Software Testing - 1

Outline

- Software testing concept
- Black-box testing
- White-box testing

A Lawsuit

The $100,000 Keying Error

Kai A. Olsen

Although it sounds disastrous, making a $100,000 mistake seems relatively minor when stockbrokers have lost millions by hitting the wrong key. The following case proved to be different, however. An ordinary bank customer, Grete Fossbakk, used Internet banking to transfer a large amount to her daughter. She keyed one digit too many into the account number field, however, inadvertently sending the money to an unknown person. This individual managed to gamble away much of the sum before police confiscated the remainder.

Subsequently, the case received extensive media coverage in Norway. The Minister of Finance criticized the bank's user interface and requested new and improved Internet banking regulations. Suddenly, the risk to Internet banking had become apparent to both the government and ordinary citizens.

Clearly, the user made a slip. She also had the chance to correct the typo before she hit the confirm button. However, as we shall see, the system also had every opportunity to catch her mistake. Yet this did not happen. The system's developers had neglected to build in a simple check that would detect if the correct input were missing.

This case raises questions about what the minimum validation procedures from a banking system developed for ordinary users should be. It also challenges us, as system designers, to help users avoid such errors.

Today's users operate alone in front of a computer, with intermediates and colleagues replaced by computer systems. This new reality makes it important to have interfaces that can offer as good as—or even better—error detection than that found in previous manual systems.

FOSSBAKK CASE

The Fossbakk case provides an illuminating example. The Internet system she employed when making her fatal mistake was one common to a large group of Norwegian banks. Reviewing this case provides insight into the types of typos that users make, the psychology behind "confirmation," and the pitfalls inherent in many Web-based transactions systems.

Fossbakk's daughter's account number was 71581555022, but she inserted an extra 5 and keyed in 71581555 5022. The user interface accepted only 11 digits in this field (the standard length of a Norwegian account number), thus truncating the number to 71581555502. The last digit is a checksum based on a modulo-11 formula. This will detect all single keying errors and errors where two consecutive digits are interchanged. Inserting an extra 5 changed both the ninth and tenth digits.

The average checksum control will catch only 93 percent of the cases in which such errors occur. For Fossbakk, the final eleven-digit number was a legal account number. However, only a small fraction of all legal account numbers are in use. Further, the chance of mistyping the account number so that it benefits a dishonest person without income or assets is overwhelmingly low in a homogeneous country such as Norway. Our user was thus extremely unlucky. The person who received her $100,000 transaction and kept the proceeds has been sentenced to prison, but this does little to help Fossbakk get her money back.

LITIGATION

Fossbakk took the case to the Norwegian Complaints Board for Consumers in Banking. This board deals with disputes between consumers and banks. The board has two representatives for the consumers and two from the banks, with a law professor as chair. In a three-to-two vote, Fossbakk lost. The chair voted for the bank, arguing that "she made an error and has to take responsibility." He also regretted that Norwegian regulations set no limit for a consumer's loss in these cases, as there would have been if Fossbakk had lost her debit card.

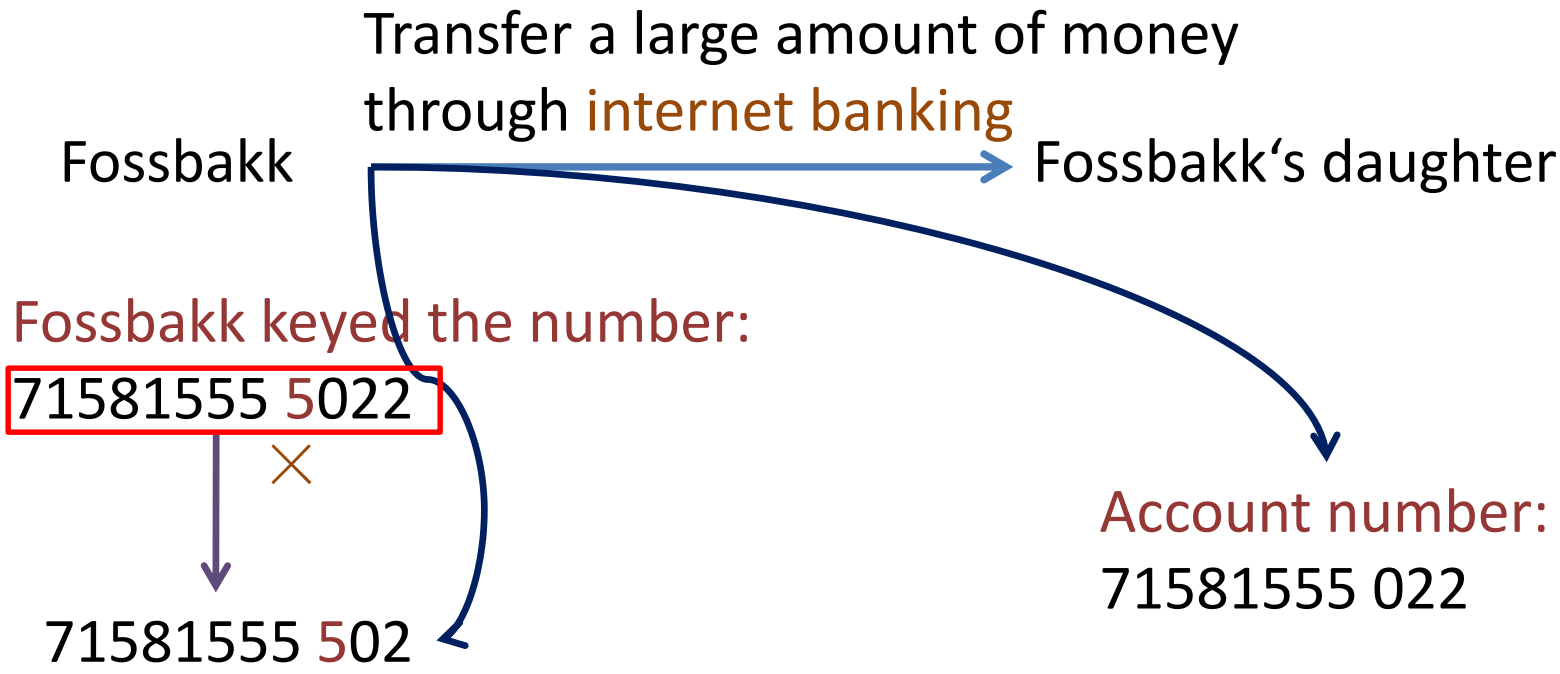
Fossbakk is now taking the case to court, backed by the Norwegian Consumer Council. She argues that she typed 12 digits and that the bank system should have given an error message in this case, instead of ignoring all typed digits after the first 11. She has acknowledged she would have no case if only 11 digits had been typed. The bank argues that she cannot prove by any measure of probability that she keyed 12 digits. They further state that there cannot be different rules of responsibility depending on the number of digits given. Finally, they stress that she confirmed the $100,000 transaction.

At this point, I was called in as an expert witness for Fossbakk. In my opinion, and I should expect that of most other computing professionals, a system should give an error message when the customer types a too-long number. Clearly, such a test can be inserted with a minimum of effort. In fact, the Financial Supervisory Authority of Norway has required all banks to implement this functionality based on the Fossbakk case.

