|  |  |
| --- | --- |
| **C:\Users\kshah3\AppData\Local\Microsoft\Windows\INetCache\IE\KWFN8A58\CHS Letterhead.png** | |
| **ASSESSMENT TASK NOTIFICATION**  **(This sheet must be handed in with your task)** | |
| **Student:** Michael Kneale | **Task Number:** 4 |
| **Subject:** Software Design and Development | **Weighting** 30 % |
| **Year / Class:** 12SDD1 | **Date Issued:** 24th June 2019 |
| **Teacher:** K Shah | **Date Due**: 6th September 2019 |
| **Task Title:** Design a software solution | **Date Submitted:** 6th Sept. |

|  |
| --- |
| **OUTCOMES ASSESSED:**  H4.1 identifies needs to which software solutions are appropriate  H4.2 applies appropriate development methods to solve software problems.  H4.3 applies a modular approach to implement well structured software solutions and evaluates their effectiveness  H5.1 applies project management techniques to maximise the productivity of the software development.  H5.2 creates and justifies the need for the various types of documentation required for a software solution.  H5.3 selects and applies appropriate software to facilitate the design and development of software solutions.  H6.2 communicates the processes involved in a software solution to an inexperienced user.  H6.4 develops effective user interfaces, in consultation with appropriate people. |
| **TASK DESCRIPTION:**  You are required to find a *real* client for a software project. Follow all the procedures for a structured approach to software design and include **all** appropriate documentation. The final product must be packaged in a way that it is ready to be produced for mass consumption. Consider the marketing of your product and as well as an appropriate user manual for your audience. |
| **TASK REQUIREMENTS:**  **al and**  Documentation;   1. Analysis (This must be checked by me before you continue as well as be signed off by your client) 2. Feasibility report 3. Gantt chart/Log Book must record conversations and clients meeting attached to the end of your report. 4. Data flow diagram 5. IPO diagram 6. Structure Chart 7. System Flow charts (for all modules as well as the main program) 8. Data Dictionary 9. Storyboard for interface design 10. Algorithm /Pseudocode 11. Source code 12. Testing documentation (approaches and results);     1. hardware test     2. software test     3. Data desk checks     4. Personnel and Procedural   **The program must be submitted on a USB Drive/Stick with an automatic setup and/or startup. Include your licence agreement. The software must be signed off as meeting specifications by your client.** |

**Assessment Marking guidelines**

**Software Documentation**

|  |  |
| --- | --- |
| **Descriptor** | **Possible Marks** |
| * identifies needs to which software solutions are appropriate * applies appropriate development methods to solve software problems. * applies project management techniques to maximise the productivity of the software development. * Comprehensively creates the various types of documentation required for a software solution. * Clearly and comprehensively communicates the processes involved in a software solution to an inexperienced user. | 13 – 15 |
| * identifies needs to which software solutions are appropriate * applies appropriate development methods to solve software problems. * applies some project management techniques to maximise the productivity of the software development. * creates the various types of documentation required for a software solution. * Generally communicates the processes involved in a software solution to an inexperienced user. | 10 - 12 |
| * identifies needs to which software solutions are appropriate * applies appropriate development methods to solve software problems. * applies project management techniques to software development. * Briefly outlines the various types of documentation required for a software solution. * Outlines the processes involved in a software solution to an inexperienced user. | 7-9 |
| * identifies needs to which software solutions are appropriate * Attempts appropriate development methods to solve software problems. * Identifies project management techniques to maximise the productivity of the software development. * Identifies the various types of documentation required for a software solution. * Identifies the processes involved in a software solution to an inexperienced user. | 3 - 6 |
| **Software Documentation Total** | /15 |

