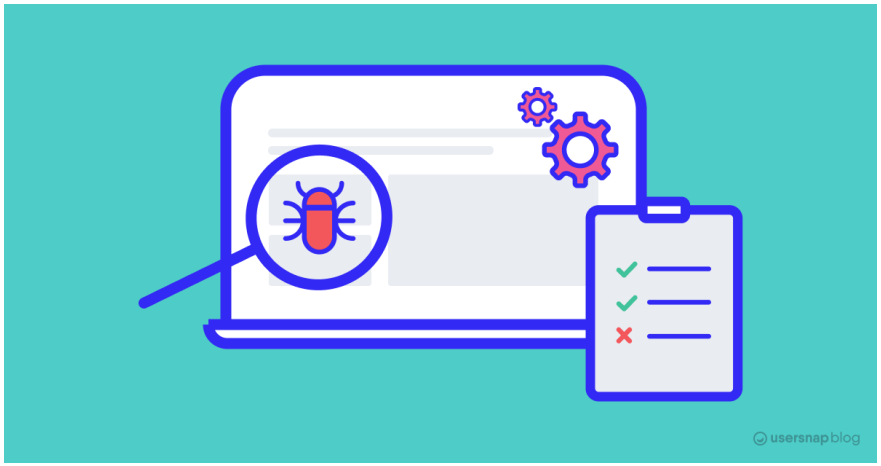
Software Testing Techniques



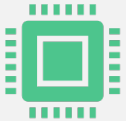
Modified from

1. Roger S. Pressman, Software Engineering: A Practitioner's Approach 8th Edition, McGraw Hill, 2014

2. Software Testing A Craftsman's Approach, Fifth Edition, CRC Press, 2021

Software Testing Fundamental

Software testing is a critical element of software quality assurance



Software development organization spends 30-40 percent of total project effort on testing

Human-related software or Safety Critical software spends 3-5 times as much as all other software engineering activities combined

Testing Objectives [Myers 1979]

"Testing is the process of executing a program with the intent of finding errors"

A good test case is one that has high probability of finding an as-yet undiscovered error

A successful test is one that uncovers an as-yet undiscovered error

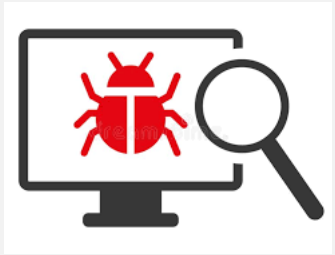
Testing Definition

- Hetzel's definition:
  - "Testing is the process of exercising or evaluating a system by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results"

Definition

Error (Mistake)

- a mistake made by a software developer or a misconception



Fault/Defect/Bug

- the manifestation of an error in the software or its documentation
- is a result (representation) of an error

Our Testing Objectives

To design tests in order to uncover different classes of errors with minimum amount of time and effort

Testing cannot show the absence of defects, it can show that software errors are present

Testing Principles

- All tests should be traceable to customer requirements
- Tests should be planned long before testing begins
- Testing should begin "in the small" and progress toward "in the large"
- Exhaustive testing is not possible
- To be most effective, testing should be conducted by an **independent third party**

Who Tests the Software?