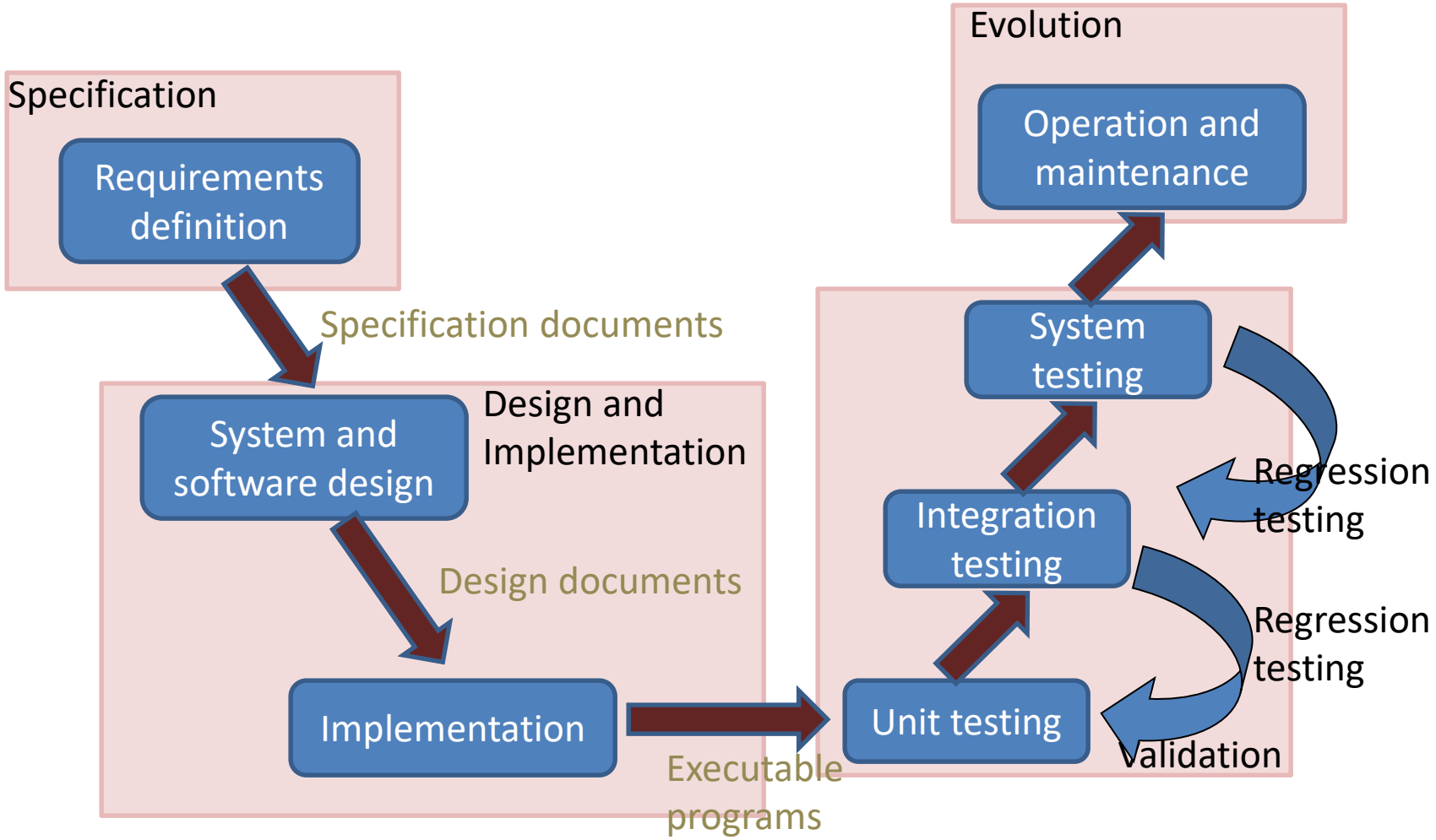
Reasonably, we could argue that the bank showed negligence when developing the user interface in question. However, can we prove, beyond doubt, that Fossbakk keyed 12 digits? Since any digits beyond 11 were stripped from the HTML form, no information log exists that can tell us what happened.

Is This Tested?

- A $100,000 keying error



Software Development Life Cycle



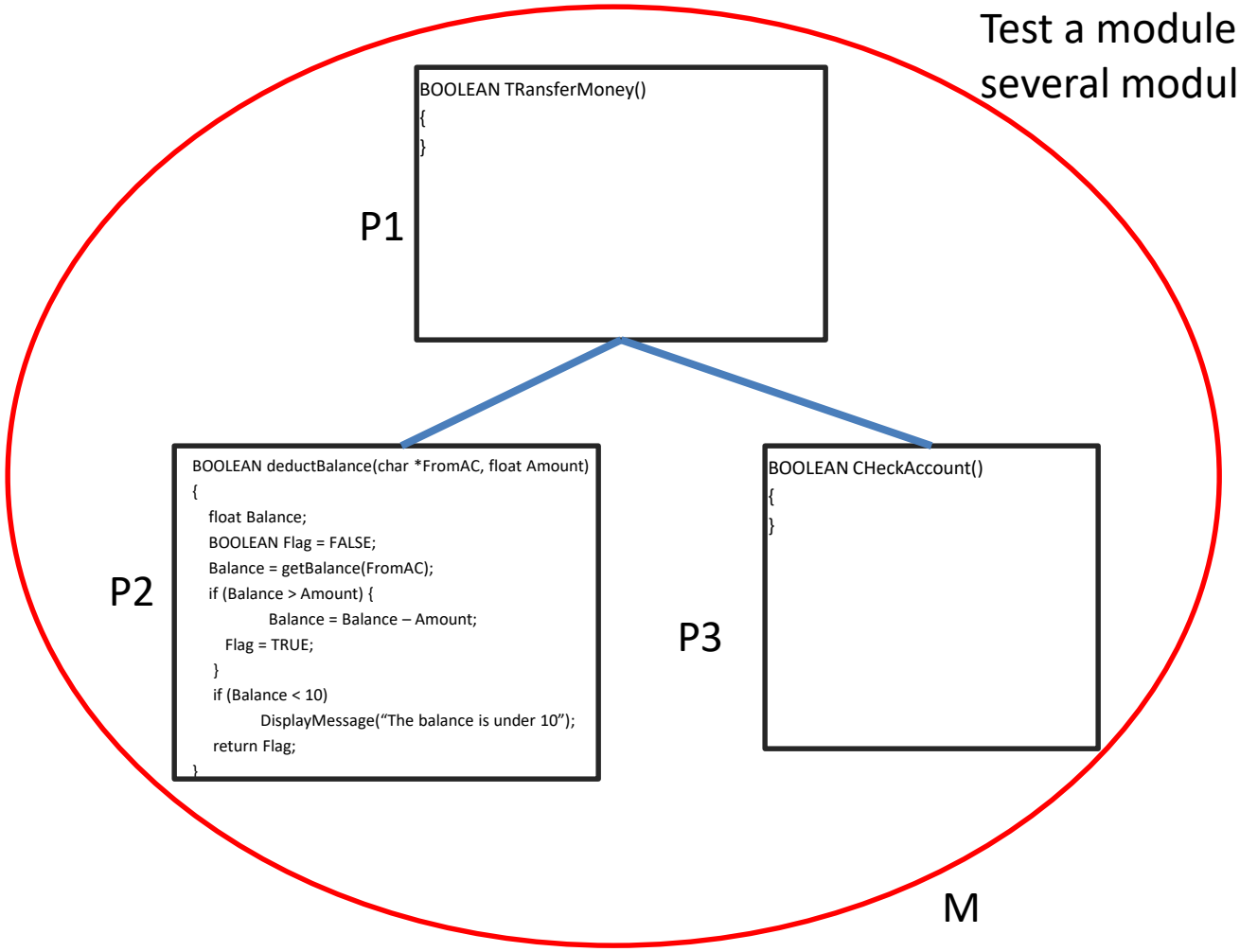
Unit （单元）Testing

```
BOOLEAN deductBalance(char *FromAC, float
    Amount)
{
   float Balance;
   BOOLEAN Flag = FALSE;
   Balance = getBalance(FromAC);
   if (Balance > Amount) {
      Balance = Balance – Amount;
      Flag = TRUE;
    }
    if (Balance < 10)
            DisplayMessage(“The balance is under
     10”);
    return Flag;
}
```

