Stakeholders: Developers, Testers, Quality Assurance Team, Product manager, clients, Operation and support team, Documentation team, security team

Regression testing: a type of sw testing conducted after a code update to ensure that the update introduced no new bugs. This is because new code may bring in new logic that conflicts with the existing code, leading to defects.

Who is responsible for unit testing?

→ Developer

→ 1st level testing, done before integration testing

→ in SDLC or V model.

Race condition occurs when the behavior of sw becomes dependent on the sequence or timing of uncontested event.

→ lead to unpredictable results and challenging debugging scenerios.

→ 2 computer program processes/threads attempt to access the same resource at the same time

→ common issue of multithreaded app

steps of (7)

SDLC - Software Development Life Cycle:

1) Project planning
2) Gathering requirement and analysis
3) Design
4) Coding / Implementation
5) testing
6) Deployment
7) maintenance

Cyclomatic complexity: a quantitative measure of the complexity of a program's control flow. It provides a way to assess the complexity of a SW system based on the number of independent paths through the source code.

formula: $E - N + 2P$ → connected component

E → edge in CFG, N → nodes

System testing is defined as testing of a complete and fully integrated SW product. This testing falls in black-box testing. (inner design of code is not needed)

8 types: 1) Usability Testing 2) Regression 3) Load 4) Functional 5) Migration 6) Compatibility 7) Boundary Value 8) Fuzz Testing

Bug model: provides a perception of the kind of bugs expected. considering the nature of all types of bugs, a bug model can be prepared that may help in deciding a testing strategy.

Exhaustive testing? Why should avoid?

→ Exhaustive or complete sw testing means that every statement in the program and every possible path combination with every possible combination of data must be executed.

→ avoid because:

(a) The domain of possible inputs → too large to test.

Input data → 4 subparts

1) Valid: case: addin 2-digit 2 numbers
range: −99 to 99 (total 199)
combination: 199 × 199
all testing? → impossible

2) Invalid: 1) numbers out of range
2) combination of alphabets and digits

iii) combo of all alphabets

iv) combo of control characters

v) combo of any other key on the keyboard.

3) <u>Edited Input:</u> eg: many space

4) <u>Race condition:</u> timing variation between multiple inputs

→ generally A before B

→ but give B before A (then race condition happens)

(b) Too many possible path

(c) Design issume (d) Time consuming

(a)

Top - Down	Bottom - UP
1) system integration testing begins with top level modules.	1) begins with lowest level modules.
2) higher level modules are tested first, then lower → then integrated	2) lower first → then higher → then integrated
3) It's beneficial when error occurs at the top	3) when at the bottom.

Top-down	Bottom up
4) main module design first, then submodules are called from it.	4) different modules are created first then integrated with main form
5) implemented on procedure oriented programming lang.	5) implemented on OOP.
6) complexity - simple	6) complex
7) works on big to small components	7) small to big
8) stub module	8) driven module
9) more expensive	9) less expensive
10) need complete testing for system.	10) early detects issues.

Testing is not a single phase in SDLC

→ It span multiple phases

1) Requirement analysis and planning: identify potential test scenerios and ensure that the requirements are testable.

2) Design: testing strategy planning

3) Coding: Unit testing → validate each units
Code reviews

4) Integration: integrate units → then test to see if work together

5) System testing: entire system

6) User acceptance: ensure if requirements are met

7) Deployment and maintenance:
regression testing: updated code test
maintenance testing:

So, testing is conducted almost all phases.

Elements of security testing:

1) Confidentiality → assessing a system's security controls to identify vulnerabilities and weaknesses that could potentially compromise data integrity
2) Integrity → data remains accurate / consistent
3) Authentication
4) Availability
5) Authorization
6) Non-repudiation → ability to prove origin or delivery of a transaction and prevent the involved parties from denying their participation

Q. wholesellen has 3 commodities to sell, 3 types of customen. Discount is given:

(i) For DGS and D ordens, 10% disscount is given innespective of value of onden.

(ii) For ordens more than 50,000, agents get discount of 15%, netailen - 10%.

(iii) For 20,000 to 50000, agents 12%.
netailen 5%.

The above nules don't apply to the funnitune items, where a flat nate of 10% discount is admissible to all customens.

