CO7095 Software Measurement and Quality Assurance
Assignment 1

Student Number: 139010381

Task 1: Project Planning

Question 1

The following is a list of activities that need to be carried out in order to produce the system as specified by the client:

LetsMove Web and Mobile App – Estate Agents and Buyers

1) **Business Analysis** – Determine what the business provides to their client base and obtain data such as property types, property locations, prices, client information etc. as this information will be analysed and entered into the database.
2) **Design GUI's** – For each web app page, login (also used on mobile app), main property listings (also used on mobile app), editing existing listings, uploading new listings and booking schedule (also used on mobile app) pages. For the mobile app pages include registration, notifications, searching, property details (also used on web app) and booking fee payment. A database GUI will also be created for use on both the web and mobile app.
3) **Client Registration** – New clients can create a username and password for their account in order to use the app and provide their preferences and income and employment documentation. Once completed they will be redirected to an external credit score check by companies such as Experian. Clients can change their preferences and other registration information on this page.
4) **Login Functionality** – Both estate agents and clients will be able to use their employee IT or newly created login credentials to access either the web or mobile app in each case.
5) **Main Property Listings** – Estate agents and clients can view the full list of existing properties with a picture and overview of information for each. From here estate agents and clients can click each listing for more information.
6) **Property Details** – Gives all information regarding the property and its pictures, from here estate agents can edit the information and clients can select book a viewing to be taken to the booking availability page.
7) **Editing Existing Listings** – Changing existing property listing information, removing pictures and existing properties from the listings.
8) **Uploading New Listings** – Upload new property listings and pictures to new and existing properties, buyers will be notified of this if it relates to their interests.
9) **Editing Client Preferences** – Clients can change their property preferences such as number of rooms, location, price, property type, etc. This can be done from the client registration page.
10) **Client Notifications** – Clients will be notified on this page on their mobile device of any new properties uploaded by estate agents on the web app that meet the client's desired preferences.
11) **Booking Schedule** – Clients and estate agents can view the schedule of bookings from this page, the schedule is printed from here and clients are directed here when adding a booking.
12) **Adding New Bookings to the Schedule** – The viewing bookings schedule can be seen by the estate agents and clients, new bookings can be created by clients on the mobile app through selection of a time and date and adding client information. The client is then directed to a booking fee page.

13) **Printing a Viewing Schedule** – Estate agents can print a daily schedule of the viewings.
14) **Booking Fee Payment** – Clients will be required to use any of the widely accepted credit and debit cards to pay a booking fee once the booking has been finalised.
15) **Searching Functionality** – Clients can search for their desired property to narrow the number of listings displayed based on details such as location, price, property type, number of rooms and other building facilities (e.g. properties with gardens). Searches based on saved, favourite and ignored listings can also be carried out.
16) **Tagging Property Listings** – Clients can tag listings as saved, favourite (allowing the property to be found quickly) or ignored (so the property will not be included in future searches).
17) **Editing Offer Status** – When offers are received from buyers (e.g. through discussion at a booked viewing or other contact), property listings can be marked with an "offer received", "sold subject to contract", "sold" or "available" status which can be viewed by estate agents.
18) **Database Creation** – A database system will be created and prepared for the database backup process.
19) **Database Backup** – All client and properties details in addition to changes made to the property listings and updates to the scheduled viewings and property lifecycles (transitions from available, sold subject to contract and sold, etc.) will be tracked and stored on the database. The database will be backed up periodically.
20) **Database Security** – All database data will be encrypted to conform to client data protection requirements.
21) **Testing** – Each part of the system will be thoroughly tested with sets of acceptable and unacceptable data to ensure the system is reliable with minimal faults. This will help to ensure only short periods of down time in the rare occurrence of a failure.

Question 2

The following is a crude estimation of the number of thousands of lines of code (KLOC) for each of the activities above that are involved in the development of the system.

Table 1: Table of activities and their respective number of lines of code, the total number of lines of code for the system was also calculated.