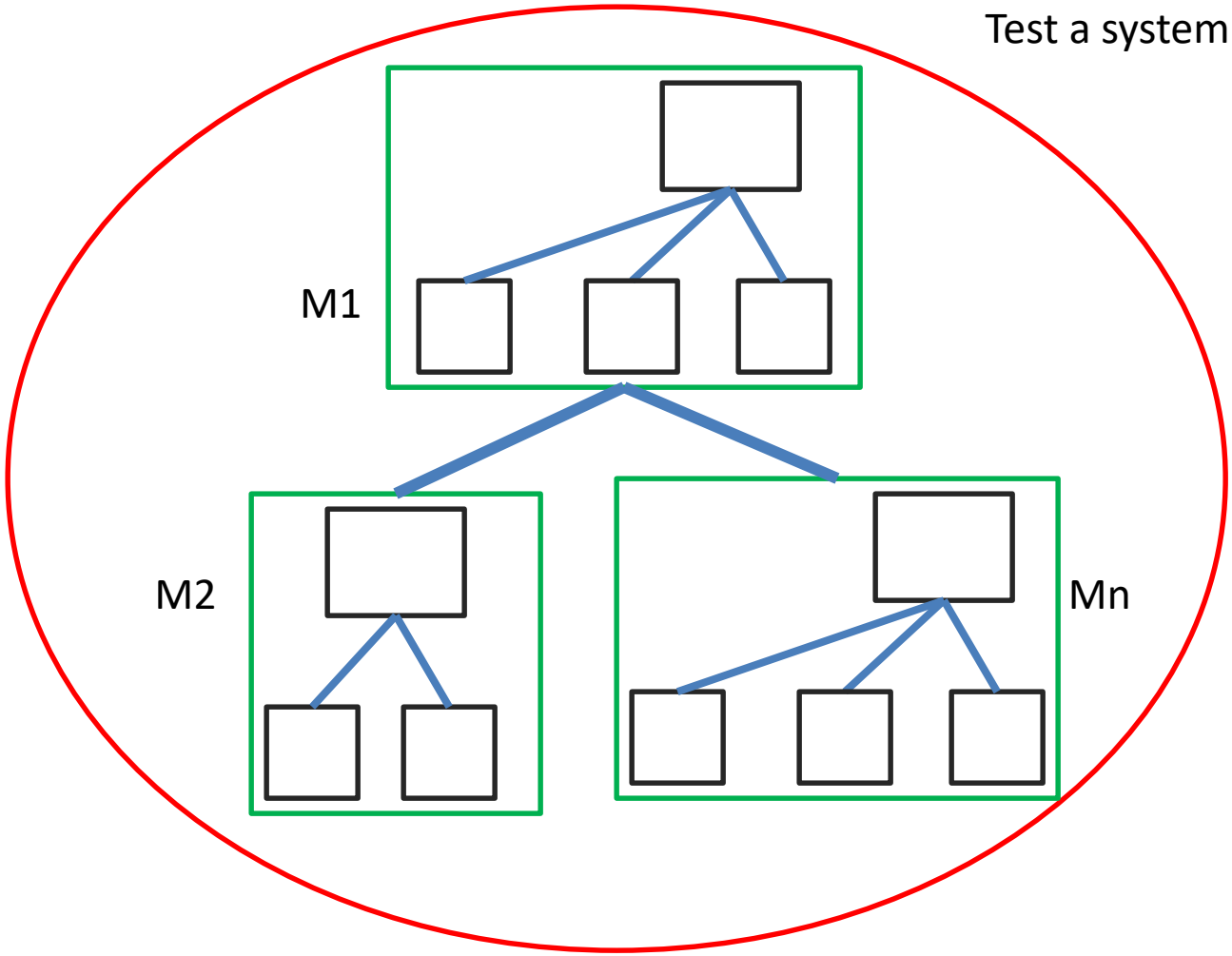
Test a single program or a function

Integration （集成） Testing

Test a module or an integration of several modules



System （系统） Testing



Regression (回归) Testing

- Regression testing is performed after some bugs （错误）are fixed （纠正）
- The same test data used before are used again to test the updated （更新）source code
- The objective is to assure the bug repair activity does not introduce （引进）more bugs

Software Testing

- A very important activity to verify (验证) the software
- It is used to check if the software contains bugs that clients do not expect
- Testing vs debugging
  - Testing: check if the system has a bug
  - Debugging: locate the position of a bug

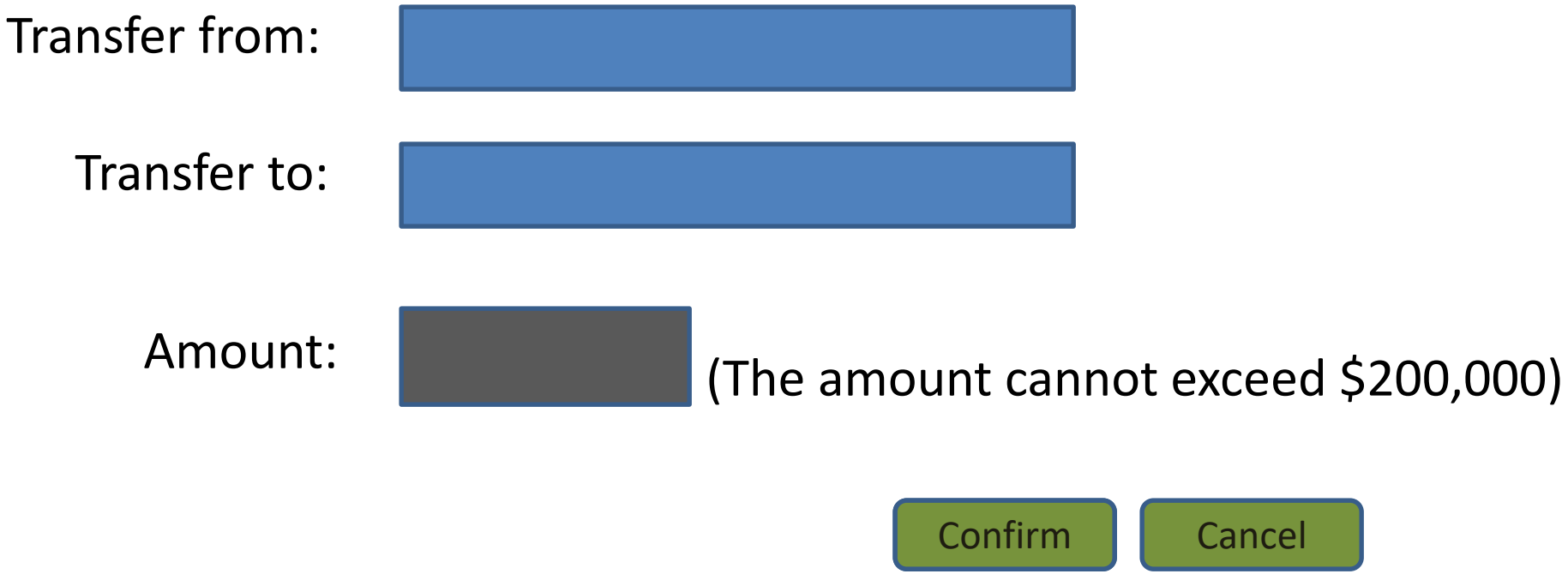
Software Testing

- V & V
  - Verification
    - Check if the system meets the specification
    - Are we building the system properly?
  - Validation
    - Check if the system meets users' requirements
    - Are we building the proper system?

