<td>✔</td>
</tr>
<tr>
<td>Extract countries</td>
<td>✔</td>
</tr>
<tr>
<td>Group extracted countries by continents</td>
<td>✔</td>
</tr>
<tr>
<td>Extract browsers information</td>
<td>✔</td>
</tr>
<tr>
<td>Extract readership profiles for the website</td>
<td>✔</td>
</tr>
<tr>
<td>Identify which other entries relate to a particular entry</td>
<td>✔</td>
</tr>
<tr>
<td>Extract ‘also like’ data</td>
<td>✔</td>
</tr>
<tr>
<td>Sort processed data</td>
<td>✔</td>
</tr>
<tr>
<td>Distinguish top 10 entries</td>
<td>✔</td>
</tr>
<tr>
<td>Show histogram of processed data</td>
<td>✗</td>
</tr>
<tr>
<td>Print results of processed data</td>
<td>✔</td>
</tr>
<tr>
<td>Read user input via UI</td>
<td>✔</td>
</tr>
<tr>
<td>Read user input via command line</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Total: 15 functional requirements</strong></td>
<td></td>
</tr>
</tbody>
</table>

Concerning the requirement “Show histogram of processed data”, which, in developers’ opinion, was not met, should be explained in more detail. The current application displays information in a bar-chart style, where every bar is a distinct data element. In developers’ opinion, histogram would give extended information to the end-user about the views by countries/continents. Despite this fact, the developer has chosen to pursue the generic representation style to all tasks that the application can handle, without distinguishing whether end-user wishes to look at distinct bars of extracted data elements or at the
combination of such (histogram). The extension of the functionality to support histogram-style representation would require considerable changes to the existing application. Overall, it can be said that this particular requirement has been met partially. Non-functional requirements were not considered widely while developing the application. The reason for that is the nature of the application and its requirements. Application is to be used for the purpose of analysis of the data presented to it from the aforementioned document tracker system. The final version is not supposed to exhibit the variety of user interface functionality nor is it to be optimized for graphics processing, therefore, as mentioned in assumptions section, the development will not consider non-functional requirements widely. Notwithstanding the fact that the non-functional requirements were not enforced by the requirements document, it is a should be noted that the processing time should not become ‘unbearable’ and the functionality should be optimized to some extent by using appropriate data structures and associated functions.

3. Design Considerations

3.1. UI design
The fact that the requirements did not put any additional pressure onto non-functional requirements, such as ‘user-friendliness’, the developed application employs minimalistic design. As a consequence, that the user of the application should not expect powerful and extensive use of UI.

As it will be shown in the user guide section, application uses one main form, which contains a frame for data representation that is updated every time user decides to use application to perform another task. User is able explicitly clear the current data representation. In addition, main window contains menu – so-called ribbon (bar), which gives opportunity to the end-user to specify which task he/she would like to perform. Optionally, user may specify additional fields, by which to refine the represented data.

3.2. Unit design
Python has dynamic notion of classes and considerable programs could be written in a single file. Nevertheless it is a good general program design approach to be able to factorize
whole application into modules for developers’ convenience, so that when making changes
to some portion of code, developer would not spend too much time understanding how the
data flows within the application.
Taking this notion into consideration, the application had been developed to support
factorization and code reuse. As a result, the overall application became more readable and
fewer functions were needed to achieve aimed tasks.
Currently, application is separated into presentation level, the UI level and processing level
as well as data reading level, giving raise to the MVC (model-view-controller) paradigm,
which is explained nicely by (Fowler, 2006).
Application employs a helper class, which contains various flags/ constants used by the
application. This design choice makes it more convenient to be able to make fast changes to
the application as well as to consult what certain variables, which are used throughout the
application, mean.

3.3. Choice of data structures
Mentioned in introduction section chosen libraries made application more usable and
accessible by providing short but powerful interfaces to functionality, such as iterations over
dictionaries, sorting of dictionaries, graphical interface to data representation and overall
data processing within the application. This resulted in application being more accessible.

Table below describes libraries/modules used for the development of the final version of
the application.

