CS20202: Software Engineering
Practise Questions

1 Software Development Life Cycle

1. Match the following for V Model of SDLC. *[LM, Marks 0.25 * 6 = 1.5]*

Characteristics	
(a)	*Acceptance Tester* is responsible for
(b)	*QA Team* is responsible for
(c)	*Building the system right* is ensured by
(d)	*Building the right system* is ensured by
(e)	*Process focus* is key for
(f)	*Product focus* is key for

Activity	
(1)	Validation
(2)	Verification
(3)	Debugging
(4)	Bug Injection

Answer:

(a)	(1)
(b)	(2)
(c)	(2)
(d)	(1)
(e)	(2)
(f)	(1)

2. Match the development activity with the appropriate SDLC Models. *[LM, Marks 0.25 * 6 = 1.5]*

Development Activity	
(a)	I work with my friend as *driver / observer*
(b)	I design test cases and then code to make them pass
(c)	I do stand-up meeting with my team every morning
(d)	I build prototype and keep refining it in quick cycles
(e)	I use the most classical model for development
(f)	I repeat *planning*, *risk analysis*, *engineering*, and *evaluation*

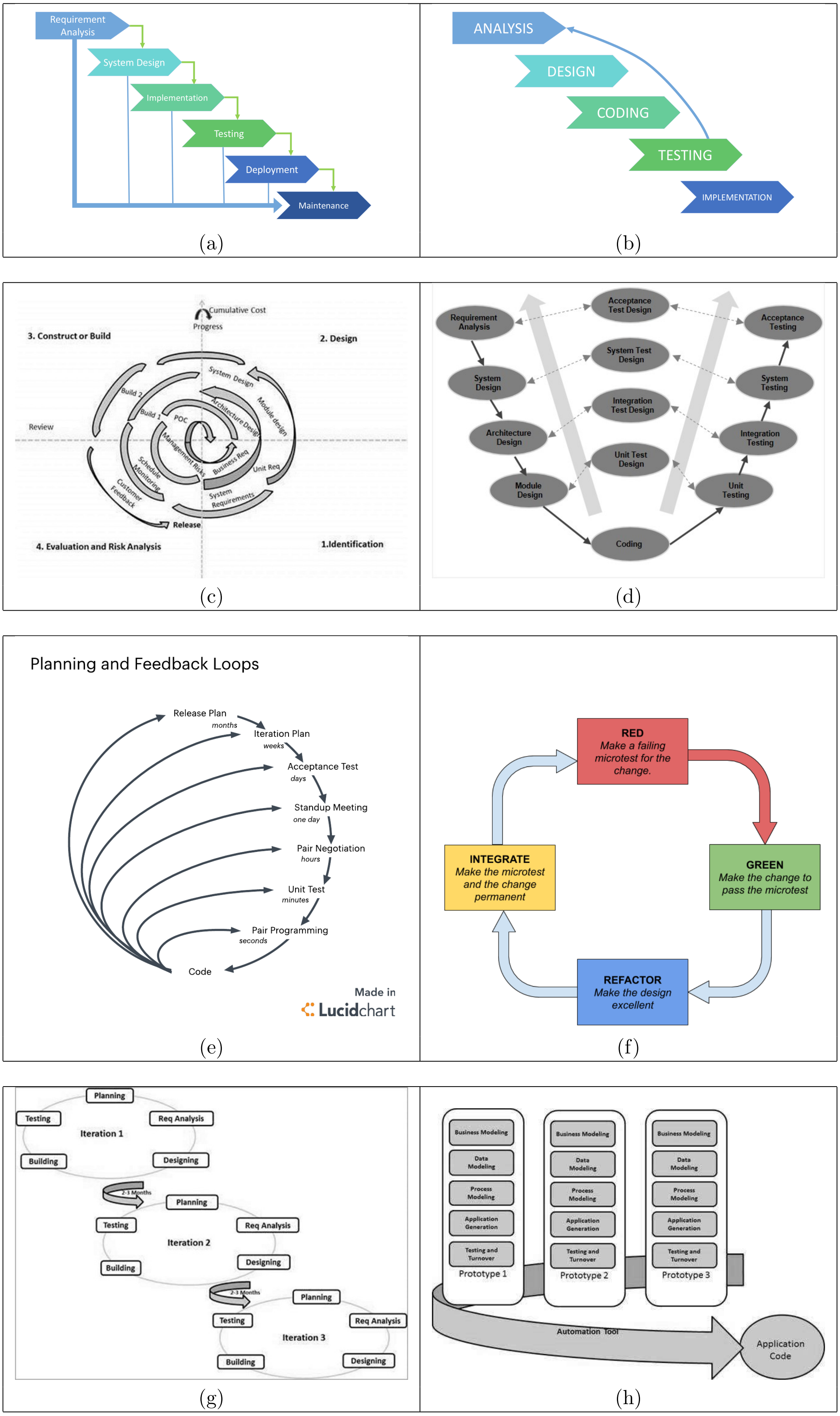
SDLC Models	
(1)	TDD
(2)	RAD
(3)	SCRUM
(4)	Spiral
(5)	Waterfall
(6)	XP

