# Software Testing.

## Goals:

(1) to show that the software performs as expected

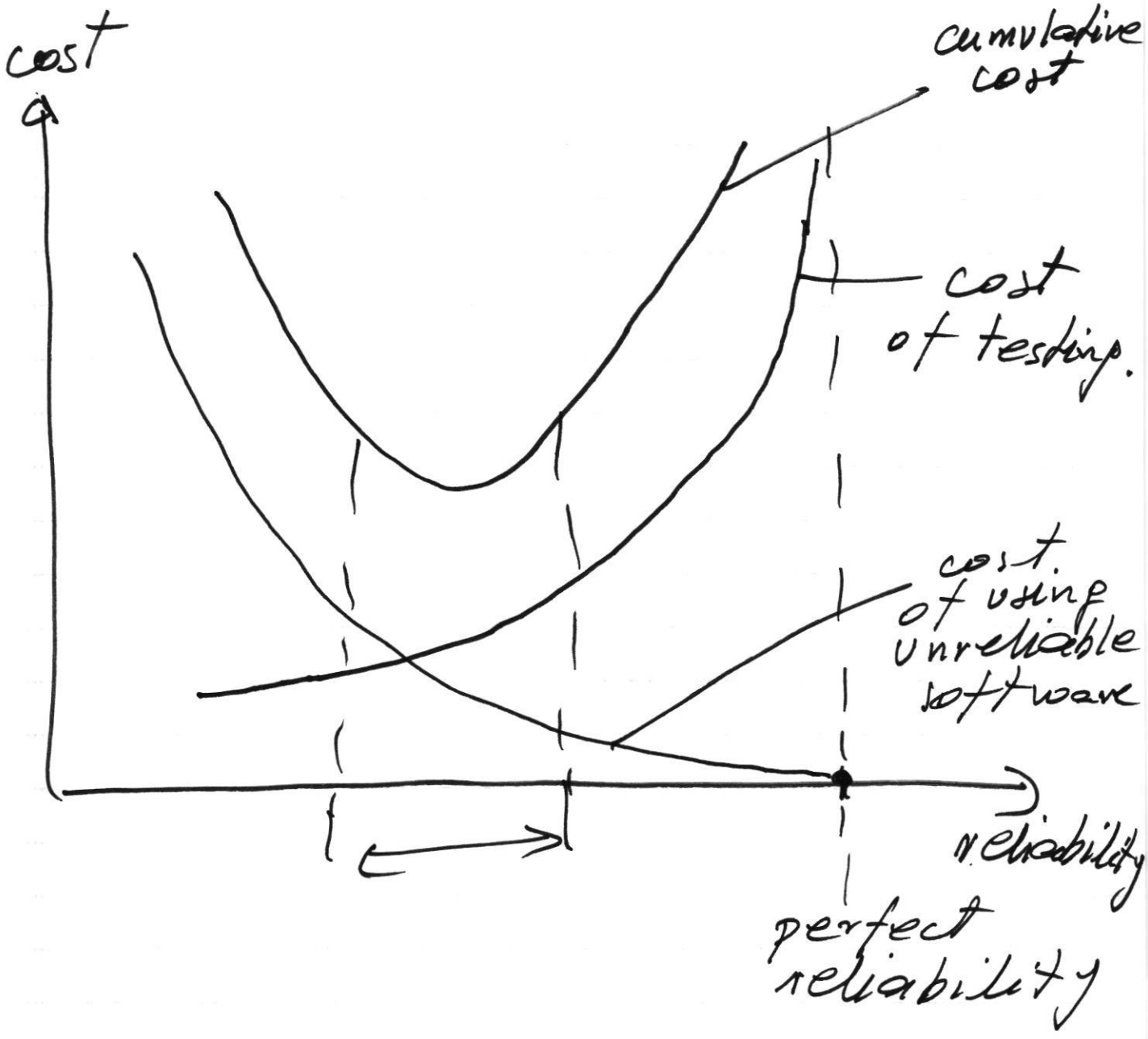
(2) to have "confidence" in software system