<table>
<thead>
<tr>
<th>Library/Module</th>
<th>Description</th>
<th>Rational</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>matplotlib</td>
<td>Gives an opportunity to present data as any widely-used graph type in just a couple of lines of code with additional functionality for adjustment of the canvas, where graph is being presented</td>
<td>Be able to present charts based on processed data</td>
<td>Listed at (Graphical Representations of Data, 2014)</td>
</tr>
<tr>
<td>tkinter</td>
<td>Is a powerful tool for construction of user interfaces, including most of the basic user interface components: buttons, labels, menus, dialogues,</td>
<td>Make UI available for the end-user to operate application</td>
<td>Listed at (GUI Programming in Python, 2014)</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
<td>Details</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>json</td>
<td>From standard Python library, provides interface for handling of internet data in JSON format in a convenient way, so the developer is not concerned with a pure file content, but an abstract JSON structure</td>
<td>Wrapper over the JSON text file with efficient functions for objects retrieval</td>
<td>Pandas</td>
</tr>
<tr>
<td>itertools</td>
<td>Helps with list comprehension and reduces the amount lines of code needed to process lists and make use by combination of various iterable data types</td>
<td>Be able to extract subset of an ordered dictionary</td>
<td>Pandas</td>
</tr>
<tr>
<td>string</td>
<td>Module contains a set of functions that make textual data processing more efficient</td>
<td>Modify JSON file content for the json module</td>
<td>stringlib</td>
</tr>
<tr>
<td>getopts</td>
<td>Module supports parsing of command line arguments</td>
<td>Process user command line input</td>
<td>Should not be considered</td>
</tr>
<tr>
<td>collections</td>
<td>Provides additional data types for efficient in-memory data storage</td>
<td>Make use of ordered dictionary data type</td>
<td>Pandas</td>
</tr>
</tbody>
</table>

**Total additional libraries/modules used: 7**

Table 2 Used libraries/modules

3.4. Use of advanced language constructs

Of all the available array of advanced language constructs in Python, only few were used for the discussed application. In particular, exceptions handling was needed for critical parts of the application, iterators provided by language modules to use lists-comprehension and higher-order functions that were used to sort the results of the analyzed data.

Requirements of the application did not require of any particular use of advanced language constructs. As a side note, advanced features should be considered for the extension of the application as their use would provide a better handling of the inner workings of the application, which will lead to more concise implementation.
4. Guides

4.1. User Guide

This section is included in the report to give the reader (whether user or developer) a clearer understanding of how application is designed, what are its possible states and how it operates.

For the purpose of clarity distinguishing various states of the application, use-case diagram should provide information regarding states of the system that the user should be concerned about. These states are of high-level, which means that they do not cover the actual states of the program under execution. If the reader wants to consult programs states on the lower level, then he/she should look into the developer guide, where states of the application are described in terms of source code.

![Use-case diagram](image)

*Figure 1 Use-case diagram*

Presented use-case diagram identifies 11 use-cases, which will not be discussed separately, since the diagram is fairly descriptive and does not require any additional explanation.
Instead, user guide subsections will present a visual and textual representation of applications state.

Important to the end-user use-cases include:

1.1. Starting application with parameters from a command line
1.2. Viewing analysis results on a chart
1.3. Changing parameters via UI
1.4. Updating chart based on task results

The rest of available functionality should become apparent based on important use-cases and general idea of the application.

Described important use-cases carry crucial functionality of the application, while the rest are merely variation of the described use-cases.

4.1.1. Start with parameters

Starting application from a command line, in this particular case will trigger data analysis process with the aim of identifying how many views were registered based on countries information inside the JSON file.

