YEAR 12 – 13
COMPUTER SCIENCE SIL

Design section of NEA report

newcollege
Bradford

Objectives:

- Decompose the problem
- Explain the structure of the solution
- Design a solution to each part of the problem
- Use algorithms in the design of the solution
- Create suitable user interface designs
- List usability features for the solution
- Identify key variables, data structures, classes and validation
- Develop test plans for each part of the development
- Justify all decisions in the design process

Help:

- Use the walkthrough book on teams **Files -> Class Materials -> AO3 – Programming Project -> "Tackling A-Level Projects OCR COM SCI."** The book goes through each section about what to do with tips and examples.
- Look at the **exemplar work** on teams of moderated work. **Files -> Class Materials -> AO3 – Programming Project -> "Exemplar Work."**
- Use the feedback excel sheet already given to you on OneDrive to help you understand what you need to do to get to the higher grade boundaries in each section.
- Email/message on teams Alex Wooding.

Decompose the Problem

You need to break down your problem in a logical way to show you understand how to tackle the project itself. This is to demonstrate that you understand how each aspect of the project works along side others using a 'decomposition diagram/s'. Another way that you can do this if you are struggling is by making a flowchart/s showing how you want the finished product to work. (You can hand draw these but if it isn't easy to read then you can lose marks easily - I would recommend making it on the PC)

You must also write a justification paragraph as to why you have split each of the tasks/objects/entities in the way you have to prove you are making a logical and justified conclusion going forward with the project. Link back to why this computational method is helping you to solve the problems in front of you. You can also have multiple versions of the same diagrams, this shows thought, planning and care being taken and provided you explain your reasoning for the changes will enable you to get the higher marks.

Types of diagrams you can use:

Please use the book for examples of each type to guide you.

- Decomposition diagrams
- Class diagrams
- Data flow diagrams
- Entity relationship diagrams (E-R Diagrams)

Think back to our **Procedural & Object-Oriented** programming lessons to help decompose and link back to theory content, this will help in your justifications. Look for the lessons on Teams and OneNote if you have forgotten how to implement this for your project.

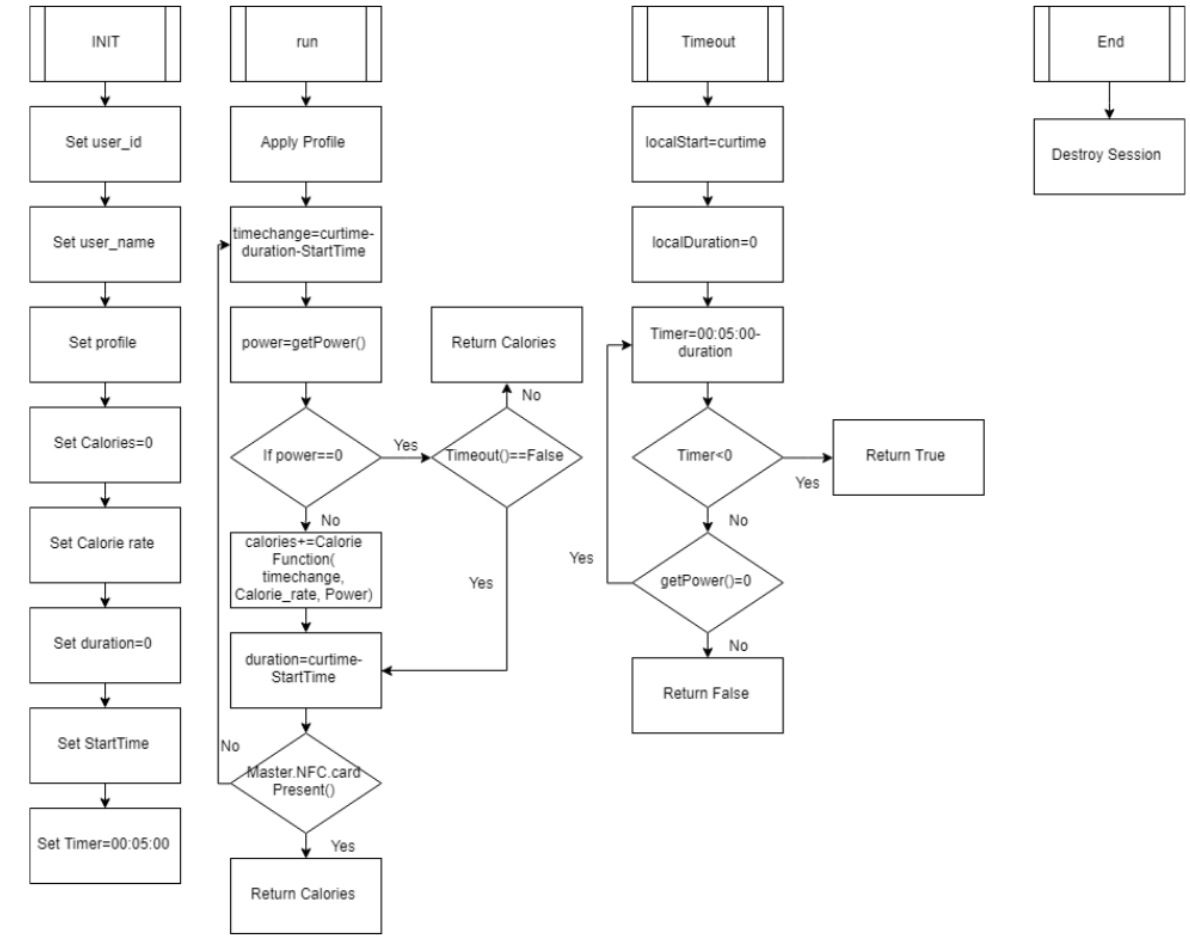
Procedural programming you're using: Variables, procedures & functions.