Design test cases for this system using decision table testing:

Ans:

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
C1: DGS/D	T	F	F	F	F	F	F	F
C2: Agent	F	T	F	T	F	T	F	I
C3: Retailen	F	F	T	F	T	F	T	I
C4: O > 50k	I	T	T	F	F	F	F	I
C5: O ≥ 20K to 50K	I	F	F	T	T	F	F	I
C6: O < 20K	I	F	F	F	F	T	T	I
C7: Funnitune	F	F	F	F	F	F	F	T

Pdf 130 page

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
A_1 : dis 5%							X	
A_2 : 8%					X	X		
A_3 : 10%	X		X					X
A_4 : 12%				X				
A_5 : 15%		X						

I → this condition doesn't matter

test case:

Type of customer	funniture	Order value	Result
DGS and D	No	51,000	10% dis
Agent	No	52,000	15%
Retailer	No	53,000	10%
Agent	No	21,000	12%
Retailer	No	22,000	8%
Agent	No	18,000	8%
Retailer	No	15,000	5%
Agent	yes	34,000	10%

PDF-119

A program reads 3 numbers, A, B, C with a range (1, 50) and prints largest number. Design test cases for this program using equivalence class testing technique.

(*) Domain of input as valid and invalid values:

Classes

$I_1 = \{ \langle A, B, C \rangle : 1 \le A \le 50 \}$

$I_2 = \{ \langle A, B, C \rangle : 1 \le B \le 50 \}$

$I_3 = \{ \langle A, B, C \rangle : 1 \le C \le 50 \}$

$I_4 = \{ \langle A, B, C \rangle : A < 1 \}$

$I_5 = \{ \langle A, B, C \rangle : A > 50 \}$

I_6, I_7 (B) $\quad I_8, I_9$ (C)

Test case:

A	B	C	Result	Classes covered
13	25	36	C is greatest	I_1, I_2, I_3
0	13	45	Invalid	I_4
51	34	17	"	I_5
29	0	18	"	I_6
36	42	0	"	I_8
36	42	53	"	I_7
36	54	18	"	I_9

* Another classes:

$I_1 = \{ \langle A, B, C \rangle : A > B, A > C \}$

$I_2 = \{ \langle A, B, C \rangle : B > A, B > C \}$

$I_3 = \{ \langle A, B, C \rangle : C > A, C > B \}$

$I_4 = \{ \langle A, B, C \rangle : A = B, A \neq C \}$

$I_5 = \{ \langle A, B, C \rangle : B = C, B \neq A \}$

$I_6 = \{ \langle A, B, C \rangle : C = A, C \neq B \}$

$I_7 = \{ \langle A, B, C \rangle : A = B = C \}$

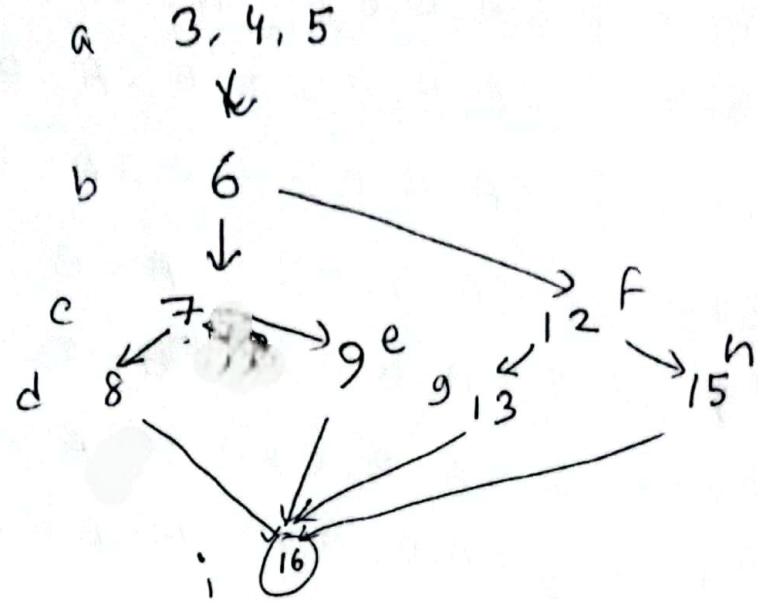
Test case

PDF-191

(10)

```
# Include <stdio.h>
main() {
float x, y, z; clrscn();
print (enter 3 variables)
scan (x, y, z)
if (x > y) {
    if (x > z)
        print (x bono)
    else
        print (z bono)
}
else {
    if (y > z)
        print (y bono)
    else
        print (z bono)
} getch();
}
```