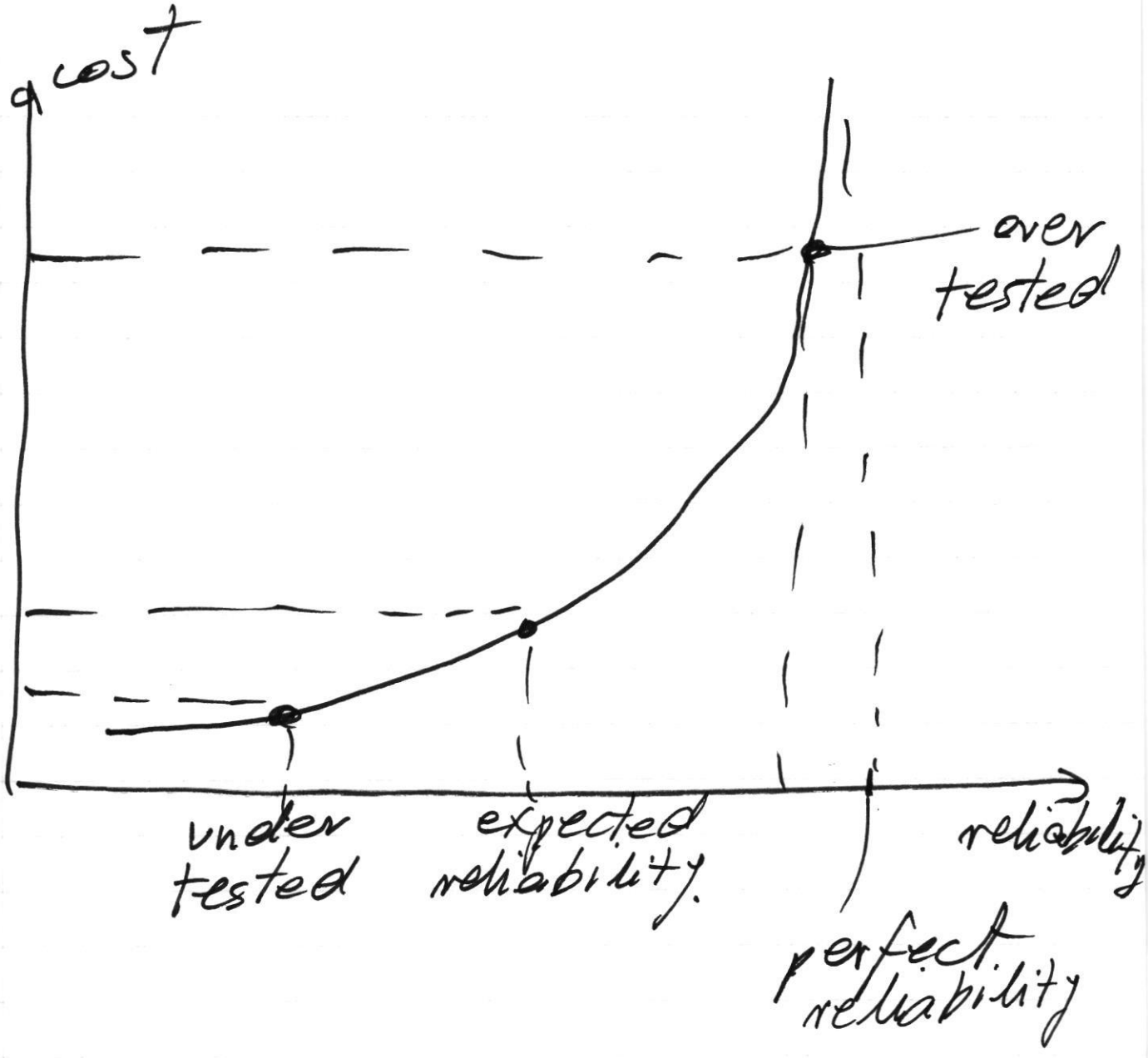
⇓

to make sure that the software does not contain any "major" defects.

Testing.

Software is the most expensive activity in software development and maintenance.


cost
cumulative cost
cost of testing
cost of using unreliable software
reliability
perfect reliability


cost
over tested
under tested
expected reliability
reliability
perfect reliability

Developers/managers want to have "confidence" in software system

* Testing should help to gain this confidence
  * using testing methods
  * hire independent testing team
  * ...

# Goals of testing.

Design a set of test cases that have a high probability of detecting defects with "minimum" amount of effort and time

"high quality" test cases.