Software Solution

|  |  |
| --- | --- |
| **Descriptor** | **Possible Marks** |
| * applies a modular approach to implement well structured software solutions and evaluates their effectiveness * selects and applies appropriate software to facilitate the design and development of software solutions. * develops effective user interfaces, in consultation with appropriate people. | 9 - 10 |
| * applies a modular approach to implement structured software solutions and evaluates their effectiveness * uses appropriate software to facilitate the design and development of software solutions. * develops effective user interfaces, in consultation with appropriate people. | 7 – 8 |
| * applies a development approach to implement software solutions and evaluates their effectiveness * uses appropriate software to facilitate the design and development of software solutions. * Develops a mostly effective user interfaces, in consultation with appropriate people. | 5 – 6 |
| * Attempts to us an appropriate approach to creating structured software solutions and evaluates their effectiveness * uses appropriate software to facilitate the design and development of software solutions. * Develops a generally effective user interfaces, in consultation with appropriate people. | 3- 4 |
| * Attempts to create a software solution and briefly evaluates its effectiveness * uses software to facilitate the design and development of software solutions. * develops an effective user interfaces | 1 – 2 |
| **Software Solution Total** | /10 |
| **Project Total** | /25 |

Assessment Task #4: Design a Software Solution

Due: 6th September 2019

By: Michael Kneale

**Feasibility report**

It is important that the value of the software solution outweighs the estimated cost and risk. In the case of this finance manager, it is important that the advantages of having one’s expenses sorted using the aforementioned program outweighs the time spent by the developer creating the program, as well as the time spent by the client entering the data on a regular basis and acclimatising themselves to using the new system. Purchasing the product is a nonissue for this particular case due to the free nature of Python and associated software, and so cost on the sides of both the developer and client are not included in this report – though it is worth mentioning that the lack of any cost makes the project considerably more feasible.

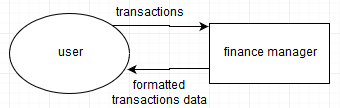
On average, it was estimated by the client that they would save an hour a week using a finance manager rather than manually trying to calculate expenses using pen and paper. The greatest benefit of such a program is to aid the client in monitoring their expenses, especially in categories that are highly variable. For example, groceries, clothing, and entertainment can be reduced once the client has a greater awareness of their current expenditure in such categories. It is estimated that between $40 and $80 per week could be saved through the software solution. Other categories, such as rates and health costs, may not be able to be reduced through utilising the financial manager. However, it is still helpful to monitor these as well, so that the client is entirely cognisant of how much they are spending. Overall, the potential for savings and increased expenditure awareness would greatly increase customer satisfaction.

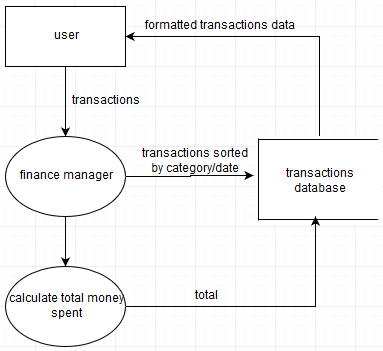
The finance manager software solution intends to offer an intuitive and user-friendly experience to the client, and requires only basic resources to use. The client already has access to these resources, that is, a computer, mouse, and keyboard, as well as a Windows 10 operating system. The client would need basic training on how to use the program, but this would be quick and inexpensive.

The finance manager will easily fit into the client’s home environment as it will be a small, unobtrusive, and easily to set up program that will not interfere with any other software installed. The program will assume in the user nothing more than a basic understanding of how computers and associated hardware work.

The benefits of the program, then, clearly outweigh the risks, because there is no cost and the client will have access to a much more time-efficient method of tracking and organising their finances than before.

