

Đề thi thử

Câu hỏi 1

In software development process. Choose the phase in which you analyze and organize the work to be achieved of software.

a. Requirements analysis

- b. Implementation / Coding
- c. External design
- d. Internal design

Câu hỏi 2

Which of the following statements **correctly** describes the characteristics of the Waterfall, Prototyping, and Spiral development process models?

a. Prototyping Model is suitable for projects with unclear or evolving requirements

- b. Spiral Model focuses on documentation and clear requirements before starting development
- c. Waterfall Model is suitable for large, complex projects with significant risk
- d. None of the above

Câu hỏi 3

Choose the testing activity corresponding to check if the system satisfies the requirements specification.

- a. Regression testing
- b. Unit testing

c. Acceptance testing

- d. Integration testing

Câu hỏi 4

Suppose we are testing a software whose behavior changes if the inputted x is greater than or equal to 0 and less than or equal to 20. Choose the test case that is **NOT related** to the boundary-value analysis method.

- a. x=1
- b. x=19
- c. x=11
- d. x=10

BVA (boundary-value analysis) test cases: 0, 1, 10, 19, 20

Câu hỏi 5

Which of the following is not a maintenance activity?

- a. Fixing of faults found after the start of operation

b. Release every few months and develop & improve the necessary functions in order by getting feedback from customers

- c. Modifications to meet changing requirements
- d. Continuous quality improvement

Câu hỏi 6

Choose the statement that **incorrectly** describes Test-Driven Development (TDD).

- a. The second step of TDD is testing
- b. The fourth step of TDD is testing
- c. A new test should fail the first time
- d. None of the above**

Test-driven development: Create test case -> Test (it should fail) -> Program -> Test

Câu hỏi 7

What should we do if there is a “**code smell**” in our software?

- a. Use pair programming to review the mistakes
- b. Write comments appropriately
- c. Use test-first to test the code continuously
- d. Refactor the code**

Câu hỏi 8

		actual		
		Buggy	Non-Buggy	total
prediction	Buggy	27		72
	Non-Buggy			
total		36		150

Suppose we predicted the fault-proneness of 150 modules using a prediction model, and the following table presents the result. Choose the F value of the prediction result.

- a. 0.75
- b. 0.25

c. 0.5

d. 0.375

$$\text{Recall} = 27/36 = 0.75$$

$$\text{Precision} = 27/72 = 0.375$$

$$\text{F-score} = 2 * \text{recall} * \text{precision} / (\text{recall} + \text{precision}) = 2 * 0.375 * 0.75 / (0.375 + 0.75) = 0.5$$

Câu hỏi 9

```
int countAndCheckVowels(const string &str) {
    int count = 0;
    for (char c : str) {
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
            c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') {
            count++;
        }
    }
    if (count >= 5) {
        return 5;
    }
    return count;
}
```

Choose the function call that achieves the branch coverage 83.33%.

- a. countAndCheckVowels("SE113.O21.CNCL(JP)");
- b. countAndCheckVowels("Review for the final exam!");
- c. countAndCheckVowels("Break a leg!");

d. All of the above answers are correct.

Branch coverage = C1

Total branches = 6

a, b, c. branches = TF, TF, (always) T or F = 5

$$\Rightarrow C1 = 5/6$$

Câu hỏi 10

```
int evaluateValues(int n, int threshold) {  
    int sum = 0;  
    for (int i = 0; i <= n; ++i) {  
        if (i % 2 == 0) {  
            sum += i;  
        } else {  
            sum += i * 2;  
        }  
        if (sum > threshold) {  
            return sum;  
        }  
    }  
    if (sum < threshold / 2) {  
        return sum * 2;  
    } else {  
        return sum;  
    }  
}
```

Suppose we tested the following function by running `evaluateValues(3,9)` and `evaluateValues(2,8)`. Choose the statement coverage.

- a. 0.8
- b. 0.73
- c. 0.89
- d. 0.83

Statement coverage = C0

Total C0 = 10

C0 of (3, 9) and (2, 8) = 8/10

Câu hỏi 11

Suppose two test teams A and B independently tested a software. Team A detected 56 bugs and team B did 42 bugs, but 21 bugs were common to both teams. Choose the total number of bugs.

- a. 31
- b. 14
- c. 112**
- d. 35

a=56, b=42, c=21

Formula: $a*b/c = 56*42/21 = 112$

Câu hỏi 12

		actual		
		Buggy	Non Buggy	total
prediction	Buggy	45		
	Non Buggy		83	
total				X

Suppose we predicted the fault-proneness of X modules using a prediction model, and the following table presents the result. The precision and recall of the prediction result are 75% and 90%, respectively. Choose the X value.

- a. 128
- b. 170
- c. 148**
- d. 138

$$\text{Recall} = 45/a = 0.9 \Rightarrow a = 50$$

$$\text{Precision} = 45/b = 0.75 \Rightarrow b = 60$$

$$c = a - 45 = 50 - 45 = 5$$

$$d = 83 + c = 83 + 5 = 88$$

$$X = b + d = 60 + 88 = 148$$

		actual		total
		buggy	non buggy	
prediction	buggy	45		b = 60
	Non buggy	c = 5	83	d = 88
total		a = 50		X = 148

Câu hỏi 13

Suppose a development team has developed many software products, and their average bug density is 1.8 [bugs/KLOC]. Now, we found a total of 26 bugs. Estimate the

percentage of the number of remaining bugs when they develop a software with 20 KLOC.

a. 72.22%

b. 10%

c. 36%

d. 27.78%

bug_rate = 1.8, loc = 20, bugs_found = 26

bugs_remain = bug_rate * loc - bugs_found = 1.8 * 20 - 26 = 10

percentage_bugs_remain = 10 / (10 + 26) = 0.278

Câu hỏi 14

Suppose we are forecasting a bug probability of a module using a logistic regression model. Choose the value of y when the bug probability is p = 0.95.

a. 1.28

b. 19

c. 0.72

d. 0.5

Formula: $y = \log_{10}(p / (1 - p)) = \log_{10}(0.95 / (1 - 0.95)) = 1.27875$

Câu hỏi 15

```

string decideDriverAction(const string& trafficLightColor, bool
pedestrianSignalOn) {
    if (trafficLightColor == "green") {
        if (pedestrianSignalOn) {
            return "Proceed with caution, watch for pedestrians.";
        } else {
            return "Proceed.";
        }
    } else if (trafficLightColor == "yellow") {
        if (pedestrianSignalOn) {
            return "Prepare to stop, watch for pedestrians.";
        } else {
            return "Prepare to stop.";
        }
    } else if (trafficLightColor == "red") {
        if (pedestrianSignalOn) {
            return "Stop and wait for pedestrians.";
        } else {
            return "Stop and wait.";
        }
    } else {
        return "Invalid traffic light color.";
    }
}

```

Suppose we tested the following function by running `decideDriverAction("green", true)` and `decideDriverAction("Red", false)`. Choose the branch coverage and condition coverage.

- a. 0.25 and 0.03125
- b. 0.25 and 0.2857**
- c. 0.3 and 0.25
- d. 0.3 and 0.3333

Condition coverage = C1

Branch coverage = C2

Total C1 = 12

Total C2 = 7

C1 of ("green", true) and ("Red", false) = 6/12

C2 of ("green", true) and ("Red", false) = 2/7

QUIZ [2]

1. Which of the following applies to email software (Outlook, Thunderbird, etc.)?

Application software.

2. Which of the following applies to operating systems (Windows, Linux, etc.)?

Basic software.

3. Which of the following applies to presentation software (such as PowerPoint)?

Application software.

4. Which of the following applies to the temperature control software for an air conditioner?

Embedded software.

5. Which of the following applies to database management systems (MySQL, PostgreSQL, etc.)?

Middleware.

6. The following text discusses the software crisis. Choose the appropriate combination of terms to fill in the blanks (a) to (d).

The software crisis refers to a sense of crisis proposed in (a). The main content of this crisis was the concern that “(b) cannot keep up with (c), potentially hindering the development of (d).” In response to this sense of crisis, the field of software engineering was established to systematically organize and formalize theories and techniques related to software development.

(a) 1968, (b) software development, (c) needs, (d) computer systems

7. What was one of the software crises that was feared?

There is a shortage of engineers, and software development is falling behind schedule.

8. Choose the word that means “a system that has been in use for a long period of time (for example, more than 20 years).”

Legacy system.

9. Choose the word that means “source code that has been copied and pasted or reused with only partial changes.”

Code clone.

QUIZ [3]

1. What step in the Waterfall model does the task of “modifying a program to address a bug reported by a user” fall into?

Operation and maintenance.

2. What step in the Waterfall model does the task of “deciding to use a hash table to manage data” fall into?

Internal design.

3. Which step in the Waterfall model does the task of “preparing a usage scenario and checking whether the software operates correctly” fall into?

Test.

4. “When developing e-mail software, we decided to develop it in four parts: (1) the part for receiving email, (2) the part for sending e-mail, (3) the part for displaying e-mail, and (4) the part for managing mailboxes.”

Which stage in the Waterfall model does this correspond to?

External design.

5. What step in the Waterfall model does the task of “drawing out the customer's requirements and considering whether the software can be developed within the time and budget” fall into?

Requirements analysis.

6. What step in the Waterfall model does the task of “writing a program in C, C++, or Python according to the contents of the design document” fall into?

Programming.

7. Play around with all the advantages of the Waterfall model (not just one):

- **The portion of the work is clear.**
- **Traditional model and easy to use.**

8. List all (or not all) of the drawbacks (bad points) of the Waterfall model.

This can lead to a design with insufficient component analysis.

- **Deepening upstream will affect the bottom.**
- **Not suitable for construction that goes back from the bottom to the top.**

9. Choose something that corresponds to “a method of proceeding with development while checking by creating a prototype.”

Prototyping.

10. Which of the following is a risk-driven development process model?

Spiral model.

QUIZ [4]

1. Which of the following is a test to check the operation of a single module?

Unit test.

2. Which of the following is a test that checks the system by having the customer actually use it?

Acceptance testing.

3. Which of the following is a method of dividing input into equivalence classes and performing tests based on the classification?

Equivalence partitioning method.

4. Which of the following is a method of testing by focusing on the transitions (boundaries) of conditions?

Boundary value analysis.

5. Which of the following corresponds to “certain input data or conditions and the expected output for them”?

Test case.

6. Among the following “relationships between x and y,” choose all that are equivalence relations (it is not necessarily only one). However, assume that both x and y are human.

- **x and y are students at the same university.**
- **x and y live in the same country.**

7. Among the relationships between x and y shown below, select all that are equivalent (there is not necessarily only one). However, assume that both x and y are integers greater than or equal to 1.

Both x and y are divisible by 3 (they are multiples of 3).

8. Suppose you are developing an application and testing a function that checks the password entered by the user. However, the password length must be between 8 and 32 characters. Select all the invalid equivalence classes for password length (it is not limited to one).

- **0 to 7 characters.**
- **33 characters or more.**

9. Suppose you are developing an application and testing a function that checks the password entered by the user. However, the password must contain all four types of characters: uppercase letters, lowercase letters, numbers, and symbols. Please select all valid passwords that meet these conditions from the following test passwords (not necessarily only one).

- **a283c#D987.**
- **example@YES123.**

10. Suppose you are developing an application and testing a function that “checks passwords entered by users.”

However,

(1) the length of the password must be between 8 and 32 characters.

(2) The password must contain all four types of characters: “uppercase letters”, “lowercase letters”, “numbers” and “symbols.”

Assume that the password a1M2H#s is entered as a test. Are these combinations valid or invalid for (1) and (2) above? Answer the combination.

(1) - Invalid (2) – Valid.