# Basic facts about testing.

1. Testing can only show a presence of a defect

2. Testing can never show the absence of defects

3. No matter how much testing is done, it can never be guanteed that all defects are detected.

# "Perfect" testing.

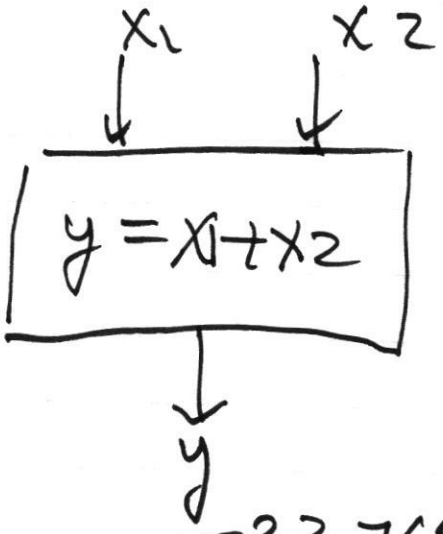
## "Exhaustive" testing.

Test the system on all possible test cases/inputs.

# of tests = $\infty$

it is not practical.

x1, x2: integer




$-32,768 \leq x_1, x_2 \leq +32,767$

$\#\text{ of tests} = (65,536)^2$

x1, x2: long integer

x1, x2: float

# Types of testing.

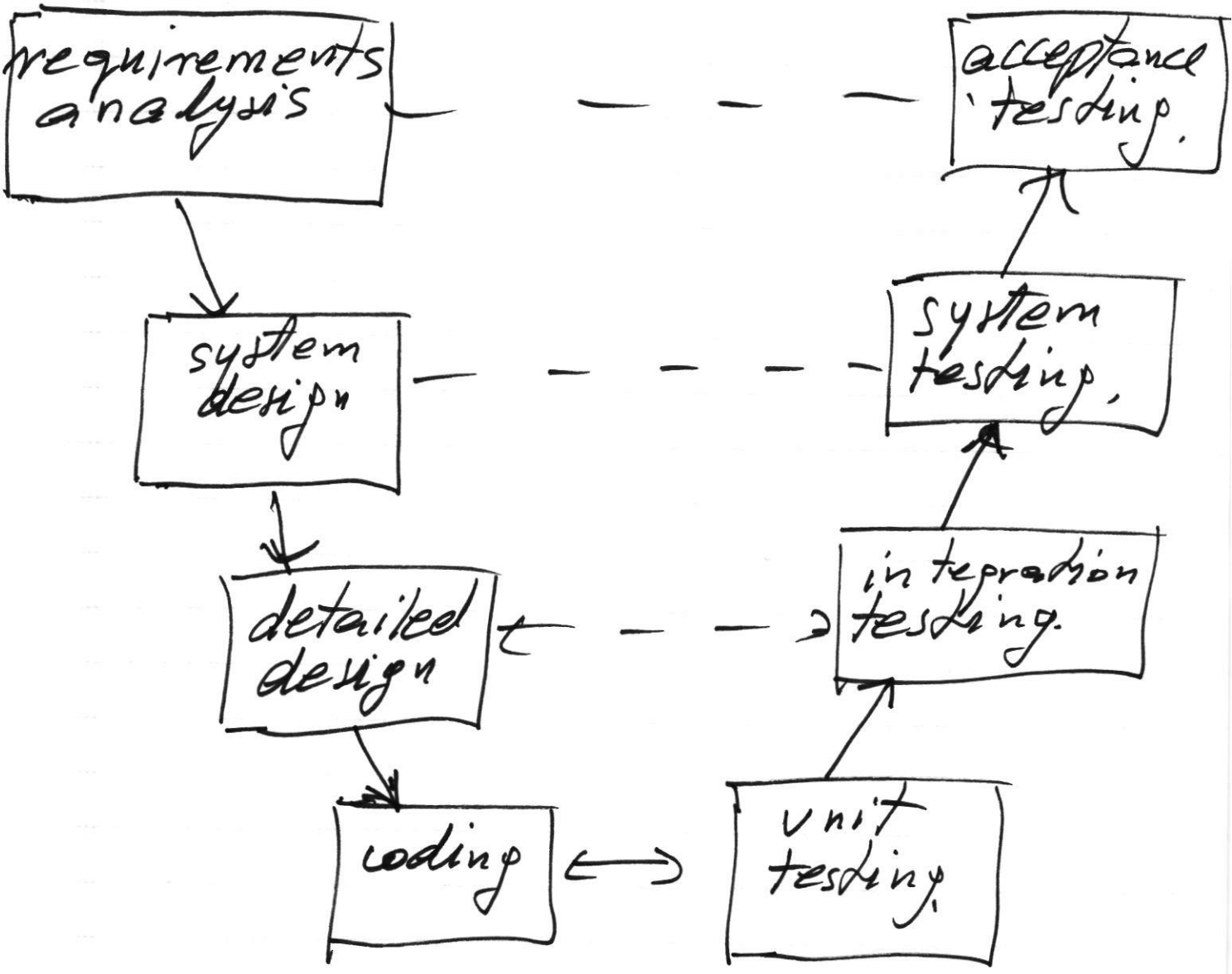
1. Dynamic testing
   (execution-oriented testing)
2. Static testing/analysis
   (no execution is involved)