a) DD graph:



b) $n = 9$, $e = 11$, $p = 1$

(1) $\therefore$ complexity $= e - n + 2p = 11 - 9 + 2 = 4$

(2) com = no. of predicate nodes + 1

$= 3 + 1 = 4$

(3) com = no. of regions $= 4$

(4) connection matrix:

	a	b	c	d	e	f	g	h	i	
a		1								1 → 0
b			1			1				2 → 1
c				1	1					2 → 1
d								1		1 → 0
e								1		1 → 0
f							1	1		2 → 1
g									1	1 → 0
h									1	1 → 0

→ CC

final count = 3 ∴ 3 + 1 = 4

Inspection testing: refers to a peer review of sw to identify bugs/defects at the early stages of SDLC.

→ resolve and identify defects earch
→ save time/resource/cost
→ improve productivity

Testing	Debugging
– helps to find bug/error	→ helps to correct the bugs that we found during the testing process
→ detect error	→ resolve error
→ For testing, design knowledge is not required	→ required
→ any body can test	→ only developers can debug
→ manual or automated testing	→ always manual
→ based on different levels of SDLC (unit, integration and system testing)	→ based on different types of bug.

Testing	Debugging
→ ensure if functional and non-functional requirements are met	→ fix defects, errors or issues in the sw system.

Dynamic Testing techniques:

→ All the methods that execute the code to test a sw are known as dynamic testing techniques.

→ 2 types:

① Black-box testing: checks functionalities only
→ functional testing
→ used for system testing under validation

② White-box testing: every design feature and code and path checked and executed logically
→ structural testing
→ used for system testing under verification.

Q 2 number A, B, range (0, 100]
calculate GCD : BVC and robust.

→ **BVC:**

	min − 1	min+ − 2	nom − 50
both A, B:	max − 100	max− − 99	

Test case:

A	B	GCD	A	B	GCD
1	50	1	50	1	1
2	50	2	50	2	2
100	50	50	50	100	50
99	50	1	50	99	1
50	50	50			

→ **robust:**

min → 1	min+ → 2	min− → 0
max − 100	max+ → 101	max− → 99
nom − 50		

Test case:

A	B	GCD	A	B	GCD
1	50	1	50	50	50
2	50	2	50	1	1
0	50	Invalid	50	0	Invalid
101	50	"	50	101	"
99	50	1	50	99	1
100	50	50	50	100	50
			50	2	2

PDF 142

(A, B, C)

(15) input : 3 angle output : determine type

all angles < 90 → acute angled

one angle > 90 → obtuse "

one " = 90 → right angled

Design test case using equivalence class testing technique.

→ classes: (input)

$I_1 = \{<A, B, C> : A > 0, A < 90\}$

$I_2 = \{<A, B, C> : B > 0, B < 90\}$

$I_3 = \{<A, B, C> : C > 0, C < 90\}$

$I_4 = \{<A, B, C> : A = 90\}$

$I_5 = \{<A, B, C> : B = 90\}$

$I_6 = \{<A, B, C> : C = 90\}$

$I_7 = \{<A, B, C> : A > 90, A < 180\}$

$I_8 = \{<A, B, C> : B > 90, B < 180\}$

$I_9 = \{<A, B, C> : C > 90, C \leq 180\}$

$I_{10} = \{A = 0\}$ $I_{11} = \{B = 0\}$

$I_{12} = \{C = 0\}$ $I_{13} = \{A = 180\}$

$I_{14} = \{B = 180\}$ $I_{15} = \{C = 180\}$

$I_{16} = \{A + B + C \neq 180\}$

Test cases:

A	B	C	Result
40	30	110	Obtuse
30	60	90	night
0	40	140	invalid
180	0	0	"
45	55	80	acute
100	80	40	invalid

PDF 142

(11) travel > 50k km per year && pay cash on travel > 8 years → free ticket India

travel < 50k km per year && railway service > 8 years → free ticket India

Design using 'equivalence class testing tech.