QUIZ [6]

Total C0 = 23 lines

Test C0 = line 1, 3, 4, 5, 9, 10, 14, 15, 16, 17, 18, 19, 23 = 13 lines

=> C0 = 13/23

Total C1 = 6 conditions (ifs, fors, whiles) = 12 branches

Test C1 = F, TF, F, F, TF, F = 8 branches

=> Total = 8/12

line	code	gcd(1,1,1)			gcd(100,50,25)			gcd(6,3,90)			total		
		run	T	F	run	T	F	run	T	F	run	T	F
1	if(x < 1 y < 1 z < 1){	O	X	O	O	X	O	O	X	O	O	X	O
2	return -1;	X			X			X			X		
	}												
3	if(x == 1 y == 1 z == 1){	O	O	X	O	X	O	O	X	O	O	O	O
4	return 1;	O			X			X			O		
	}												
5	if(x < y){	X	X	X	O	X	O	O	X	O	O	X	O
6	swap = x;	X			X			X			X		
7	x = y;	X			X			X			X		
8	y = swap;	X			X			X			X		
	}												
9	r = x % y;	X			O			O			O		
10	while (r != 0){	X	X	X	O	X	O	O	X	O	O	X	O
11	x = y;	X			X			X			X		
12	y = r;	X			X			X			X		
13	r = x % y;	X			X			X			X		
	}												
14	if(y < z){	X	X	X	O	X	O	O	O	X	O	O	O
15	swap = y;	X			X			O			O		
16	y = z;	X			X			O			O		
17	z = swap;	X			X			O			O		

	}											
18	r = y % z;	X		O			O			O		
19	while (r != 0){	X	X	X	O	X	O	O	X	O	O	X
20	y = z;	X		X			X			X		
21	z = r;	X		X			X			X		
22	r = y % z;	X		X			X			X		
	}											
23	return z;	X		O			O			O		

QUIZ [8]

1. Group A and Group B test a piece of software separately. A finds 10 bugs and B finds 12 bugs. However, 5 bugs are found by both A and B.

Choose the answer that is closest to your estimate of the total number of bugs (those already found and those yet to be found).

a=10, b=12, c=5

Formula: $a*b/c = 10*12/5 = 24$

24.

2. Group A and Group B test a piece of software separately. A finds 21 bugs and B finds 15 bugs. However, 8 bugs are found by both A and B.

Choose the answer that is closest to your estimate of the total number of bugs (those already found and those yet to be found).

a=21, b=15, c=8

Formula: $a*b/c = 21*15/8 = 39.375$

39.

3. Group A and Group B test a piece of software separately. A finds 12 bugs and B finds 13 bugs. However, 6 bugs are found by both A and B.

Choose the answer that is closest to your estimate of the number of bugs that remain to be found.

a=12, b=13, c=6

Formula: $a*b/c - a - b + c = 12*13/6 - 12 - 13 + 6 = 7$

7.

4. Group A and Group B test a piece of software separately. A finds 20 bugs and B finds 19 bugs. However, 6 bugs are found by both A and B.

Choose the answer that is closest to your estimate of the number of bugs that remain to be found.

a=20, b=19, c=6

Formula: $a*b/c - a - b + c = 20*19/6 - 20 - 19 + 6 = 30.333$

30.

5. Suppose a project team's previous developments had a bug density of 2.9 [items/KLOC]. The team has just developed a new piece of software with 26 KLOC. Choose the answer that is closest to your estimate of the total number of bugs.

bug_rate = 2.9, loc = 26

total = bug_rate * loc = 2.9 * 26 = 75.4

75.

6. Suppose a project team's previous developments had a bug density of 3.2 [cache/KLOC]. The team develops a new piece of software that is 15 KLOC long. And when they test the software, they find 21 bugs.

Choose the answer that is closest to your estimate of the number of bugs that remain to be found.

bug_rate = 3.2, loc = 15, bugs_found = 21

bugs_remain = bug_rate * loc - bugs_found = 3.2 * 15 - 21 = 27

27.

7. Download srgm-data3.csv (included in lecture08-materials.zip) and fit the exponential reliability growth model to the data. Then, find a 90% confidence interval for the total number of bugs and select all the bug counts that fall within this interval.

- 1220.
- 1278.
- 1279.

8. Download srgm-data3.csv and fit a delayed sigmoidal reliability growth model to the data. Then, find the 90% confidence interval for the total number of bugs, and select all the bugs that fall within this interval. *

- 152.
- 173.
- 190.
- 194.

QUIZ [9]

1. What term is explained in the following passage?

“Writing numbers and data directly into source code. If such numbers and data appear multiple times, it is better to use a macro to centrally manage them.”

Hard coding.

2. What term is explained in the following passage?

“It means a program whose execution flow is complicated and very difficult to read. It means a bad program.”

Spaghetti program.

3. What term is explained in the following passage?

“The rules for writing source code (including variable naming and comment writing)”.

Coding standard.

4. What term is explained in the following passage?

“Preparing a test program prior to programming.”

Test first.

5. What term is explained in the following passage?

“A modification to improve the readability and maintainability of source code (while preserving functionality).”

Refactoring.

6. Choose all the terms that mean “bad” programming.

- **Hard coding.**
- **Spaghetti programming.**

7. Choose an action that was explained in class as “bad programming (rather bad)”.

Write comments on every line of a program.

8. Choose all that are correct to explain pair programming.

- **Code reviews are also conducted at the same time as development.**
- **Simple mistakes are reduced and development becomes more efficient.**
- **Two people use one PC to program.**

9. What is the first thing to do after preparing a test in test-driven development?

Run the test and verify that it fails.

QUIZ [13]

Recall = bug đoán đúng / tổng bug thực tế

Precision = bug đoán đúng / tổng bug dự đoán

F-score = $(2 * \text{recall} * \text{precision}) / (\text{recall} + \text{precision})$

$y = \log_{10}(p / (1 - p))$

		Đoán đúng 10 bug		Tổng dự đoán 41 bug
Results of predictions using training data		Actual		total
		Buggy	Non Buggy	
Prediction	Buggy	10		41
	Non Buggy			
total		30		

Tổng số bug thực tế 30 bug

$$\text{Recall} = \frac{10}{30} \cong 0.333 \quad \text{Precision} = \frac{10}{41} \cong 0.244$$

Recall = Đoán đúng bug / Tổng số bug thực tế

Precision = Đoán đúng bug / Tổng dự đoán

F-score

□ **Harmonic mean of recall and precision**

$$\frac{1}{F-score} = \frac{\frac{1}{recall} + \frac{1}{precision}}{2}$$



$$\begin{aligned} F-score &= \frac{1}{\frac{1}{2} \left(\frac{1}{recall} + \frac{1}{precision} \right)} \\ &= \frac{2 \cdot recall \cdot precision}{recall + precision} \end{aligned}$$

$$y = a_1x_1 + a_2x_2 + a_3x_3 + b$$



$$\log \frac{p}{1-p} = a_1x_1 + a_2x_2 + a_3x_3 + b$$

□ This is called a **logistic regression**

model.

Software Testing – Review

Review for the Final Exam

Review for the Final Exam

- [2] Software Engineering
- [3] Software Development Process
- [4] Black Box Testing Techniques
- [6] White Box Testing Techniques
- [8] Evaluation of Testing and Reliability
- [9] Programming Tips and Techniques
- [12] Software Quality Management & Software Metrics
- [13] Bug Prediction and Test Plan

[2] Software Engineering (1/6)

□ Hệ thống máy tính

- Software và Hardware
- Software: Vai trò là làm chủ Hardware (Nghĩa là tận dụng tối đa Hardware)

□ Kỹ thuật phần mềm (Software Engineering)

- Các lĩnh vực / ngành học để phát triển hiệu quả phần mềm chất lượng cao
- Ước lượng man-month/hour/day cũng là 1 nhánh
- The Mythical Man-Month: Thêm người vào 1 dự án đang trễ chỉ làm cho nó càng trễ thêm

[2] Software Engineering (2/6)

□ Độ tin cậy phần mềm (Software Reliability)

- Lỗi phần mềm gây ra lỗi nghiêm trọng trong đời sống
- → Phần mềm cần phải được kiểm thử (testing) ở nhiều tình huống thực tế khác nhau

□ Đặc trưng phần mềm (Software Features)

- Khó nắm bắt tình hình thực tế
- Tập trung vào quá trình phát triển
- Thời gian vận hành và bảo trì lâu dài
- Ít tái sử dụng

[2] Software Engineering (3/6)

□ Software Reuse (Tái sử dụng phần mềm)

- Blackbox reuse: Frameworks, Libraries
- Whitebox reuse: Copy paste và chỉnh sửa lại

□ Good Software (Phần mềm tốt)

- User / Client:
 - Đáp ứng các yêu cầu đặc tả
 - Dễ sử dụng, thân thiện, nhanh, nhẹ, ...
- Dev:
 - Chi phí thấp và thời gian ngắn trong khi vẫn đáp ứng yêu cầu
 - Khả năng bảo trì dễ dàng

[2] Software Engineering (4/6)

□ Phân loại phần mềm

- Basic software
 - OS, Compiler/Interpreter, Service Programs (Utilities), ...
- Application software
 - Phần mềm bảng tính, trình chiếu, ...
- Middleware
 - DBMS, Web server, Application server ...
- Embedded software
 - Phần mềm cho Thiết bị điện, ô tô, thang máy
 - Phần mềm cho Thiết bị sức khỏe, viễn thông, cảm biến, ...

[2] Software Engineering (5/6)

□ Vấn đề

- Nhu cầu về các hệ thống ngày càng tăng, phần mềm ngày càng trở nên lớn và phức tạp hơn
- Chi phí phát triển và bảo trì phần mềm tăng lên theo thời gian

□ Giải pháp

- Thiết lập các lý thuyết và kỹ thuật liên quan đến phát triển phần mềm
- Hệ thống hóa thành các "kỹ thuật, tiêu chuẩn" chung
→ **Software Engineering**

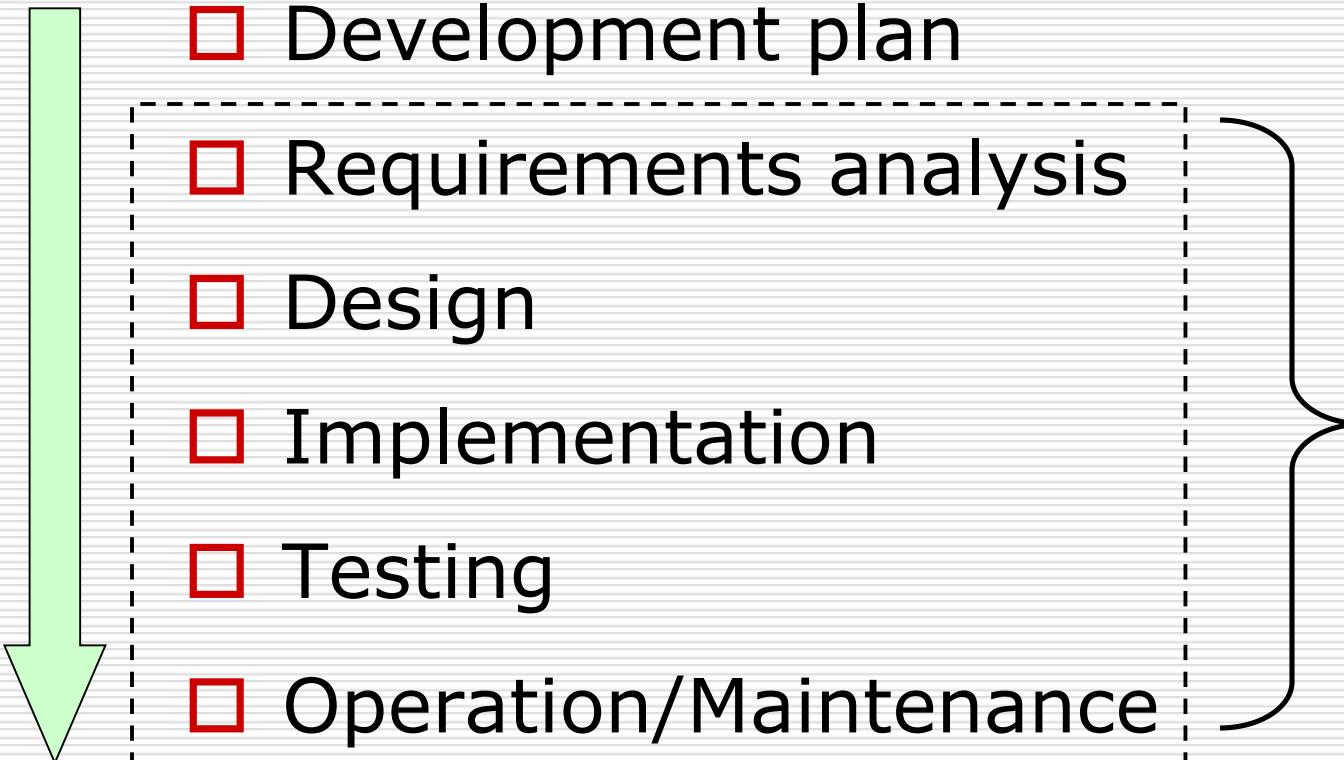
[2] Software Engineering (6/6)

