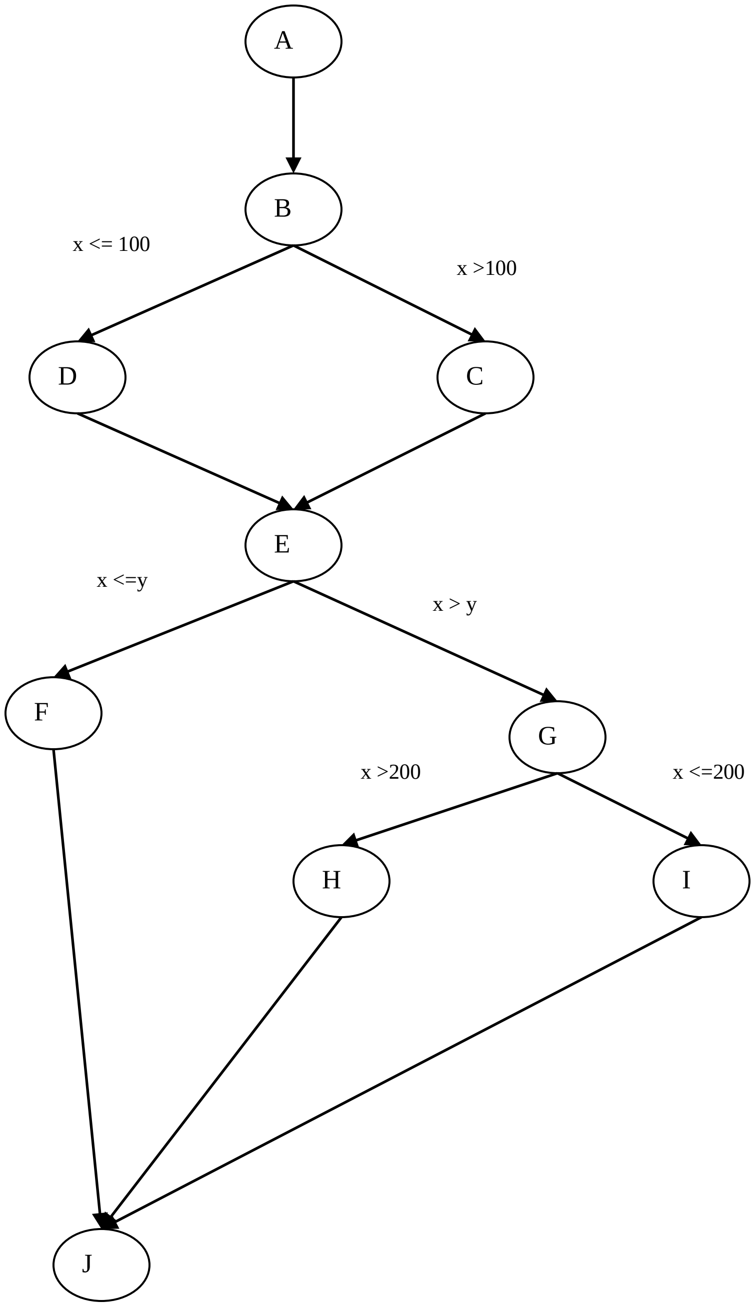
SEENG 426: Final Exam Review Solutions

Exercise 1:

void prog(int X, int Y) { **x = X; y = Y;**	A
if (x > 100) {	B
x = x – 100; **}**	C
else { **x = x + 100;** **}**	D
if (x <= y) {	E
y = 2*y; **}**	F
else { **if (x > 200) {**	G
x = x - 100; **}**	H
else { **x = x*x;** **}** **}**	I
printf(“x=%d. y=%d”,x, y); **}**	J



Test input 1: covers path ABDEGIJ
Test input 2: executes path ABCEFJ

1. Both test inputs combined execute segments A, B, C, D, E, F, G, I, J

Segment H is not covered. So statement coverage = 9/10=90%

2.We have the following branches:

AB, BCE, BDE, EFJ, EG, GHJ, GIJ

Test input 1covers: AB, BDE, EG, GIJ
Test input 2 covers: AB, BCE, EFJ

Both test inputs combined cover: AB, BCE, BDE, EFJ, EG, GIJ

Branch GHJ is not covered.