**Data flow diagrams**





**IPO diagram**

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Amount spent on:   * cars * clothes * electronics * rates * donations * gas/electricity * entertainment * groceries * gifts * health * holidays * insurance * house/garden * books/music * petrol * school * phone/internet * transport * water   Transaction dates  Exit program | Put given data into database for each transaction  Programmatically retrieve individual items  Calculate total money spent | Screen with expenditure sorted by category and date  Total money spent |

**Structure chart**

A close up of text on a whiteboard

Description automatically generated

**System flowcharts**

**Data dictionary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data item** | **Type** | **Length** | **Valid values** | **Description/purpose** |
| Category | string | 30 | |A-Z|, |a-z|, |/| | Stores the category under which the transaction falls (gifts, health, etc.). Input by the user |
| Cost | floating point | 10 | |0-9|, |.| | Stores the cost of the transaction. Input by the user |
| Date | string | 8 | |0-9|, |/| | Stores the date that the transaction was made. Input by the user |
| variable | string | 20 | |A-Z|, |a-z|, |/| | Allows the members of a dropdown menu to be called |
| self.i | integer | 3 | |0-9| | Allows for the incrementation of the “ID number” going down the database table |

**Storyboard**

A close up of text on a white background

Description automatically generated

**Algorithm (pseudocode)**

BEGIN financeManager

IF enterExpenses IS PRESSED

submitExpenses

ELSEIF viewExpenses IS PRESSED

showExpenses

IF viewTotal IS PRESSED

findSum

ENDIF

ELSEIF exitButton OR xButton IS PRESSED

exitApplication

ENDIF

END financeManager

BEGIN submitExpenses

GET transaction\_category FROM user

GET transaction\_amount FROM user

GET transaction\_date FROM user below thing is submitsdsssdadsdsa

TRANSFER transaction\_category AND transaction\_amount AND transaction\_date INTO database

END submitExpenses

BEGIN showExpenses

IF data IN table

DELETE data IN table

ENDIF

CREATE table

ENTER transaction\_category INTO column1

ENTER transaction\_amount INTO column2

ENTER transaction\_date INTO column3

END showExpenses

BEGIN findSum

sum = 0

FOR i IN table

sum = sum + cost[i]

DISPLAY sum

END findSum

BEGIN exitApplication

PROMPT user: “are you sure you wish to exit?”

IF user INPUTS “yes”

EXIT program

ELSEIF user INPUTS “no”

PRINT “you will now return to the program”

ENDIF

END exitApplication

**Source code (comments indicated by hashes)**

#Most later widgets aren't commented on because their functions (or the logic behind them) have been explained in earlier widgets and get repetitive

#Imports all of the tkinter functions, allowing for the creation of a GUI. tkinter is installed by default, but must be imported in order to be used

import tkinter as tk

from tkinter import ttk

from tkinter import \*

from tkinter import messagebox

#Imports sqlite3, which allows for the implementation of database functionality

import sqlite3

#Defining font variables of different sizes (and, in the case of LARGE\_FONT, a different boldness) so that they can be used later on

TITLE\_FONT= ("Verdana", 30)

LARGE\_FONT= ("Verdana", 20, 'bold')

MEDIUM\_FONT= ("Verdana", 16)

SMALL\_FONT= ("Verdana", 8)

#Creates database first time around (afterwards it connects to the one already created)

connection = sqlite3.connect('address\_book.db')

#Creates a 'cursor' (i.e. allows for traversing a database later on)

c = connection.cursor()

#Creates table (in which database info will be stored) the first time running the program, otherwise does nothing (as per the "IF NOT EXISTS")

c.execute("""CREATE TABLE IF NOT EXISTS addresses (

date integer,

cost integer,

category text

)""")

#Defines the base frame off of which the windows are based

class finance(tk.Tk):

#Defines what the initiation function does, and allows args (arguments) and kwargs (keyword arguments) to be passed

def \_\_init\_\_(self, \*args, \*\*kwargs):

#Initiates screens

tk.Tk.\_\_init\_\_(self, \*args, \*\*kwargs)

#Titles the window

tk.Tk.wm\_title(self, "Finance Manager")

container = tk.Frame(self)

#Ensures that the container (where widgets are able to be placed) fills the window

container.pack(side="top", fill="both", expand = True)

#Weight means that the container will grow given there is space to do so

container.grid\_rowconfigure(0, weight=1)

container.grid\_columnconfigure(0, weight=1)

self.frames = {}

for F in (StartPage, EnterExpenses, ViewExpenses):

frame= F(container, self)