Answer:

(a)	(6)
(b)	(1)
(c)	(3)
(d)	(2)
(e)	(5)
(f)	(4)

3. Match the following SDLC life-cycle diagrams with their respective names. *[LM, Marks 0.25 * 8 = 2]*

SDLC Life-cycle Diagrams



(a) (b) (c) (d) (e) (f) (g) (h)

SDLC Models

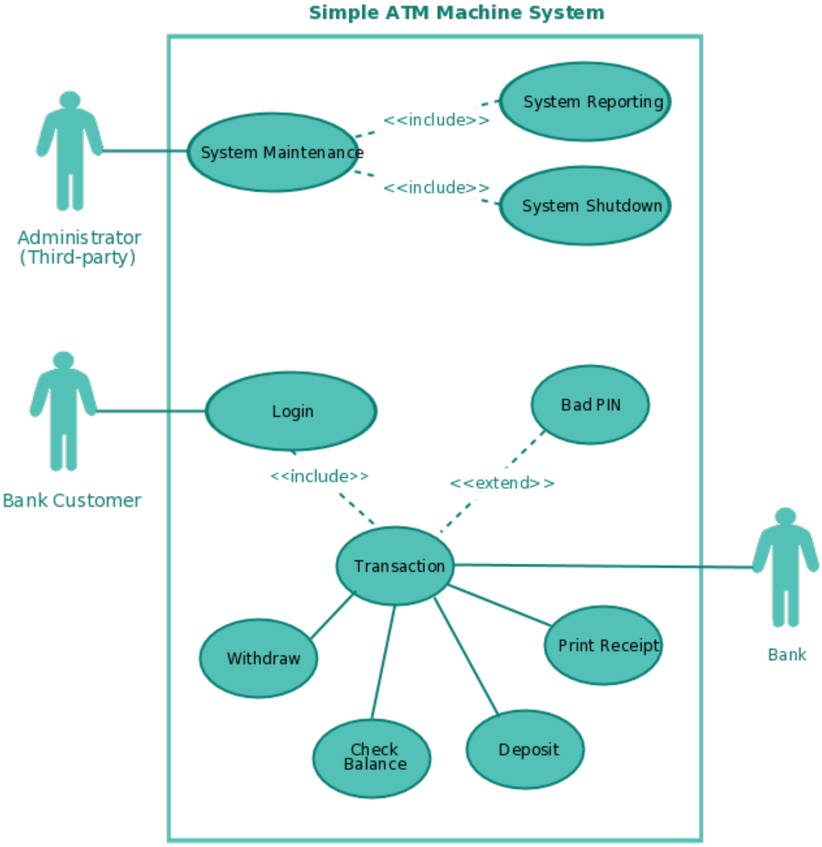
(1)	Agile
(2)	Iterative
(3)	RAD
(4)	Spiral
(5)	TDD
(6)	V
(7)	Waterfall
(8)	XP