So branch coverage = 6/7=85.71%

3. We have the following paths:

ABCEFJ, ABCEGHJ, ABCEGIJ, ABDEFJ, ABDEGHJ, ABDEGIJ

Both test inputs combined cover: ABDEGIJ, ABCEFJ

So path coverage = 2/6=33.33%

4.

Test input (X=201, Y=100) covers path ABCEGIJ

Test input (X=301, Y=100) covers path ABCEGHJ

Test input (X=10, Y=110) covers ABDEFJ

Path ABDEGHJ will never be taken; try test input (X=100, Y=100)

So the maximum path coverage that can be achieved is 5/6=83.33%

Exercise 2:
First we need to derive the logic function corresponding to the alarm function.
This can be done using KV matrix:

AB

	00	01	11	10
00			1	
CD 01			1	1
11			1	1
10			1	1

Based on that we can derive: Z=AC + AB + AD

Input vector Number	Conditions				Actions	
	DeviceCtl (A)	**TempLevel (B)**	**PressureLevel (C)**	**GasVolume (D)**	**Alarm (Z)**	**ShutdownCtl (W)**
0	0	0	0	0	0	0
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	1	1	0	0	1	0
4	0	0	1	0	0	0
5	1	0	1	0	1	0
6	0	1	1	0	0	0
7	1	1	1	0	1	1
8	0	0	0	1	0	0
9	1	0	0	1	1	1
10	0	1	0	1	0	0
11	1	1	0	1	1	1
12	0	0	1	1	0	0
13	1	0	1	1	1	1
14	0	1	1	1	0	0
15	1	1	1	1	1	1

We then determine the unique true points:
A product term that makes a product term true but no other term true in a sum-of-products formula.

Unique true points for:

- AC: {5}
- AB: {3}
- AD: {9}

Near false point: a variant, containing a negated literal from a product term, which makes the logic function, evaluates to zero for the negated term. Negating each literal in the term, one-by-one, identifies near-false points.

Product term Negation	Variants Containing this negation	Test candidate set: Variants containing This negation, where Z=0	Candidate set number
A C			1
A ~C	1, 3, 9, 11	1	2
~A C	4, 6, 12, 14	4, 6, 12, 14	3

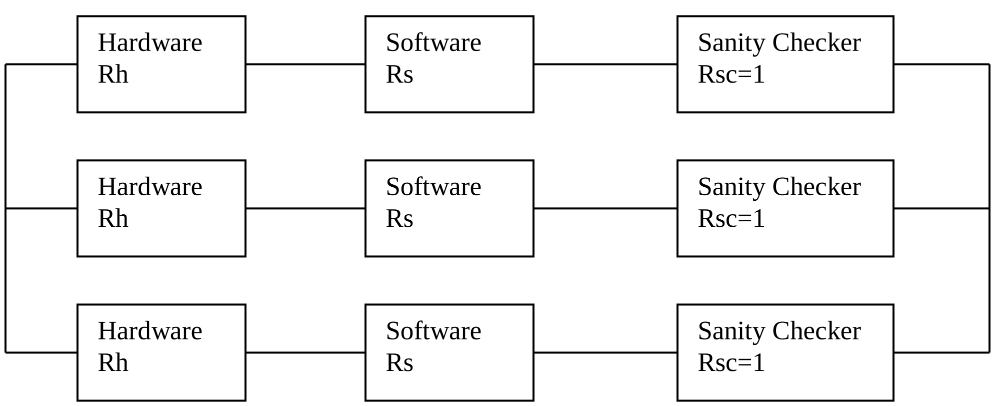
Product term Negation	Variants Containing this negation	Test candidate set: Variants containing This negation, where Z=0	Candidate set number
A B			4
A ~B	1, 5, 9, 13	1	5
~A B	2, 6, 10, 14	2, 6, 10, 14	6