#Sets frames to white with its hexadecimal colour value

frame.configure(bg="#FFFFFF")

self.frames[F] = frame

#Makes the grid (in which widgets are organised) cover the entirety of the frame. Sticky="nsew" means it is being stretched to each side of the window (north, south, east, west sides)

frame.grid(row=0, column=0, sticky="nsew")

#Shows StartPage by default

self.show\_frame(StartPage)

def show\_frame(self, cont):

frame = self.frames[cont]

frame.tkraise()

#StartPage screen

class StartPage(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

#Makes the variable label a tkinter label displaying the text "Finances 2019-2020", using the large title font, and appearing in the zeroth column and zeroth row

label = tk.Label(self, text="Finances 2019-2020", font=TITLE\_FONT)

label.grid(row=0, column=0)

button1 = ttk.Button(self, text="Enter expenses",

command=lambda: controller.show\_frame(EnterExpenses))

button1.grid(row=1, column=0, padx=25, pady=20)

button2 = ttk.Button(self, text="View expenses",

command=lambda: controller.show\_frame(ViewExpenses))

button2.grid(row=2, column=0, padx=25)

canvas1 = tk.Canvas(self, width = 300, height = 300, bg="#FFFFFF", highlightthickness=0)

canvas1.grid(row=3, column=0, padx=25)

#Function to exit the program that is triggered when the "Exit application" button (which is defined below) is pressed.

def ExitApplication():

MsgBox = messagebox.askquestion ('Exit Application','Are you sure you want to exit the application?',icon = 'warning')

if MsgBox == 'yes':

quit(app)

else:

tk.messagebox.showinfo('Return','You will now return to the application screen')

button1 = tk.Button (self, text='Exit application',command=ExitApplication)

canvas1.create\_window(150, 150, window=button1)

#EnterExpenses screen

class EnterExpenses(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

label1 = tk.Label(self, text="Enter expenses", font=LARGE\_FONT)

label1.grid(row=0, column=0)

label1.config(bg="#FFFFFF")

label2 = tk.Label(self, text="Date", font=MEDIUM\_FONT)

label2.grid(row=1, column = 0, sticky="sw")

label2.config(bg="#FFFFFF")

label3 = tk.Label(self, text="Cost", font=MEDIUM\_FONT)

label3.grid(row=2, column = 0, sticky="sw")

label3.config(bg="#FFFFFF")

label4 = tk.Label(self, text="Category", font=MEDIUM\_FONT)

label4.grid(row=3, column = 0, sticky="sw")

label4.config(bg="#FFFFFF")

button1 = ttk.Button(self, text="Back",

command=lambda: controller.show\_frame(StartPage))

button1.grid(row=5, column=1, pady=50)

dateEntry = ttk.Entry(self)

dateEntry.grid(row=1, column=1)

variable = tk.StringVar()

drop = ttk.OptionMenu(self, variable, "cars", "cars", "clothes", "electronics", "rates", "donations", "gas/electricity", "entertainment", "groceries", "gifts", "health", "holidays", "insurance", "house/garden", "books/music", "petrol", "school", "phone/internet", "transport", "water")

drop.grid(row=3, column=1, sticky="ew")

costEntry = ttk.Entry(self)

costEntry.grid(row=2, column=1)

#Defines the submit function, which inserts data

def submit():

connection = sqlite3.connect('address\_book.db')

c = connection.cursor()

c.execute("INSERT INTO addresses VALUES (:date, :category, :cost)",

{

'date': dateEntry.get(),

'category': variable.get(),

'cost': costEntry.get()

})

connection.commit()

connection.close()

dateEntry.delete(0, END)

costEntry.delete(0, END)

submitButton = ttk.Button(self, text="Add record to database", command=submit)

submitButton.grid(row=4, column=1, pady=20)

#ViewExpenses screen

class ViewExpenses(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

label = tk.Label(self, text="View expenses", font=LARGE\_FONT)

label.grid(row=0, column=0)

label.config(bg="#FFFFFF")