Answer:

(a)	(7)
(b)	(2)
(c)	(4)
(d)	(6)
(e)	(8)
(f)	(5)
(g)	(1)
(h)	(3)

2 UML

1. Choose the **correct** statements below based on the Use Case Diagram. *[LM, Marks 1]*



(a) *Bank Customer* may not login for doing a *Transaction*

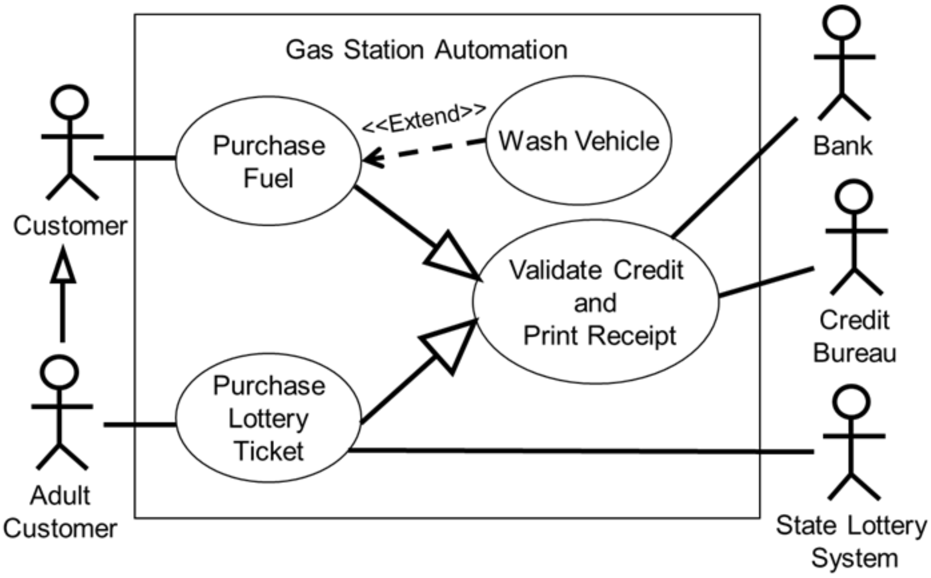
(b) There are two *human actors* in the system

(c) *Bank Customer* can generate *PIN*

(d) *Bank* manages every *Transaction*

Answer: (b), (d)

2. Choose the **correct** statements below based on the Use Case Diagram. *[MSQ, Marks 1]*



(a) *Wash Vehicle* is optional during *Purchase Fuel*

(b) Use-case *Validate Credit and Print Receipt* is specialization of *Purchase Lottery Ticket* use case