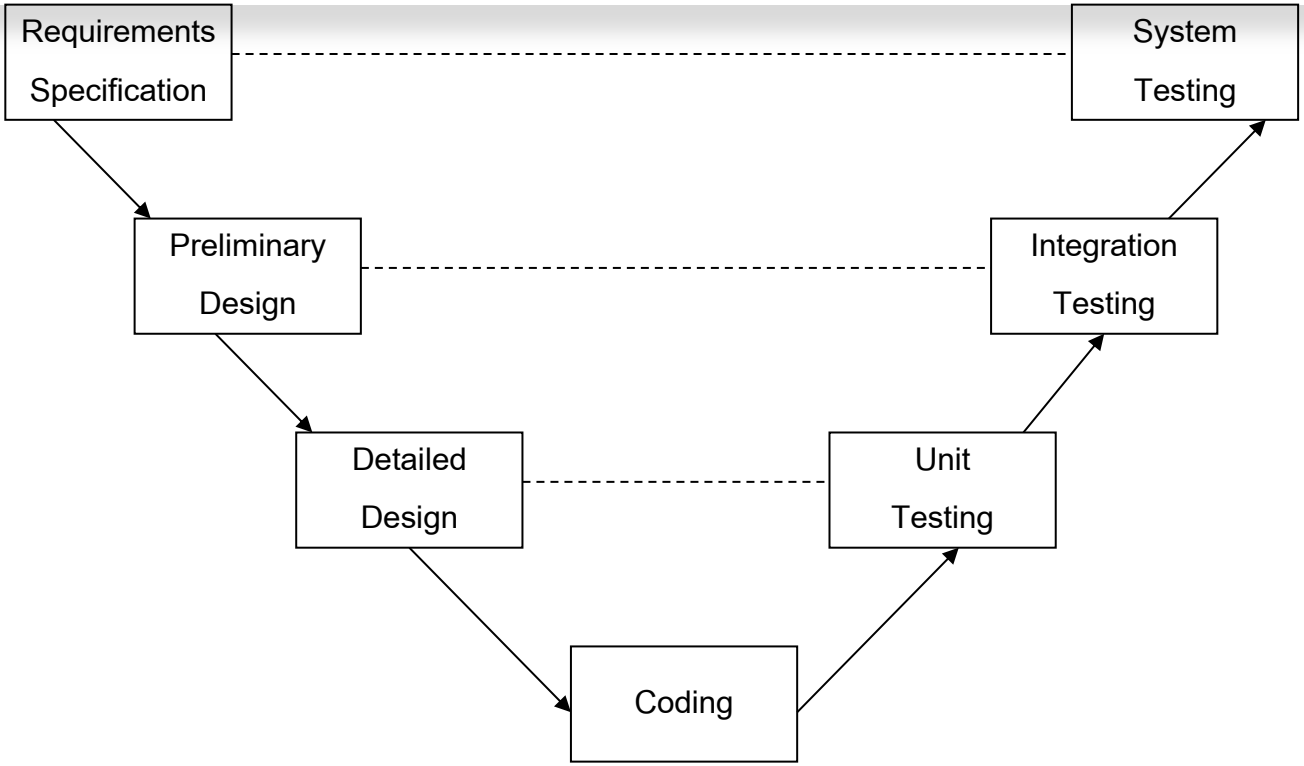
Developer

- Understand the system, but will test "gently and, is driven by "delivery"

Independent Tester

- Must learn about the system, but will attempt to break it and, is driven by quality

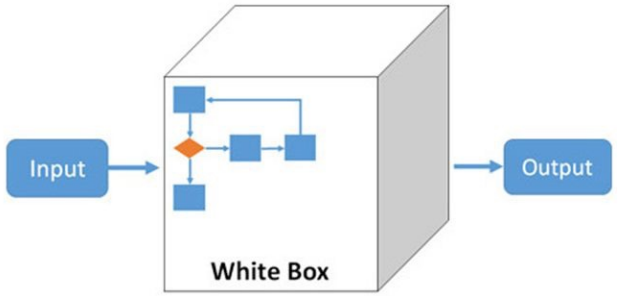
Levels of Testing



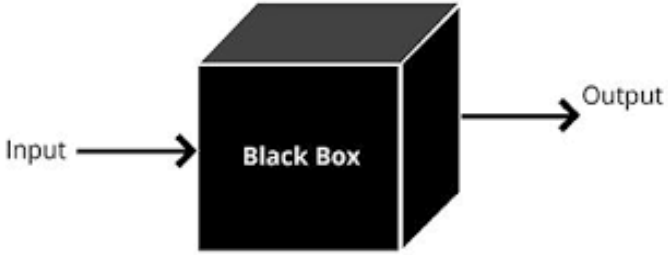
Test Case Design

- **2 approaches**:

➢ **White-Box Testing**

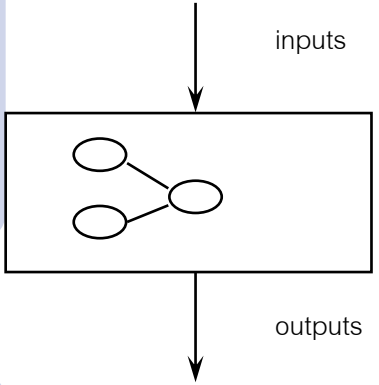


➢ **Black-Box Testing**



White-Box Testing

is also called glass-box or structural testing



uses the control structure of the procedural design to derive test cases

White-Box Testing (cont.)

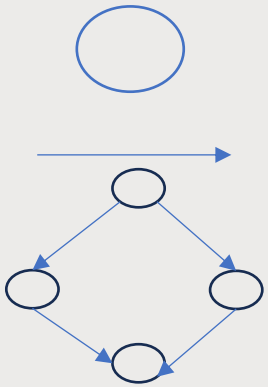
- Test cases are derived so that
  - all statements are exercised
  - all logical decisions (both true and false) are exercised
  - all independent paths are exercised at least once
  - all loops are executed
  - internal data structures are exercised

White-Box Testing (cont.)