□ Khó khăn và hướng giải quyết

- Khó khăn trong phân tích yêu cầu
 - Tạo các Artifacts một cách rõ ràng và chi tiết, ...
- Khó khăn trong tái sử dụng mã nguồn
 - Nắm bắt Design Patterns, System Architecture, ...
- Khó khăn trong quản lý dự án
 - Theo dõi tiến độ, nhân sự, trao đổi giữa các bên liên quan, các quy trình phát triển tiên tiến (Agile), ...
- Khó khăn trong việc ước tính thời gian và chi phí
 - Sử dụng các mô hình, phương pháp thống kê để ước lượng và dự đoán, ...

[3] Software Development Process (1/8)

□ Quy trình phát triển phần mềm



[3] Software Development Process (2/8)

□ Kế hoạch phát triển (Development Plan)

■ Lên kế hoạch (Setup)

- Loại phần mềm
- Thời hạn giao cho khách hàng

■ Ước lượng (Estimation)

- Man-hours / chi phí
- Khả năng mở rộng

■ Chuẩn bị (Preparation)

- Nguồn nhân lực
- Ngân sách
- Môi trường phát triển

[3] Software Development Process (3/8)

□ Phân tích yêu cầu (Requirements Analysis)

- Yêu cầu nghiệp vụ
 - Yêu cầu chức năng
 - Yêu cầu phi chức năng
- Mô tả yêu cầu
 - Mô tả chính xác, không mơ hồ
 - Xem xét tính khả thi: chức năng, chi phí, thời hạn
- Artifact/Deliverable: Đặc tả cho các yêu cầu
 - Sơ đồ Use-case, Đặc tả Use-case, ...
 - Class / Activity / Sequence Diagram, ...
 - UI, Dev/Deploy Environment, ...

[3] Software Development Process (4/8)

□ Thiết kế (Design)

Artifact

- Kiến trúc hệ thống, Cấu trúc chương trình
 - Đặc tả phần mềm: Dependencies, Libraries, Frameworks, Services, ...
 - Đặc tả phần cứng: Máy khách, máy chủ, môi trường mạng, ...
- Các chức năng, UI, cách vận hành và trao đổi của từng thành phần
- Cấu trúc dữ liệu và giải thuật, ...

[3] Software Development Process (5/8)

❑ Hiện thực (Implementation)

Artifact

- Source code
- Đặc tả chương trình
 - ❑ Các scripts thực thi, tệp cấu hình, ...
 - ❑ Tài liệu kỹ thuật: Mô tả hành động nội bộ của phần mềm (flowchart, kiến trúc, thuật toán, cấu trúc dữ liệu và UI), mô tả về API, ...

[3] Software Development Process (6/8)

□ Kiểm thử (Testing)

Artifact

■ Đặc tả kiểm thử (Test Specifications)

- Test Plan (Cách tiếp cận, mục tiêu, phạm vi và lịch trình tổng thể của các hoạt động kiểm thử)
- Test Scripts (Cho kiểm thử tự động)
- Test Data (dữ liệu mẫu, các trường hợp/điều kiện biên và các kịch bản thực tế)

■ Báo cáo kiểm thử (Test Reports)

- Các Test cases và Kết quả (Báo cáo lỗi, phạm vi/độ phủ của việc kiểm thử)
- Các báo cáo về Hiệu suất, Bảo mật, sự chấp nhận của user

[3] Software Development Process (7/8)

□ Vận hành & bảo trì (Operation & Maintenance)

Artifact

- Tài liệu hướng dẫn
 - Cách cài đặt, định cấu hình, cách sử dụng các chức năng
 - Hướng dẫn khắc phục sự cố
 - Bảo trì và hỗ trợ
- Báo cáo lỗi, Báo cáo phản hồi, ...

[3] Software Development Process (8/8)

□ Development Process Model

- Thác nước (Waterfall)
 - Tiến hành tuần tự từ trên xuống
 - Dễ nắm bắt tiến độ nhưng giai đoạn sau cần chờ giai đoạn trước đó hoàn thành
- Tạo mẫu (Prototyping)
 - Dựa trên các nguyên mẫu (prototype)
→ Đễ xác nhận và đánh giá thông số kỹ thuật và thiết kế
- Xoắn ốc (Spiral)
 - Có sự kiểm tra và lặp lại ở các bước (theo chu kỳ)

[4] Black Box Testing Techniques (1/8)

□ Kiểm thử (Testing)

- Tìm lỗi
- Test case < Test suite < Test domain
- Kiểm thử hộp đen (Blackbox Testing)
 - Dựa trên các đặc tả
- Kiểm thử hộp trắng (Whitebox Testing)
 - Dựa vào cấu trúc của chương trình
- Kiểm thử ngẫu nhiên (Random Testing)
 - Tạo ra các test cases ngẫu nhiên và kiểm thử

Lớp tương đương: Chương trình sẽ xử lý chúng theo cùng một cách thức

[4] Black Box Testing Techniques (2/8)

□ Kỹ thuật Blackbox Testing 1 – Phân vùng tương đương (Equivalence Partitioning)

- Chia input thành các lớp tương đương bao gồm hợp lệ (Valid Input) và không hợp lệ (Invalid Input)
- **B1.** Đặt lớp tương đương hợp lệ và lớp tương đương không hợp lệ cho từng điều kiện đầu vào.
- **B2.** Tạo tất cả các kết hợp của **các lớp tương đương hợp lệ**
- **B3.** Tạo tất cả các kết hợp **chỉ chứa một** lớp tương đương không hợp lệ

[4] Black Box Testing Techniques (3/8)

□ Kỹ thuật Blackbox Testing 1 – Phân vùng tương đương (Equivalence Partitioning)

■ Ví dụ:

- Input 1 có 3 lớp valid – 2 lớp invalid
- Input 2 có 2 lớp valid – 1 lớp invalid
- → Số test case valid: $3 \times 2 = 6$
- → Số test case invalid: $2 \times 2 + 1 \times 3 = 7$
- **→ Tổng: 13 (Đây là số lượng cơ bản của phương pháp này, còn nhiều biến thể khác sẽ có số lượng test cases khác)**

■ Thực hiện kiểm thử theo từng cột

[4] Black Box Testing Techniques (4/8)

□ Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)

- Kiểm thử dựa trên các **giá trị biên/ranh giới** của các lớp tương đương hợp lệ và không hợp lệ
- **B1.** Xác định **miền giá trị** cho từng điều kiện đầu vào
- **B2.** Với từng input, xem xét 5 giá trị: [**min, min+, nominal, max-, max**] (nominal là giá trị trung bình)
- **B3.** Cố định các giá trị **nominal** của $n-1$ biến và thực hiện kết hợp với từng giá trị của biến còn lại
- Với Standard BVA, số test cases là **$4n + 1$**

[4] Black Box Testing Techniques (5/8)

□ Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)

- Ví dụ: Giả sử có 3 biến đầu vào là X, Y, Z
 - **X: 0 → 100.**
 - **Y: 20 → 60.**
 - **Z: 80 → 100.**
- Sử dụng Standard BVA để tạo các test cases

[4] Black Box Testing Techniques (6/8)

□ Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)

- Ví dụ: 3 biến đầu vào là X, Y, Z đã cho
- B1: Tìm các giá trị cho mỗi biến

	X	Y	Z
Min	0	20	80
Min+	1	21	81
Nominal	50	40	90
Max-	99	59	99
Max	100	60	100

[4] Black Box Testing Techniques (7/8)

□ Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)

- Ví dụ: 3 biến đầu vào là X, Y, Z đã cho
- B2: Cố định nominal của X, Y và tạo kết hợp với mỗi giá trị của Z
- B3: Thực hiện tương tự cho các cặp còn lại: Y, Z và X, Z

[4] Black Box Testing Techniques (8/8)

□ Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)

- Ví dụ: 3 biến đầu vào là X, Y, Z đã cho

Kết quả

(Lưu ý: Đây là Standard BVA, còn nhiều biến thể khác sẽ có số lượng test cases khác)

Test case #	X	Y	Z
1	50	40	80
2	50	40	81
3	50	40	90
4	50	40	99
5	50	40	100
6	0	40	90
7	1	40	90
8	99	40	90
9	100	40	90
10	50	20	90
11	50	21	90
12	50	59	90
13	50	60	90

[6] White Box Testing Techniques (1/3)

- Tập trung vào cấu trúc của chương trình
- C0 (Instruction Coverage)
 - Số dòng lệnh được thực thi ít nhất 1 lần ÷ Tổng số dòng lệnh
- C1 (Branch Coverage)
 - Số nhánh True, False được thực thi ít nhất 1 lần của các nhánh điều kiện (if, for, while) ÷ Tổng số nhánh True, False của các nhánh điều kiện
- C2 (Condition Coverage)
 - Số các kết hợp True / False **CÓ THỂ CÓ** giữa các nhánh điều kiện được thực thi ít nhất 1 lần ÷ Tổng số các kết hợp True / False **CÓ THỂ CÓ** của các nhánh điều kiện

[6] White Box Testing Techniques (2/3)

□ Mối quan hệ

- Tuân theo 1 chiều:
C2 100% → C1 100%, C1 100% → C0 100%
(Các trường hợp khác đều không đúng)
- Độ phủ đạt 100% (bất kể C0, C1, C2 hoặc cả 3)
KHÔNG có nghĩa là chương trình không có lỗi
- Việc viết test case theo điều kiện nào trong C0, C1, C2
cũng sẽ dẫn đến **độ chính xác và số lượng test case khác nhau**
→ Cần xác định và thống nhất việc kiểm thử

[6] White Box Testing Techniques (3/3)

□ Random Testing

- Tạo tự động một lượng lớn các test cases
- Tìm ra các kịch bản không ngờ đến
- Không thể kiểm thử theo đặc tả
- Tỷ lệ bao phủ thấp

□ Blackbox Testing

- Được dùng từ Unit Testing đến System Testing

□ Whitebox Testing

- Chủ yếu trong Unit Testing và Small Integration Testing

[8] Evaluation of Testing and Reliability (1/5)

- **Đánh giá hoạt động kiểm thử**
 - Dựa trên tỷ lệ phủ (Coverage rate)
 - Dựa trên tỷ lệ xử lý lỗi (Bug resolution rate)
 - Số bugs đã xử lý ÷ Tổng số bugs
 - Không biết chính xác Tổng số bugs → Ước lượng
 - Dựa trên vận hành (Operability)
 - MTBF (Mean Time Between Failures): Thời gian trung bình giữa các lần xảy ra lỗi