Product term Negation	Variants Containing this negation	Test candidate set: Variants containing This negation, where Z=0	Candidate set number
A D			7
A ~D	1, 3, 5, 7	1	8
~A D	8, 10, 12, 14	8, 10, 12, 14	9

Variant	Test candidate set									Test case
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
0										
1		x			x			x		√
2						x				
3				x						√
4			x							
5	X									√
6			x			x				
7										
8									x	
9							x			√
10						x			x	
11										
12			x						x	
13										
14			x			x			x	√
15										

A complete test suite should include at least one variant from each test candidate set. In this example a suitable test suite would be variants {1, 3, 5, 9, 14}.

Exercise 3:



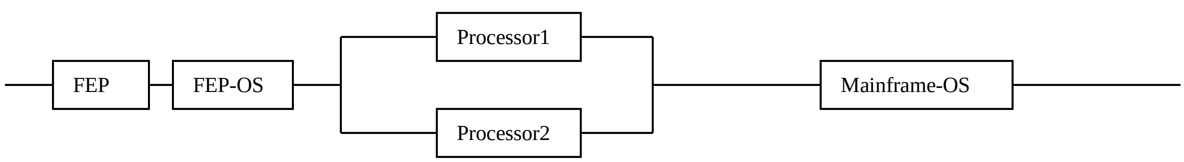
The system reliability $R = 1 - (1 - R_h R_s R_{sc})^3 = 1 - (1 - R_h R_s)^3 = 0.999$

$\Rightarrow (1 - R_h R_s)^3 = 0.001 \Rightarrow (1 - R_h R_s) = 0.1 \Rightarrow R_h R_s = 0.9$

We start by arbitrary values for Rh and Rs and start playing around that. For instance selecting Rh=0.95 and Rs=0.95 yields requisite system reliability. Since the cost of hardware is low, we can slightly increase the hardware and slightly decrease the software component, and play round that. Doing so lead us finally to Rh=0.97 and Rs=0.93, with a total cost of 4.3M.

Steps	**Rh**	**Rs**	**Rsys**	**Total cost**
1	0.95	0.95	0.999	4.5M
2	0.96	0.94	0.999	4.35M
3	0.97	0.93	0.999	4.3M
4	0.98	0.92	0.999	4.75M

Exercise 4:
Naturally, the mainframe OS will run on both processors; hence, a more complete event diagram is given below:



System reliability:
$Rsys = R_{fep} \times R_{fep\text{-}os} \times (1 - (1 - R_{mf\text{-}proc})^2) \times R_{mf\text{-}os}$

- ⇨ $R_{mf\text{-}os} = 0.90/(0.99 \times 0.95 \times (1-(1-0.98)^2)) = 0.9/0.94 = 0.957$
- ⇨ The mainframe OS reliability should be 0.957

Exercise 5:
The average number of failure per system is 0.1 failure/week.
For all the doctors this gives 60 failure/week.
The average service person can make 2 service calls daily, assuming a 5-days week, he average service person would handle 10 service calls/week.
So this means that we need 6 service persons.

Assuming 52-weeks in a year, we have 52×0.1= 5.2 calls annually per system.
So to make a 20% profit, we need to charge cost= 5.2×200×1.2=$1,248

Exercise 6:

1. Number of failures at the end of testing:

$$\Delta\mu = \frac{\nu_0}{\lambda_0} \times (\lambda_1 - \lambda_2) = \frac{100}{10} \times (10 - 0.0004) = 99.99 \approx 100 \quad failures$$

2. Corresponding execution time:

$$\Delta\tau = \frac{\nu_0}{\lambda_0} \times \ln\left(\frac{\lambda_1}{\lambda_2}\right) = \frac{100}{10}\ln\left(\frac{10}{0.0004}\right) = 101\ CPUhr$$