- **Coverage criteria**
  - **Statement coverage**: requires every statement to be executed at least once
  - **Branch coverage**: requires each branch to be traversed at least once
  - **Path coverage**: requires all possible control flow paths through the program from entry to exit is executed once

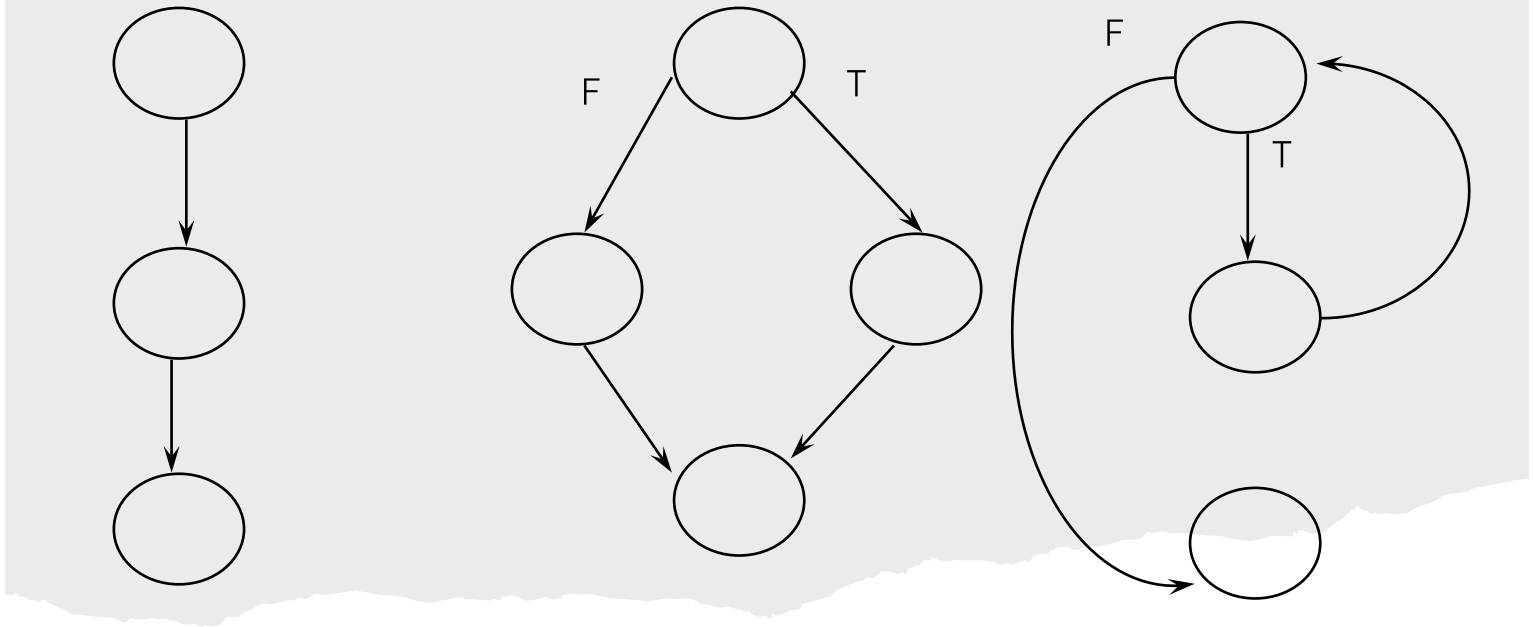
White-Box Testing (cont.)

- use control flow to derive test cases and to measure test coverage
- control flow notation:



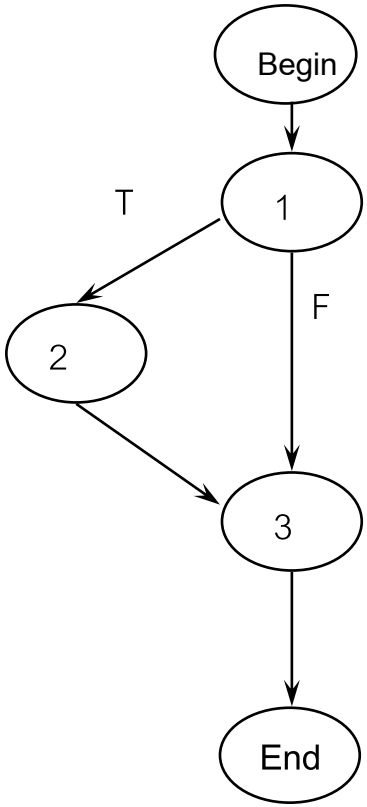
- **circles (nodes)** : program statements
- **arrows (edges)**: flow of control (edges must be terminate at a node)
- **region**: bounded by edges and nodes