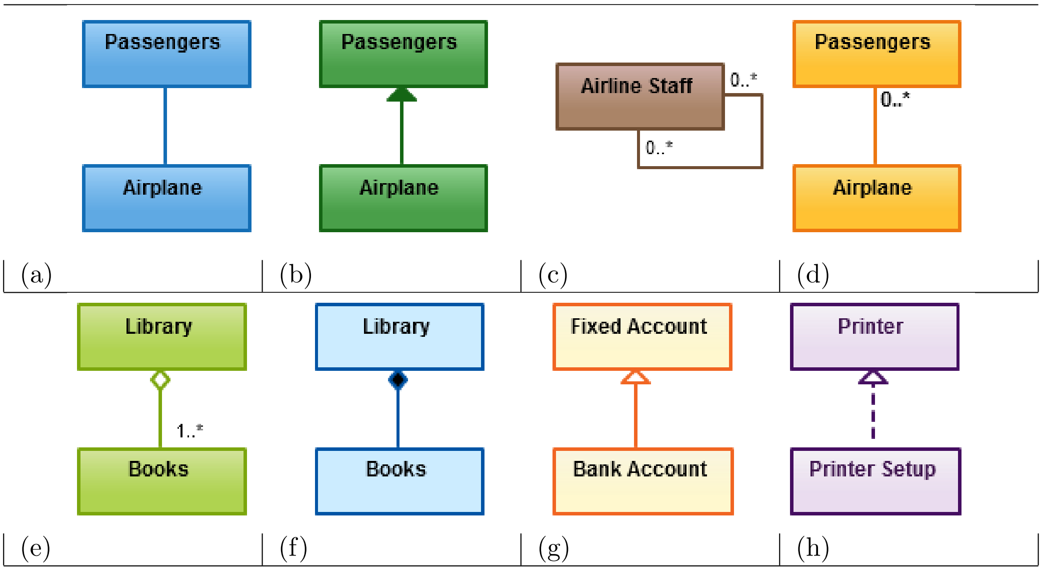
(c) *Adult Customer* ISA *Customer*

(d) *Credit Bureau* manages *Purchase Fuel*

Answer: (a), (c)

3. Match the illustrative examples below with UML Class Diagram Relationships. *[LM, Marks 0.25 * 8 = 2]*

Examples of Relationships in UML Class Diagrams



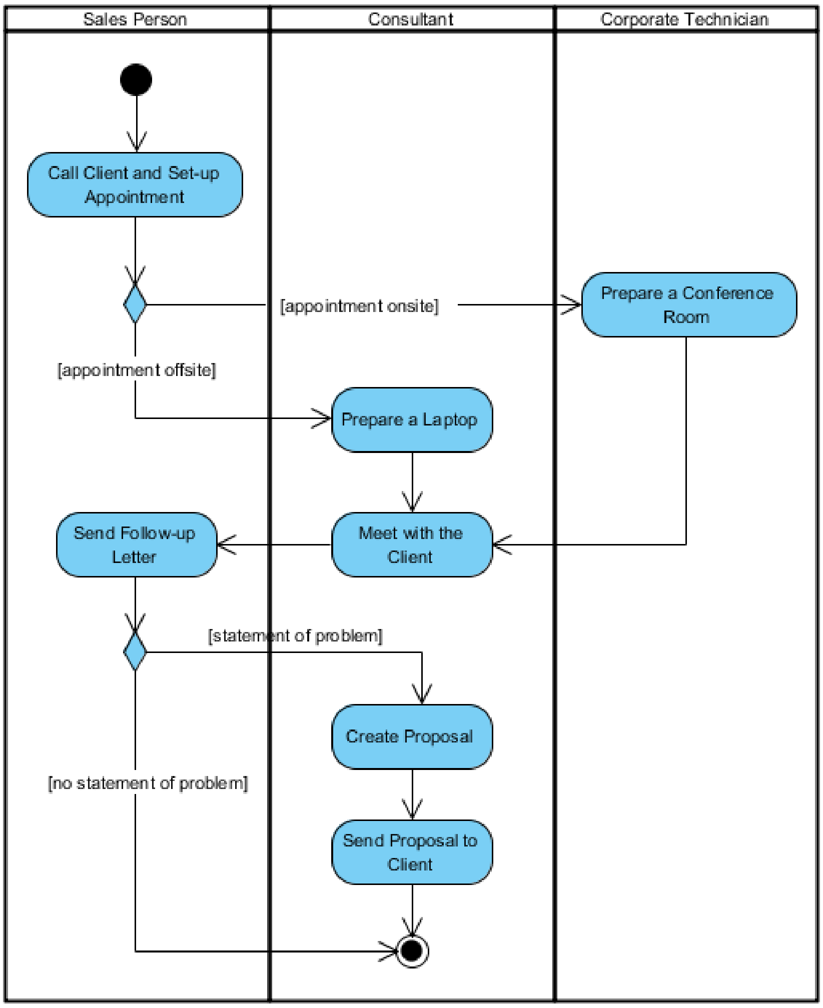
Class Diagram Relationships

(1)	Realization
(2)	Association
(3)	Inheritance / Generalization
(4)	Aggregation
(5)	Reflexive Association
(6)	Multiplicity
(7)	Composition
(8)	Directed Association