Activity	Thousand Lines of Code (KLOC)
Business Analysis Functionality	0.5
Design GUI's	3.3 (0.3KLOC per page GUI, 11 pages in total)
Client Registration	0.5
Login Functionality	0.2
Main Property Listings	0.4
Property Details	0.2
Editing Existing Listings	0.5
Uploading New Listings	0.4
Editing Client Preferences	0.2
Client Notifications	0.3
Booking Schedule	0.2
Adding New Bookings to the Schedule	0.3
Printing a Viewing Schedule	0.1
Booking Fee Payment	0.3
Searching Functionality	0.5
Tagging Property Listings	0.3

Editing Offer Status	0.3
Database Creation	0.3
Database Backup	0.5
Database Security	0.4
Testing	9.7 (1:1 ratio of code to test code for 20 activities)
	Total: 19.4

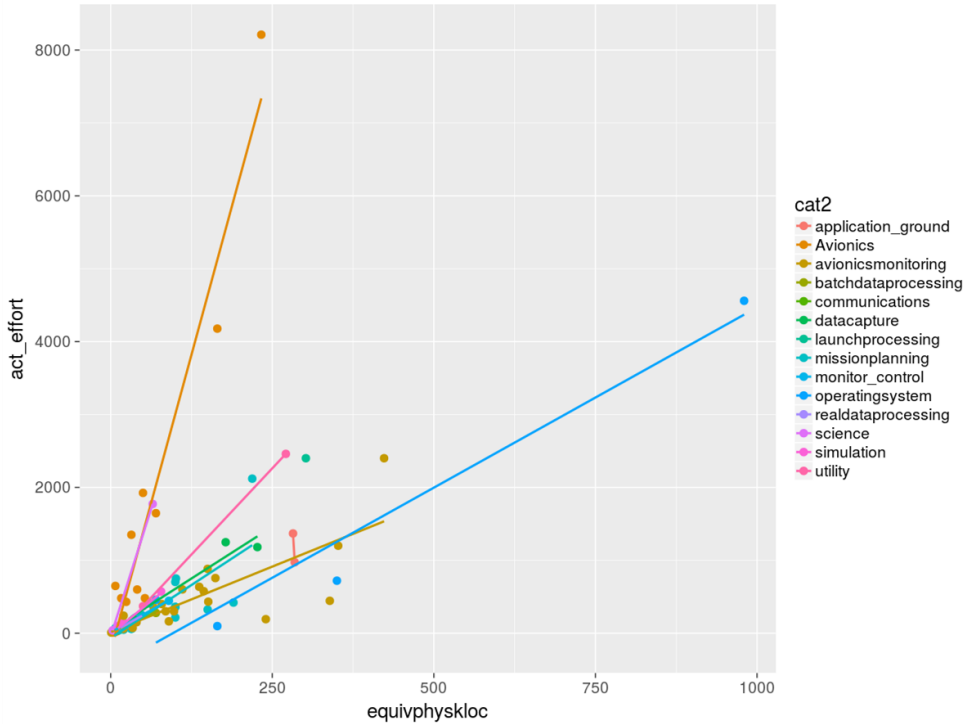
Question 3

Using the NASA93 historical data and computing a linear regression model (Figure 1) for effort and KLOC the cost per KLOC (gradient a) and initial cost (intercept c) were determined. The effort for each of the activities (where "s" is the number of lines of code in LOC) was calculated using:

$$\boldsymbol{E = a \times s + c}$$

The results were as follows:

Figure 1: The plotted linear regression model for effort and KLOC from different system categories.



The gradient and intercept were found for the "datacapture" category which was selected to match the LetsMove system as it is one of the smaller systems in the NASA93 data set. The "datacapture" category also has a mid-range gradient and so a more reasonable cost per line of code for a web/mobile app compared to complex scientific or simulation systems. The linear regression fit closely represents the data points in comparison to some of the other categories and if the number of KLOC for LetsMove were scaled up ($19.4 \times 10 = 194KLOC$) this would more closely match the "datacapture" 225 KLOC so the systems are of a similar size (31 KLOC difference).

For the "datacapture" category the intercept (initial cost c = 0) and the gradient (cost per line of code a = 6.0). Gradient was calculated using (change in y/change in x = 1350/225 = 6.0).