Control Flow Graph of 3 Prime Structures



Sequence Condition Iteration

White-Box Testing: Statement Coverage

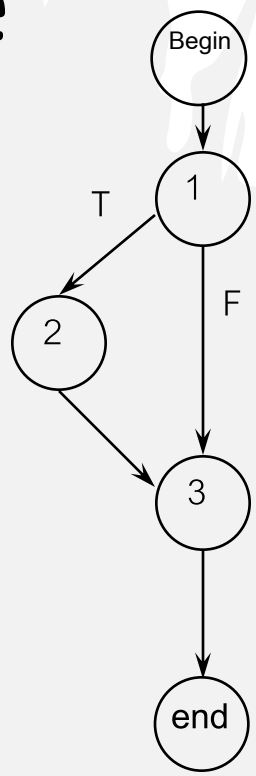



```
function abs (x:int): integer
begin
 1   if x < 0 then
 2      x := 0-x
 3  abs := x
end
```

test case x = - 1 satisfy statement coverage

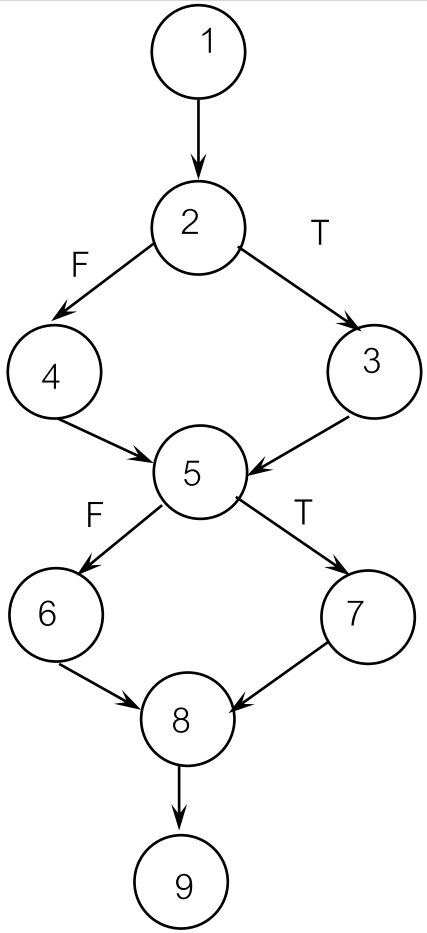
White-Box Testing: Branch Coverage

```
function check (x:integer) boolean
begin
   1  if (x >=0) and (x <= 100)
   2  then check = true
   3  else  check = false
end
```



test case x = 5 and x = -5 satisfy branch coverage

White-Box Testing: Coverage Criteria



100% Statement coverage

A: 1-2-4-5-6-8-9

B: 1-2-3-5-7-8-9

100% Branch coverage

A: 1-2-4-5-6-8-9

B: 1-2-3-5-7-8-9

100% Path coverage

(all possible path)

A: 1-2-4-5-6-8-9

B: 1-2-3-5-7-8-9

C: 1-2-4-5-7-8-9

D: 1-2-3-5-6-8-9

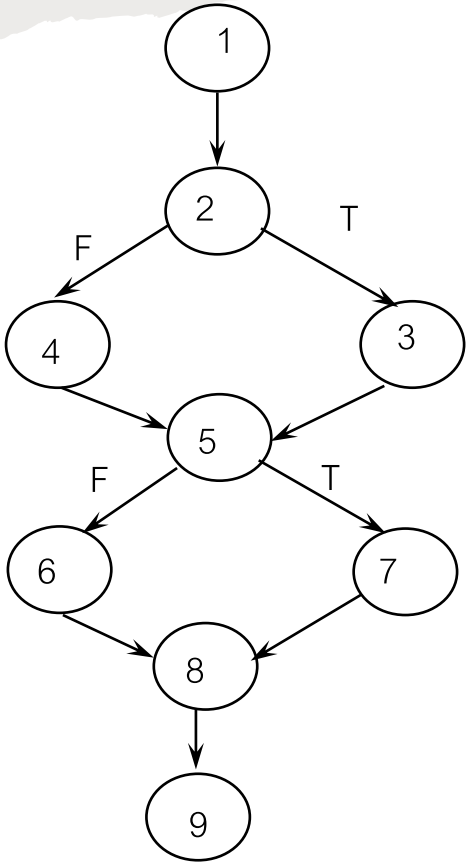
White-Box: Basis Path Testing

- **McCabe's Cyclomatic Complexity (C)**
  - will put a lower bound on number of independent paths (an independent path must move along at least one edge that has not been traversed before path is defined)
  - C = #edges -#nodes + 2
  - C = #regions + 1

Basis Path Testing

- was proposed by Tom McCabe in 1976
- generate program control flow graph
- calculate McCabe's cyclomatic complexity (C or V(G))
- generate **C** independent paths
- we derive **C** test cases

Basis Path Testing (cont.)



C = 10 - 9 + 2 = 3

or C = 2 + 1 = 3

generate 3 independent paths

- Path 1: 1-2-4-5-6-8-9
- Path 2: 1-2-4-5-7-8-9
- Path 3: 1-2-3-5-7-8-9

White-Box: Loop Testing (Simple Loop)

- Simple loop with n passes, derive test cases to test the loop with
  - no iterations of the loop (skip the loop)
  - one pass
  - two passes
  - n, n + 1, n - 1

White-Box Testing (cont.)