Object-Oriented you're using: Attributes & methods.

If you have/or need to do research into any technologies i.e Databases, Object-Oriented programming or suitable languages for your project then I suggest writing about it here in a similar way to the research conducted in the analysis. You can refer to this research when in the 'variables/data structures/validation' part of the section also.

Example '**Decompose the problem**', '**Algorithm**' & '**Variables/Data Structures/Validation**' for the same project so you can see how they link together.

Session Object: Flowchart Design



Describe the solution

Algorithms

You must describe the solution for your project code. Depending on the complexity of your project this can be quite extensive and if it is mentioned in your diagrams in the previous section, you need to be at least attempting to implement a basic solution here.

This means you must use the diagrams/flowcharts you made earlier to inform how your code will work at a very basic level using pseudocode. The better your 'picture' of how you want it to look now is the more aware you are of the tasks/challenges ahead of you and the more prepared you will be. I have placed some examples below but suggest you look at the examples on Teams to give you a better idea of the level of detail required of the top level bands.

Example '**Decompose the problem**', '**Algorithm**' & '**Variables/Data Structures/Validation**' for the same project so you can see how they link together.

Session Controller: Pseudocode Design

Pseudo.txt - Notepad

File Edit Format View Help

```
CLASS Session() //session class
    String uid //user id
    String user_name
    String profile
    Real calories
    Real rate //calorie burn rate in cal/s
    TimeStamp duration
    TimeStamp startTime
    TimeStamp timer
    Controller master



    Session(c_master, name, user_id, user_profile,) //constructor, master is the top gym machine controller object
                                                    //master is needed to access functions like getPower()
        user_name=name
        uid=user_id
        profile=user_profile
        calories=0
        rate=c_master.rate
        duration="00:00:00.0000"
        startTime="00:00:00.0000"
        timer="00:05:00.0000"
        master=c_master //set master controller
```