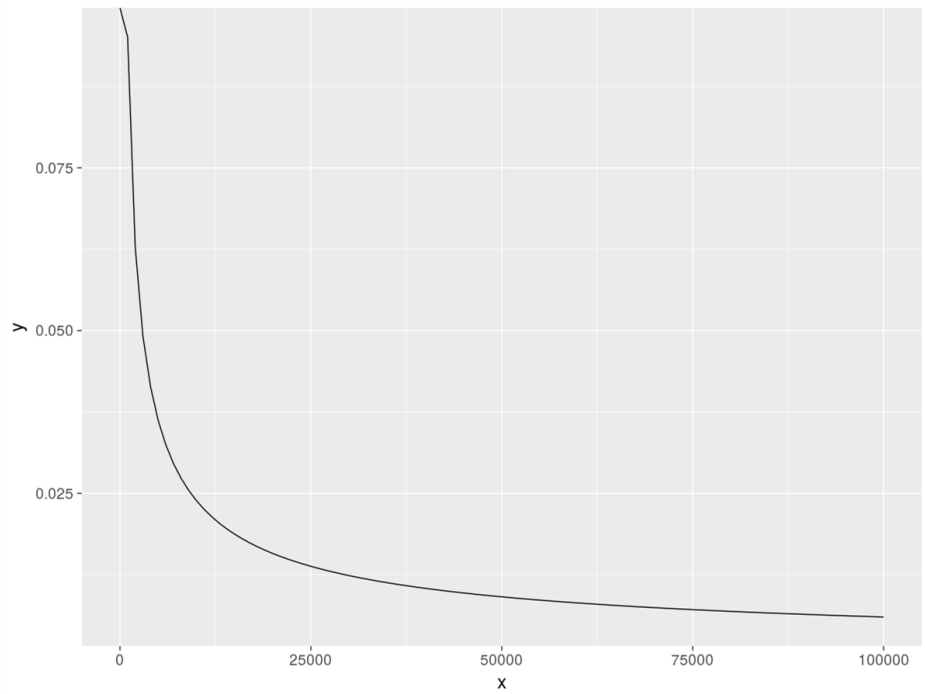
Table 2: Estimation of effort for each activity (requirement) using the KLOC estimates from Table 1, initial cost and cost per LOC from Figure 1 and the effort equation $E = a \times s + c$. Initial cost (c = 0) is applied and therefore the equation $E = a \times s$ is used for each individual activity.

Activity	Effort Estimation
Business Analysis Functionality	3000
Design GUI's	19800
Client Registration	3000
Login Functionality	1200
Main Property Listings	2400
Property Details	1200
Editing Existing Listings	3000
Uploading New Listings	2400
Editing Client Preferences	1200
Client Notifications	1800
Booking Schedule	1200
Adding New Bookings to the Schedule	1800
Printing a Viewing Schedule	600
Booking Fee Payment	1800
Searching Functionality	3000
Tagging Property Listings	1800
Editing Offer Status	1800
Database Creation	1800
Database Backup	3000
Database Security	2400
Testing	58200
	Total: 116400

Question 4

As the development costs will decrease as the size of the system increases, different scale factor values (variable b) were tested with the NASA93 historical data to determine which values of b provide a better approximation or the "normal" predicted effort for each activity. For decreasing cost (effort) as size of the system (KLOC) increases the scale factor is b < 1. With a straight line produced at b = 1 and effort increasing with KLOC when b is positive, then b must be negative.

Figure 2: The following graph shows the plot for scale factor b = -0.6 for the NASA93 data (where the x and y axes are KLOC and effort respectively). The scale factor is used to calculate the "normal" predicted effort for each activity in Table 3 below.



The above plot in Figure 2 shows a reasonable (not extreme) decrease in effort cost as the size of the system in KLOC increases for the chosen "datacapture" category (approximately 225 KLOC). Therefore it is a good estimation for the LetsMove system. The "normal" predicted effort for each activity in Table 3 was calculated using:

$$\boldsymbol{E = a \times s^{b} + c}$$

Table 3: The following table contains the "normal" predicted effort values for each activity with scale factor b = -0.6. As the initial cost is zero the equation used is simply $E = a \times s^{b}$. The table shows clearly that for larger activities the effort cost is lower.

Activity	"Normal" Predicted Effort
Business Analysis Functionality	0.14
Design GUI's	0.05
Client Registration	0.14
Login Functionality	0.25
Main Property Listings	0.16
Property Details	0.25
Editing Existing Listings	0.14
Uploading New Listings	0.16
Editing Client Preferences	0.25
Client Notifications	0.20
Booking Schedule	0.25
Adding New Bookings to the Schedule	0.20
Printing a Viewing Schedule	0.38
Booking Fee Payment	0.20

Searching Functionality	0.14
Tagging Property Listings	0.20
Editing Offer Status	0.20
Database Creation	0.20
Database Backup	0.14
Database Security	0.16
Testing	0.02
	Total: 3.83

Question 5

The following table (Table 4) was created from the normal time estimates taken from Table 3, in each case the estimates were multiplied by 10 to scale the values to be more readable in the PERT chart (expected estimates with only two decimal places). For each activity the optimistic and pessimistic time estimates were added. The expected estimates were then calculated using the equation:

$$\frac{(\boldsymbol{Optimistic}+(\mathbf{4}\times\boldsymbol{Normal})+\boldsymbol{Pessimistic}}{\mathbf{6}}$$

Table 4: This table provides the time estimates for each activity given in months.