#Back button to return to StartPage

button1 = ttk.Button(self, text="Back",

command=lambda: controller.show\_frame(StartPage))

button1.grid(row=6, column=0, pady=50)

#Defines the 'view' function, which is later called through a button. It essentially shows the database data in table form

def view():

#Establishes connection to database

connection = sqlite3.connect('address\_book.db')

c = connection.cursor()

#Clears table in case there is already data in it (if this line were not included, there would likely end up being duplicate rows in the table)

tree.delete(\*tree.get\_children())

#Retrieves all data from the database to display in the table

c.execute("SELECT \* FROM addresses")

rows = c.fetchall()

self.i = 0

for row in rows:

print(row)

tree.insert("", tk.END, text=str(self.i), values=row)

self.i = self.i + 1

#Ceases connection to database

connection.commit()

connection.close()

#Treeview allows for the creation of a table. The columns are defined and made the right size

tree = ttk.Treeview(self)

tree.grid(row=1, column=0, pady=20, padx=20)

tree['columns'] = ('cost', 'category', 'date')

tree.heading('#0', text='ID number', anchor='w')

tree.column('#0', minwidth=0, width=65, anchor='w')

tree.heading('cost', text='Date')

tree.column('cost', width=75)

tree.heading('category', text='Category')

tree.column('category', width=100)

tree.heading('date', text='Cost')

tree.column('date', width=100)

self.Treeview = tree

#Finds sum of all "cost" values in treeview and displays it in the totalLabel widget defined below

def findSum():

sum1 = 0.0

for child in tree.get\_children():

sum1 += float(tree.item(child, "values")[2])

totalLabel.config(text=sum1)

total.set(sum1)

total = StringVar()

totalLabel = ttk.Label(self, text='Press "view total" to add up all transactions')

totalLabel.grid(row=4, column=0)

viewTotal = ttk.Button(self, text="View total", command=findSum)

viewTotal.grid(row=3, column=0, pady=20)

#Allows for the "view" command to be called through the press of a button

viewButton = ttk.Button(self, text="View data", command=view)

viewButton.grid(row=2, column=0, pady=20)

#Application becomes the finance function i.e. the main frame through on which all the windows are defined/built

app = finance()

#Ensures the application cannot be resized (as resizing is unnecessary and could cause formatting issues)

app.resizable(False, False)

#Defines the resolution of the program (x pixels wide by y pixels high)

app.geometry("410x600")

#Same as the earlier ExitApplication function (in that it closes the program), except it is triggered when pressing the X button in the top right (which can be done from any screen)

def on\_closing():

if messagebox.askokcancel("Exit Application", "Are you sure you want to exit the application?", icon = 'warning'):

app.destroy()

else:

tk.messagebox.showinfo('Return','You will now return to the application screen')

app.protocol("WM\_DELETE\_WINDOW", on\_closing)

#Makes the program and its windows run in an infinite loop

app.mainloop()

#Commits changes made to database/spreadsheet

connection.commit()

#Closes connection to database

connection.close()

**Testing documentation**

**Hardware test**

The program was tested on the client’s computer and ran smoothly, without any issues. The program was also tested on a number of other computers, so if the client wishes to move the program to any other Windows computer, it should continue to run well. .exe and .db (database) files easily run on all Windows computers, so there was no problem in that regard. The small size of the program’s graphical user interface means it can run on virtually any desktop screen without being cut off or otherwise misshapen.

**Software test**

The program was tested on several computers with a large number of other programs installed and running. The program worked on a computer with well over a terabyte’s worth of applications installed, so the program is unlikely to stop working based on what is installed on a computer.

Additionally, the program was tested on computers without Python installed, verifying that the executable is able to be run without the need for any prior installations.

The program also worked on the client’s device whilst other applications were running, such as Google Chrome and Microsoft Outlook. There were no conflicts, and the program did not run any slower.