Available parameters include:

1.4.1. ‘-t’ – task ID, where:
   1 – views by continents
   2 – views by browsers
   3 – avid readers
   4 – related documents sorted by readership profiles
   5 – related documents sorted by number of readers)

1.4.2. ‘-u’ – user UUID
1.4.3. ‘d’ – document UUID

1.4.4. ‘r’ – representation ID, where:
   0 [default] – for chart
   1 – print out to console

4.1.2. Show views by continents

![Chart showing views by continents]

*Figure 3 Views by continents*

The results from previous step (starting application with parameters) will in turn be mapped onto another JSON data – countries-continents mapping, which is used to present to the user visits by continents, rather than countries.
4.1.3. Changing parameters

Change of parameters can be done via File -> Settings navigation. Once user enters desirable parameters and clicks on update button, the previous chart will be updated with the new results.
4.1.4. Update presentation

![Figure 5 Updated presentation](image)

Updated analysis presentation made available to the end-user by substituting previous chart with the new fresh results.

4.2. Developer Guide

As a continuation of design considerations chapter, this section will provide the actual source code that will show a particularly interested reader the implementation details that may inspire, direct or simply inform the reader of how this particular application is constructed.

This chapter will follow the same use-case diagram, which was presented in user guide section as well as unit diagram, given in Appendix A, which should provide more detailed view over the applications’ structure with underlying functions and attributes.
4.2.1. Start with parameters

When user starts application from the command line with additional parameters, main program object is created with underlying additional components.

```python
try:
    program = Main(sys.argv)
except ValueError as e:
    print("oops")
    print(e)
```

Code snippet 1 Triggering main class

In code snippet 1, `Main` object is created with additional parameters. Further, in the `__init__` function, parameters are extracted and parsed, updating appropriate global variables. If user makes a mistake in parameters specification, error message is printed out in the console and a reason is specified to make next attempt to be correct.

```python
#extract parameters
try:
    myopts, args = getopt.getopt(additional_arguments[1:],"u:d:t:r:")
except:
    #unexpected input detected
    print("Unexpected format."
    +program_helper.program_accepts_parameters_string)
    sys.exit()

#parse parameters:
for o, a in myopts:
    if o == '-u':
        self.userUUID = a
    elif o == '-d':
        self.docUUID = a
    elif o == '-t':
        self.statisticsType = a
    elif o == '-r':
        self.representation = a
    else:
        print("Usage: %s -i input -o output" % sys.argv[0])

self.update_canvas()
```

Code snippet 2 Extracting and parsing command line parameters

4.2.2. Show views by continents

When user specifies appropriate parameters to programs’ initialization, following procedure will be executed, which will extract data from a JSON file and pass `json` object to a function that will parse the object, process it according to specified task and return results to the caller main class.

```python
json_data = data_file_extractor.extract_from_file(program_helper.default_json_file_reference,
                                                  self.statisticsType, self.docUUID, self.userUUID)
if json_data != None:
    data, data_type_representation = program.extract_data(json_data, self.statisticsType, self.docUUID, self.userUUID)
```

Code snippet 3 Extract data from JSON file
The central method, which parses data accordingly and creates in-memory structures that will be used for construction of a chart.

```python
def extract_data(json_data, aType, aDocUUID, aUserUUID):
    data_type_representation = -1
    for item in json_data:
        if int(aType) == program_helper.Represent_Type.Countries:
            data_type_representation = program_helper.DataRepresentation.FList
        elif int(aType) == program_helper.Represent_Type.Browsers:
            data_type_representation = program_helper.DataRepresentation.FList
        elif int(aType) == program_helper.Represent_Type.Avid_readers:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes1:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes2:
            ...
        return data, data_type_representation
```

Code snippet 4 Create in-memory supporting data structure for views by continent task

Chart is created and returned to the main caller object, which puts the chart object onto the main form.

```python
self.plt = chart_helper.createGraphCanvas(data, self.representation, data_type_representation)
```

Code snippet 5 Pass extracted data to chart-rendering unit

4.2.3. Show views by browsers

Code snippet 3 extracts data from the JSON file and passes json object to a central function in the application, which in turn processes the object and creates appropriate in-memory data, which is returned to the caller.

```python
def extract_data(json_data, aType, aDocUUID, aUserUUID):
    data_type_representation = -1
    ...
    for item in json_data:
        if int(aType) == program_helper.Represent_Type.Countries:
            ...
        elif int(aType) == program_helper.Represent_Type.Browsers:
            ...
            ...
        elif int(aType) == program_helper.Represent_Type.Avid_readers:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes1:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes2:
            ...
        return data, data_type_representation
```

Code snippet 6 Create in-memory supporting data structure for views by browsers task
Code snippet 5 sends created in-memory data for creation of a chart object and returns that object to the main caller class, which, in turn, places the new chart object onto the main form.