Activity	Time Estimates (months)			Expected (months)
	Optimistic	Normal	Pessimistic	
A: Business Analysis Functionality	1.2	1.4	1.9	1.45
B: Design GUI's	0.3	0.5	0.9	0.53
C: Client Registration	1.0	1.4	1.8	1.40
D: Login Functionality	2.0	2.5	2.9	2.48
E: Main Property Listings	1.3	1.6	1.9	1.60
F: Property Details	2.1	2.5	2.8	2.48
G: Editing Existing Listings	1.1	1.4	1.7	1.40
H: Uploading New Listings	1.2	1.6	1.8	1.57
I: Editing Client Preferences	2.3	2.5	2.8	2.52
J: Client Notifications	1.8	2.0	2.3	2.02
K: Booking Schedule	2.2	2.5	2.7	2.48
L: Adding New Bookings to the Schedule	1.6	2.0	2.5	2.02
M: Printing a Viewing Schedule	3.3	3.8	4.0	3.75
N: Booking Fee Payment	1.7	2.0	2.4	2.02
O: Searching Functionality	1.2	1.4	1.8	1.43
P: Tagging Property Listings	1.8	2.0	2.4	2.03

Q: Editing Offer Status	1.7	2.0	2.3	2.00
R: Database Creation	1.5	2.0	2.2	1.95
S: Database Backup	1.2	1.4	1.9	1.45
T: Database Security	1.4	1.6	2.0	1.63
U: Testing	0.2	0.3	0.8	0.37

Question 6

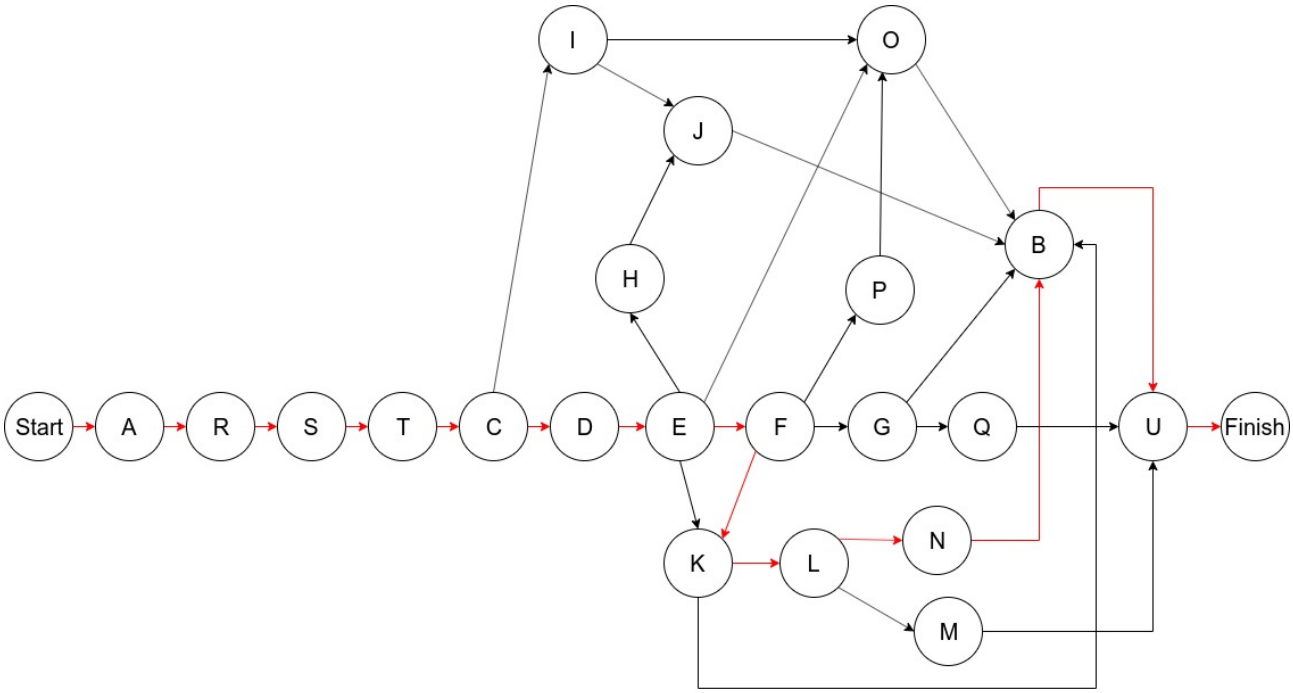
For each activity the dependencies were determined and a short justification was provided. Some assumptions are made in the cases of testing and GUI design such that testing is only carried out once every part of the system has been completed and GUI's for each page are only created once the functionality of each requirement is in place (to allow interface elements to function).