**Data desk check**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Line | Date | Cost | Category | Total | Output (from database) |
| 1 | 12/12/19 | 35 | gas/electricity | 35 | 0 | 12/12/19 | gas/electricity | 35 |
| 2 | 6/2/19 | 5 | petrol | 40 | 1 | 6/2/19 | petrol | 5 |
| 3 | 5/1/19 | 50 | groceries | 90 | 2 | 5/1/19 | groceries | 50 |
| 4 | 8/1/19 | 177 | cars | 267 | 3 | 8/1/19 | cars | 177 |
| 5 | 23/2/19 | 359 | insurance | 626 | 4 | 23/2/19 | insurance | 359 |
| 6 | 500 | 34 | cars | 660 | 5 | 500 | cars | 34 |
| 7 | 28/7/19 | 9.5 | house/garden | 669.5 | 6 | 28/7/19 | house/garden | 9.5 |
| 8 | 15/7/19 | 999 | holidays | 1668.5 | 7 | 15/7/19 | holidays | 999 |
| 9 | 14/5/20 | abc | gifts | N/A | 8 | 14/5/20 | gifts | abc |
| 10 | 14/5/20 | 24 | house/garden | N/A | 9 | 14/5/20 | house/garden | 24 |

As shown above, entering an incorrect date value will not cause any issues. Entering an incorrect cost value will stop the total from being able to be calculated, but the program will still run. An incorrect category cannot be entered as a dropdown menu is used to choose one. An incorrect ID number (which precedes the date in the output) cannot be incorrect as they are automatically generated and correctly assigned.

**Personnel and procedures**

The program was beta tested by users outside of the development process, who had no prior experience with the program and varying degrees of skill regarding using computers. The consensus regarding the user interface in particular was that it was simple, user-friendly, and fairly intuitive. However, it was noted that some of the buttons should probably have been made larger so that they were easier to click.

The rest of the program was also fairly intuitive, however it was made clear that the program would have greatly benefitted from having a table of data that updated automatically, rather than needing to press “View data” all the time. The instantaneous response times of all the buttons in the program improved the user experience.

**User manual**

The user need only double click on the .exe file (named “finance\_manager”) for the program to run.

The program can be installed anywhere and moved to anywhere, however upon running the program for the first time, a database is created in which transactions data is kept. Should the user wish to move the program to another folder/directory, they should move the database file (“address\_book”) along with it and place it in the same folder/directory so it can be called on later.

The rest of the program is straightforward. The user can click the “Enter expenses” button to be brought to a window where they can enter the date, cost, and category of their transactions. They can also press “View expenses” to see a table with their transactions data in it (though the “View data” button must first be pressed). If the user wishes to update the table, they must press “View data” again, as it does not update automatically.

The “View total” button in the “View expenses” screen can be used to see the sum of all the transactions’ costs. If pressed again, it will update the total displayed to account for any new data that may have been added.

To exit the application, the user can press the “Exit application” button on the start screen or the X button in the top right corner. There is no difference between the two. There is also no risk in doing so, because the user is asked whether they are sure they wish to exit the program, before the program finally exits.