---

# Testing on different levels

* unit level testing
* integration testing.
* system testing.
* acceptance testing.

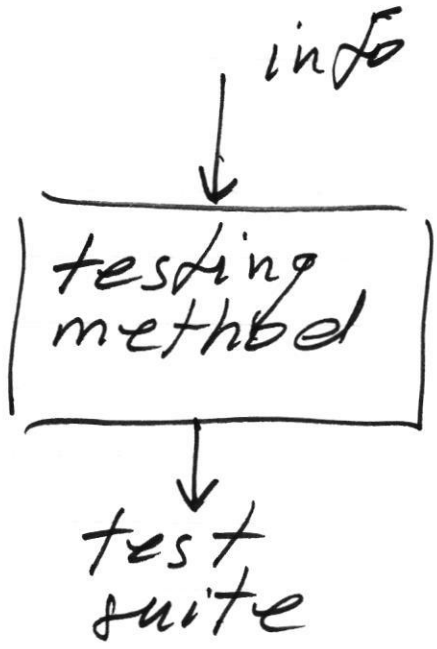
# V - model


requirements analysis
acceptance testing
system design
system testing
detailed design
integration testing
coding
unit testing

# Types of testing.

* testing functionality.
* performance testing.
* stress — " —
* security — " —
* robustness testing.
* ...
* ...

# Testing functionality.




A set of test cases

Test case:

1. input
2. expected output/outcome

source of information that can be used by testing methods!

1. specification
2. source code
3. no information (random testing)

1. Specification-based testing methods

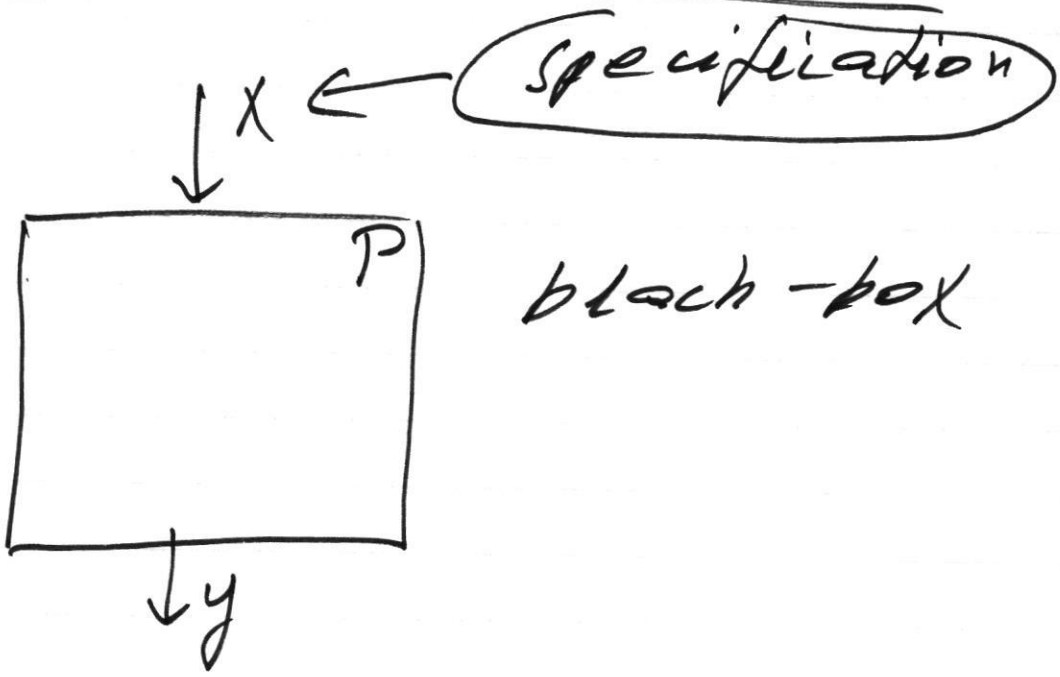
(black-box testing)
(functional testing)