Table 5: This table describes the dependency relations (activities that must happen before a given activity).

Activity	Dependencies	Description
A: Business Analysis Functionality	-	Before the system is created data regarding the property and client information must be determined and analysed which will guide the creation of the database and the information entered into the database fields.
B: Design GUI's	**C, D, E, F, G, H, J, K, N, O, R**	Dependent on client registration, login, main property listings, editing and uploading new listings, property details, notifications, booking schedule, booking fee, searching and the database. As each page should be implemented for buttons etc. to function.
C: Client Registration	**T**	Once the database has been created and its functionality implemented this is the first point of entry for the mobile application.
D: Login Functionality	**C**	Used for web and mobile but is dependent on registration for the mobile app.
E: Main Property Listings	**D**	First page for web and mobile so requires correct login to access.
F: Property Details	**E**	Page for further details after property is selected on the main property listing page.
G: Editing Existing Listings	**F**	Dependent on property details as this is the page to be edited.
H: Uploading New Listings	**E**	Dependent on main property page as new listings are added there.
I: Editing Client Preferences	**C**	Dependent on client registration as clients select preferences as part of their registration and can edit them from this page.
J: Client Notifications	**H, I**	Dependent on client preferences and uploading new listings as clients are only notified of new listings that match preferences.

K: Booking Schedule	**F, E**	Dependent on property details page as clients select bookings from there and main properties page as estate agents can view the schedule from there.
L: Adding New Bookings to the Schedule	**K**	Dependent on the booking schedule page as clients select bookings from there.
M: Printing a Viewing Schedule	**K, L**	Dependent on new bookings being added to the schedule and the Schedule booking page as schedules are printed from there.
N: Booking Fee Payment	**L**	Dependent on a new booking being added to take the client to the booking fee page.
O: Searching Functionality	**E, I, P**	Dependent on editing preferences, the main property listings page and tagging properties as searches are carried out on preferences and tags and are displayed on the main properties page.
P: Tagging Property Listings	**F**	Dependent on the properties details page as this is where the property can be tagged.
Q: Editing Offer Status	**F, G**	Dependent on the properties details page and editing existing listings as offer information is visible on the details page.
R: Database Creation	**A**	The database will be created first in preparation for all other parts of the system being created and will allow data to be input as each activity is completed.
S: Database Backup	**R**	Dependent on database creation. When client registration, login, main property listings, property details, editing listings, uploading listings, editing preferences, booking schedule, new bookings, booking fee payment, tagging properties and offer status are implemented, all of these can be tracked and changes can be saved.
T: Database Security	**S**	Dependent on database backup so that data can be encrypted when it is entered.
U: Testing	**B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T**	Dependent on all other activities being completed as testing is completed for each individual activity.

Figure 3: The complete network diagram for all 21 activities including the highlighted critical path.



Question 7

A PERT Chart was constructed for the network diagram in Figure 3 but due to the complexity of the diagram all PERT Chart time constraint information was condensed into Table 6 below. The following calculations were used to identify each time constraint:

Earliest Start = Largest Earliest Finish Amongst Predecessors
Earliest Finish = Earliest Start + Expected Duration
Latest Finish = Minimum Latest Start of Succeeding Activities
Latest Start = Latest Finish – Expected Duration
Slack = Latest Start – Earliest Start

Table 6: PERT Chart in table format identifying time constraints for each activity node from A – U.