[8] Evaluation of Testing and Reliability (2/5)

- Phương pháp ước lượng Tổng số bug 1 –
Capture / Recapture
 - Chuẩn bị n bug nhân tạo
 - Giả sử kiểm thử tìm thấy x bug nhân tạo và y bug thật
 - Tổng số bug = $\frac{n \cdot y}{x}$
 - → Các lỗi nhân tạo thiếu thực tế và có chủ định trước

[8] Evaluation of Testing and Reliability (3/5)

□ Phương pháp ước lượng Tổng số bug 2 – Two-step Editing

- Số lỗi được tìm thấy bởi nhóm A = **a**
- Số lỗi được tìm thấy bởi nhóm B = **b**
- Số lỗi được tìm thấy trong cả hai nhóm = **c**
- Tổng số bug = $\frac{ab}{c}$
- Số bug còn lại = $\frac{ab}{c} - a - b + c$

- → Tốn nhiều thời gian và chi phí

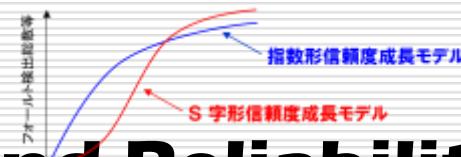
[8] Evaluation of Testing and Reliability (4/5)

□ Phương pháp ước lượng Tổng số bug 3 – Past Statistical Data (Density)

- Mật độ lỗi trước đó x [bugs/KLOC]
- Quy mô phát triển y [KLOC]
- Tổng số bug = $x * y$

□ Đường cong tăng trưởng (Growth Curve)

- Một đường cong mô hình cách dữ liệu mục tiêu thay đổi theo thời gian
- Sử dụng giá trị tích lũy để bắt kịp tốc độ tăng trưởng của dữ liệu mục tiêu (số bugs)



[8] Evaluation of Testing and Reliability (5/5)

□ Software Reliability Growth Model (SRGM)

- 1 mô hình điển hình về mối quan hệ giữa thời gian kiểm thử và tổng số bug được phát hiện
 - Exponential SRGM: $a(1 - e^{-bt})$
 - Delayed S-shaped SRGM: $a \{ 1 - (1 + bt)e^{-bt} \}$
 - ...
- Mục tiêu: Ước tính các tham số của đường cong để khớp với dữ liệu thực tế nhất có thể
- → Từ đó có thể ước lượng tổng số bug trong tương lai

[9] Programming Tips and Techniques (1/5)

□ Programming

- Là 1 bước trong quy trình phát triển phần mềm
- Mục tiêu:
 - Đáp ứng các đặc tả yêu cầu
 - Dễ hiểu
 - Dễ bảo trì
- Phương pháp:
 - Code conventions
 - Các quy định chung của nhóm, dự án, công ty, ...

[9] Programming Tips and Techniques (2/5)

□ Dễ đọc và hiểu

- Thụt lề, Khoảng trắng, Dòng mới
- Đặt tên hàm, tên biến mang ngũ nghĩa

□ Ngăn ngừa lỗi

- Đóng mở ngoặc đầy đủ
- Không đặt tên biến trùng nhau

□ Cấu trúc đơn giản

- Tránh dùng goto
- Tránh quá nhiều điều kiện rẽ nhánh

[9] Programming Tips and Techniques (3/5)

□ Độc lập với môi trường

- Sử dụng các cú pháp, hàm dùng chung

□ Comment thích hợp

- Tránh “Thiếu Why Thừa What”

□ Dễ dàng thay đổi

- Tránh Hardcode
- Sử dụng các Macros

[9] Programming Tips and Techniques (4/5)

□ XP (eXtreme Programming)

- Dùng trong quy mô nhỏ (Thời gian ngắn và nhân lực ít)

□ XP Practice 1 – Test-first

- Viết các test cases trước rồi mới viết chương trình
- Dùng trong Test-Driven Development

□ XP Practice 2 – Refactoring

- Tinh chỉnh code, tránh duplicate code, tách code, ...
- Tăng cường tính dễ đọc dễ hiểu
- Cải thiện hiệu suất và khả năng bảo trì chương trình

[9] Programming Tips and Techniques (5/5)

□ XP Practice 3 – Pair Programming

- 1 người viết, 1 người quan sát/đánh giá/góp ý/trao đổi
→ Đề xuất cải tiến và sửa lỗi trong quá trình thực hiện, sau đó hoán đổi vị trí cho nhau
- Chia sẻ kiến thức/kinh nghiệm, phát hiện lỗi sớm

□ XP Practice 4 – Coding Standards

- Áp dụng các quy tắc về Code Conventions của ngôn ngữ sử dụng
- Một nhóm phải có các phương pháp viết mã chung, sử dụng cùng định dạng và phong cách viết mã

[12] Software Quality Management and Software Metrics (1/6)

□ Đảm bảo chất lượng phần mềm

■ Kiểm tra

- Kiểm thử đầy đủ trước khi release
- Fix các lỗi phát hiện được

■ Giám sát và cải tiến

- Ghi lại, phân tích, đánh giá và cải thiện các công việc khác nhau trong quá trình phát triển

■ Bảo trì

- Đảm bảo phần mềm hoạt động một cách phù hợp

■ Kiểm thử hồi quy (Regression Testing)

[12] Software Quality Management and Software Metrics (2/6)

Bảo trì thích ứng

- Đáp ứng sự thay đổi của môi trường (OS, libraries, framework, ...)

Bảo trì khắc phục

- Sửa các lỗi xảy ra trong lúc sử dụng phần mềm

Bảo trì khẩn cấp

- Sửa các lỗi bất chợp, tạm thời

Bảo trì tăng cường

- Cải thiện tính năng, yêu cầu mới

Bảo trì hoàn hảo

- Tinh chỉnh mã, cải thiện hiệu suất

Bảo trì phòng ngừa

- Rà soát và sửa các lỗi trước khi phát sinh

[12] Software Quality Management and Software Metrics (3/6)

□ TQC (Total Quality Control)

- Kiểm soát chất lượng với sự tham gia của tất cả các bên liên quan, bao gồm cả các nhà quản lý, lãnh đạo,

...

□ Tập trung vào quản lý

- Ưu tiên: Quality, Cost, Delivery

□ Quản lý ngược dòng

- Xác định và giải quyết nguồn gốc ngược dòng vấn đề

□ Phòng ngừa tái diễn

- Học hỏi từ những vấn đề đã xảy ra trong quá khứ

[12] Software Quality Management and Software Metrics (4/6)

□ Nắm bắt vấn đề từ dữ liệu

- Số hóa những gì đang thực sự xảy ra và xử lý các vấn đề bằng cách đánh giá chúng một cách logic, khách quan và thống kê

□ Tiêu chuẩn hóa

- Manuals, Rules, Patterns

□ PCDA cycle

- Plan → Do → Check → Act

[12] Software Quality Management and Software Metrics (5/6)

□ Nguyên lý Pareto

- Quy tắc 80 : 20
- 80% số lỗi của phần mềm đến từ 20% modules
- → Biết được nơi cần cải thiện để đạt hiệu quả cao nhất

□ Chỉ số phần mềm (Software Metrics)

- Đo lường các đặc tính chất lượng phần mềm
- LOC (Lines of Code) / KLOC
- Cyclomatic Number (McCabe)
- CALL, ...

[12] Software Quality Management and Software Metrics (6/6)

□ Phân tích dữ liệu (Data Analysis)

- Histogram: Biểu đồ thể hiện tần suất của dữ liệu
- Kernel Density Estimation:
 - Ước tính hàm mật độ xác suất của dữ liệu
 - Phân phối lệch phải (**Right-Skewed**) là phổ biến nhất

□ Mean và Median

- Mean: Giá trị trung bình của một tập dữ liệu
- Median: Trung vị là giá trị ở trung tâm khi tập dữ liệu được sắp xếp từ nhỏ đến lớn (Nếu có 2 vị trí ở trung tâm thì cộng lại chia trung bình)

[13] Bug Prediction and Test Plan (1/4)

□ Fault-Prone Module Analysis

- Ý nghĩa: Tìm các đặc trưng của những module dễ bị lỗi theo số liệu
- Tập trung vào phân bố của lỗi
- Sử dụng thống kê để dự đoán xu hướng tổng thể

□ Phương pháp

- Chọn 1 loại dữ liệu (LOC, CC, CALL, ...)
- Giả định nếu nó lớn hơn một giá trị nhất định → lỗi
- Chia dữ liệu thành 2 phần: Train và Test

[13] Bug Prediction and Test Plan (2/4)

□ Recall

- Số dự đoán đúng được dự đoán là lỗi / Tổng số lỗi thực tế

□ Precision

- Số dự đoán đúng được dự đoán là lỗi / Tổng số dự đoán

□ F-score

- Precision và Recall càng cao càng tốt. Tuy nhiên trong thực tế nếu ta điều chỉnh model để tăng Recall quá mức có thể dẫn đến Precision giảm và ngược lại.
- Cần tìm 1 đại lượng trung hòa / cân bằng 2 giá trị này
→ **F-score**: Trung bình điều hòa

[13] Bug Prediction and Test Plan (3/4)

□ Test Prioritization

- Kiểm thử dựa trên các giá trị chỉ số giảm dần
- Mục tiêu: Cần kiểm thử bao nhiêu phần trăm để có thể tìm thấy 80% tổng số lỗi? → Số càng thấp càng tốt

□ Dự đoán với nhiều số liệu

- Kết hợp LOC, CC, CALL, ... trong cùng 1 mô hình
- Mỗi đại lượng sẽ có các trọng số khác nhau
- Các phép biến đổi như Logarit, Sigmoid, ... là để đưa miền giá trị của hàm dự đoán y về [0,1]

[13] Bug Prediction and Test Plan (4/4)

□ Vấn đề

- Mô hình dự đoán sẽ không tốt nếu dữ liệu bị mất cân bằng (Overwhelming) → Dẫn đến Overfitting

□ Cách giải quyết:

- 1. Thay đổi trọng số của các đại lượng trong mô hình
- 2. Thay đổi lượng dữ liệu để cân bằng lại
 - Undersampling: Giảm số lượng lớp đa số
 - Oversampling: Tăng số lượng lớp thiểu số
→ Thuật toán SMOTE là 1 trong số đó



Hỏi & Đáp



Cảm ơn!

Chúc các bạn thi tốt!

ソフトウェアテスト

[2] ソフトウェア工学の概要

Software Testing
[2] Overview of Software Engineering

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Role of software

- A computer system is a combination of hardware and software

Computer system



The role of software is to make full use of the hardware.

How to utilize hardware depends on software

Games are a typical example

Software Engineering

- Academic field for efficiently developing high-quality software
 - It's not a way of thinking that 'if only a few people try their best, something will happen.'
 - Academic field to make development work

Theory and technology for efficient and high-quality requirements analysis, design, programming, testing, etc.



Theory and techniques for successfully managing development activities (development management)

The Mythical Man-Month

- **Man-month**=number of people x number of months
 - * (Ex) 1 man-month = 1.2 million yen
- Unit of labor hours required for development

(Ex) 6 man-months = 2 months of work with 3 engineers

10 more man-months of work left! However, there is only half a month left until the delivery date (0.5 month)

Bad manager

So, should we increase the number of staff by 20 people?

$$\text{※ } 10/0.5 = 20$$

There is chaos at the scene!