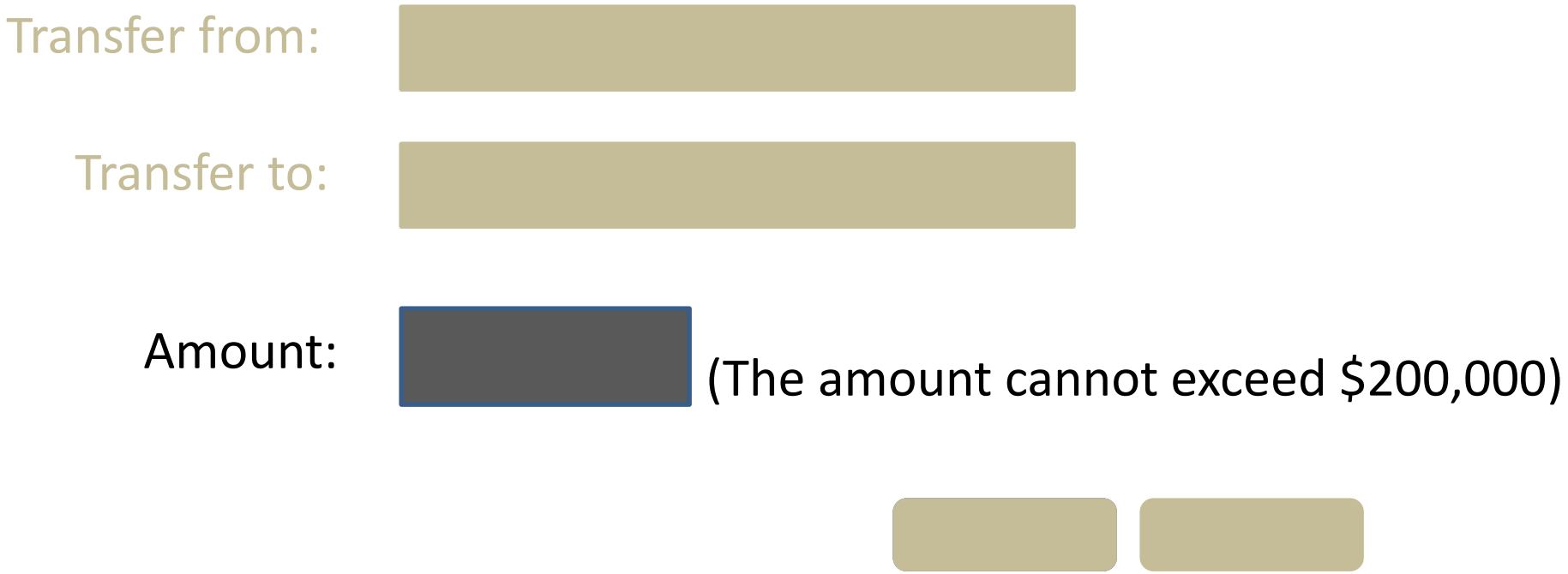
Software Tester

- Software tester is an important role in software development
- There are 9000 testers at Microsoft

Interface – Money Transfer



Interface – Money Transfer



Test Cases

- Are these data enough?
  - 190000
  - 200000
  - 200000.01
  - 199999.99
  - -100
  - 199.9.9

Test Cases

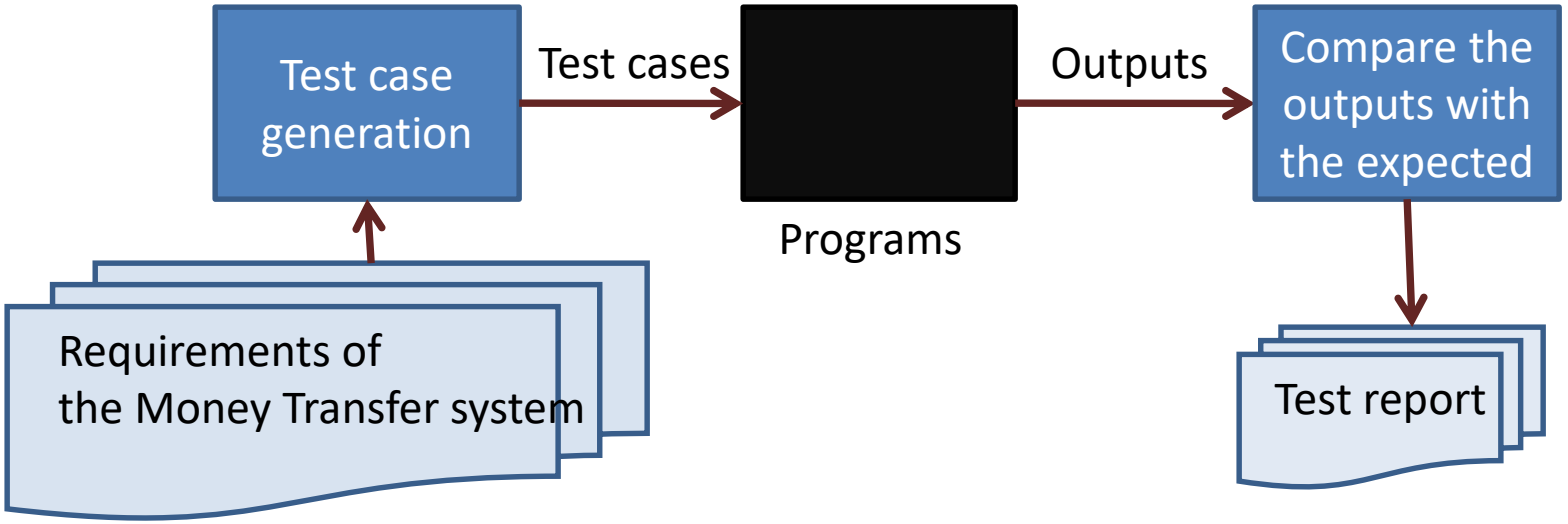
- A test case （测试案例） is a combination （组合）of input variable values of a program
  - (FromAC = 12345678901, ToAC = 23456789012, Amount = 10) is a use case
- Different test cases can have different effectiveness （有效性） in finding the bugs of software
  - Many test strategies have been proposed to generate the suitable test cases,
  - but not all of them are good at finding bugs in the software.