Node	Earliest Start	Expected Duration	Earliest Finish	Latest Start	Slack	Latest Finish	Predecessors
START	0	0	0	0	0	0	-
A	0	1.45	1.45	0	0	1.45	START
R	1.45	0.53	1.98	1.45	0	1.98	A
S	1.98	1.40	3.38	1.98	0	3.38	R
T	3.38	2.48	5.86	3.38	0	5.86	S
C	5.86	1.60	7.46	5.86	0	7.46	T
D	7.46	2.48	9.94	7.46	0	9.94	C
E	9.94	1.40	11.34	9.94	0	11.34	D
F	11.34	1.57	12.91	11.34	0	12.91	E
G	12.91	2.52	15.43	15.98	3.07	18.50	F

Q	15.43	2.02	17.45	18.50	3.07	20.52	G
I	7.46	2.48	9.94	14.39	6.93	16.87	C
H	11.34	2.02	13.36	14.85	3.51	16.87	E
P	12.91	3.75	16.66	13.71	0.8	17.46	F
J	13.36	2.02	15.38	16.87	3.51	18.89	I, H
O	16.66	1.43	18.09	17.46	0.8	18.89	P, E, I
K	12.91	2.03	14.94	12.91	0	14.94	E, F
L	14.94	2.00	16.94	14.94	0	16.94	K
N	16.94	1.95	18.89	16.94	0	18.89	L
M	16.94	1.45	18.39	19.07	2.13	20.52	L
B	18.89	1.63	20.52	18.89	0	20.52	O, J, G, N, K
U	20.52	0.37	20.89	20.52	0	20.89	B, Q, M
FINISH	20.89	0	20.89	20.89	0	20.89	U

Question 8

From Table 6 we can determine that the critical path of the development of the LetsMove system is the path through the system with zero slack for each activity. A point of zero slack indicates that the activity cannot be delayed without affecting the entire duration of the project. The critical path therefore follows the nodes:

START, A, R, S, T, C, D, E F, K, L, N, B, U, FINISH.

Other paths through the system have activities with slack > 0 indicating that they can be delayed by the specified amount of time and the project can still be completed in the given time frame.

Task 2: Black-box Testing

Question 1

A set of test inputs was identified to achieve Modified Condition Decision Coverage (MC/DC) for the validate method in the code provided in the brief, the overview of this can be seen in Table 7 below.

Table 7: The following table includes the branches covered for each set of test inputs, branches are identified by line number, the condition and the outcome.

Line Numbers	Condition	Test Inputs (Number, Type)	Condition/ Branch Changes Tested	Condition Elements					Condition Outcome
				A	B	C	D	E	
11, 12, 13	A != 16 || B<51 || C >55	5312345678910111, MASTERCARD	A (ELSE), B (ELSE), C (ELSE)	F	F	F	-	-	F
		5412345, MASTERCARD	A (IF)	T	F	F	-	-	T

		5012345678910111, MASTERCARD	B (IF)	F	T	F	-	-	T
		5612345678910111, MASTERCARD	C (IF)	F	F	T	-	-	T
19, 20	(A != 13 && B != 16) || C != 4	412345, VISA	A (IF), B (IF)	T	T	F	-	-	T
		4123456789101, VISA	A (ELSE)	F	T	F	-	-	F
		4123456789101112, VISA	B (ELSE), C (ELSE)	T	F	F	-	-	F
		3123456789101112, VISA	C (IF)	T	F	T	-	-	T
26, 27, 28	A != 15 || (B != 34 && C != 37)	3712345, AMEX	A (IF)	T	T	F	-	-	T
		3012345678910011, AMEX	B (IF), C (IF)	F	T	T	-	-	T
		3412345678910011, AMEX	B (ELSE)	F	F	T	-	-	F
		3712345678910011, AMEX	A (ELSE), C (ELSE)	F	T	F	-	-	F
34, 35	A != 16 || B != 6011	601112345, DISCOVER	A (IF)	T	F	-	-	-	T
		6011123456789101, DISCOVER	A (ELSE), B (ELSE)	F	F	-	-	-	F
		5013123456789101, DISCOVER	B (IF)	F	T	-	-	-	T
41, 42, 43, 44, 45	A != 14 || ((B != 36 && C != 38) && (D < 300 || E >305))	30212345678910, DINER	A (ELSE), D (ELSE), E (ELSE)	F	T	T	F	F	F
		30312345, DINER	A (IF)	T	T	T	F	F	T
		29012345678910, DINER	D (IF)	F	T	T	T	F	T
		30612345678910, DINER	C (IF), B(IF), E (IF)	F	T	T	F	T	T
		36012345678910, DINER	B (ELSE)	F	F	T	F	T	F
		38012345678910, DINER	C (ELSE)	F	T	F	F	T	F

Question 2

Using the Category Partition Method, a set of categories and constraints were found for a single independently testable feature, the validate method which successfully checks if a card is a valid instance of that type of card. The number of test cases was identified and the effect of constraints considered. A test case has also been provided.