3. Testing cost:

$$\begin{aligned} Cost_{test} &= test\_effort \times \Delta\mu \times wage + computer\_cost \times \Delta\tau \\ &= 6 \times 100 \times 100 + 50 \times 101 = \$65,050 \end{aligned}$$

4. Testing duration:

$$Duration_{test} = \frac{test\_effort \times \Delta\mu}{size\_test\_team} = \frac{6 \times 100}{4} = 150hrs = 3weeks\ and\ 4\ days(business)$$

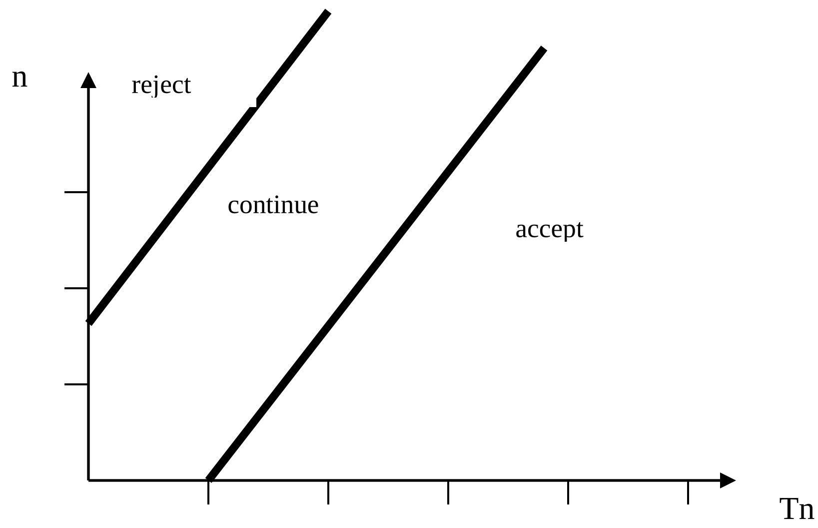
Exercise 7:

A= ln(β/(1-α))= ln(0.1/0.9)=-2.2
B= ln((1-β)/α)=ln(0.9/0.1)=2.2

Accept-continue boundary: Tn= (A-nlnγ)/(1-γ)= 2.2+0.69n => (n=2, Tn=3.58),
Reject-continue boundary: Tn= (B-nlnγ)/(1-γ)= -2.2+0.69n => (n=4,Tn=0.56), (n=6, Tn=1.94)

Failure number	**Failure time (CPU hr)**	**Normalized Failure time**	**Region**
1	8	0.8	Continue
2	19	1.9	Continue
3	60	6	Accept

So we can assume at this stage of the testing that the failure intensity objective is met.



Exercise 8:

a.
Initial Release:

-SSI(1) = 50 KLOC
-DEF(1)= 2 def/KSSI
therefore, (2 * 50) = 100 defects

Second Release:

-DEF(2) = 90% * DEF(1) = 1.8 Def/KSSI
-SSI(2) = SSI(1) + CSI(2) - deleted(2) - changed(2)
= 50 + 20 - 0 - (0.20 *20)
= 66 KLOC

Total Defects in Second Release:
= 1.8 * 66 = 119 defects

Third Release:

SSI(3) = SSI(2) + CSI(3) - deleted(3) - changed(3)
= 66 + 30 - 0 - (30 * 0.20)
= 66 + 30 - 6
= 90 KLOC
DEF(3) = 90% * DEF(2)
=0.90 * 1.8 = 1.62 def/KSSI

Total Defects in Third Release:
= 1.62 * 90 = 146 defects

Therefore, total number of additional defects in the third release is 146.

b.
DEF = (Total # defects)/Size = N/size = #def/KSSI

N(2) = 66 * 1.8 = 119 def
90*DEF(3) = N(3) <= N(2) = 119
DEF(3) <= 119/90 = 1.32def/KSSI

Exercise 9:
The example assume here the following default values:
MAX_WEIGHT = max=400lbs; HIGHEST_FLOOR= high=100;
LOWEST_FLOOR=low=1