4.2.4. Show readership profile

Before procedure, code snippet 3 repeats its job (same in all the cases)

```python
def extract_data(json_data, aType, aDocUUID, aUserUUID):
    data_type_representation = -1
    data = []
    # iterating data in json
    for item in json_data:
        if int(aType) == program_helper.Represent_Type.Countries:
            ...
        elif int(aType) == program_helper.Represent_Type.Browsers:
            ...
        elif int(aType) == program_helper.Represent_Type.Avid_readers:
            data_type_representation = program_helper.DataRepresentation.FDictionary
            # if type of request is to show top 10 readers based on their time spent reading
            visitor = -1
            readtime = 0
            if program_helper.JSONIdentifiers.visitor_id in item:
                visitor = item[program_helper.JSONIdentifiers.visitor_id]
            if program_helper.JSONIdentifiers.read_time in item:
                readtime = item[program_helper.JSONIdentifiers.read_time]
            data.append((visitor, readtime))
        # last when reading last element of the array
        if item == json_data[-1]:
            data = extract_number_of_readers_for_documents_mapping(data, program_helper.DictionaryConstruction.Use_Number, top=10)
        elif int(aType) == program_helper.Represent_Type.Also_likes1:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes2:
            ...
    return data, data_type_representation
```

Code snippet 7 Create in-memory supporting data structure for readership profile task

After procedure, code snippet 5 repeats its job (same in all the cases)

4.2.5. Show related documents sorted by \( f(x) \)

Before the procedure, code snippet 3 repeats its job
def extract_data(json_data, aType, aDocUUID, aUserUUID):
    ...
    for item in json_data:
        if int(aType) == program_helper.Represent_Type.Countries:
            ...
        elif int(aType) == program_helper.Represent_Type.Browsers:
            ...
        elif int(aType) == program_helper.Represent_Type.Avid_readers:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes1:
            ...
        elif int(aType) == program_helper.Represent_Type.Also_likes2:
            ...
        else:
            data_type_representation = program_helper.DataRepresentation.FDictionary
            # relate docs by who read them - also likes
    visitor = -1
    document = -1
    if program_helper.JSONIdentifiers.visitor_id in item:
        visitor = item[program_helper.JSONIdentifiers.visitor_id]
    if program_helper.JSONIdentifiers.document_id in item:
        document = item[program_helper.JSONIdentifiers.document_id]
    data.append((visitor, document))
    # last when reading last element of the array
    if item == json_data[-1]:
        data = extract_also_like(data, aDocUUID, aUserUUID, \( f(x), \text{is\_reversed=True, top=10} \))
    return data, data_type_representation

Code snippet 8 Create in-memory supporting data structure for the related documents tasks, sorted by a function \( f(x) \)

The actual piece of code that does the search for the related documents and sorts them by a specific function is given below

def extract_also_like(data, doc_UUID, visitor_UUID, sorting_function=None, *arguments, **keywords):
    is_reversed = False
    top = -1
    for kw in keywords.keys():
        if kw == program_helper.VariousParameters.reversed_sort:
            is_reversed = keywords[kw]
        elif kw == program_helper.VariousParameters.select_top:
            top = keywords[kw]
    result = extract_all_visitor_UUIDs(data, doc_UUID) # everyone else who have read this document
    result2 = []
    for visitor in result:
        result2.extend(extract_all_doc_UUIDs(data, visitor)) # list of other documents read by the readers of this document
    result2 = list(set(result2))
    sorting_functions_result = sorting_function # top 10 read documents (expanded) & sorted (doc, readtime)
    result3 = []
    for item in result2:
        for item2_a, item2_b in sorting_functions_result.items()[:top:-1]:
            if item == item2_a and item != doc_UUID:
                result3.append((item, item2_b))
    result4 = OrderedDict(itemdata.isokeys(OrderedDict(sorted(result3, key=lambda an_item: an_item[1], reverse=True, is_reversed))).iteritems()
    # top3(len(result) if top3(len(result) else top))
    result = result4
    return result