2. code-based testing methods

(white-box testing)
(structural -"-)

3. Random testing.

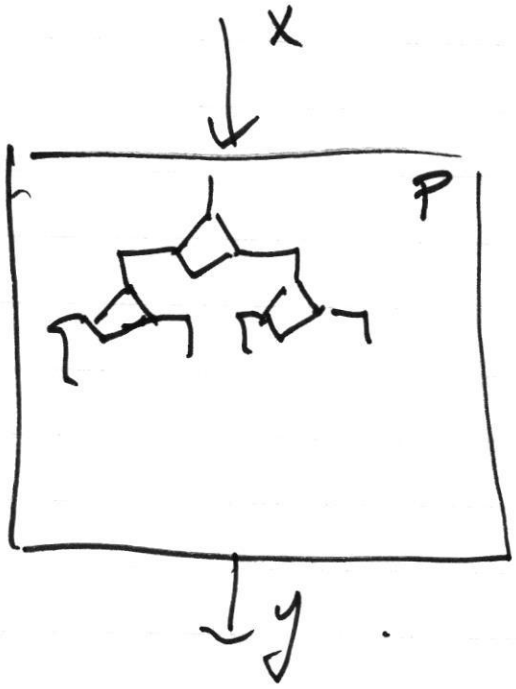
# Specification based testing.




black-box

---

# code-based testing.




source code

All these testing methods should be used.