- Random Testing
  - require tools to generate large volumes of test inputs
  - needs parameters to guide input generation
  - does not guarantee coverage criteria
  - may not exercise all of code

A SIMPLE LOOP EXAMPLE*

```
for (var counter = 1; counter < 5; counter++) {
    console.log('Inside the loop:' + counter);
}
console.log('Outside the loop:' + counter);
```

*https://www.javascripttutorial.net/javascript-for-loop/

THE FOR LOOP WITHOUT THE INITIALIZATION PART EXAMPLE*

```
var j = 1;
for (; j < 10; j += 2) {
    console.log(j);
}
```

*https://www.javascripttutorial.net/javascript-for-loop/

THE FOR LOOP WITHOUT THE CONDITION EXAMPLE*

```
for (let j = 1;; j += 2) {
    console.log(j);
    if (j > 10) {
        break;
    }
}
```

*https://www.javascripttutorial.net/javascript-for-loop/

THE FOR LOOP WITHOUT ANY EXPRESSION EXAMPLE*

```
// initialize j variable
let j = 1;
for (;;) {
    // terminate the loop if j is greater than 10;
    if (j > 10) break;
    console.log(j);
    // increase the counter j
    j += 2;
}
```

*https://www.javascripttutorial.net/javascript-for-loop/

THE FOR LOOP WITHOUT THE LOOP BODY EXAMPLE*

```
let sum = 0;
for (let i = 0; i <= 9; i++, sum += i);
console.log(sum);
```

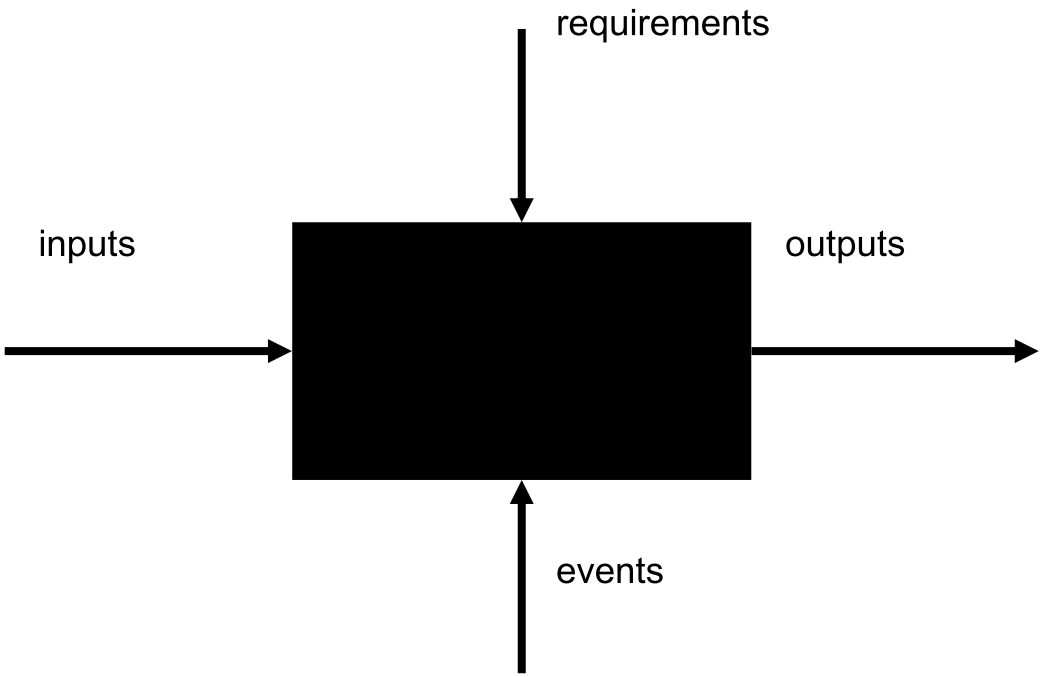
*https://www.javascripttutorial.net/javascript-for-loop/