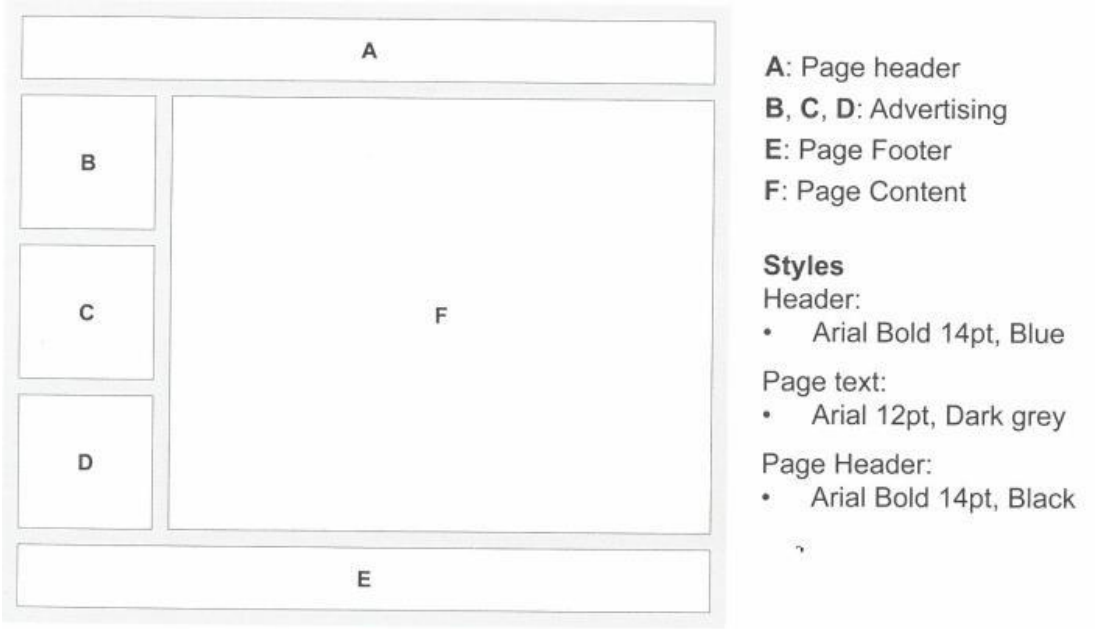
```
    run() //run a session
        startTime=getCurrentTime() //set the start time
        master.lcd.writeNew("Session Begun")    //update lcd
        noCards=TRUE    //assume no cards to start loop
        WHILE noCards //while there are no nfc cards
            timeChange=seconds(getCurrentTime()-duration-startTime) //the interval since the last measurement
            power=master.getPower() //get power
            IF power!=0 THEN  //check power is not zero
                calories+=CalorieFunction(timeChange,power) //increment the calorie count based on a function I will determine
            ELSE
                IF timeout() THEN //see if the machine times out due to inactivity, if so...
                    RETURN (calories, duration, startTime) //then return the session data
                ENDIF
            ENDIF
            duration=geCurrentTime-startTime  //update the duration
            noCards=NOT(!master.nfc.cardPresent()) //check if there are any cards
        ENDWHILE
        RETURN (calories, duration, startTime)  //return data if a card is present (loop broken)

    timeout() //timeout function
        localStart=getCurrentTime() //set timer start time
        o_timer=timer
        WHILE TRUE
            timer=0_timer-(getCurrentTime()-localStart) //update timer
            master.lcd.writeNew(timer+" until you are logged out")
            IF timer<0 THEN //If timer passed zero
                RETURN TRUE //then it has timed out
            ELSE IF master.getPower()!=0 THEN //if power is no longer zero...
                timer=o_timer
                RETURN FALSE //say it has not timed out
            ENDIF
        ENDWHILE

    end() //end session
        self.destroy() //destroy object
```

This is the pseudocode Session class. Much like with the Controller class, all the variables are now listed. If any more are needed in the Python program, it will be due to the syntax of python or any library methods. That is one of the great benefits of pseudocode over flowcharts, it clearly outlines the data flow of the program. Below is another list of variables:

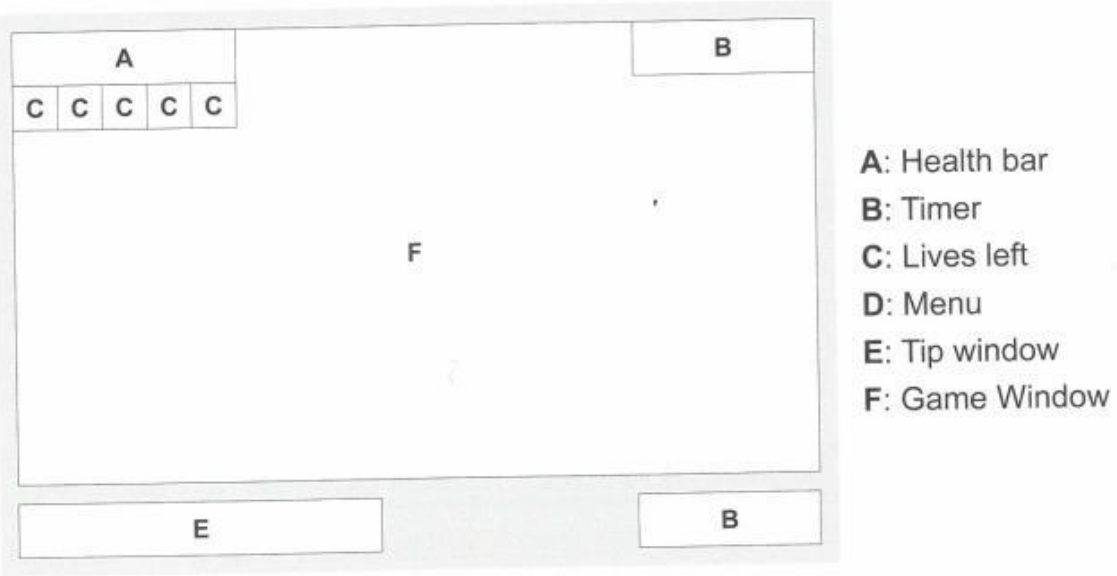
User Interface Designs

Below are examples of a bare minimum to include on your User Interface Designs. This should not be high quality and needs to show more of a skeleton view of the overall ideas that you wish to put in place. You get marks of the content not the overall aesthetics of it at this point. You must ensure that each of the diagrams you make are well thought out and explained in a paragraph linking back to the research and stakeholders constantly. You should make a few diagrams based on the nature of your

project. E.g. A game would have a home screen, a game view screen and possibly a settings screen.



Sample GUI layout for web page



Sample GUI layout for web page

Usability Features

You must discuss the usability features you're going to put in place to ensure a good user experience.
You need to answer (without directly answering these questions in the report, i.e. Don't copy and paste them in then answer them):

1. **What makes your program accessible to the user?**
2. **What allows them to use your program more easily?**

3. **What have you found out in your research which will help a user understand, navigate and control your program?**

You should link back to your user interface designs and justify why you made decisions on layout/button size etc. If you are unsure of what specifically to write, think about how people with a disability can use software already available in the public domain and see their 'usability features' that have been put in place to make their lives easier using the software.
E.g.

- Text to speech options
- Ability to enter data again without the program crashing
- Colours and fonts to help highlight core information
- Layout of the GUI at certain points (Enlarged areas/boxes where data needs to be entered)
- Location of menus and core content
- Ideas to help specific users (Disabilities)

If you want some more ideas about accessibility please watch this video:
https://www.youtube.com/watch?v=IkQk8ZbToNo

Variables/Data Structures/Validation

1. **Variables**: You must think carefully about the variables that you are intending to use. You will have already thought about them in the section '**Algorithms**' above. This is the opportunity to explain the '**Name**', '**Data Type**' and '**Use**' for each of them.

Example '**Decompose the problem**', '**Algorithm**' & '**Variables/Data Structures/Validation**' for the same project so you can see how they link together.