Test Case Generation

- Two main categories
  - Black-box testing （黑盒测试）(Functional testing)
  - White-box testing （白盒测试）(Structural testing).
- Gray-box testing
  - Combination of black and white

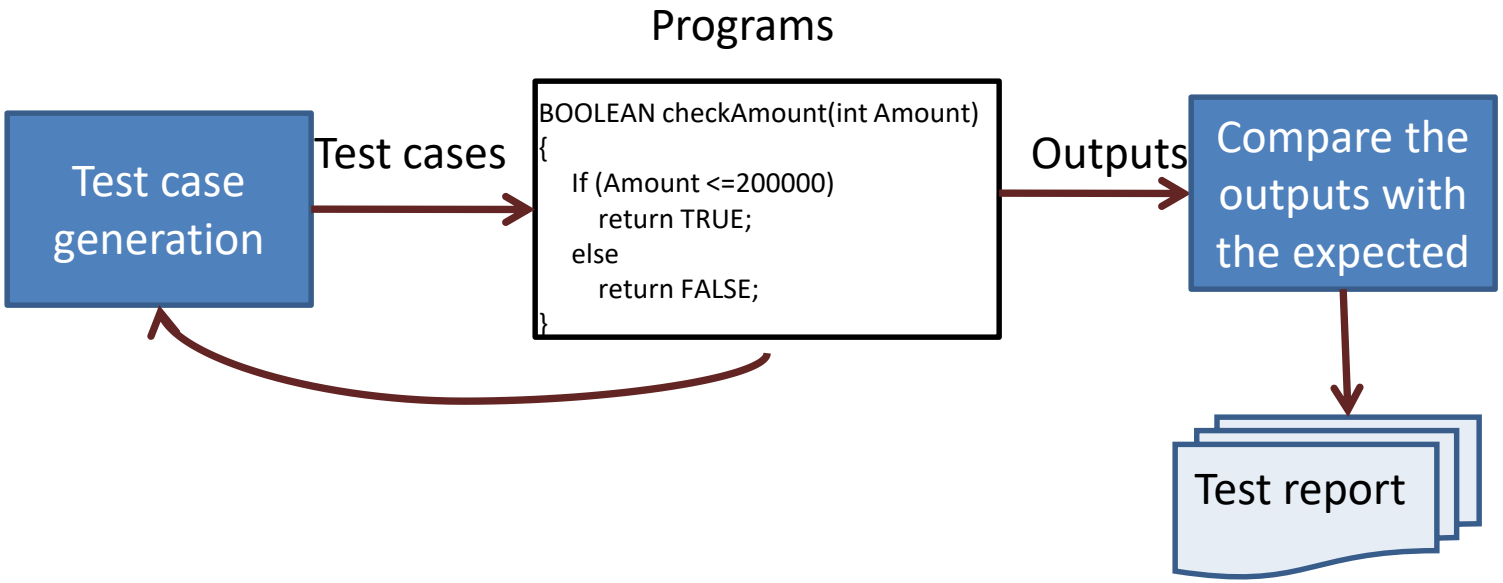
Black-Box Testing

- Test cases are generated based on the requirements specification of software, rather than the source code

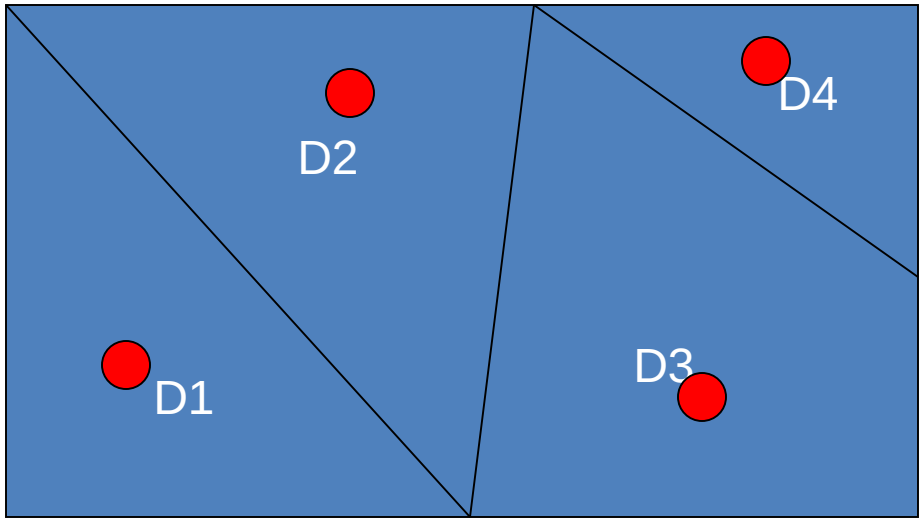


White-Box Testing

- Test cases are generated based on the source code of software

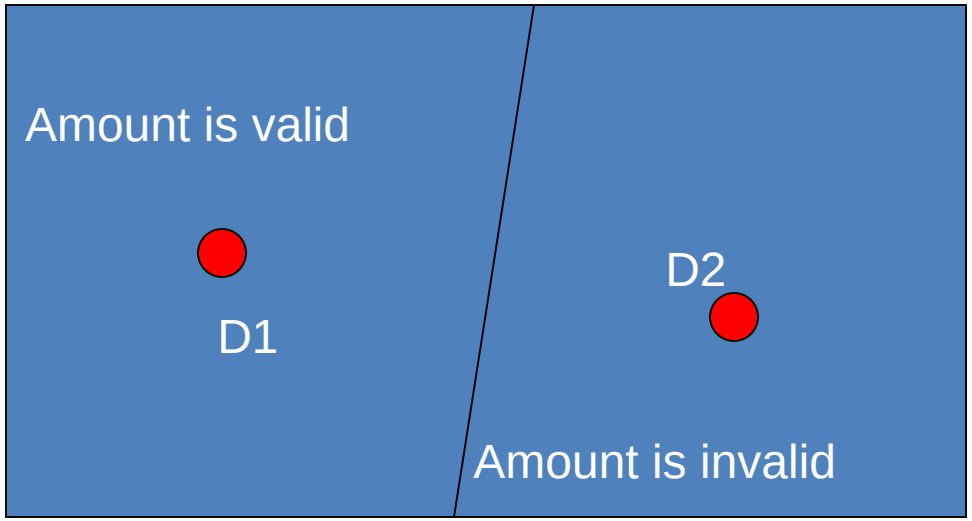


Equivalence-Class Testing （等价类）



Input domain D is divided into disjoint sub-domains according to some relation. Then from each sub-domain, a representative is chosen as a test case

Equivalence-Class Testing



Input domain D

D1: {<F, T, A>|A is invalid}

D2: {<F, T, A>|A is valid}

D1 and D2 can be further divided.