→ I_1 = {travel > 50k km}

I_2 = {travel ≤ 50k km}

I_3 = {pay cash}

I_4 = travelling > 8 years

I_5 = train > 8 years

I_6 = {not cash}

I_7 = {pay any}

I_8 = travel < 8

I_9 = train < 8

Test case!

travel km	pay	travel year	railway	free ticket?
≥ 50K	cash	7	7	yes
> 50k	cash	8	8	yes
> 50k	cash	8	7	yes
> 50K	notcash	5	6	no
≤ 50K	cash	8	8	yes
⋮				

Validation Testing Technique example: (5)

1) Unit validation Testing
2) Integration "
3) Function "
4) System "
5) Acceptance "

Software Crisis highlights the challenges of managing the exponentially increasing complexity of modern sw systems. It includes -

1) complexity overload
2) Limited Reusability
3) Absence of standards
4) Lack of Formalization

High Level Design: refers to the overall system, a design that consists description of the system architecture and design and is a generic system design. It includes:

1) System Architecture 2) Database Design
3) Brief description of system, service, platform and relationships among modules.

If we test the whole system directly:

Pros: 1) Comprehensive Coverage ; 2) Efficiency in certain cases ; 3) Early detection of global issues

Cons: 1) Complex Debugging 2) Resource bottleneck 3) Limited Traceability 4) Incomplete Testing Coverage due to time constraints 5) More risk

→ suitable for small projects

"Nested loops are problematic areas for testers".

1) increase complexity - large no. of combinations of loop iteration

2) Increased execution paths

3) More time and memory

4) Achieving complete path coverage is difficult.

5) Less readable
6) more challenging to maintain
7) complex error-handing mechanisms are needed

B CMMI (Capability Matunity Model Integration)
→ a framework that provides organizations with guidance on how to improve their processes and enhance the quality of their products and services.
→ 5 matunity level (low-high)
1) Initial 2) managed 3) Defined
4) ~~Qualit~~ Quantitavely managed
5) Optimizing

To move from CMMI level 3 to 5, an organization needs to focus on achieving the optimizing level. This involves:
1) Quantitative process management
2) continuous process improvement
3) Defect prevention
4) Technology change management
5) organizational Innovation and deployment

SW crisis example: Therac-25 → a computer controlled medical linear accelerator used for radiation therapy → cancer patient treatment.

* 6 patients died → overdose of radiation
* mid-1980s

Reason:
1) SW bugs → magnitude of radiation higher than intended.
2) Lack of proper testing
3) Inadequate SW engineering practice
4) Insufficient UI design.
5) Inadequate Documentation
6) Untrained operators

Failure: inability of a SW system or components to perform its required functions within specified performance requirements.

Defect reaches → end customer

Ex: //runtime failure
int a[5]; print (a[10])

→ not behave as expected

// Functional failure

func add_num(a,b)
return a * b;

Bug: Unexpected problem with sw that was not anticipated by developer
→ login button not working

Error: flaw in code that may lead to unexpected or incorrect behavior.

1) Syntax error // print "Hello"
2) Logical error // int a = 10/0;

SDLC: cost-effective and time-efficient process that development teams use to design and build high-quality sw.

→ minimize risks
→ expected requirements are met.
→ structured process used by sw developers to design, build, test, deploy, maintain sw system.

Ex: Developing an e-commence website

1) Requirements: identify products, shopping cart, checkout; collect input on desired product categories and payment options.
2) Design: user-friendly interface, database and structure plan.
3) Implementation: code, technology
4) Testing: Ensure smooth shopping experience
5) Deployment: launch, accessible for customers

6) **Maintenance:** addness/fix ~~fix~~ issues
update features

STLC (Software Testing Life Cycle)

→ a process used to test sw and ensure that quality standards are met.

→ may be developed multiple times until a product is suitable for release.

Ex: 1) Requirement Analysis
2) Test Planning
3) Test case Design
4) Test Environment setup
5) Test execution
6) Defect reporting
7) Regression testing
8) Test closure
9) Release and Deployment testing

(*) Decision table formation:

		Entry		
		Rule 1	2	3
Condition stub	C1	T	T	F
	C2	T	F	F
	C3	F	T	T
Action stub	A1		X	
	A2	X		
	A3			X

(4) components

(1) **Condition stub**
→ input condition list
→ complex combo of input

(2) **Action stub**
→ output list

(3) condition entry: - has rules, particular combo of inputs (T/F)

(4) Action entry: X

Diffenence between white and black-box testing.