Variables	**Conditions**	**Type**	**Test Cases**							
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Weight	>0	On	0							
		Off		1						
	≤MAX_WEIGHT	On			max					
		Off				max +1				
	Typical	In					50	100	90	95
selectedFloor	≥LOWEST_FLOOR	On					low			
		Off						low +1		
	≤HIGHEST_FLOOR	On							high	
		Off								high -1
	!=currentFloor	On								
		Off								
		Off								
	Typical	In	2	3	4	3				
door	isClosed()	On								
		Off								
		In	true	true	true	true	tru e	true	true	true
inEmergency	=false	On								
		Off								
		In	fals	fals	fals	fals	fals	fals	fals	fals
currentFloor	Typical	In	1	2	3	4	10	low	high -1	20
Expected Result			reje	acc	acc	rej	acc	acc	acc	acc
Expected Output (Elevator Run)			No	yes	yes	no	yes	yes	yes	yes

Variables	Conditions	Type	Test Cases						
			9	**10**	**11**	**12**	**13**	**14**	**15**
Weight	>0	On							
		Off							
	≤MAX_HEIGHT	On							
		Off							
	Typical	In	85	79	60	99	250	200	150
selectedFloor	≥LOWEST_FLOOR	On							
		Off							
	≤HIGHEST_FLOOR	On							
		Off							
	!=currentFloor	On	10						
		Off		11					
		Off			9				
	Typical	In			9	10	95	99	10
door	isClosed()	On				true			
		Off					fals		
		In	true	true	true			true	
inEmergency	=false	On						fals	
		Off							true
		In	fals	fals	fals	fals	fals		
currentFloor	Typical	In	10	10	10	5	90	90	80
Expected Result			rej	acc	acc	acc	rej	acc	rej
Expected Output (Elevator Run)			no	yes	yes	yes	no	yes	no

Exercise 10:
Through fault injection, the total number of defects in the field is:
N= (25/20) x 55= 69 defects

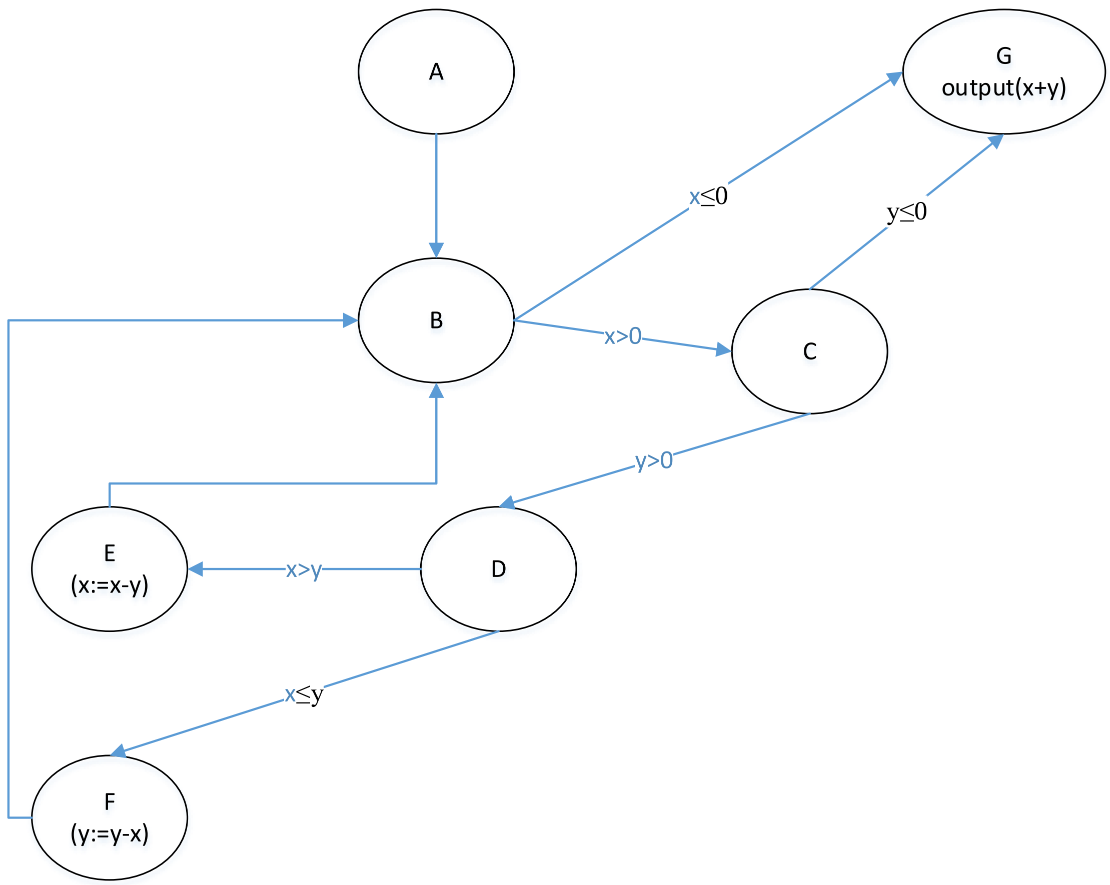
1. The overall DRE(overall) = (163 + 186 + 271)/(163 + 186 + 271 +69) = 90%
2. The DRE (ST) =271/(271 +69) = 80%
3. Residual defect density d = 69/500 = 0.138 defects/KLOC which is very low; so the product was ready to be released; field quality would most likely be good.