A test suite should contain

(1) specification-based tests

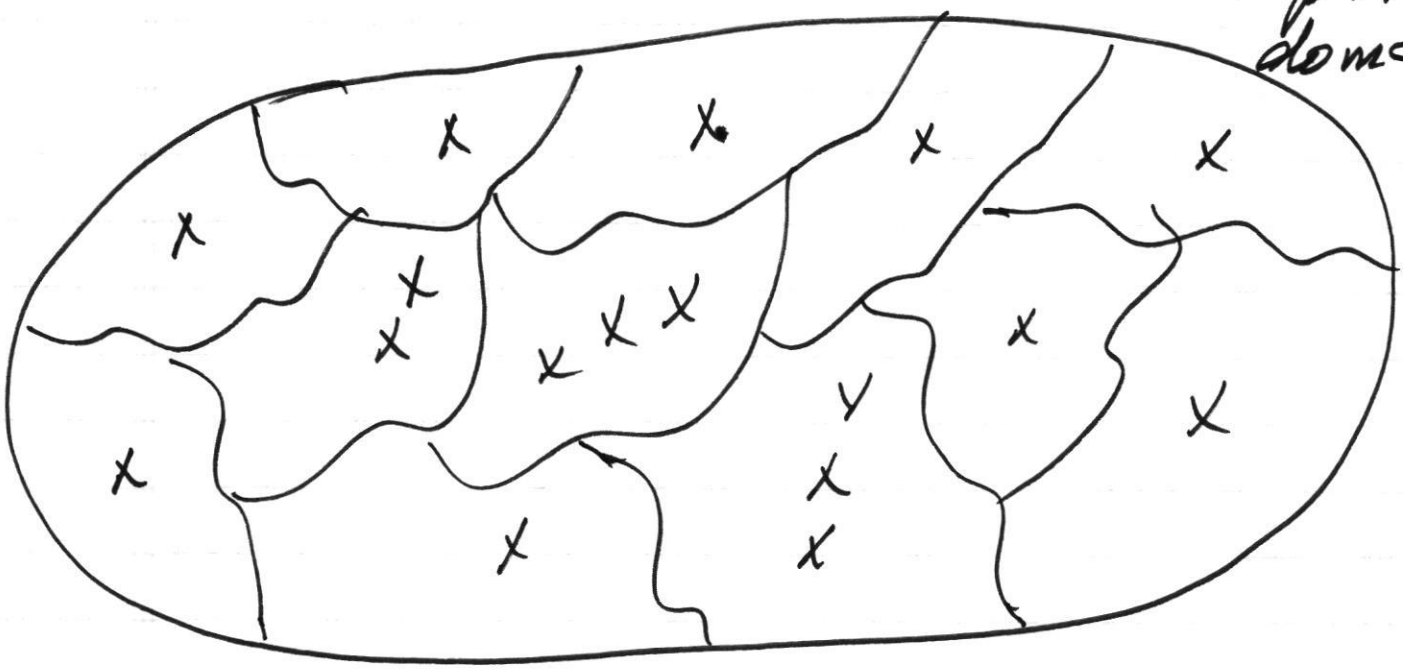
(2) code-based tests

(3) random test (2-5%)

# specification-based testing methods

1. equivalence partitioning.
2. boundary value analysis testing.
3. decision-table based testing.
4. cause-effect graphing.
5. combinatorial testing.
6. model-based -11-
7. . . .

# Equivalence partitioning.

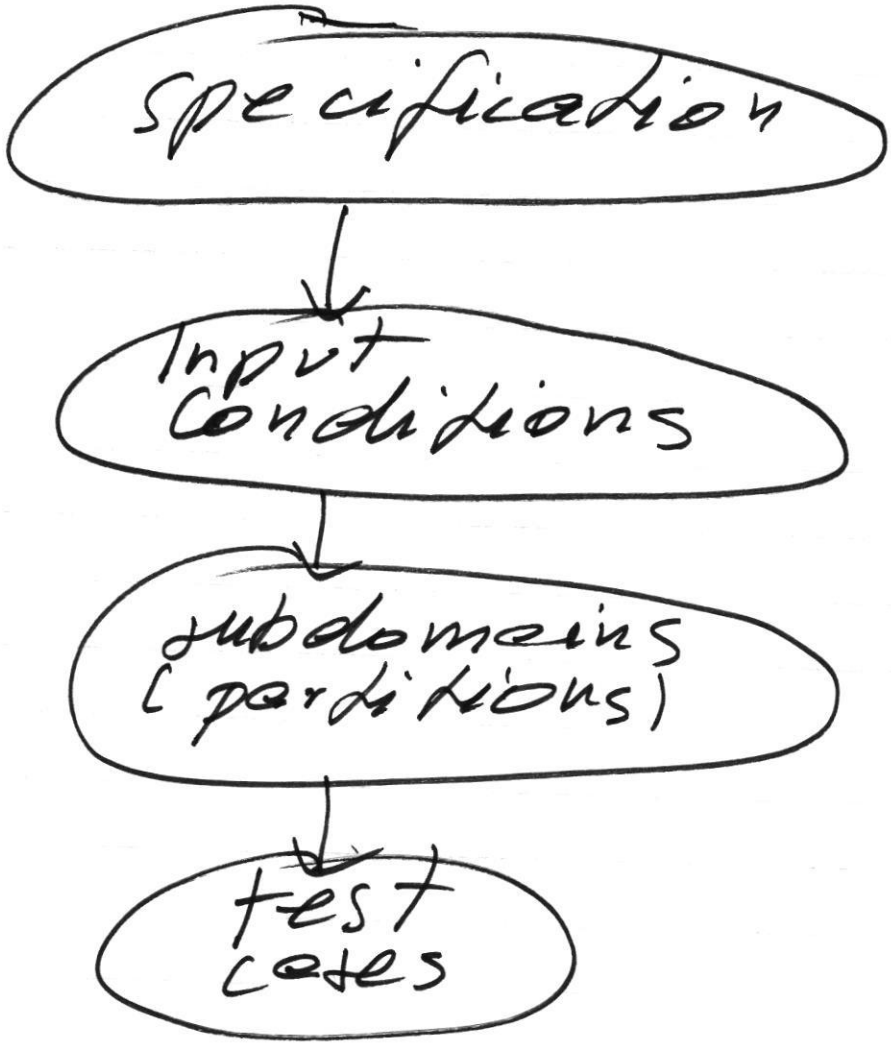



Input domain is divided into finite number of subdomains (equivalence partitions)

A sub-domain (equivalence partition) contains a set of inputs that have a common property.