Answer:

(a)	(2)
(b)	(8)
(c)	(5)
(d)	(6)
(e)	(4)
(f)	(7)
(g)	(3)
(h)	(1)

4. Choose the **correct** statements below based on the Activity Diagram. *[MSQ, Marks 1]*



(a) There are 4 *swim-lanes*

(b) Choice of swim-lane may depend on *appointment* being *onsite* or *offsite*

(c) *Consultant* creates proposals

(d) *Sales Person* prepares a Conference Room

Answer: (b), (c)

3 Testing

1. Consider the following *Quadratic Equation Solver* (QES) function `Solve` that takes 3 `double` parameters `a`, `b`, and `c` for solving equations of the form $ax^2 + bx + c = 0$. The solutions are passed back through output parameters `r1` and `r2`. The function returns a value designating the equivalence class of the root/s.

*[LM, Marks 0.25 * 6 = 1.5]*

The `Solve` *function code is used in other questions too. So if you are getting this for the first time, you may study it well. Of course, the same will be provided in the other questions too where it is used.*

```
00: unsigned int Solve(double a, double b, double c, double& r1, double& r2)
01: {
02:     unsigned int retVal = 0;
03:     if (0 == a) {
04:         if (0 == b) {
05:             if (0 == c) {
06:                 retVal = 5;
07:             } else {
08:                 retVal = 0;
09:             }
10:         } else {
11:             retVal = 1;
12:             r1 = -c/b;
13:         }
14:     } else {
15:         double disc = b*b - 4*a*c;
16:         if (0 == disc) {
17:             retVal = 2;
18:             r1 = r2 = -b/(2*a);
19:         } else {
20:             if (disc > 0) {
21:                 retVal = 3;
22:                 r1 = (-b + sqrt(disc))/(2*a);
23:                 r2 = (-b - sqrt(disc))/(2*a);
24:             } else {
25:                 retVal = 4;
26:                 r1 = -b/(2*a); r2 = sqrt(-disc))/(2*a);
27:             }
28:         }
29:     }
30:
31:     return retVal;
32: }
```

For checking the statement coverage of `Solve`, a set of 6 test cases are designed below. Match the test cases with the statements it covers in the above code.

	Coefficients		
	a	**b**	**c**
(a)	0	3	6
(b)	1	0	1
(c)	0	0	0
(d)	1	2	-35
(e)	1	-6	9
(f)	0	0	5

	Statements Covered
(1)	2,3,4,5,6,31
(2)	2,3,4,5,8,31
(3)	2,3,4,11,12,31
(4)	2,3,15,16,17,18,31
(5)	2,3,15,16,20,21,22,23,31
(6)	2,3,15,16,20,25,26,31

Answer:

(a)	(3)
(b)	(6)
(c)	(1)
(d)	(5)
(e)	(4)
(f)	(2)

2. Consider the following *Quadratic Equation Solver* (QES) function `Solve` that takes 3 `double` parameters `a`, `b`, and `c` for solving equations of the form $ax^2 + bx + c = 0$. The solutions are passed back through output parameters `r1` and `r2`. The function returns a value designating the equivalence class of the root/s. *[LM, Marks 0.25 * 6 = 1.5]*

   The `Solve` *function code is used in other questions too. So if you are getting this for the first time, you may study it well. Of course, the same will be provided in the other questions too where it is used.*

```
00: unsigned int Solve(double a, double b, double c, double& r1, double& r2)
01: {
02:     unsigned int retVal = 0;
03:     if (0 == a) {
04:         if (0 == b) {
05:             if (0 == c) {
06:                 retVal = 5;
07:             } else {
08:                 retVal = 0;
09:             }
10:         } else {
11:             retVal = 1;
12:             r1 = -c/b;
13:         }
14:     } else {
15:         double disc = b*b - 4*a*c;
16:         if (0 == disc) {
17:             retVal = 2;
18:             r1 = r2 = -b/(2*a);
19:         } else {
20:             if (disc > 0) {
21:                 retVal = 3;
22:                 r1 = (-b + sqrt(disc))/(2*a);
23:                 r2 = (-b - sqrt(disc))/(2*a);
24:             } else {
25:                 retVal = 4;
26:                 r1 = -b/(2*a); r2 = sqrt(-disc))/(2*a);
27:             }
28:         }
29:     }
30:
31:     return retVal;
32: }
```