Variable	Type	Use
uid	Integer	Holds the user id.
user_name	String	User name
profile	String	Holds the difficulty profile
calories	Real	Holds the number of calories burned in the session.
rate	Real	Rate of calories burnt in cal/s
duration	TimeStamp	Duration of the session
startTime	TimesStamp	Session start time
master	Controller Object	References the controller object so that the session controller can access the NFC reader and LCD.
nocards	Boolean	Card Present True, No Card False
timeChange	Real	Time in a real amount of seconds since the last reading.
power	Real	Current power reading
localStart	TimeStamp	When the timeout began
o_timer	TimeStamp	Initial timer value

This links to the following criteria:

- Machine can apply a difficulty setting
- Machine can count calories burnt

2. **Data Structures**: This can consist of any **Databases, Stacks, Queues, Graphs, Lists, Tuples and Hash Tables** you wish to or have considered incorporating to the project. If you have done some research in any of these technologies I would suggest linking back to it to justify your decisions going forward. E.g. When determining the next player in the game, you can talk about the ability to use enqueue and dequeue to move them appropriately.
   If it makes more sense to add an E-R Diagram with the database content here, then that is fine to do so.

Example '**Decompose the problem**', '**Algorithm**' & '**Variables/Data Structures/Validation**' for the same project so you can see how they link together.

Database

I first decided to make a list of all the fields that I would need and justify them to ensure I had a minimal number:

Field	Reason
user_id	Every user must be uniquely identified in some way. As users can have the same names, user_id is more appropriate. Moreover, the id will be of the same format and one field instead of two.
nfc_id	An nfc_id will be associated with every user, linking their account to a physical NFC card. The card cannot be made the user_id as a card may be damaged or broken. They also may change cards and Mifare 1K cards cannot have their ID changed.
name	The name of the user is needed for personalisation and would be needed for sending letters etc.
address	This is needed in case a letter were to be sent etc.
telephone	This would be needed for communication.
email	This too would be needed for communication.
user_active	This would be a flag for whether the account is active or not.
machine_id	This would uniquely identify every machine.
machine_type	Gives the type of machine e.g. stepper, rower
machine_model	The specific model e.g. Robson Rower III
machine_muscles	This would contain the calorie rates for each muscle for each model of machine.
machine_serial	This would be a physical number associated with the machine as opposed to machine_id (software)
session_id	Uniquely identify each session
session_date	The date and time when the session took place.
session_data	The field will contain the calorie burns for the muscles for the session.
profile	Difficulty profile for the user on a machine type.

If all these fields were in a flat file database, it would be in zeroth normal form. This form is not useful however. Every time a session is created, all the user data and machine data would be repeated. This would be prone to errors due to mistakes in repeated data reducing integrity. However, the law states that data must be accurate and kept up to date (Data Protection Act 1998), thus it would be in the gym's best interests to normalise the database. Normalisation will reduce redundancy too as well as file size.

I could also include fields relating to medical and health information about the users. This of course could be useful as a trainer would need to take such information into consideration when suggesting new profiles to avoid their client undergoing injury or harm. However, the trainer will likely have records of the client's health information in some other form anyway and thus does not need to be included in the database. Another consideration is BMI score and height etc. These could be stored over time to create a plot of how a client's BMI score has changed. This could be useful as it would serve as a measure of progress. However, due to my resources and time, I will not include this feature, though in future maintenance it could be added.

1NF

Data must be atomic and related attributes must be together. All tables must have a primary key:

users	**machines**	**sessions**	**profiles**
user_id	machine_id	session_id	user_id
first_name	machine_type	session_data	machine_type
middle_names	machine_model	session_date	profile
last_name	machine_serial	duration	
house_no	machine_muscles	user_id	
street		machine_id	
city			
postcode			
telephone			
email			
nfc_id			
active			

Above I have split down fields like address and name to make them atomic. I have separated the data into four tables because they were the four logical groups that stood out. All primary/composite keys are underlined. Now the database must be put into 2NF.

3. **Validation:** You must ensure that your solution has some inbuilt validation to ensure that bugs, errors or general issues do not occur. You will need to explain how and why you are putting these measures in place. If you have not already talked about this while making your pseudocode you must reiterate your design ideas and reasoning behind doing this. E.g. A user inputs an impossible number such as a float that your software cannot handle. You code must have a way of ensuring that incorrect input from users would not cause the software to stop, crash or output an incorrect result.