Equivalence-Class Testing

Input domain D

Equivalence-Class Testing

Amount valid		Amount invalid	
"From" account valid	"From" account invalid	"From" account invalid	"From" account valid

Input domain D

Equivalence-Class Testing

Amount valid				Amount invalid			
"From" account valid		"From" account invalid		"From" account invalid		"From" account valid	
"To" account valid	"To" account invalid	"To" account valid	"To" account invalid	"To" account valid	"To" account invalid	"To" account valid	"To" account invalid

Equivalence-Class Testing

D1_1_1: {<F, T, A>|"From" account is valid, "To" account is valid, amount is valid}

D1_1_2: {<F, T, A>|"From" account is valid, "To" account is valid, amount is invalid}

D1_2_1: {<F, T, A>|"From" account is valid, "To" account is invalid, amount is valid}

D1_2_2: {<F, T, A>|"From" account is valid, "To" account is invalid, amount is invalid}

D2_1_1: {<F, T, A>|"From" account is invalid, "To" account is valid, amount is valid}

D2_1_2: {<F, T, A>|"From" account is invalid, "To" account is valid, amount is invalid}

D2_2_1: {<F, T, A>|"From" account is invalid, "To" account is invalid, amount is valid}

D2_2_2: {<F, T, A>|"From" account is invalid, "To" account is invalid, amount is invalid}

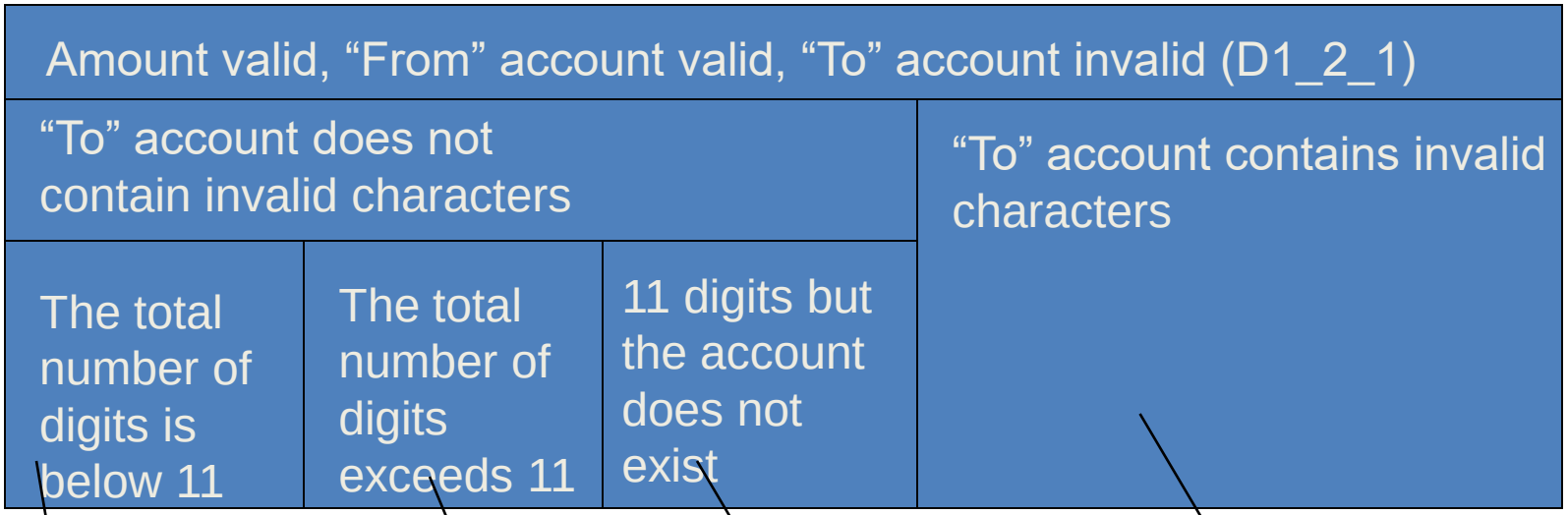
Equivalence-Class Testing

Amount valid				Amount invalid			
"From" account valid		"From" account invalid		"From" account invalid		"From" account valid	
"To" account valid	"To" account invalid	"To" account valid	"To" account invalid	"To" account valid	"To" account invalid	"To" account valid	"To" account invalid
D1_1_1	D1_2_1	D2_1_1	D2_2_1	D2_1_2	D2_2_2	D1_1_2	D1_2_2

- 8 sub-domains are obtained.
- They are disjointed.
- We can choose a representative from each sub-domain, then 8 test cases are generated.

Further Division – Strong equivalence class testing

Further division on D1_2_1:



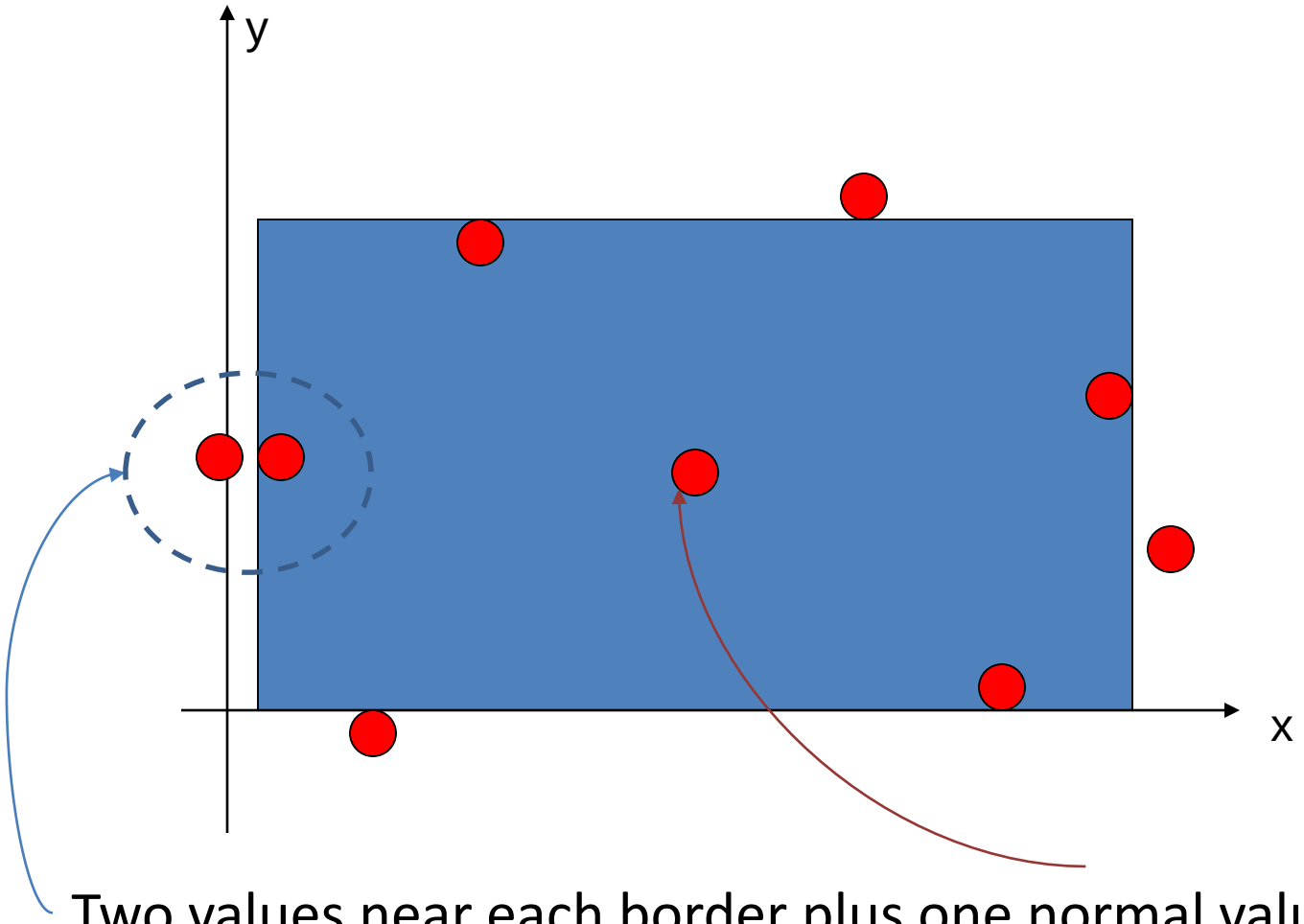
<12345678910, #(234567891, 200.10>

<12345678910, 11111111111, 200>

<12345678910, 12345678910123, 200>

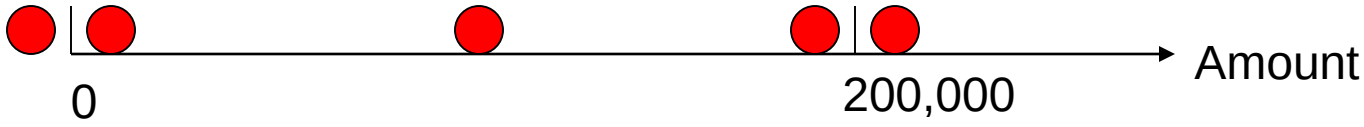
<12345678910, 2345, 300>

Boundary Value (边界值)Testing



Two values near each border plus one normal value

Boundary Value Testing



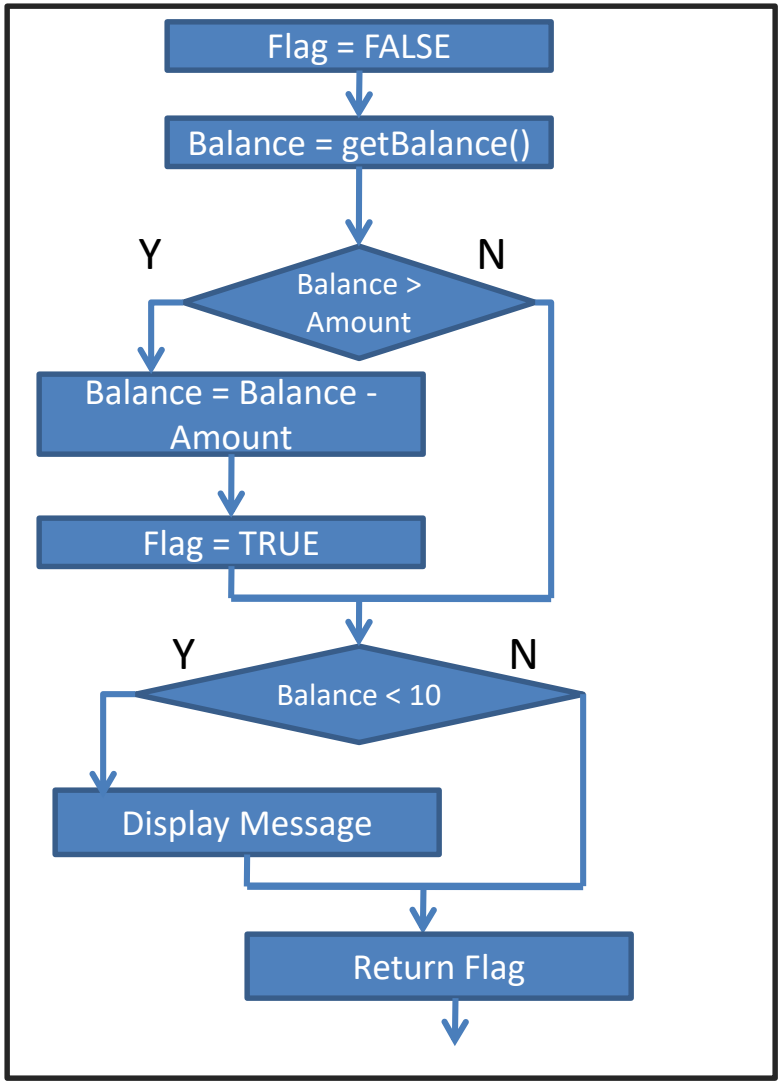
The number of the test cases = 4 * *n* + 1
where *n* is the number of input variables

White-Box Testing

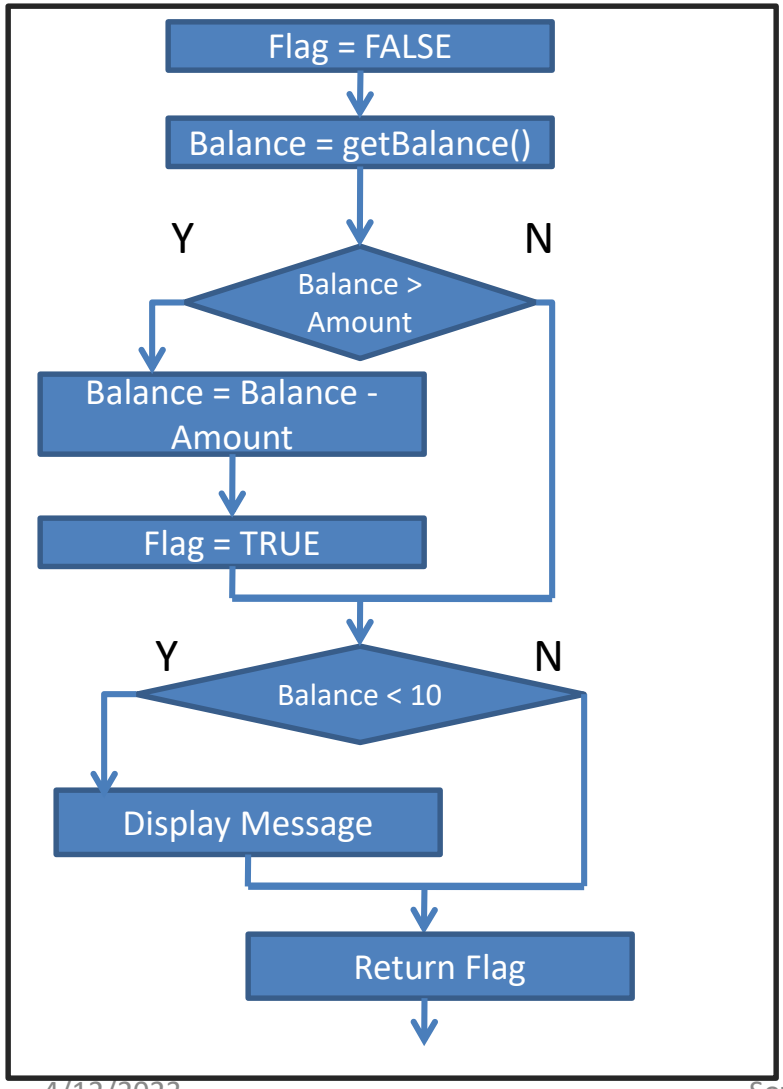
- Statement testing
  - Every statement must be executed （运行）at least once
- Branch testing
  - Every branch must be executed at least once
- Path testing
  - Every path must be executed at least once
- ……

Source Code for deductBalance

```
BOOLEAN deductBalance(char *FromAC, float
    Amount)
{
   float Balance;
   BOOLEAN Flag = FALSE;
   Balance = getBalance(FromAC);
   if (Balance > Amount) {
      Balance = Balance – Amount;
      Flag = TRUE;
    }
    if (Balance < 10)
            DisplayMessage(“The balance is under
     10”);
    return Flag;
}
```