Test cases are selected from each sub-domain.

subdomains are derived from "input conditions" that need to be identified from the specification.



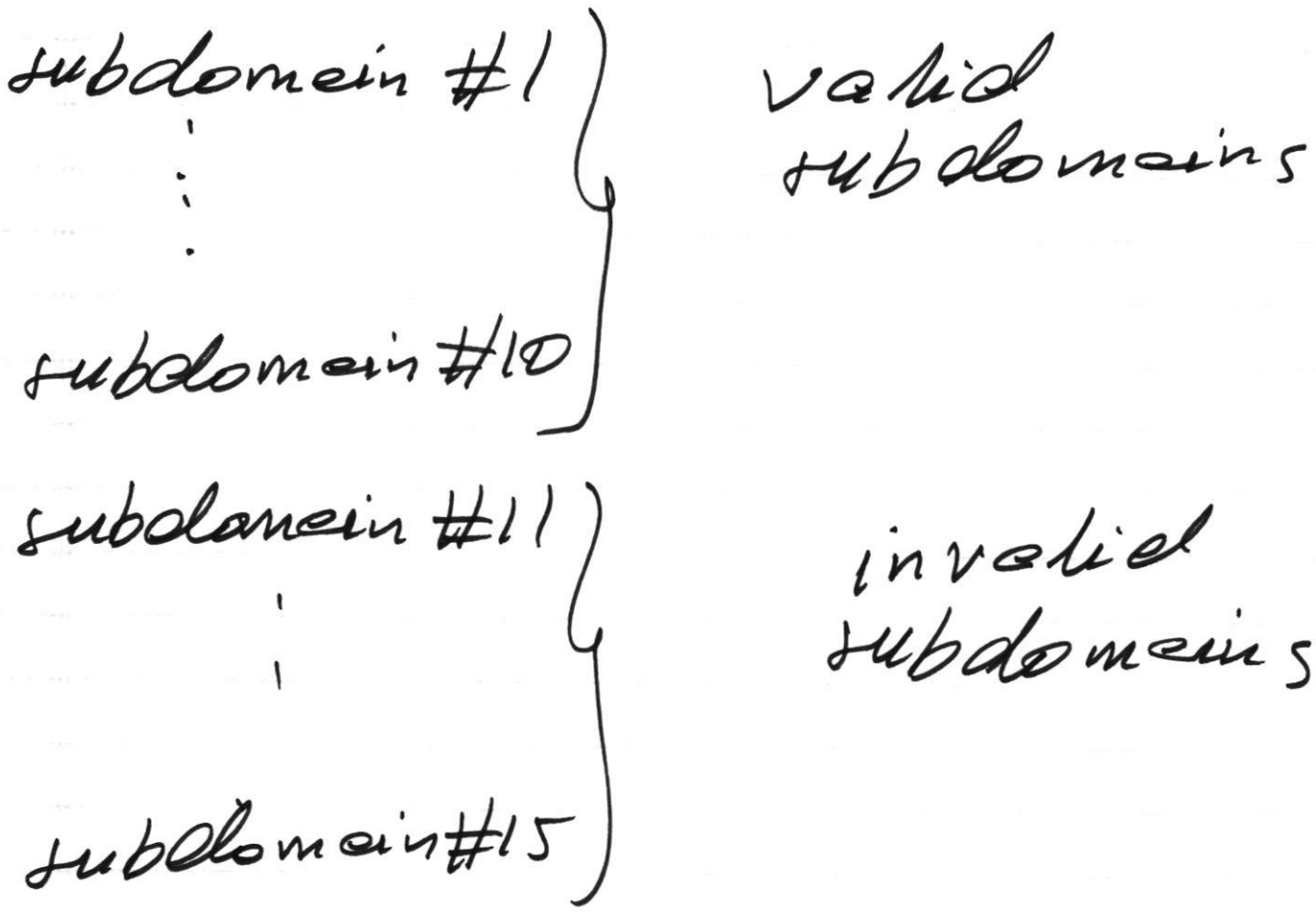
# Given input condition

Two types of subdomains.