HOMEWORK: JAVASCRIPT SWITCH STATEMENT*

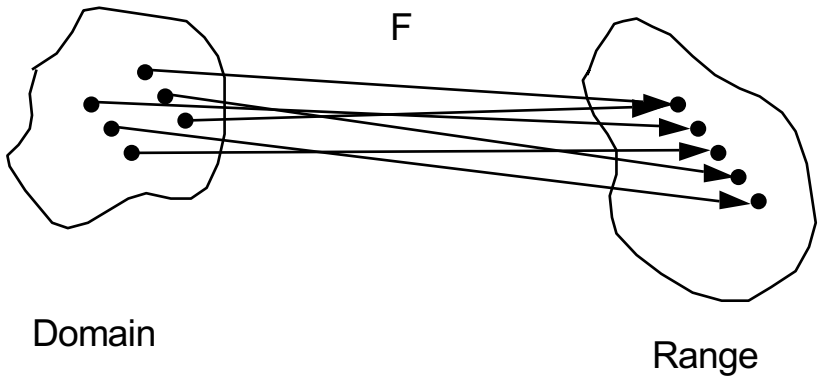
```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

*https://www.w3schools.com/js/js_switch.asp?fbclid=IwAR1E7IoPyqj9MHKdYUTCFNcbS72rjeQHPs0A12n2woSL0ddbyu2xvzdlcdU

Black-Box Testing (Functional Testing)



Black-Box Testing (Functional Testing)



The rationale for referring to specification-based testing as “functional testing” is likely due to the abstraction that any program can be viewed as a mapping from its Input Domain to its Output Range:

Output = F (Input)

Black-Box Testing (cont.)

- focus of functional requirements of the software
- tester develops set of input conditions that fully exercise the functional requirements
- is not an alternative approach to white-box testing
- is a complementary approach that is likely to uncover a different class of errors than white-box testing
- **incorrect or missing functions can be found**
- use **equivalence classes partitioning** and **boundary value analysis** approach to develop test cases

Black-Box Testing (cont.)

- **Input Equivalence Classes Partitioning**
  - divide all input domain into set of equivalence classes
  - identify classes such that the success of test case in a class implies the success of the others
  - an equivalence class represents a set of valid and invalid states for input conditions

Input Equivalence Classes

EC1: X < 0 (Invalid)

EC2: X > 200 (Invalid)

EC3: 0 <= X =< 200 (Valid)

- **TC1**
  - Inputs: X = -1
  - Expected Outputs: invalid
  - Classes covered: EC1
- **TC2**
  - Inputs: X = 203
  - Expected Outputs: invalid
  - Classes covered: EC2
- **TC3**
  - Inputs: X = 30