An organized development process is important. (Man-hour/month estimation is also a field of software engineering)

Software Liability Is Serious

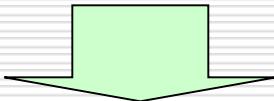
- The key to safety and security in everyday life
- Software bugs can cause serious damage
 - Possibility of impact on electricity, gas, water supply, transportation, and finance
 - (Ex) Stock exchange stops → Major damage
 - may cause accidents
 - (Ex) Elevator malfunction → Personal injury

Software reliability

Needs to be tested for a variety of situations

Software Features (1/4)

□ It is difficult to grasp the actual situation

- Software is logical entity, not physical entity
- In short, **It does not actually exist as a “thing”**
- It is difficult to know to what stage development has progressed and whether products are manufactured according to specifications

Software Features (2/4)

- Work concentrated on the development process
 - Hardware, the flow is to develop a product and manufacture it in a factory.
 - For software, manufacturing is just a copying
 - Quality and cost are centered on the development process

Hardware: Defective products → During the manufacturing process
Software: Defective products = "Everything is defective from the beginning" → It's all about not failing during development

Software Features (3/4)

- Long period of operation and maintenance
 - Operation (Actually used by users)
 - Maintenance (Modify, improve, and extend functions)

The period of doing is much longer

Ease of maintenance is the key

Software Features (4/4)

Low reuse

- ✓ Hardware: It is common to reuse existing parts
- ✓ Software: It is difficult to complete just by combining parts (requires a lot of customization)
- There is software that can be reused in the form of libraries, but it cannot be used without programming.

Reuse

□ There are two types of software reuse

■ Black Box

No need to worry about the contents, use it like a part

■ White Box

Copy and paste the program and rewrite it as needed

Black box reuse

□ Library form

Example, by using an existing function as is

The content of the source code itself does not matter.

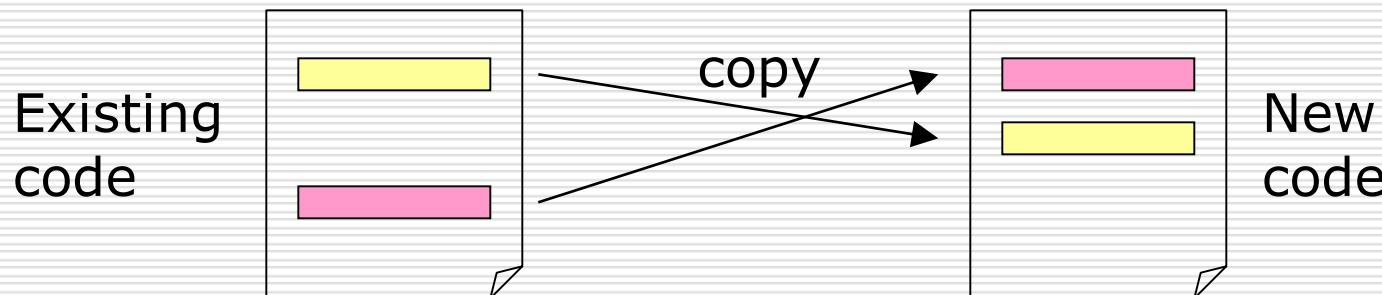
Although it can only be used in a predetermined manner (according to the manual), it is useful.



White box reuse

- "Copy and paste" the source code
 - For example, using part of existing source code.
 - Copy the codes in a book and web page
 - There is also a risk of copying mistakes (bugs) and misuse.

Example: Even if you copy and paste a program that you do not understand, you will only get confused.



[Exercise 1]

Consider the code clone problem

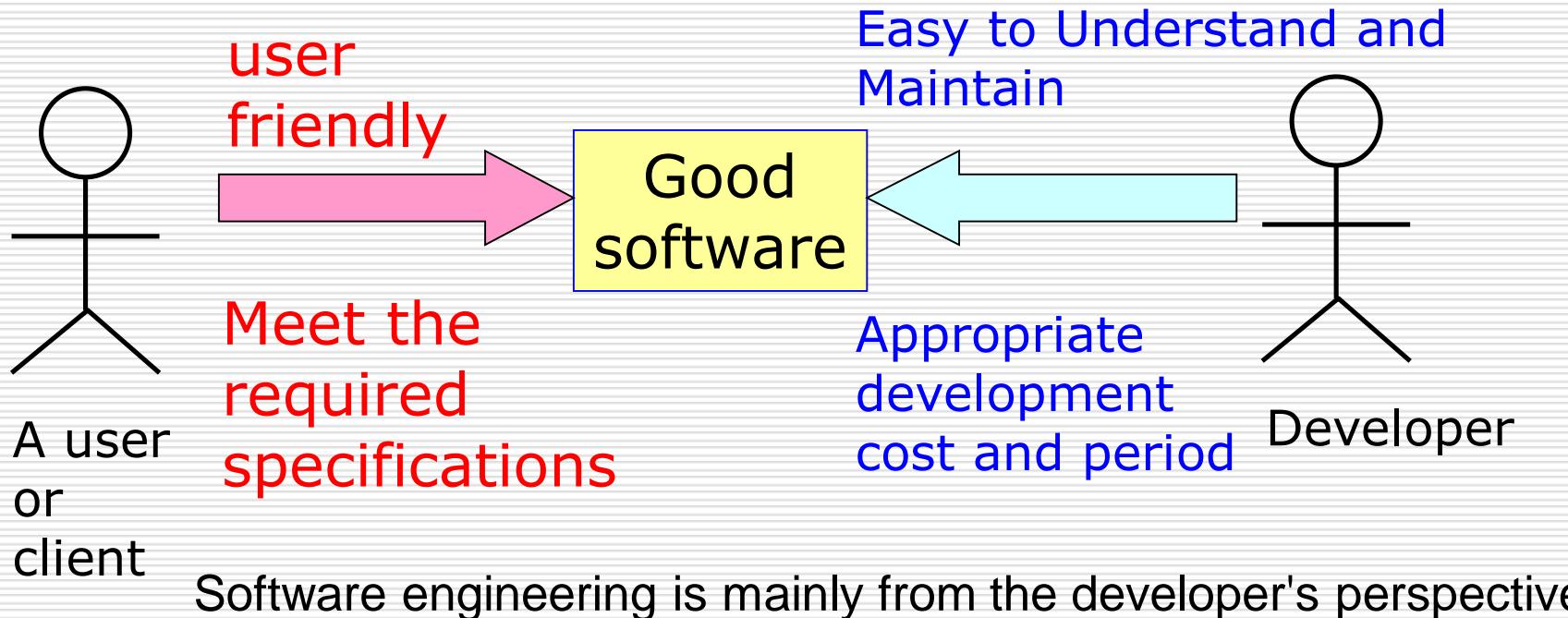
- Copy and paste the source code as it is, or change part of it and reuse it
→ It is called a "**code clone**"

- In the software industry, software with many code clones is regarded as a problem

Why?

What is good software?

- Note that the concept differs depending on the perspective



User's Perspective (1)

Satisfaction of required specifications

Requirements can be broadly divided into two types:

- **Functional requirements**: Functions to be achieved

Ex: User authentication

- **Non-functional requirements** : "How" the requirements should be realized

Ex: Authentication for 500 users must be completed within 3 seconds

It is not about the function itself, but about how to realize it such as performance and security

User's Perspective (2)

Usability

- In a word, "ease of use"

- Easy to operate
 - Easy to understand operation
 - Indirectly, operating speed and memory usage are also related.

(I don't want to use so-called "heavy" systems)

Developer's Perspective (1)

Cost and Period

- Low development cost is desirable
- The development period is within the deadline (by the delivery date)
 - It is rare to be able to develop slowly over time
 - Need to finish on time with limited resources and time

The goal is to develop software that can be developed at **low cost, delivered on time, and meets the required specifications.**

Developer's Perspective (2)

Maintainability

- Once completed and put into operation, the required specifications almost always change.
 - Request for detailed corrections: "I want you to change this"
 - Request for addition or expansion of functions: "More such functions"
- Bugs may be found

Easy-to-main software is “good” software & Easy-to-understand is also important because it may be maintained by another person

Three characters that appear in software development workplaces

□ QCD

- Quality
- Cost
- Delivery

It means three important pillars in the manufacturing industry.

□ KKD

- Intuition
- Experience
- Courage

It means a workplace that is not managed well.

Classification of software

- Basic software
 - Application software
 - Middleware
 - Embedded software
- Image of these three layers built into hardware
-
- The diagram illustrates the three layers of software as stacked horizontal bars. From top to bottom, the colors are light blue, orange, and light green. The text labels for each layer are: 'Application software' in the blue bar, 'Middleware' in the orange bar, and 'Basic software' in the green bar. A thick grey double-headed vertical arrow is positioned to the right of the bars, pointing both upwards and downwards, symbolizing that these three layers are integrated into a single physical hardware device.
- What the user wants to do
- What the calculator can provide?

Basic software

- Software that is the basis for using a computer and executing software
 - Operating System

- Compiler/Interpreter
 - Service program (utility)

(Ex) File editing/searching

The basic software cannot perform any specific tasks, but without it, the story cannot begin in the first place.

(Ex) Just because an OS (Windows, etc.) is installed doesn't mean you can do anything. However, if it isn't installed in the first place, nothing will work.

In other words, "App"

Application software

- Software with a specific purpose
 - For example,
 - Securities transaction management
(industry specific software)
 - Budget management (business software)
 - Spreadsheets, presentations (common application software)
 - Software has become a tool for accomplishing certain tasks or purposes.

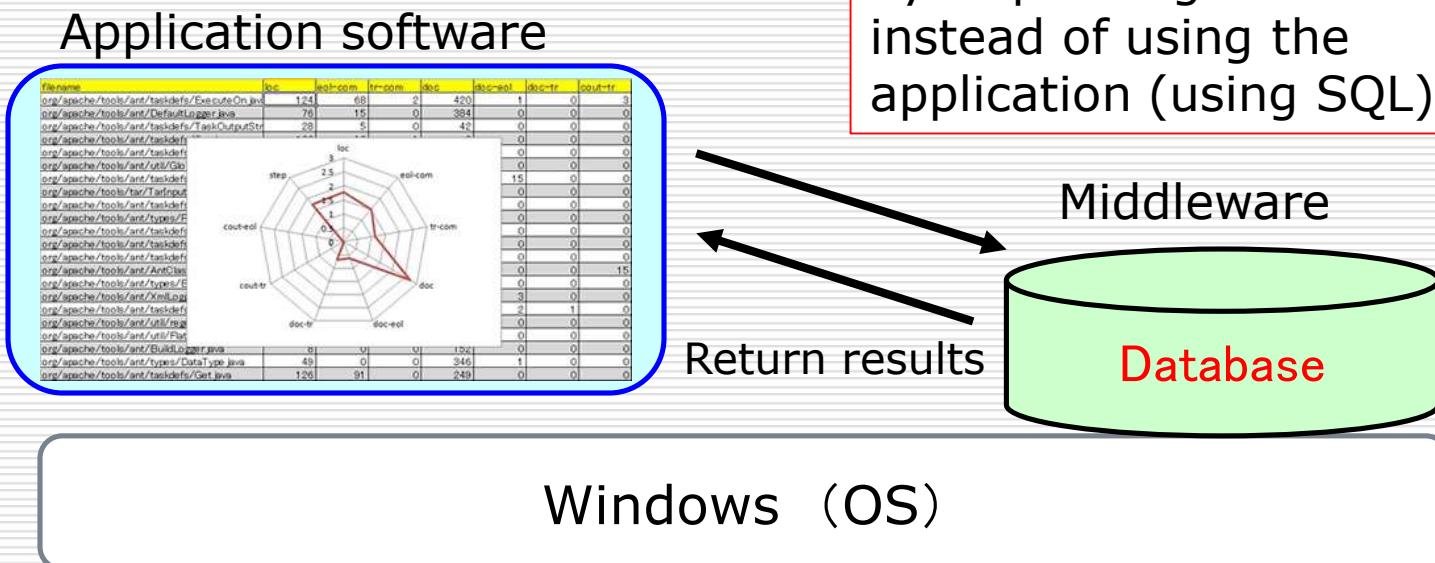
Middleware

- Intermediate between basic software and application software
 - Serves as the basis for application software
 - For example,
 - Database management system
 - Web server
 - Application server
 - Rather than using OS functions directly (system calls, etc.), use higher-level concepts: For example, in SQL

Software “for apps”
rather than
software that users
use directly

Positioning of middleware

- Ex, suppose you have an application that processes large amounts of data on Windows.