Black	White
Knowledge about Intennal code structure is hidden/not needed	~~not~~ needed
code implomentation knowledge not needed	
functional test of sw	structural test of sw
done by sw testers	done by sw developers
behavior testing	logic testing
Ex: search something on google using keywords	Ex: input check and verify loops
Techniques:	Techniques
1) Boundary Value Analysis	1) Basis Path Testing
2) Equivalence class Testing	2) Loop "
3) Decision Table Based "	3) Data flow "
4) state " " "	4) mutation "
5) Cause-Effect Graphing Based Testing	

Waterfall Model:

traditional and linear sequential approach to sw development, where progress is seen as flowing steadily downwards through phases like a waterfall.

→ each phase must complete → before moving to next

→ once a phase is complete → changes in requirement not accepted.

6 phases: Requirements, Analysis, Design, Coding, Testing, Deployment.

Characteristics

1) sequential and linear: one full complete → go to next

2) Rigidity

3) well-Defined Phases: goals is specific

4) Detailed Documentation

5) Suitable for small projects/stable requirements

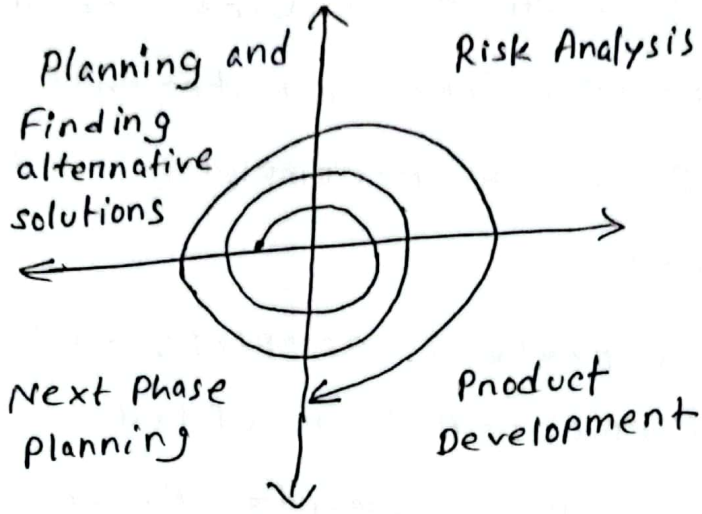
Pros

1) Clean and well-structured process
2) Easy to understand/use
3) suitable for small projects

Cons

1) Inflexibility in changes
2) Long time for delivery
3) If requirements are not understood well initially
high risk of customer dis-satisfaction

Spinal model: 4 basic activities
Planning, risk analysis, engineering, evaluation.



Why process of underlying sw be written?

→ code documentation needed

Reason: 1) understand purpose, functionality and structure of code

2) It's a communication tool within dev team.

3) New team members can know about codebase

4) Assists in debugging / troubleshooting

Software Quality Assurance: ongoing process that ensures the sw product meets and complies with the organization's established and standardized quality specifications.

→ assure quality in sw

→ Process which works parallel to sw development.

In path-based integration

Source node : A source node in a program is a statement fragment at which program execution begins or resumes.

Sink Node: A sink node in a program is a statement fragment at which program halts or terminates.

How to verify a code?

→ ① check that every design specifications in high-level and low-level design has been coded using traceability matrix

② check misunderstood/incorrect arithmetic precedence

③ check mixed mode operations

④ check incorrect initialization, loop, exit etc.

2 techniques: ① static ② Dynamic testing

Static testing: a type of testing that's performed on a piece of sw without executing the actual code.

→ process of checking code
→ " " designing document
→ " " checking requirement
→ not executing code

Steps of inspection process : 6

Goal: detect bugs

1) Planning : Plan for meeting
2) Overview : Define purpose
3) Individual preparation: Identify potential error
4) Inspection meeting : item is inspected
5) Rework : Reported bugs are fixed
6) Follow-UP : planning and overview

Logic coverage criteria in white-box:

1. Statement coverage:
2. Decision or branch coverage
3. condition coverage
4. Decision/condition coverage
5. Multiple condition coverage

Equivalence Class Testing : a method for deriving test cases wherein classes of input conditions called equivalence classes are identified such that each member of the class causes the same kind of processing and output to occur.