Code snippet 9 Extract related data elements and sort by the sorting function

The function ‘extract_also_like(…)’ uses 2 underlying functions, namely, ‘extract_all_visitor_UUIDs(…)’ and ‘extract_all_visitor_UUIDs(…)’, which together perform search for other elements in the data set that are related to the original element, specified by the user (document UUID). Later, the accumulated data is sorted on the basis of the sorting function, which is given as a parameter to the ‘extract_also_like(…)’ function. After all the above processing, code snippet 5 repeats its job.
4.2.6. Change of parameters

Change of the parameters via UI destroys currently displayed chart and updates global variables in the main unit – cw2

```python
def trigger_update(self):
    # destroy previous graph
    self.close_current_view()
    self.statisticsType = self.task_entry.get()
    self.docUUID = self.doc_uuid_entry.get()
    self.userUUID = self.reader_uuid_entry.get()
    self.update_canvas()
```

Code snippet 10 Changing parameters

4.2.7. Update presentation

Procedure that creates a chart object from supporting in-memory data structure for a particular task, and updates the current chart view with the newly created one.

```python
self.plt = chart_helper.createGraphCanvas(data, self.representation, data_type_representation)
self.frame = Frame(self.window)
self.frame.pack()
# embed graph into main window
if self.plt != None:
    canvas = FigureCanvasTkAgg(self.plt.figure(1), master=self.frame)
    canvas._tkcanvas.pack(side=Tkinter.TOP, fill=Tkinter.BOTH, expand=True)
```

Code snippet 11 Update chart view

4.2.8. Data modification before rendering chart

Before the data is used for the creation of a chart object, data is transformed to a unified representation, which is used when rendering a chart object.

Data, given to ‘chart_helper’ unit arrives in either a dictionary format or a list format and both need to be re-transformed.

```python
if data_type == program_helper.DataRepresentation.FList:
    result_dict = dict((i, data.count(i)) for i in set(data))
elif data_type == program_helper.DataRepresentation.FDictionary:
    result_dict = data
else:
    return
```

Code snippet 12 Data re-transformation prior creation of a chart

4.2.9. Rendering of the actual chart object

After the data been re-transformed, it is ready to be used for the creation of a chart, which will present analysis results to the end-user in a form of bar chart. Following code shows how chart object can be rendered.
def createGraphCanvas(data, representation, data_type):
    if data != None and len(data) > 0:
        # create a dictionary of found data, where key is category and value is occurrence number

        # lists to hold dictionary values and dictionary keys
        dictListValues = []
        dictListKeys = []

        # populate lists
        for key, value in result_dict.items():
            dictListValues.append(value)
            if len(key) > 5:
                key = ".." + key[-5:]
                # better would be to change rotation or width if text is too long
                dictListKeys.append(key)

        # amount of categories
        N = len(dictListValues)
        menMeans = tuple(dictListValues)

        ind = list(range(N));

        # specifying width of bars
        width = 0.7

        # make 2 subplots
        fig, ax = plt.subplots()

        # create graph bars
        rects1 = ax.bar(ind, menMeans, width, color="y")

        # various settings
        ax.set_xticks([1]*len(ind))
        ax.set_xticklabels( tuple(dictListKeys) )

        # specify position of ticks
        ax.tick_params(top = 'off', bottom = 'off', right = 'off')
        ax.grid(axis = 'y', linestyle = '-')
        ax.set_yticks(rotation=70)

        # adjust y-ticks values so that only integers will be shown
        if max(dictListValues) > 10:
            ax.set_yticks(0, max(dictListValues) + max(dictListValues)*0.1)
        else:
            ax.set_yticks(0, 10)

        # put labels on graph
        make_labels(ax, rects1)

        return plt;
5. Testing & Proof

Testing proceeded in the standard fashion, where input to the application was given and output was compared to the expected output. The difference considered being the failure to meet a particular requirement.