Embedded software

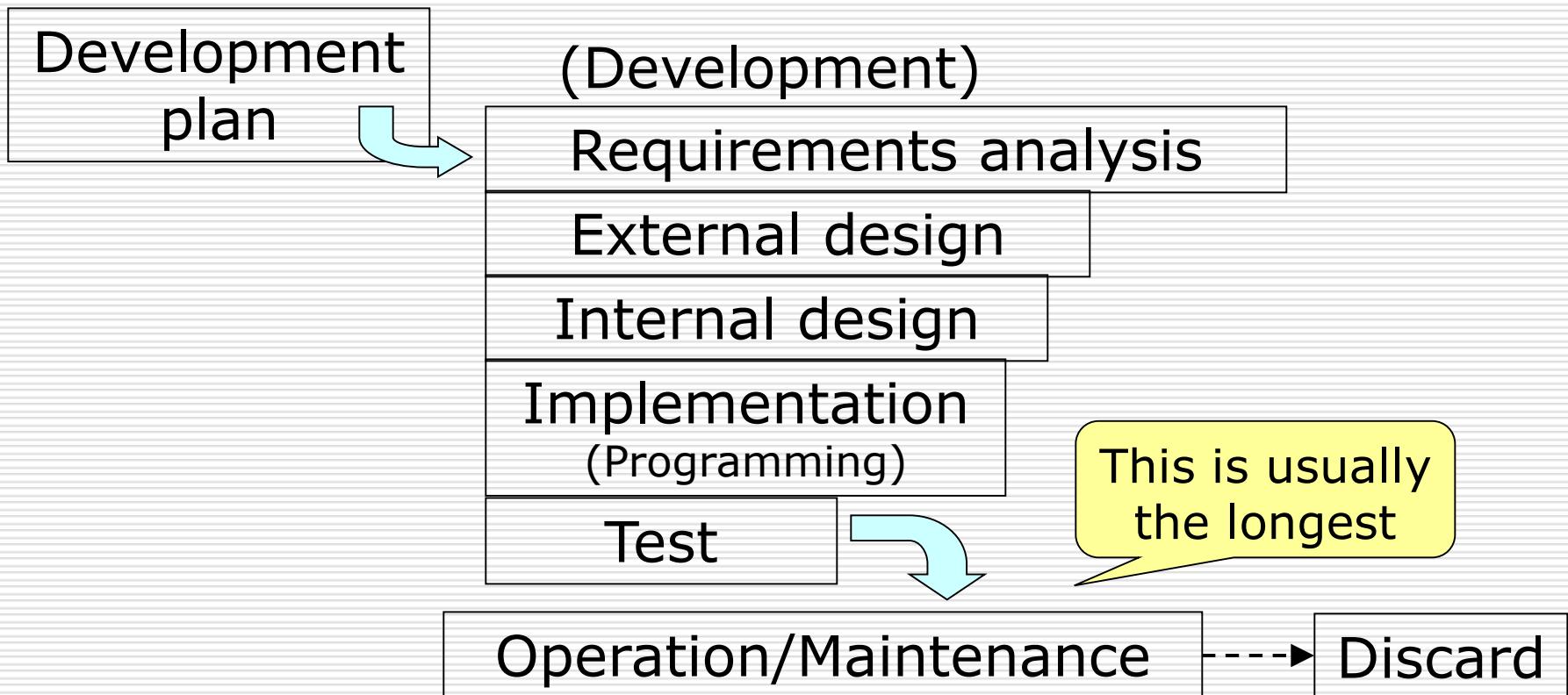
- Software mainly used to control devices in electronic devices that implement specific functions (※It does not run on a general-purpose computer, but is built into a specific device)
 - Ex: Electrical appliances, car navigation systems, elevators, etc
 - Supplied as an integral part of the hardware
 - It is not easy to install a modified version
 - Operating environments are diverse and high reliability is required

[Exercise 2] List the challenges in developing and maintaining embedded software.

- Thinking about embedded software for elevator control
- When developing this software, what is more difficult than general software?
- Also, what about maintenance?

Life cycle

□ Life cycle: The Life of Software



Software Crisis

□ A sense of crisis proposed in 1968

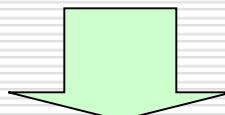
Software development cannot keep up with needs,
hindering the development of computer systems.

- Software development is slow and
hinders computer progress
- Enlargement → Frequent bugs →
Developing into social problems
- Increased development costs

In short, the software is the bottleneck!

(1) Obstructing the progress of computers

- The trend was to let computers do many things (at the time)



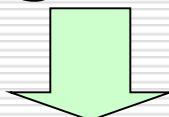
- Let's cheaply mass-produce general-purpose hardware and let software handle the details

But software is basically "handmade"

- Development is not in time!
- Not enough engineers!

(2) Fear of development into social problems

- As needs for systems increase, software becomes larger and more complex.



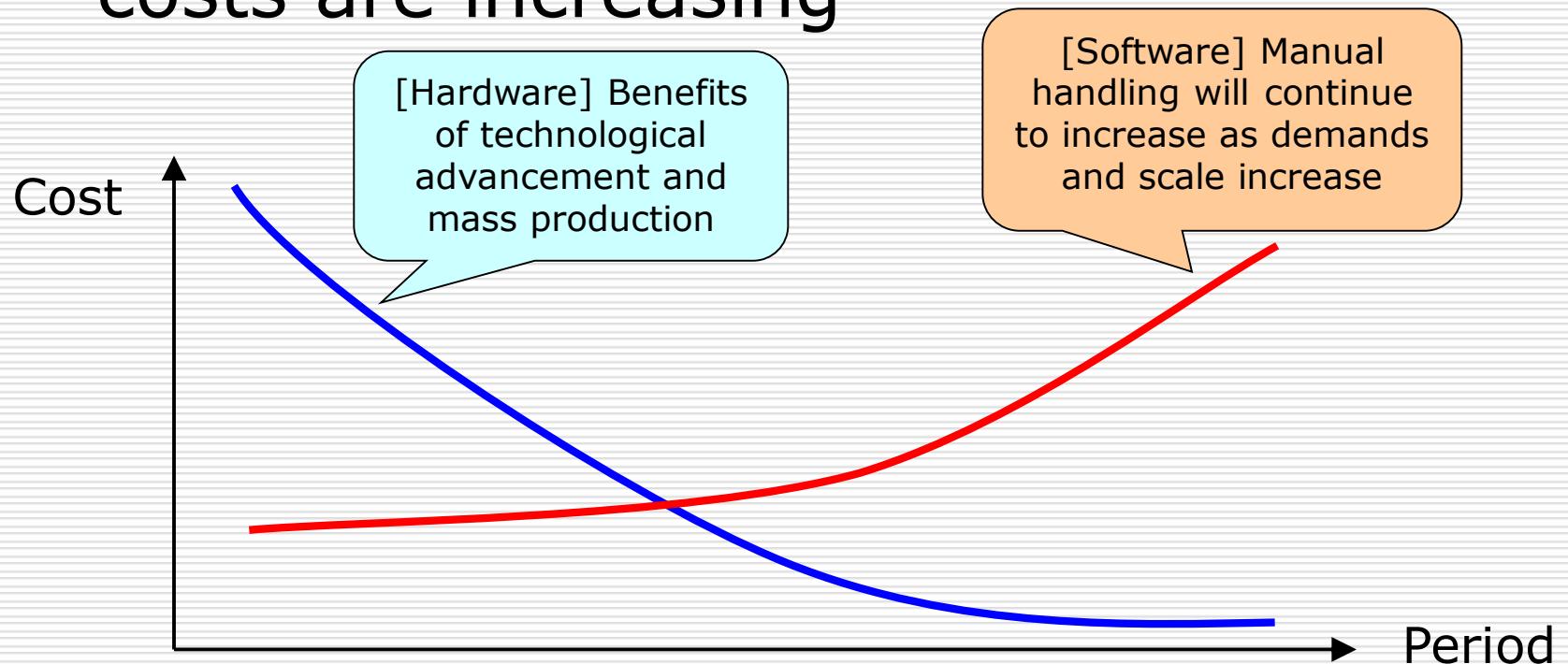
- The risk of human error (so-called bugs) also increases.

Depending on the system and the type of bug, it may cause serious damage to social life.

Possibility of affecting electricity, gas, water, transportation, finance, etc.

(3) Increase in costs

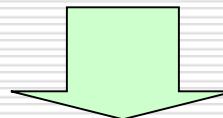
- Software development and maintenance costs are increasing



Birth of “Software Engineering”

- Measures are needed to solve such problems

Researching theories and techniques for efficiently developing high-quality software (until then, it was more like "craftsmanship")



- Establish theories and techniques related to software development, systemize them as "engineering" (the science of manufacturing) to achieve high productivity and high quality!

Current/Future Challenges

- The software industry has developed through various research and technological innovations, but difficult issues still remain.
 - Difficulty in Requirements Analysis
 - Difficulty in Reusing
 - Difficulty in Project Management
 - Difficulty in Estimating

Difficulty in requirements analysis

- Analyze customer requirements,
 - Correctly
 - Without ambiguity
 - Appropriate as a system for development purposes
- It must be written as a specific specification.

This is the first step,
but it is difficult because
it is the closest to humans.

If the customer is a software amateur, he or she will make unexpectedly unreasonable demands.

Confusion arises because different fields (called domains) have different cultures and languages.

Stories like "I can't do that"

There is also a story that 'Even if it is obvious to the other person, we do not know it.'

Difficulty in reusing

□ Source code reuse

- Although it is relatively easy to practice, simple copying and pasting can also cause bugs → **Code cloning**
- **Which code should I use?**

□ Design and architecture reuse

- This is reusing knowledge and know-how at a more abstract level than code, and the hurdles are quite high.
- **Knowledge of design patterns and frameworks required**

Difficulty in project management

- In development project management
 - Progress management
 - Quality management
 - Staffing, communication management

Items such as these are issues for project managers

Difficulty in estimating

- In reality, there is still a large degree of individuality
- In other words, it largely depends on the experience and abilities of individual engineers.
- It is difficult to estimate man-hours, period, and costs from this → **Delays and cost overruns**
 - **Man-hour estimation model (cocomo)**
 - Development organization maturity model (**CMM, CMMI**)
 - **Prediction using statistics, neural networks,..**

Summary

- Software engineering is not only the study of how to create software, **but also the management that leads development projects to success.**
- Although the software industry is developing, there are still many issues to be solved:**requirements analysis, reuse, project management, estimation, etc.**

Homework

Answer “[2] quiz”
by tomorrow 13:00pm

(Notes: Your quiz score will be a part of your final project evaluation)

ソフトウェアテスト

[3] ソフトウェア開発プロセス

Software Testing
[3] Software Development Process

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Outline of software development

□ Development planning (overall plan)

□ Requirements analysis

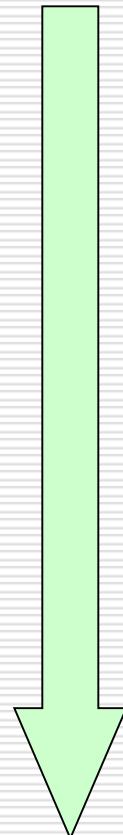
□ Design

□ Implementation

□ Testing

□ Operation/Maintenance

Development process



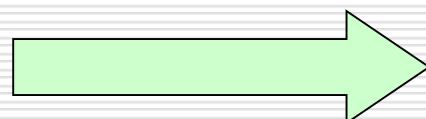
(0) Development planning

- Development planning
- Requirements analysis
- Design
- Implementation (Programming)
- Testing
- Operation/Maintenance

(0) Development planning

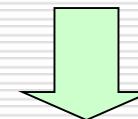
□ Settings → Estimation → Preparation

Set what to make and by when



Determination of development method

Estimation of man-hours and costs



Since it is not always possible to realize all the functions you want to create on your own within the time period and budget, you can also set the extent to which you want to create them.