  - Expected Outputs: valid
  - Classes covered: EC3

Black-Box Testing (cont.)

- **Boundary Value Analysis**
  - experience shows that a greater number of errors tend to occur at the boundaries of the input domain, rather than the center
  - **min-, min, min+, nom, max-, max, max+**
  - use this technique to complement equivalence class partitioning

Black-Box Testing (cont.)

Input: 0 <= X =< 200

Test cases: -1, 0, 1, 100, 199, 200, 201

Decision Table

test cases for black-box testing can be generated from **a decision table**

is used when there are a number of combination of actions under sets of condition

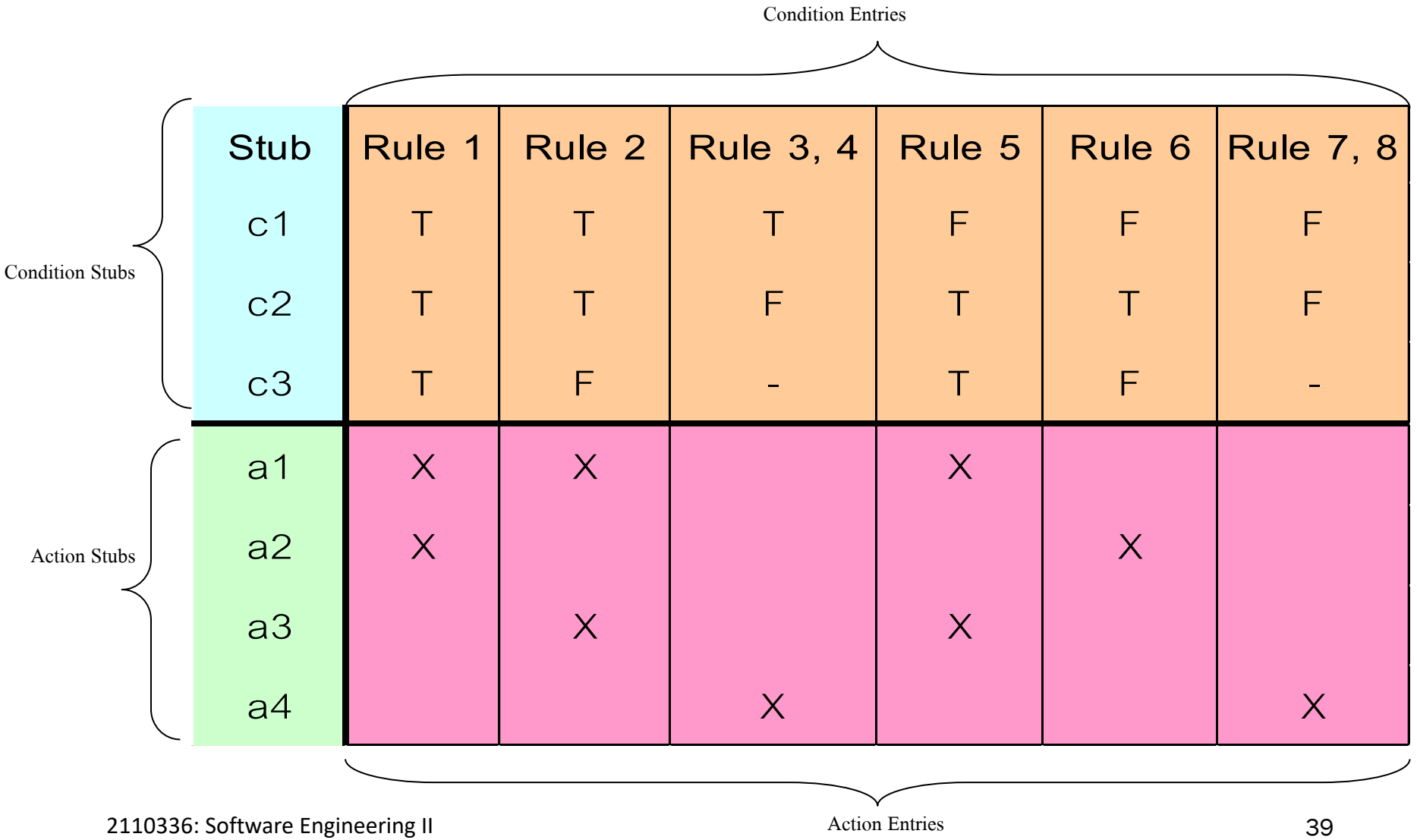
4 parts:

- conditions (condition stubs)
- rules
  (condition entries)
- actions
  (action stubs)
- sets of actions (action entries)

Decision Table

- A condition that is not specified as T or F is called "don't care"
  - the condition is irrelevant, or the condition does not apply
- Types of decision tables
  - **limited entry**: all conditions are binary
  - **extended entry**: conditions have several values

Portions of a Decision Table



Stub	Rule 1	Rule 2	Rule 3, 4	Rule 5	Rule 6	Rule 7, 8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	-	T	F	-
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

Decision Table Testing Technique

- **Conditions stubs** are input condition
- **Condition entries**
  - can be T or F (limited entry decision table)
  - sometimes are equivalence classes of inputs (limited entry decision table)
- **Actions** are major functional processing portions of the item tested
- **Action entries** are output of test cases
- testers have to add action to show when a rule is logically impossible