Statement Testing



One test case is enough:

(Balance = 550, Amount = 545)

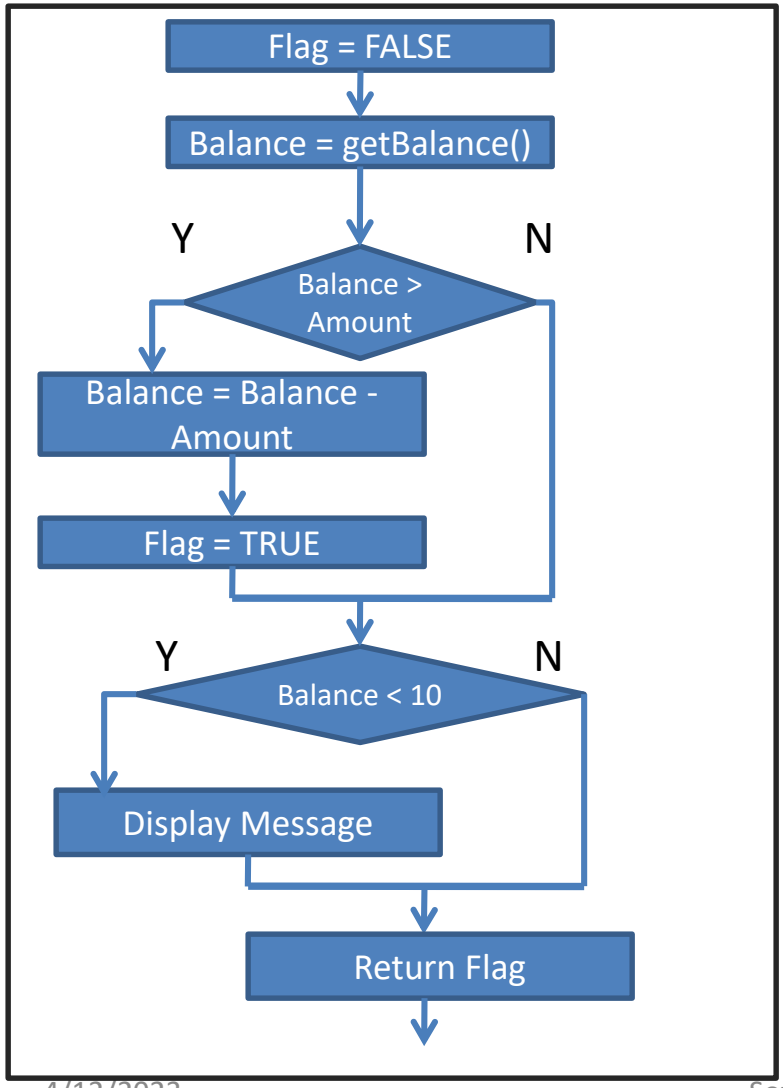
Two test cases to satisfy the criteria:

(Balance = 550, Amount = 500)

(Balance = 9, Amount = 10)

It is expected to use least number of test cases to fulfill a testing criteria.

Branch Testing

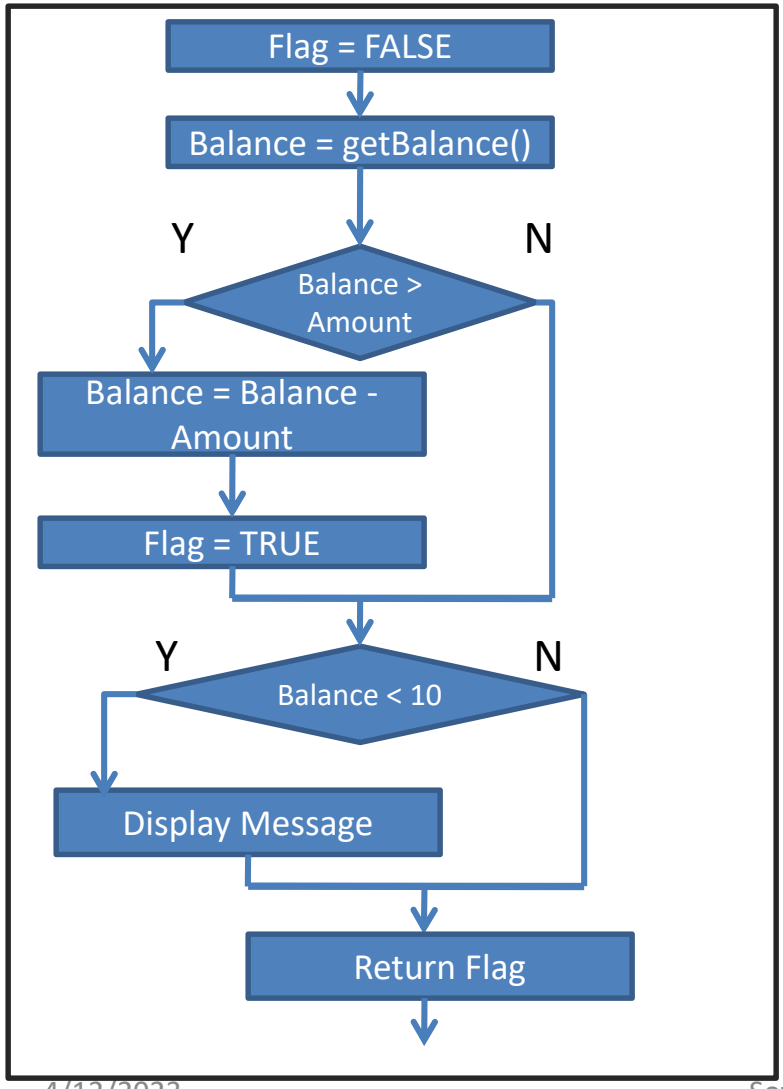


At least two test cases are needed:

(Balance = 550, Amount = 545)

(Balance = 500, Amount = 560)

Path Testing



At least four test cases are needed:

(Balance = 550, Amount = 545)
(Balance = 600, Amount = 560)
(Balance = 500, Amount = 600)
(Balance = 9, Amount = 10)

Other Testing Methods

- Transaction flow testing
- Transition testing
- Logic-based testing
- Fault-based testing
- Metamorphic testing

Class Exercise

- IsTriangle function

  Specification: The lengths of three edges are used to decide if the three edges can form a triangle. If any of the length is less than or equals 0, output the warning information and return false. If the sum of two edges is less than or equals the third edge, they cannot form a triangle; otherwise, they can form a triangle.

  1) Please use the equivalence-class testing method to generate a set of test cases. Please also specify the sub-domains that are obtained from the classification of the domain.

  2) Please generate the least number of test cases using statement testing, branch testing and path testing. The code is in the next slide.

Code

```
BOOL IsTriangle (int a, int b, int c)
{
    If (a <= 0 || b <= 0 || c <= 0) {
        printf("the lengths are out of range);
        return FALSE;
    }
    If ((a >= b + c) || (b >= a + c) || (c >= a + b)) {
        printf("It is not a triangle");
        return FALSE;
    }
    else {
        printf("It is a triangle");
        return TRUE;
    }
}
```

Summary

- Verification and validation
- Black-box testing and white-box testing
- Black-box testing strategies
  - –Equivalence class testing
  - –Boundary value testing
- White-box testing
  - –Statement testing
  - –Branch testing
  - –Path testing