Example '**Decompose the problem**', '**Algorithm**' & '**Variables/Data Structures/Validation**' for the same project so you can see how they link together.

Validation

This component of the overall system has a large range of inputs and each one must be validated. I will cover each input type and then explain the validating methods that I will use.

Buttons:

Buttons, of course, need little validation as there is no variable input for buttons. The user either clicks, or they don't and that is managed automatically. What must be validated, however, is that when the button is pressed, the correct event needs to occur with relation to that button.

Radio:

Radio buttons again, need little validation as the only possible inputs are those the radio has as options. I must, however, ensure that the mutual exclusivity is implemented. It would not be desirable if multiple inputs could be selected simultaneously. As they would overwrite each other, potentially confusing the user.

Slider:

Sliders also need little validation as they have a specified range and the user can only drag the bar between it. To ensure errors do not occur, I need to set the range to match that which my functions are expecting. I must then validate what the slider returns to ensure it is of the correct data type.

Entries:

Entries are where the user will type something. This does require validation. For example, user IDs are integers and therefore the input must be validated to ensure it is an integer. I need to ensure that no 'illegal' characters are entered such as ';' or ':' as I use them in my data formatting functions and such characters would corrupt the data.

Drop-downs:

Drop downs are effectively buttons and require no validation other than that the correct variables are modified and are assigned 'legal' values.

Describe the approach to testing

In-Development test plan

The purpose of this task is to ensure that you are using 'iterative testing' for everything that you make. This shows that if you have tried and tested all aspects of the code before moving onto the next decomposed task. This uses 'White-box testing as it is based on the code that is written for individual parts of the algorithm or sections.

What needs to be tested?

- **Syntax** errors – Normally detected by the IDE – **not** needed.
- Logical errors – Logical paths, e.g. Boolean expressions have been used – needed.
- **Runtime** errors – Unexpected events happen, e.g. dividing by zero error which hasn't been considered – needed.
- **Validation** –
    - **Normal** – test data should be accepted by validation rules without causing errors.
    - **Boundary** – test data check the max and min values that can be entered. The data type should also be correct/
    - **Invalid** – test data is of the correct data type, but just outside of the validation rules.
    - **Erroneous** – test data should not be accepted. It will be of the wrong data type.
- **Output** – Results show where errors may exist. It is possible that each function or class you test works individually but causes errors when linked together.

This is an example of how you would test your user interface but testing should also be conducted on code to include the potential errors listed above.

Testing

This component will also require extensive testing. I will perform the usual tests on all the algorithms.

What is different about testing this component, however, is the UI aspect. I will again require my stakeholders to test the UI extensively and then give any feedback they might have so that I can improve the system. I will also request that they try to cause an error so that I can reduce the risk of the system failing.

I myself will also perform tests on every widget to ensure that its functionality is correct. Such a test table could look like this:

Widget	Event	Expected Outcome	Outcome	Action
Add Button	Click	Update designed profile		
Menu Button	Click	Display drop down menu		

This table will allow me to document all my tests and record the action which I need to take.

The examples of testing tables are not from the example project used throughout the rest of this document. This is an example of it done particularly well in another project.

Testing data

I will be testing my program using tables to make sure all aspects of the program are functioning accordingly to the requirements.

Function/Method being tested	Scenario	Input (optional)	Expected results
Player movement	In game	up	player will move up, and play the walk up animation
		down	player will move down, and play the walk down animation
		left	player will move left, and play the walk left animation
		right	player will move right, and play the walk right animation

Player attack	In game	space	player will attack
Player damage(taken)	In game	-	players health will reduce according to the damage done by the enemy
Player death	In game	-	game will end
Enemy movement	in game	-	enemy will move towards the position of the player, is animation will move respectively to the left or right in the direction of movement, and colliders will be flipped for each direction
Enemy attack	In game	-	enemy will attack every a set amount of time
Enemy damage(taken)	In game	-	enemy's health will be reduced by the player's damage
Enemy death	In game	-	points will be awarded to player and the enemy object will be destroyed
Camera movement	In game	-	camera will move towards the player, as to follow it. With the stationary point of the player in the middle
Game controller	In game	-	x enemy will be spawned after all are destroyed x+1 are spawned, Initial delay of "initialwait" and delay between waves of "wavewait"