Due to the fact that provided test data was not suitable for the overall test, especially for the related elements extraction task, the original test data was modified to include the information, which could be used during the testing of the application.

Excerpt of the test data that was modified is given in the Appendix B. Changes to the test data included insertion of additional entries in the JSON format, which included the other documents that were read by the readers of the specified document. In particular, for the document UUID: ‘140228202800-6ef39a241f35301a9a42cd0ed21e5fb0’, following entry was added: The same user that read mentioned document, also read ‘130705172251-3a2a725b2b53aa3f2af810acf0aeabo’, which symbolizes that both document UUIDs are related.

After identifying which documents are related, they are sorted according to the sorting function provided.

The table below gives test cases that were conducted during the testing phase. The test data given in Appendix B was used for the testing of the application.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Expected results</th>
<th>Actual results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show views by continent</td>
<td>MX-5; VE-7</td>
<td>NA-5</td>
</tr>
<tr>
<td></td>
<td>maps onto -&gt;</td>
<td>SA-7</td>
</tr>
<tr>
<td></td>
<td>NA-5; SA-7</td>
<td></td>
</tr>
<tr>
<td>Show views by browser</td>
<td>Mozilla-12</td>
<td>Mozilla</td>
</tr>
<tr>
<td></td>
<td>Firefox-7</td>
<td>Firefox</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other</td>
</tr>
<tr>
<td>Show readership profiles</td>
<td>...c1-797</td>
<td>...c1-797</td>
</tr>
<tr>
<td></td>
<td>...c2-797</td>
<td>...c2-797</td>
</tr>
<tr>
<td></td>
<td>...c7-797*5+79799+7978=91762</td>
<td>...c7-91762</td>
</tr>
<tr>
<td>Print related documents</td>
<td>..bc-79799</td>
<td>..bc</td>
</tr>
</tbody>
</table>
Therefore:
2. ...bc
3. ...bb
4. the rest

Print related document
(sorted by the number of readers)

Change parameters via UI
Updated results of the new task are shown on the chart

Start the application via command line w/parameters
Results of the task are shown on the chart

**Total test cases performed:** 7

**Total test cases passed:** 7
Testing covered the main functionality of the application, while additional explicit border cases were omitted due to the lack of time. Implicit additional test cases were performed during the development of the application, where unexpected input had been given at various stages of the development and output tested. Developer of the application is confident in the application to some extent, but unexpected results may occur during the execution.

It is strongly advised to perform additional testing before extending the existing application, because application lacks extensive test cases as it can be seen from the testing results table.

Specifically, application must be tested with regards to its boundaries: what happens if JSON file of wrong formatting is given, what happens if application is given a very complex problem, would it run out of memory?, what happens if end-user will try to inject the command line parameters/input?

6. Future Work

The future work should consider extending the application and adjusting current functionality to be able to meet the stated requirements.

In particular, for meeting current requirement ‘to the word’, it is advised to:

4.2.1. Implement histogram/bar chart representation (be able to switch)
4.2.2. Parse ‘visitor_useragent’ precisely
4.2.3. Find use for the ‘visitor_UUID’ in related documents refinement
4.2.4. Optimize and improve GUI usage
4.2.5. Port the application to Python3.x
4.2.6. Display continent names instead of abbreviations

In addition, following ideas may be appropriate for the functional extension of the current application:

1. Be able to control the presented chart (zoom/move/extract/etc.)
2. Provide parameter setting for displaying top N elements
3. Optimize related elements functionality (current implementation may seem to be redundant)
4. Implement other sorting functions and try re-using the ‘extract_also_like’ function
5. Make parameters uniform, so that application may be able to behave dynamically and be able to parse other documents to some extent
6. Improve performance-related issues (if such will be found)

7. Discussion

As mentioned before, this work is a continuation of the overall course of the software development, where the developer had to grasp new programming languages relatively fast and be able to construct complex applications, using learned languages. The first implementation was done in C# programming language (Simple Web Browse) and the second one had been in Python (Data Analysis Application).

During the course of the implementation of both applications in different programming languages, the developer learned and appreciated the differences between scripting languages (i.e. Python) and compiled languages (i.e. C#).