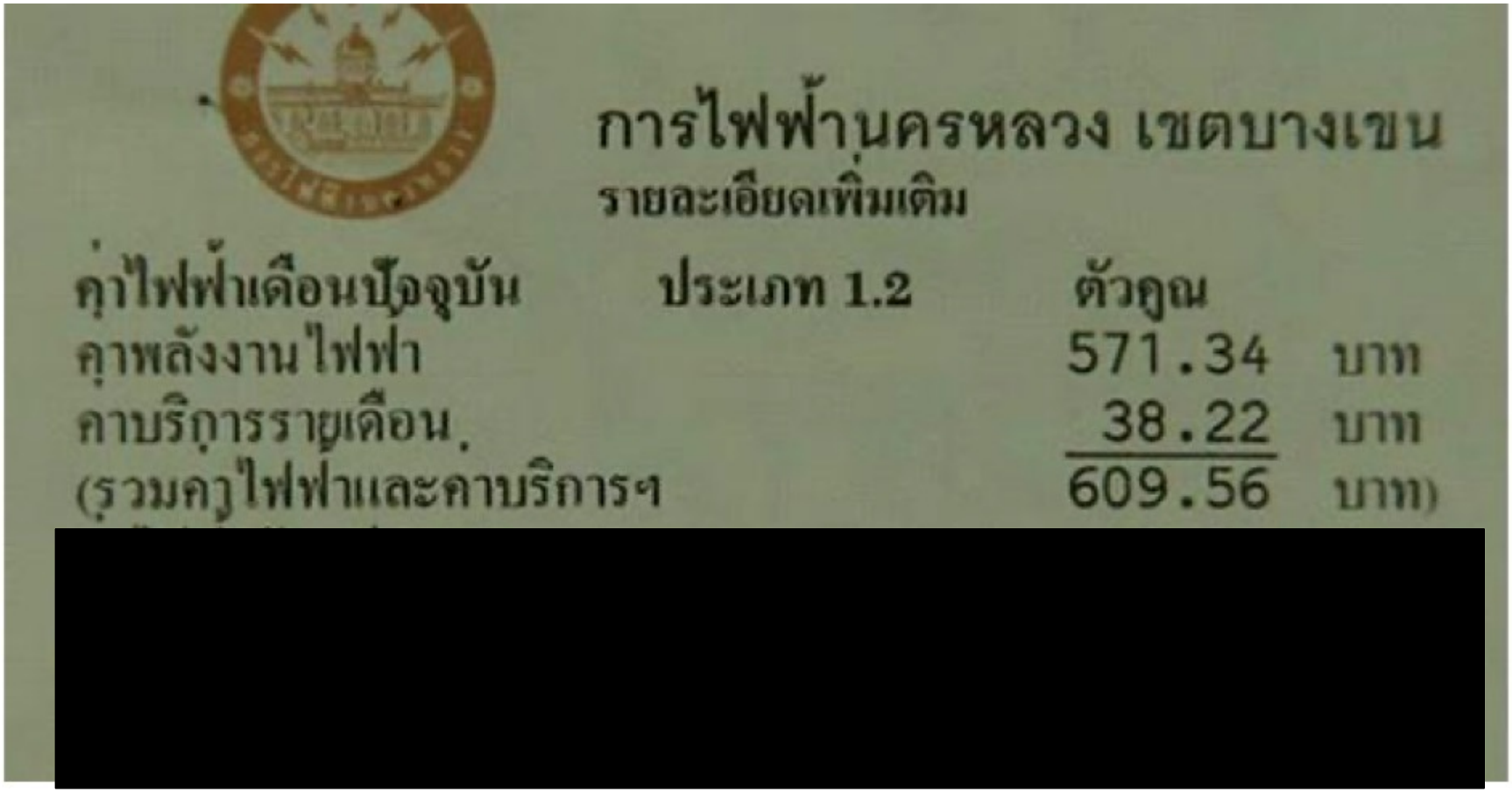
Decision Table: Example

ปัจจุบันค่าบริการรายเดือน สาหรับผู้ใช้ไฟฟ้าประเภทบ้านอยู่อาศัยแบ่งเป็น 2 ประเภท คือ

- ผู้ใช้ไฟฟ้าประเภท 1.1 คือ บ้านอยู่อาศัยที่ติดตั้งมิเตอร์ไม่เกิน 5 แอมป์ และมีการใช้ไฟฟ้า ไม่เกิน 150 หน่วย/เดือน จะคิดค่าบริการรายเดือน เท่ากับ 8.19 บาท/เดือน
- ผู้ใช้ไฟฟ้าประเภท 1.2 คือ บ้านอยู่อาศัยที่ติดตั้งมิเตอร์เกิน 5 แอมป์ และบ้านอยู่อาศัยที่ ติดตั้งมิเตอร์ไม่เกิน 5 แอมป์ แต่มีการใช้ไฟฟ้าเกิน 150 หน่วย/เดือน จะคิดค่าบริการ รายเดือน เท่ากับ 38.22 บาท/เดือน

ค่าไฟตามอัตราก้าวหน้า อ้างอิงจากการไฟฟ้านครหลวง (กฟน.) คือ

- 35 หน่วยแรก เหมารวมทั้งสิ้นเป็นจำนวนเงิน 85.21 บาท
- 115 หน่วยต่อไป หน่วยละ 1.1236 บาท
- 250 หน่วยต่อไป หน่วยละ 2.1329 บาท
- ที่เกินกว่า 400 หน่วย หน่วยละ 2.4226บาท


การไฟฟ้านครหลวง เขตบางเขน
รายละเอียดเพิ่มเติม
ค่าไฟฟ้าเดือนปัจจุบัน ประเภท 1.2 ตัวคูณ
ค่าพลังงานไฟฟ้า 571.34 บาท
ค่าบริการรายเดือน 38.22 บาท
(รวมค่าไฟฟ้าและค่าบริการฯ 609.56 บาท)

Decision Table: Example

Account type = 1.1	T	T	T	T	F	F	F	F	F			
Account type = 1.2	F	F	F	F	T	T	T	T	F			
KWH Used <= 35	T	F	F	F	T	F	F	F	-			
35 < KWH <= 150	F	T	F	F	F	T	F	F	-			
150 < KWH <= 400	F	F	T	F	F	F	T	F	-			
KWH > 400	F	F	F	T	F	F	F	T	-			
ServiceCharge = 8.19	X	X	X	X								
ServiceCharge = 38.22					X	X	X	X				
Total = ServiceCharge + 85.21	X				X							
Total = ServiceCharge + (85.21) + ((KWH - 35) * 1.1256)		X				X						
Total = ServiceCharge + (85.21) + (115 * 1.1236) + ((KWH - 150) * 2.1329)			X				X					
Total = ServiceCharge + (85.21) + (115 * 1.1236) + (250 * 2.1329) + ((KWH - 400) * 2.4226))				X				X				
prompt error message 'Invalid Account type'									X			
impossible										X	X	X