1. valid subdomain(s)
   contains inputs that satisfy the input condition

2. invalid subdomain(s)
   contains inputs that violate the input condition

## Input condition




## Test case selection:

select at least one test case from each subdomain

# Types of Testing

1. Normal Equivalence testing

   selecting tests from valid subdomains

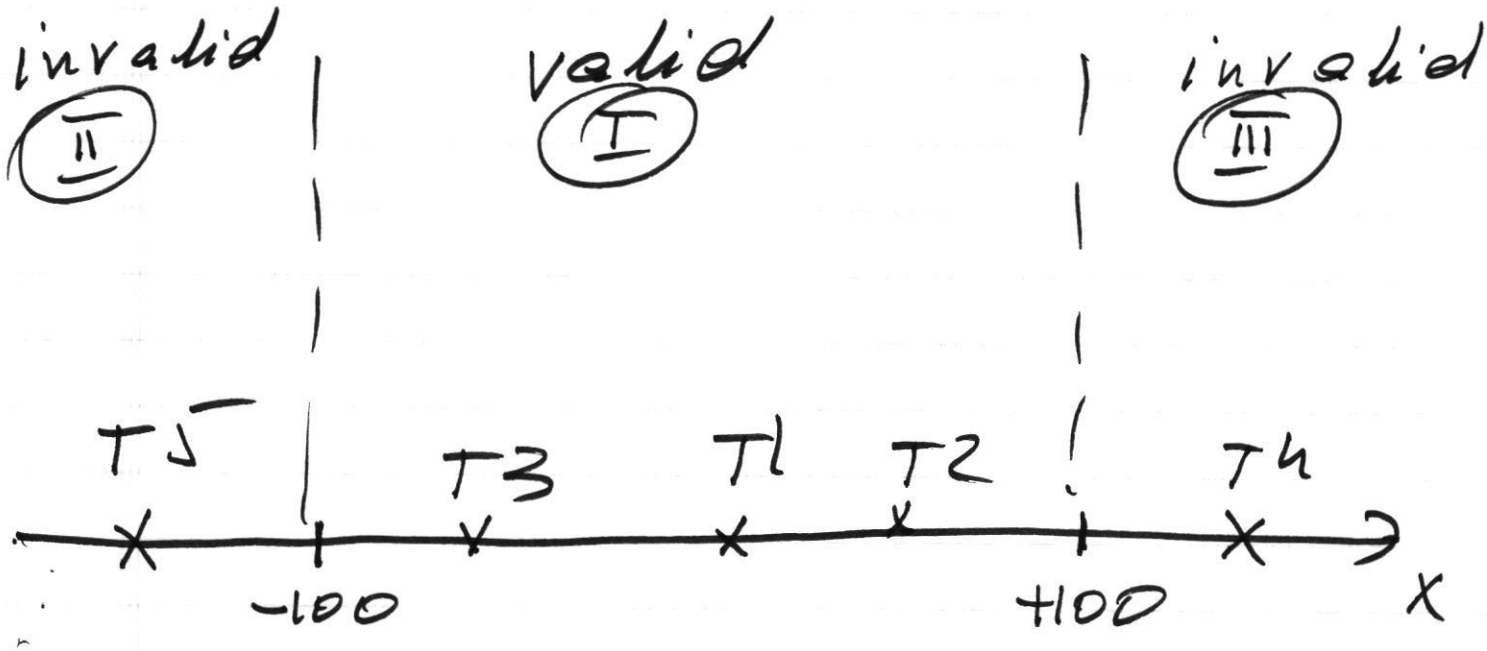
2. Robust Equivalence testing.

   selecting tests from invalid subdomains

# input condition
## range of values

Program input: x : integer

Input Condition: $-100 \leq x \leq 100$




## 1. Normal Eq. tests

T#1: x = 5

T#2: x = 75

T#3: x = -51

## 2. Robust Eq. tests

T#4: x = 120

T#5: x = -115

(1) valid subdomain:

several normal eq. tests

(2) invalid subdomain

one robust eq. test is sufficient.