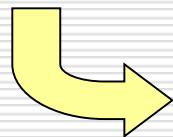
Preparation

- Staff assignment
- Budgetary measures
- Environment arrangement

(0) Development planning: Determination of development method

Basic policy

- **Develop something new?**
- **Based on an existing system?**



Calculate the scale of development
(e.g.) How much code do I need to write?

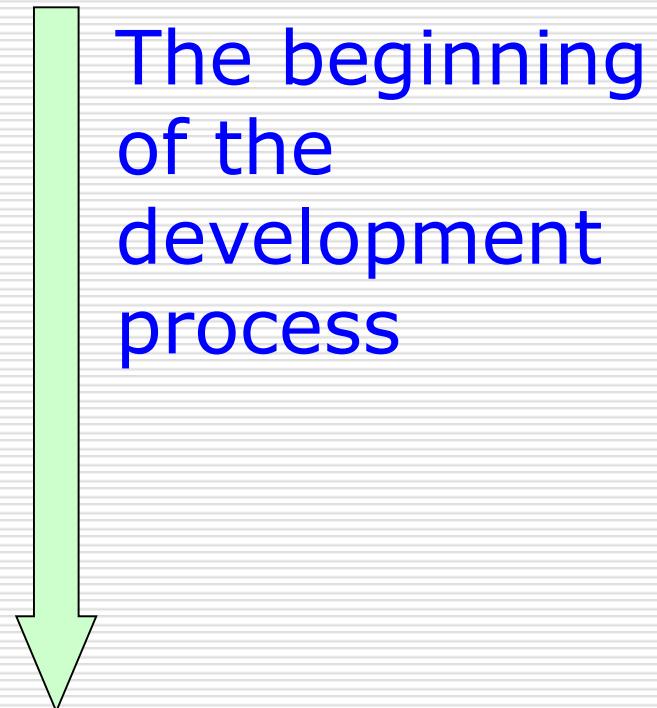


Effective in estimating man-hours and costs

Many other issues that need to be considered
Securing human resources & structure? Environment? Budget?

(1) Requirements analysis

- Development planning
- Requirements analysis
- Design
- Implementation
(Programming)
- Testing
- Operation/Maintenance



The beginning
of the
development
process

(1) Requirements analysis

Clarify what you are trying to achieve with the software

□ Business analysis

- Analyze and organize the work to be achieved of software

□ Description of request content

- Write accurately without ambiguity
- Consider feasibility: function, cost, delivery date

[Example] Bulletin notification system

(1) Requirements analysis

- I want a system that notifies me on my smartphone when a notice related to me appears.
 - Is it real time? Do you summarize it by day?
 - How would you like to be notified?
 - How do I register/change notification destinations?
 - How to prevent spoofing and protect security?
 - In what format should you enter the content of the bulletin board?
 - What are the conditions for “related postings”?
 - Is it necessary to link with the student database?

[Exercise 1]

Requests from customers

“I want you to create a system that allows you to check the congestion status of the cafeteria with your smartphone.”

Consider and list the items that should be confirmed with the customer when creating the requirement specifications.

(2) Design

- Development planning
- Requirements analysis
- Design
- Implementation (Programming)
- Testing
- Operation/Maintenance

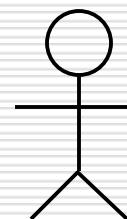
(2-1) External design

Divide target system → **subsystems**

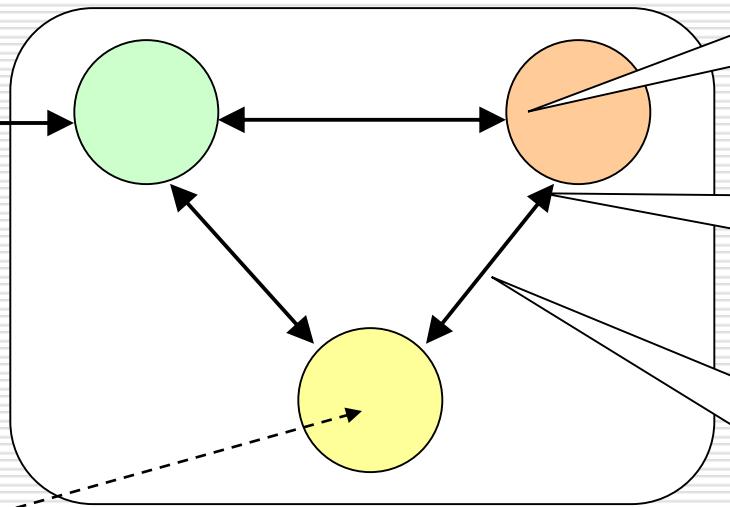
Design **external specifications** for each subsystem

Users & external systems

Whole system



How does it operate?



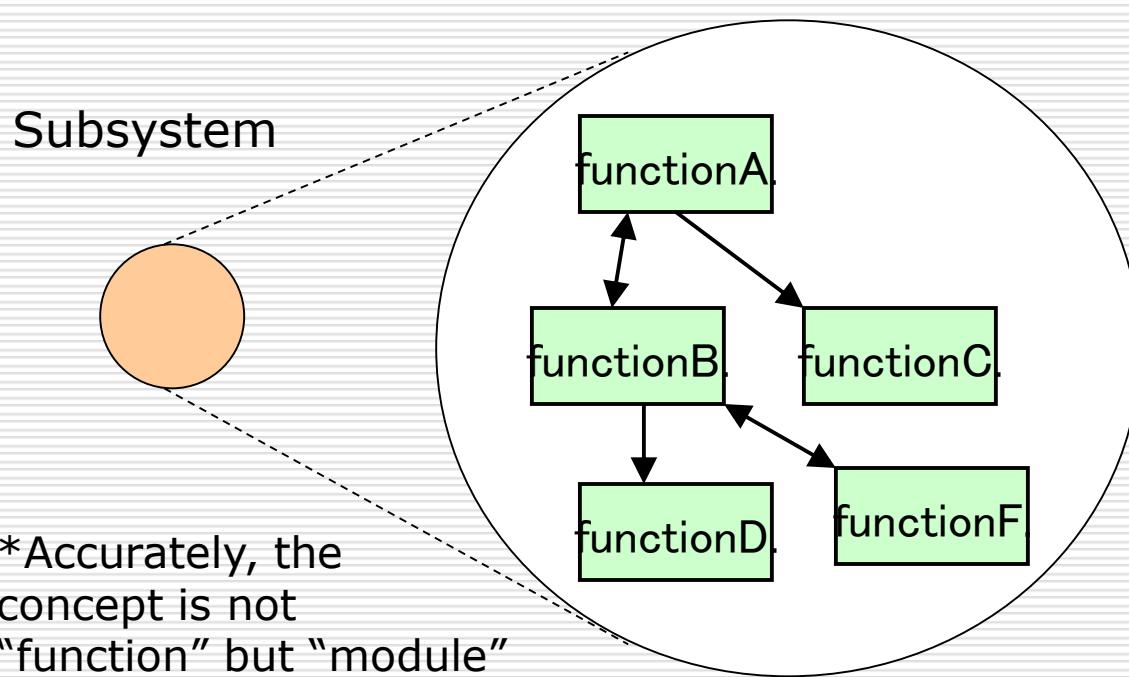
What kind of function should it have?

What kind of interface should I use?

How will data and control be exchanged?

(2-2) Internal design

About the contents of each external design
Design detailed **Program specifications**

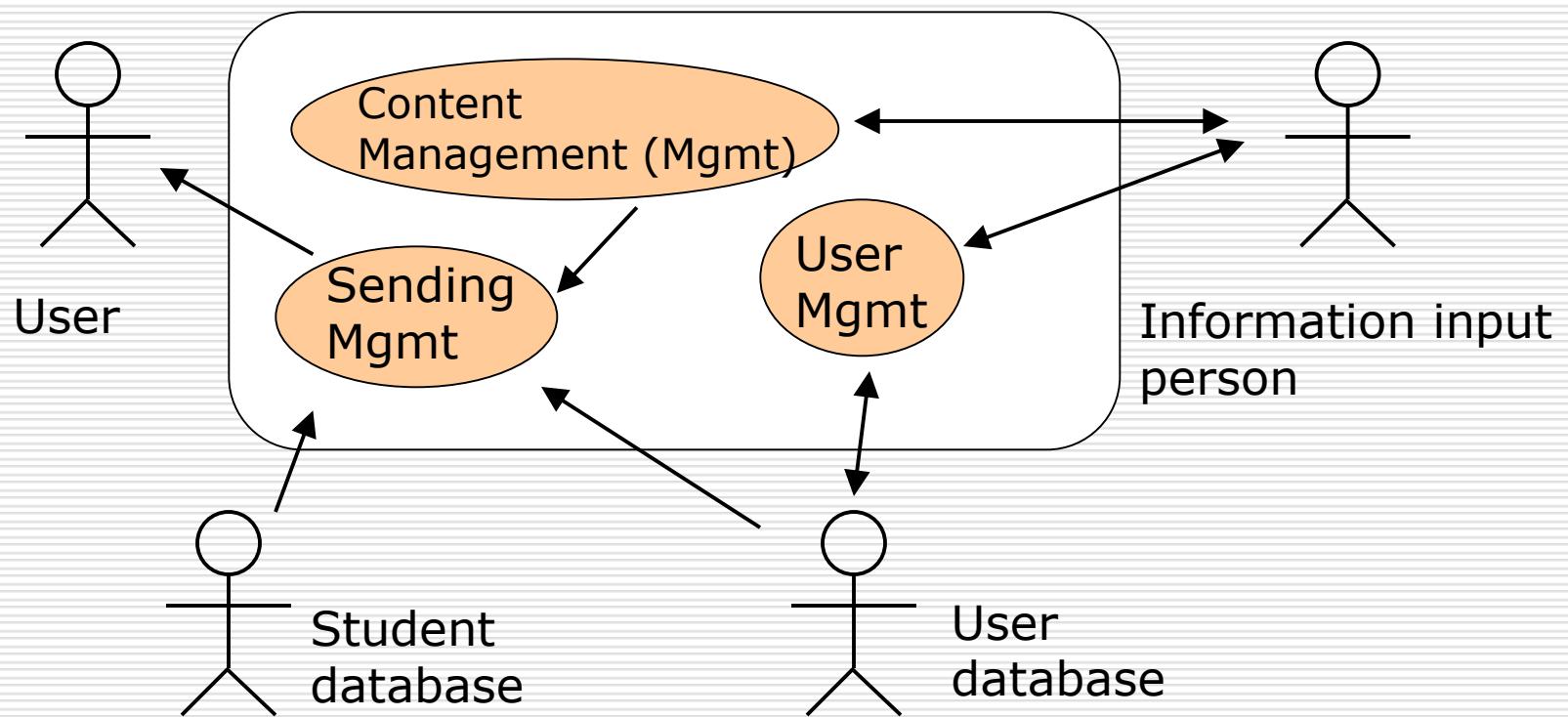


- Program structure (allocation of work as a function)
- Specifications of each function (input / output, functions)
- Data structures and algorithms