→ no need to test every input

→ only execute one test case from each partitioned class.

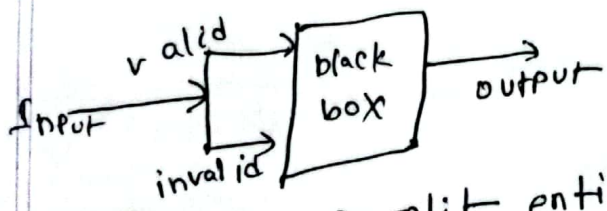
→ complete testing, non-redundant

Identification of equivalent class:

2 types of class :

1) valid equivalence class : valid inputs

2) Invalid " " : consider invalid inputs that'll generate error conditions or unexpected behavior.



Guideline: 1) split entire range into 2 or more class.

2) if handles differently → define 1 valid eq. class per valid input

3) BVA : if $0 \le a \le 100$: valid class : $0 \le a \le 100$
invalid : $a < 0$ and $a > 100$

4) For 1 input variable, more than 1 class : → 3 class
if input is a character, it can be alpha, num, special char.

Types of integration testing :

1) ~~Decou~~ Decomposition based testing

2) call Graph - based Integration

3) Path - based integration

Ramp - up period : period of time between when the test begins and the time when it reaches level.

→ time between initial development and maximum capacity utilization of a product.

Properties of CFG:

1) process oriented

2) shows all the paths that can be traversed during a program execution.

3) directed graph 4) has nodes and edges

Defect an error, flaw, failure or fault in a computer program that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

How Driven class helps in unit testing?

→ ① isolates the module under test from the rest of the system

② acts as a substitute for the actual calling module that interacts with testing module

③ generates controlled input data

④ captures output data

⑤ stimulates the execution of module

⑥ provides a consistent and repeatable testing environment

⑦ early detection of issues

Building team for static process:

1) Author/Owner/Producer :- produce program/document
- fix defects

2) Inspector → not manager/supervisor
→ not directly directed to program
→ finds error/omission/inconsistency

3) moderator: schedules, leads, controls sessions
→ plan/execute inspection

4) Recorder: record all result

→ test Driven Development

Benefits of TDD over ~~on~~ traditional unit testing:

→ **TDD**: approach in sw development where tests are written before the actual code is implemented

In **unit** test: tests written after code development

→ ①

TDD	Traditional Unit
① tests written before code implementation, so early detect defects. If the code doesn't pass initial test, it's not integrated into the system	① Defects are discovered after code has been developed. More tough to identify / fix issues early.
② focus on writing modular and loosely coupled code	② write tests based on existing code
③ optimal design	③ not optimal
④ Higher test coverage	④ lower.
⑤ Quickly find regression	⑤ not find
⑥ faster debugging	⑥ slower.

Types of test plan: ③

1) master: overall plan, roadmap and STLC outline.
2) Phase: focus on one sprint at a time
3) specific: details of 1 functionality and how it's tested

Incremental Integration testing is preferred over non-incremental.

→ integrating and testing units/modules one by one, starting from basic to complex/dependent.

pros: 1) identify/fix errors early.

2) Isolate faults easily.

trace them back to specific module

3) monitor the progress of integration process.

4) Integrate and test small units at a time reduces the complexity of the testing process.

5) Different teams/developers work on different components concurrently. (In non-incre, sequential)

6) Easier Debugging

In non-incremental, there may not be a clear picture of functionality of each module until they're all integrated and tested.

Rapid Prototyping: agile strategy in product development, involving the creation and testing of 3-D prototypes for optimizing shape, size and usability. It's a dynamic process that utilizes multiple iterations based on user feedback and analysis.

→ useful when the requirements are not well-understood or are likely to change during the development process

→ offers a flexible and collaborative approach.

→ repeats development, evaluation and feedback cycle iteratively.

→ each iteration aims to enhance the prototype based on the evolving understanding of user requirements.

Effective	Exhaustive (complete)
1) Targets critical and high-priority aspects.	1) Aims to cover all possible combinations.
2) optimize resource	2) not optimize
3) time - less	3) time - more
4) may not detect rare defects	4) uncover even the rarest defects.
5) more scalable	5) less scalable
6) not impossible, but hard to implement	6) nearly impossible
7) some defects may go undetected	7) detect all defects.