Table 8: This table provides the categories, partitions and constraints for each input parameter and includes justification in each case.

Parameters	Categories	Partitions	Constraints	Justification
String (number)	Length	Zero	[Single] [Property Empty] [If Acceptable] [Property WrongLength]	Zero length should be tested once to determine if the entered parameter is empty. An empty parameter cannot be tested for content.
		Too Short	[Property NonEmpty] [If Acceptable] [Property WrongLength]	Length of the input parameter should be tested to determine if length is too short in the case of a non-empty acceptable type.
		Too Long	[Property NonEmpty] [If Acceptable] [Property WrongLength]	Length of the input parameter should be tested to determine if length is too long in the case of a non-empty acceptable type.
		Possible Length	[Property NonEmpty] [If Acceptable] [Property NotWrongLength]	A selection of possible parameter lengths would be correct for a given acceptable type.
	Content	Spaces	[If NonEmpty][If Acceptable] [Property WrongContent]	Input parameter could contain blank spaces in the event that a string is entered. Spaces should be considered wrong content.
		Special Characters	[If Non Empty][If Acceptable] [Property WrongContent]	Special characters such as “/”, “+”, “&” etc. should be considered as they could be used in conjunction with letters such as “/n” which denotes a new line in the event that a string parameter was provided. This is also considered to be incorrect content.
		Uppercase Letters	[If NonEmpty] [If Acceptable] [Property WrongContent]	Uppercase letters should be tested in the case of a string parameter but are still considered incorrect content.
		Lowercase Letters	[If NonEmpty] [If Acceptable] [Property WrongContent]	Lowercase letters should be tested in the case of a string parameter but are also considered incorrect content.
		Negative Numbers	[If NonEmpty] [If Acceptable] [Property WrongContent]	Negative numbers should be tested in the event that a string or int parameter is passed. Negative numbers are considered to be incorrect content.
		Positive Numbers	[If NonEmpty] [If Acceptable] [Property NotWrongContent]	Positive numbers should be tested in the event that a string or int parameter is passed. Positive numbers are considered to be correct content.
		Possible Content	[If NonEmpty] [If Acceptable] [Property NotWrongContent]	A selection of possible content for an input parameter is available depending on an acceptable type being provided.

Int (type)	Type	Acceptable Type	[Property Acceptable]	A selection of acceptable types are possible and each accepted type should be tested against length and content partitions.
		Non Acceptable Type	[Single] [Property NonAcceptable]	Non acceptable types should only be tested once as they should be discarded and not tested against other partitions.

Due to attempting to keep the categories and partitions generalised (not specific to the program provided in the assignment question) and the difficulty of defining partitions such as "possible length" without being able to specify a range (as this would be related to the program), the number of test frames may be much smaller than expected.
The number of test frames was calculated from Table 8 using the TSLgenerator to combine all of the possible variations. The results were as follows:

Total Test Frames (Without Constraints) = 56 test frames
Total Test Frames (With Constraints) = 23 test frames

By including the constraints this reduces the total test frames by 33 as many of the combinations are not tested when including constraints. For example [Single] ensures that a "Non Acceptable Type" or "Zero" length are only tested once and it would not be beneficial to combine these with other partitions. Partitions involving content and length are only tested for acceptable types and "Possible Length" and "Possible Content" are only tested in the event that content or length are not incorrect (content should be a number and length should not be too long or too short).

Below is a test case with a set of inputs and outputs for the Category Partition Method, any outputs that are not entered exactly as listed in Table 9 would not pass the test case.
Test case 22
type : acceptable type
length : possible length
content : positive numbers

Table 9: In the case of the validate program an example set of inputs and outputs is as follows.

Type Input	**Length Input**	**Content Input**	**Output**
MASTERCARD	16	First two digits between 51 and 55	Test Passes
VISA	13 or 16	First digit is 4	Test Passes
AMEX	15	First two digits 34 or 37	Test Passes
DISCOVER	16	First four digits 6011	Test Passes
DINERS	14	First two digits 36 or 38 or the first three digits are greater than 300 but less than 305	Test Passes

Task 3: White-box Testing
Question 1
A JUnit test suite for 100% branch coverage was created in Java and the .java files have been provided with the submission of this assignment.