For checking the branch coverage of `Solve`, a set of 6 test cases are designed below. Match the test cases with the branches it covers in the above code.

Coefficients			
	a	**b**	**c**
(a)	6	17	12
(b)	0	3	-9
(c)	0	0	3
(d)	1	10	25
(e)	1	-5	6
(f)	0	0	0

Branches Covered	
(1)	2-3,3-4,4-5,5-6,6-31
(2)	2-3,3-4,4-5,5-8,8-31
(3)	2-3,3-4,4-11,11-12,12-31
(4)	2-3,3-15,15-16,16-17,17-18,18-31
(5)	2-3,3-15,15-16,16-20,20-21,21-22,22-23,23-31
(6)	2-3,3-15,15-16,16-20,20-25,25-26,26-31

Answer:

(a)	(6)
(b)	(3)
(c)	(2)
(d)	(4)
(e)	(5)
(f)	(1)

3. Consider the following *Quadratic Equation Solver* (QES) function `Solve` that takes 3 `double` parameters `a`, `b`, and `c` for solving equations of the form $ax^2 + bx + c = 0$. The solutions are passed back through output parameters `r1` and `r2`. The function returns a value designating the equivalence class of the root/s. *[LM, Marks 0.25 * 6 = 1.5]*

   The `Solve` *function code is used in other questions too. So if you are getting this for the first time, you may study it well. Of course, the same will be provided in the other questions too where it is used.*

```
00: unsigned int Solve(double a, double b, double c, double& r1, double& r2)
01: {
02:     unsigned int retVal = 0;
03:     if (0 == a) {
04:         if (0 == b) {
05:             if (0 == c) {
06:                 retVal = 5;
07:             } else {
08:                 retVal = 0;
09:             }
10:         } else { // Linear equation
11:             retVal = 1;
12:             r1 = -c/b;
13:         }
14:     } else {
15:         double disc = b*b - 4*a*c;
16:         if (0 == disc) {
17:             retVal = 2;
18:             r1 = r2 = -b/(2*a);
19:         } else {
20:             if (disc > 0) {
21:                 retVal = 3;
22:                 r1 = (-b + sqrt(disc))/(2*a);
23:                 r2 = (-b - sqrt(disc))/(2*a);
24:             } else {
25:                 retVal = 4;
26:                 r1 = -b/(2*a); r2 = sqrt(-disc))/(2*a);
27:             }
28:         }
29:     }
30:
31:     return retVal;
32: }
```

For checking the path coverage of `Solve`, a set of 6 test cases are designed below. Match the test cases with the paths it covers in the above code.

Coefficients			
	a	**b**	**c**
(a)	4	-12	9
(b)	4	0	9
(c)	6	-22	20
(d)	0	0	0
(e)	0	5	3
(f)	0	0	27

Paths Covered	
(1)	2-3-4-5-6-31
(2)	2-3-4-5-8-31
(3)	2-3-4-11-12-31
(4)	2-3-15-16-17-18-31
(5)	2-3-15-16-20-21-22-23-31
(6)	2-3-15-16-20-25-26-31

Answer:

(a)	(4)
(b)	(6)
(c)	(5)
(d)	(1)
(e)	(3)
(f)	(2)

4. We need to perform black box testing for a login screen which allows a maximum of three attempts before the login is locked. Assuming that the user-id is correct, how many test cases will be needed at the minimum for this test? *[SA, Marks 0.5]*

   Answer: 4