|  |  |
| --- | --- |
| **Date** | **Tasks done/communication with client** |
| **15/7/19** | Discussed with client the specifics of the problem that has to be solved. Client specified that the program include major household expenditure categories, sorts, and total (i.e. sum of all costs).  Created Gantt chart detailing the interrelationship and sequencing of tasks, and making it easier to stick to schedule and be on time regarding the completion of tasks. Ensured that all necessary tasks were included so that all the documentation necessary would end up being included. |
| **20/7/19** | Finished and submitted analysis of problem to be solved, which is an outline of the design requirements and specifications communicated by the client.  Researched suitable programming languages. In particular, looked for something that allows the implementation of databases and graphical user interfaces (as well as quick interaction between the two), and is easy to use. |
| **22/7/19** | Completed feasibility report, detailing the potential use of the program and, in particular, weighing up whether it is worth making in the first place. Concluded that it would be well worth the time spent, especially given that neither the developer nor the client would need to spend any money on the project. The pros and cons were considered whilst communicating with the client about how often, for how long, and on what kind of hardware they would be using the finances manager software. |
| **25/7/19** | Continued researching suitable programming languages and settled on Python and the built-in GUI module “tkinter”, given prior experience with the language and its simplicity relative to other programming language.  Talked with the client and made sure that they were fine with the look of several programs made with tkinter, so that they would likely be fine with the final product made for them. |
| **31/7/19** | Discussed the categories that should be included in the program, and settled on a list of ones that encapsulate all possible expenditure. Thought of a number of things that one could spend money on to make sure that all bases were covered.  Completed IPO diagram, which can be referred to later on as it is the simplest form of the program, showing only the rudimentary relationships between input, process(es), and output. |
| **1/8/19** | Completed data flow diagrams, which showed the flow of data of the program in different levels of complexity.  Completed structure chart, giving a more hierarchical view of the program. It shows what modules underpin what other modules, as well as the order in which modules are called.  Completed system flowcharts, giving a detailed look at the flow of data, without a focus on the order in which the many modules are carried out. Additionally, they show the decisions that are made, controlling what events happen when. |
| **24/8/19** | Completed pseudocode, both for the overarching program, and each module individually. Underlined module names so it is obvious when they are being called.  Completed a data dictionary, showing the name, type, size, purpose, and valid characters of each data item. It ended up being fairly small due to the heavy use of GUI widgets over more traditional data items. |
| **26/8/19** | Completed the storyboard i.e. rough drawings/designs of the windows and the sequence in which they occur, so that there is a definitive point of reference when designing the actual GUI with tkinter. |
| **30/8/19** | Tested the software in relation to other software, that is, having a number of other programs installed and/or open. Focused on the client’s computer and their software, and didn’t run into any issues.  Tested how well the software ran on different computers in order to test its hardware compatibility. Focused on the client’s hardware, and found no issues. |
| **31/8/19** | Tested the program with regard to personnel, that is, with a number of users not involved in the project and who had different levels of computer skills. |
| **1/9/19** | Performed a desk check and recorded it in the documentation. Used both valid data (such as 12/12/12 for date) and invalid data (such as abc for cost). All the valid data was accepted by the program and its database, and all valid cost data was able to be used to correctly calculate the total. Invalid data did not stop the program from running, but did, in the case of cost, stop the total from being calculated. |
| **6/9/19** | Added source code to documentation as per the requirements of the project.  Created user manual, detailing how the program is used, as well as how it should be moved to other files/directories, should the user have a need to do so.  Client expressed satisfaction with end product, and confirmed they would make consistent use of it going forward. |

**SOFTWARE DESIGN REPORT – GANTT CHART**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15/7 | 20/7 | 22/7 | 24/7 | 26/7 | 28/7 | 30/7 | 1/8 | 3/8 | 5/8 | 7/8 | 9/8 | 11/8 | 13/8 | 15/8 | 17/8 | 19/8 | 21/8 | 23/8 | 25/8 | 27/8 | 29/8 | 31/8 |
| Defining and Understanding the problem  Identify a problem/valid need |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Generate ideas to solve need and complete project analysis |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Research possible programming tools |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| IPO chart |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Data flow diagram |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Structure chart |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| System flowcharts |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Data dictionary |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Pseudocode |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Implementing  Design GUI (storyboard) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Include common modules where possible |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Coding solution |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Systematic removal of errors |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Testing and Evaluating  Hardware test |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Software test |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Desk check(s) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Personnel and procedural |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Maintaining  Check all needs have been met with client |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Licence agreement |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Marketing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| User manual |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

A close up of text on a white background

Description automatically generatedA close up of text on a whiteboard

Description automatically generatedA close up of text on a whiteboard

Description automatically generatedA close up of text on a whiteboard

Description automatically generatedA close up of text on a white background

Description automatically generatedA close up of text on a whiteboard

Description automatically generated