Identified differences include:

1. Strong typing (C#) vs absence of strong typing (Python)
2. Speed of execution slow (Python) vs fast (C#)
3. Access to the lower-level OS components; present (C#) vs not present (Python)
4. ‘Glueware’-like behavior (Python) vs building algorithms and data structures from scratch (C#)
5. Rapid prototyping development methodology (Python) vs full solid implementation approach (C#)
6. Safety-focused and generally has stronger focus on non-functional aspects of software (C#) vs lesser non-functional requirements needs (Python)

Based on the identified differences, it is crucial what the developer want to achieve in the development of a software and what are the requirements and which assumptions are being made in the process of analysis of the requirements. In addition, performance-related
issues should be of stronger concern when choosing a programming language, since scripting languages are interpreted runtime, therefore the execution time is relatively slower and additional effort would be needed to make code optimized as much as possible to be able to reach the optimal implementation.

As development of the both application proceeded on different platform, namely, Windows and Linux, it can be noted that, when working on Linux platform, developer should expect more command line usage and less graphical support for the task at hand, while on the Windows platform, user is guided through by a rich spectrum of graphical interface, which could be seen as an advantage for a less experienced user.
8. Conclusion

In conclusion, the overall aim of the application had been met with minor deviations from the identified requirements.

This report had presented the developed application at a higher-level for the end-user as well as at a lower-level for the fellow developer. The structure of this report had given the reader overview over which requirements were stated by the client together with the information and elaboration over which requirements were met and which were not. Design considerations section had listed most of the important design decisions that were made during the course of the application development. Alternatives for the chosen development components are given for a reader to be able to decide whether named alternatives are better than those that were chosen for this project. Both users and developers guides had presented the application from end-user point of view and from developer’s point of view. It is strongly encouraged that both of the sections should be read together, taking information from the both in order to build a better understanding of what application is actually capable of and how this capability is achieved. Testing section fairly presented testing results to the reader, which should be used as a reference in making a decision on whether developed application met posed requirements. Future work section had presented some ideas for the reader about how the current version can be updated or improved. The discussion section had taken into consideration the previous work done by the same developer – the C# implementation and compared the work done in this iteration with the work done in the previous iteration. That particular section should give the reader understanding of differences between two applications and their development processes. The reader is encouraged to read the report produced at the previous iteration, describing development on another platform, using different programming language.
References


Bibliography


https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Appendix B

```json
[ "ts": 139361098, "visitor_userid": "745409913574d4c7", "visitor_usernamed": null, "visitor_source": "external", "visitor_device": "browser", "visitor_ussertagent": "Mozilla/5.0 (iPhone; CPU iPhone OS 7_6_3 like Mac OS X) AppleWebKit/537.15 (KHTML, like Gecko) Mobile/11B651 [FBAN/FBIOS;FBAV/7.6.6;FBID/2; FBGR/telco;FB1 D;phone;FLS/ES;FB0/3]; visitor_ip": ".", visitor_country": "MX", visitor_referrer": ".", "env_type": "reader", "env_doc_id": "140228202800", "env_adds": null, "event_type": "impresion", "subject_type": "doc", "subject_doc_id": ".", "subject_page": 23, "cause_type": "page" };

[ "ts": 139361098, "visitor_userid": "745409913574d4c7", "visitor_usernamed": null, "visitor_source": "external", "visitor_device": "browser", "visitor_ussertagent": "Mozilla/5.0 (iPhone; CPU iPhone OS 7_6_3 like Mac OS X) AppleWebKit/537.15 (KHTML, like Gecko) Mobile/11B651 [FBAN/FBIOS;FBAV/7.6.6;FBID/2; FBGR/telco;FB1 D;phone;FLS/ES;FB0/3]; visitor_ip": ".", visitor_country": "MK", visitor_referrer": ".", "env_type": "reader", "env_doc_id": "140228202800", "env_adds": null, "event_type": "impresion", "subject_type": "doc", "subject_doc_id": ".", "subject_page": 23, "cause_type": "page" ]
```

Code snippet 14 JSON test data