Exercise 11:

The code segments are identified as follows:

Begin Input(x,y)	A
While(x>0 and y>0) Do	B, C
If (x>y)	D
Then x := y -y	E
Else y := y-x	F
Output(x+y) End	G

The CFG for the code is as follows:



The entry-exit paths are as follows:

ABG
ABCG
ABCDEBG
ABCDEBCG
ABCDFBG
ABCDFBCG

Statement coverage can be achieved using 2 paths:
ABCDEBG
ABCDFBG

Edge coverage can be achieved using 3 paths:
ABCDFBG
ABCDEBCG

Condition coverage can be achieved using 2 paths

ABCDFBG
ABCDEBCG

Path coverage requires covering all the paths listed above.

The test cases can be identified through path sensitization. For instance, a test case for path ABG is as follows:

[x=0; y=5; expected output = 5]

Exercise 12:

Quadratic equation will be of type:
*ax*2 + *bx* + *c* = 0
- Roots are real if $(b^2 - 4ac) > 0$
- Roots are imaginary if $(b^2 - 4ac) < 0$
- Roots are equal if $(b^2 - 4ac) = 0$
- Equation is not quadratic if $a=0$

Using the extreme point combination, we should identify N=4n+ 1 test cases, where n is the number of input variables; n=4 => N=13; the test cases are as follows:

Test Case	a	b	c	Expected output
1	0	50	50	Not Quadratic
2	1	50	50	Real Roots
3	50	50	50	Imaginary Roots
4	99	50	50	Imaginary Roots
5	100	50	50	Imaginary Roots
6	50	0	50	Imaginary Roots
7	50	1	50	Imaginary Roots
8	50	99	50	Imaginary Roots
9	50	100	50	Equal Roots
10	50	50	0	Real Roots
11	50	50	1	Real Roots
12	50	50	99	Imaginary Roots
13	50	50	100	Imaginary Roots

The above test cases correspond to inputs that are all at the boundary or in the domain. For more robust test cases, we need to include test cases slightly outside the domain; this means 2 additional test cases for each input variable. Likewise, for robust test cases, we need a total N2= 6n+1=19, as given below:

Test Case	A	B	C	Expected Output
1	-1	50	50	Invalid Input
2	0	50	50	Not Quadratic Equation
3	1	50	50	Real Roots
4	50	50	50	Imaginary Roots
5	99	50	50	Imaginary Roots
6	100	50	50	Imaginary Roots
7	101	50	50	Invalid Input
8	50	-1	50	Invalid Input
9	50	0	50	Imaginary Roots
10	50	1	50	Imaginary Roots
11	50	99	50	Imaginary Roots
12	50	100	50	Equal Roots
13	50	101	50	Invalid Input
14	50	50	-1	Invalid Input
15	50	50	0	Real Roots
16	50	50	1	Real Roots
17	50	50	99	Imaginary Roots
18	50	50	100	Imaginary Roots
19	50	50	101	Invalid Input

Exercise 13:

i. Current failure intensity
$\lambda(\mu) = \lambda_0 [1 - \mu / v_0] = 16 [1 - 50 / 200] = 12$ failure / CPU hr

ii. Decrement of failure intensity
$d\lambda / d\mu = - \lambda_0 / v_0 = 16 / 200 = -0.08$ CPU hr

iii. Failure intensity at 100 CPU hr
$\lambda(\tau) = \lambda_0 e^{(- \lambda 0 / v0) \times \tau}$
$= 16\, e^{(-16 \times 100 / 200)} = 16\, e^{(-8)}$

Exercise 14:
Choose the correct or best alternative in the following:

1. All the modules of the system are integrated and tested as complete system in the case of
   (A) Bottom up testing (B) Top-down testing
   (C) Sandwich testing (D) Big-Bang testing
   Ans: D

2. The level at which the software uses scarce resources is
   (A) reliability (B) efficiency
   (C) portability (D) all of the above
   Ans: B

3. Modifying the software to match changes in the ever changing environment is called
   (A) adaptive maintenance (B) corrective maintenance
   (C) perfective maintenance (D) preventive maintenance
   Ans: A

4. Alpha and Beta Testing are forms of
   (A) Acceptance testing (B) Integration testing
   (C) System Testing (D) Unit testing
   Ans: C

5. Changes made to the system to reduce the future system failure chances is called
   (A) Preventive Maintenance (B) Adaptive Maintenance
   (C) Corrective Maintenance (D) Perfective Maintenance
   Ans: A

6. The problem that threatens the success of a project but which has not yet happened is a
   (A) bug (B) error
   (C) risk (D) failure

Ans: C

7. The main purpose of integration testing is to find
   (A) design errors (B) analysis errors
   (C) procedure errors (D) interface errors
   Ans: D

8. For a function of two variables, boundary value analysis yields
   (A) 4n + 3 test cases (B) 4n + 1 test cases
   (C) n + 4 (D) None of the above
   Ans: B

9. Site for Alpha Testing is
   (A) Software Company (B) Installation place
   (C) Any where (D) None of the above
   Ans: A

10. Which is not a size metric?
    (A) LOC (B) Function count
    (C) Program length (D) Cyclomatic complexity
    Ans: D

11. As the reliability increases, failure intensity
    (A) decreases (B) increases
    (C) no effect (D) none of the above
    Ans: A

12. Software deteriorates rather than wears out because
    (A) software suffers from exposure to hostile environments.
    (B) defects are more likely to arise after software has been used often.
    (C) multiple change requests introduce errors in component interactions.
    (D) software spare parts become harder to order.

    Ans: B

13. Which of these terms is a level name in the Capability Maturity Model?
    (A) Ad hoc (B) Repeatable
    (C) Reusable (D) Organized
    Ans: B

14. The ISO quality assurance standard that applies to software engineering is
    (A) ISO 9000 (B) ISO 9001
    (C) ISO 9002 (D) ISO 9003

    Ans: B

15. What is the normal order of activities in which software testing is organized?
    (A) unit, integration, system, validation
    (B) system, integration, unit, validation
    (C) unit, integration, validation, system
    (D) none of the above
    Ans: A

16. The goal of quality assurance is to provide management with the data needed to determine which software engineers are producing the most defects.
    (A) True (B) False
    Ans: B

17. Units and stubs are not needed for unit testing because the modules are tested independently of one another
    (A) True (B) False
    Ans: A