Question 2

A total of 6 tests were created to ensure that only one branch for each method was tested in each case and therefore each test can only fail for one reason. The following tests covered all method branches:

1) testPopValid() - Tests the ELSE branch of the pop() method.
2) testPopInValid() - Tests the IF branch of the pop() method.
3) testPushValid() - Tests the ELSE branch of the push() method.
4) testPushInValid() - Tests the IF branch of the push() method.
5) testIsEmptyValid() - Tests the TRUE branch of the isEmpty() method.
6) testIsEmptyInValid() - Tests the FALSE branch of the isEmpty() method.

Question 3

Mutation testing is the process of making copies of the source code to introduce minor changes (mutations) in an attempt to simulate general errors and coding mistakes (such as mistyping variable sizes or operators etc.). This is a good indication of the ability of a test suite to detect future errors based on previous mistakes that have been made. While coverage-driven testing may execute all of the code this does not imply that the code has been sufficiently tested. The test suite provided for the StringStack class utilises Assert methods to check the expected outcome at different stages within a test which ensures the test captures the desired errors and does not just execute the code. Mutation testing is therefore a more systematic and rigorous test and is far more useful than coverage-driven testing for ensuring error detection and correct code functionality.

Question 4

Table 10: This table identifies a mutant that would be killed by each of the 6 tests. The table includes the line number for the code that would be changed in the StringStack.java file and the change that would be made to the code (The code itself has not been changed but comments indicate where mutations would occur). A justification for each mutant is also included.

Test	Line Number	Change	Mutant Killed	Justification
testPopValid()	17	--pointer to ++pointer	Mutant3	Mutant 3: When the pointer is incremented instead of decremented with pop the stack is considered to still contain items. As the stack should be empty when all elements are removed (pointer value 0) this causes the test to fail. Mutant 4-6: This also applies if the isEmpty() method is altered to expect a positive pointer value but the pointer is decremented.
	21	pointer <= 0 to pointer < 0	Mutant4	
	21	pointer <= 0 to pointer > 0	Mutant5	
	21	pointer <= 0 to pointer != 0	Mutant6	
testPopInValid()	5	pointer = 0 to pointer = 10	Mutant1	Mutant 1: When checking that the stack is empty
	21	pointer <= 0 to pointer < 0	Mutant4	

	21	pointer <= 0 to pointer > 0	Mutant5	(pointer 0) before the pop method is called, the test fails as pointer = 10 suggests there are 11 elements in the stack. Mutant 4-6: isEmpty() expects 0 when exception is thrown, as pointer does not change the test fails when not expecting 0.
	21	pointer <= 0 to pointer != 0	Mutant6	
testPushValid()	21	pointer <= 0 to pointer > 0	Mutant5	Mutant 5-6: Expects not empty for isEmpty() as elements have been entered (pointer is positive), changing isEmpty() to expect non-zero numbers means that empty is then denoted by positive or negative numbers and the test fails.
	21	pointer <= 0 to pointer != 0	Mutant6	
testPushInValid()	21	pointer <= 0 to pointer > 0	Mutant5	Mutant 5-6: Expects not empty for isEmpty() as elements have been entered (pointer is positive), changing isEmpty() to expect non-zero numbers means that empty is then denoted by positive or negative numbers and the test fails.
	21	pointer <= 0 to pointer != 0	Mutant6	
testIsEmptyValid()	5	pointer = 0 to pointer = 10	Mutant1	Mutant 1: When pointer is 10 the test fails as isEmpty() believes there are elements in the stack. Mutant 4-6: When pointer is 0 (empty) but isEmpty() expects positive numbers the test returns false and fails.
	21	pointer <= 0 to pointer < 0	Mutant4	
	21	pointer <= 0 to pointer > 0	Mutant5	
	21	pointer <= 0 to pointer != 0	Mutant6	
testIsEmptyInValid()	11	pointer++ to pointer --	Mutant2	Mutant 2: When a new element is pushed to the stack but the pointer is decremented (negative) then as isEmpty() expects a negative it returns true when an item is in the stack, failing the test. Mutant 5-6: a positive pointer then denotes empty.
	21	pointer <= 0 to pointer > 0	Mutant5	
	21	pointer <= 0 to pointer != 0	Mutant6	