Scene	Action	Expected results
Main menu	Play button	load game scene
	Exit button	application closes
	Music toggle	music is muted/unmuted
Paused menu	Resume button	un-pauses the game
	Main menu button	load main menu scene
	Exit button	application closes
	Music toggle	music is muted/unmuted
	SFX toggle	SFX is muted/unmuted
Game over	press enter	load main menu scene

Post-Development test plan

This is a form of 'Black-box testing' that is carried out when the entire solution is complete. Good test plans are designed to try to make the system fail as this would closer resemble what would happen in the real world when people are using the finished software.

This section should focus on what you will do to focus on qualitative testing, i.e. the speed, feel and accessibility of the software. You need to write a few paragraphs

about how you have considered this type of testing and what this might entail for your project specifically.

Feedback from the stakeholder should include:

- What worked
- What errors there were
- How these errors impacted the stakeholder
- What areas for improvement they noticed
- What their overall feel for the success of the system was

Example '**Decompose the problem**', '**Algorithm**' & '**Variables/Data Structures/Validation**' for the same project so you can see how they link together.

Post-Development Plan

When I finish the development stage, testing and evaluation will follow, however, it would be good to have a plan in place and a procedure to follow when I get there.

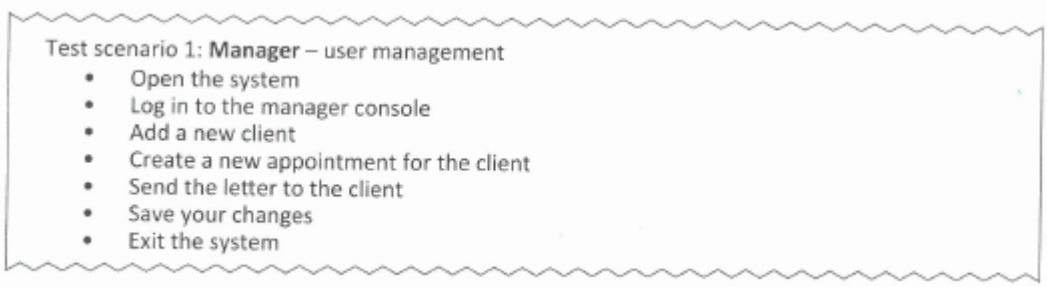
My first step would be to get stakeholder feedback as there would be no point in carrying out an evaluation if there are changes to be made as required by the user which would then require a re-evaluation. Upon receiving the user feedback I will then consider each response in turn looking for potential changes and determining whether it is realistic or not. After this stage, I would then make all changes that I had decided on and move onto the full evaluation stage.

I would then evaluate each phase: analysis, design and development and testing with a particular focus on development and testing and the final product functionality.

Post Development Test Plan

The testing of the final product would involve looking at each success criteria and user requirement in turn and testing to see if it has been met. Then I would perform destructive testing on the program where I will try to break it and see how easy it is to make it crash. I would then identify and explain the system's usability features. The testing performed in the post development phase will mostly be black box. This means that I will perform a specific action or give a specific input and see what the output is and whether it is correct. Exactly how the output was achieved does not matter. This is what part of the destructive testing will be. If when trying to break the system all the correct outputs are given, I will assume it is functional. That would be a black box test.

Finally, I would consider maintenance and the overall limitations of the system.


Test scenario 1: **Manager** – user management

- Open the system
- Log in to the manager console
- Add a new client
- Create a new appointment for the client
- Send the letter to the client
- Save your changes
- Exit the system

An example post-development test scenario for a manager of the system

Extra beneficial work to your learning:

There are a number of practice exam papers available on Teams for you as well as hard copies in class. If you would prefer a hard copy printed out, please ask for one.

Otherwise they can be found on teams: **Class Materials -> Past Papers -> AS**