Design up to the first step of programming

[Example] Bulletin notification system

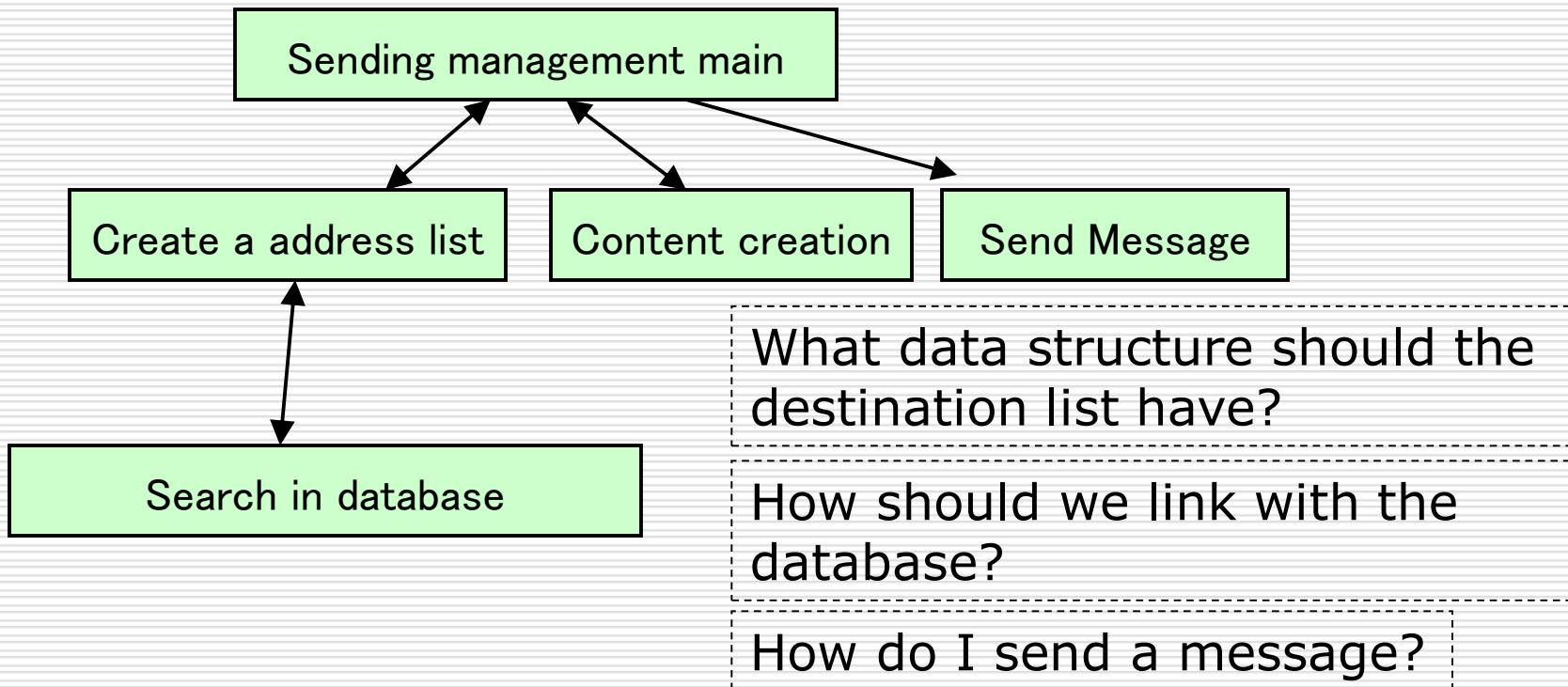
(2-1) External design



[Example] Bulletin notification system

(2-2) Internal design

□ Sending management subsystem



(3) Implementation

- Development planning
- Requirements analysis
- Design
- Implementation (Programming)
- Testing
- Operation/Maintenance

(3) Implementation

Coding based on internal specifications

- Using a programming language
- Create **Program specifications, flowcharts, etc.** as related documents

[Example] Bulletin notification system

(3) Implementation

- Use the appropriate libraries and programming languages to achieve the desired functionality
 - Take advantage of OS-specific features → C
 - Databases Integration → Java or Python
- Proactively create documents such as program specifications and flowcharts with consideration for later maintenance.

(4) Testing

- Development planning
- Requirement analysis
- Design
- Implementation (Programming)
- Testing
- Operation/Maintenance

(4) Testing

Artifact

Test specification
Test report

Based on various specifications
Test if the software works properly

- Functions (modules) according to the program specifications?
→ Unit test
- Subsystem meet Internal specifications?
→ Integration test
- System meet External specifications?
→ System test
- System meet Requirement specifications?
→ Acceptance test

(4) Testing

Unit test

Does it work correctly at the level of one part (function, etc.)?

Integration test

Does it work correctly when several parts are connected (a function calls another function, etc.)?

System test

Does the system work as designed and according to specifications?

Acceptance test

Does the system work in a way that satisfies the customer?

(5) Operation and maintenance

- Development planning
- Requirement analysis
- Design
- Implementation (Programming)
- Testing
- Operation/Maintenance

(5) Operation and maintenance

User Operation & Maintenance according to discovery of defects and correction requests

- Prior to operation, create **operation manual**
- If a failure occurs, **record and retain the report and response.**

(Supplementary) Use of terminology

□ Malfunction, failure

Phenomenons where the software does not work properly

□ Defect, fault

Direct cause of failure

Bug is an ambiguous term that includes these three things.

□ Error

The mistake that created the defect

[Example] Bulletin notification system (5) Operation and maintenance

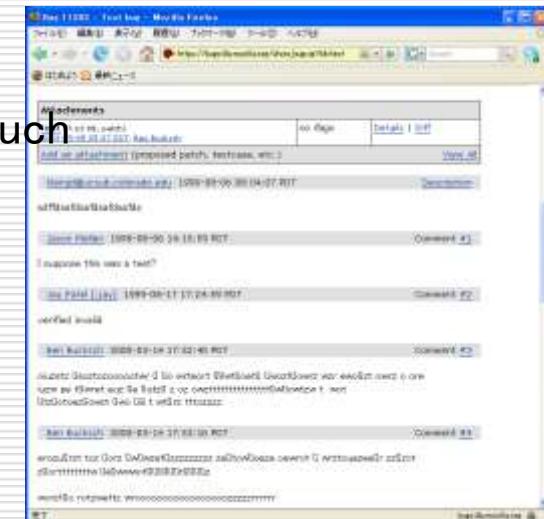
□ Operation (Have them actually use it)

- Report any problems
- If there is a request, ask for it

□ Maintenance

- Bug fixes
- Responding to requests

Bugzilla or JIRA
(There are also bug management systems such as)



[Exercise 2]

Answer the relevant process

- Answer which process the following work corresponds to.

"Cafeteria congestion check system"

① Congestion data will be structured data of measurement time (d,h,m), seat occupancy rate (%), and will be recorded in a file at 3-minute intervals.

③ System configuration
(1) Congestion degree data management unit
(2) User interface section
(3) Controll (Controller) section

④ Conducted a user survey & identified issues that need improvement

② Before going live, we checked the behavior by allowing 1,000 people to access at the same time and by allowing them to access outside of business hours.

[Exercise 2]

Answer the relevant process

① Congestion data will be structured data of measurement time (d,h,m), seat occupancy rate (%), and will be recorded in a file at 3-minute intervals.

[Internal design]

③ System configuration

- (1) Congestion degree data management unit
- (2) User interface section
- (3) Controll (Controller) section

[External design]

④ Conducted a user survey & identified issues that need improvement

[Operation/Maintenance]

② Before going live, we checked the behavior by allowing 1,000 people to access at the same time and by allowing them to access outside of business hours.

[Test]

Process models

- Software development process
(Basically, the following 5 steps)
 - Requirement analysis
 - Design
 - Implementation
 - Testing
 - Operation/Maintenance

A model that expresses **how to proceed** is called a **process model**

Typical examples of process models

- Waterfall model

(One variation is the V model)

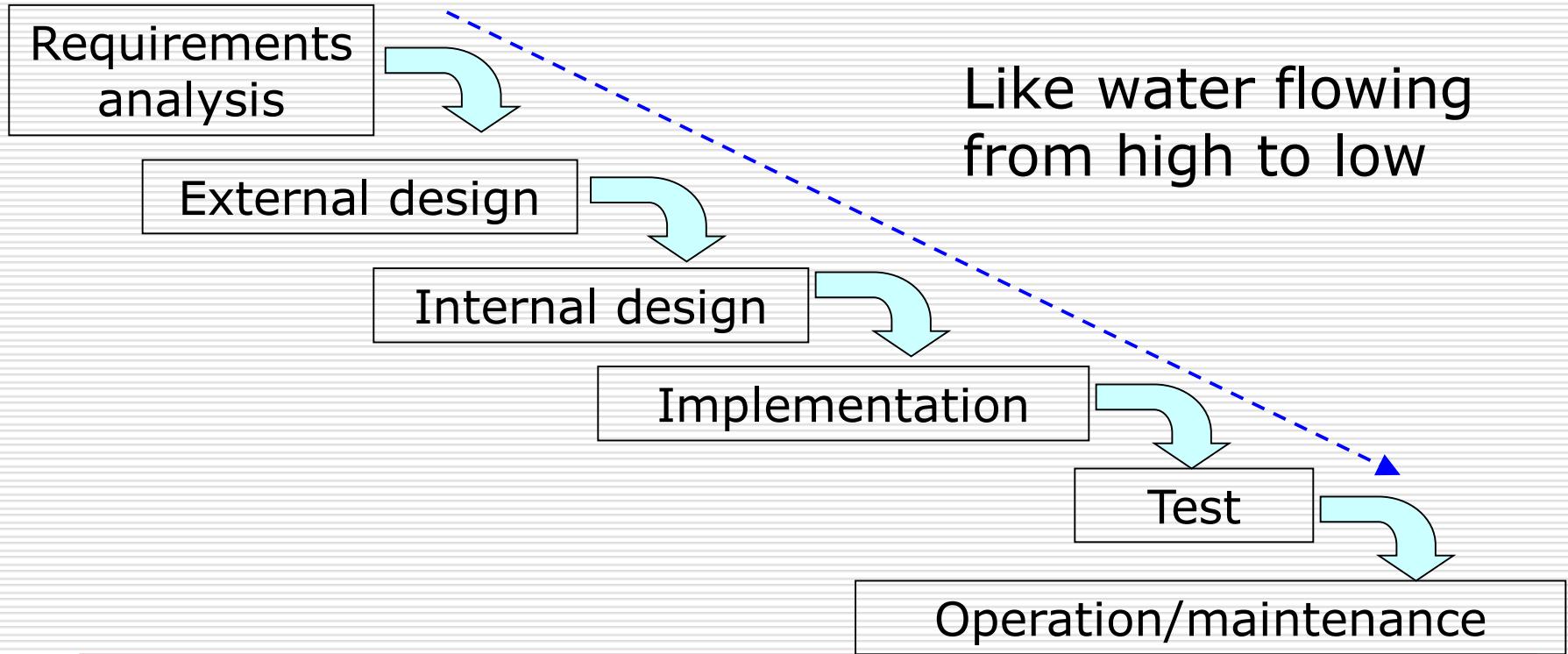
- Prototyping

(This itself is more of a development method than a process model)

- Spiral model

Waterfall Model

- Proceed step by step from requirements analysis to operation and maintenance



Waterfall Model: Features

- Once the "requirements analysis" is completed, proceed to the next "external design", and so on, and **perform each step in order**.
 - Easy to understand project progress status
 - The division of work is clear.
 - Traditional “standard” model and easy to teach (*First document published in 1970)

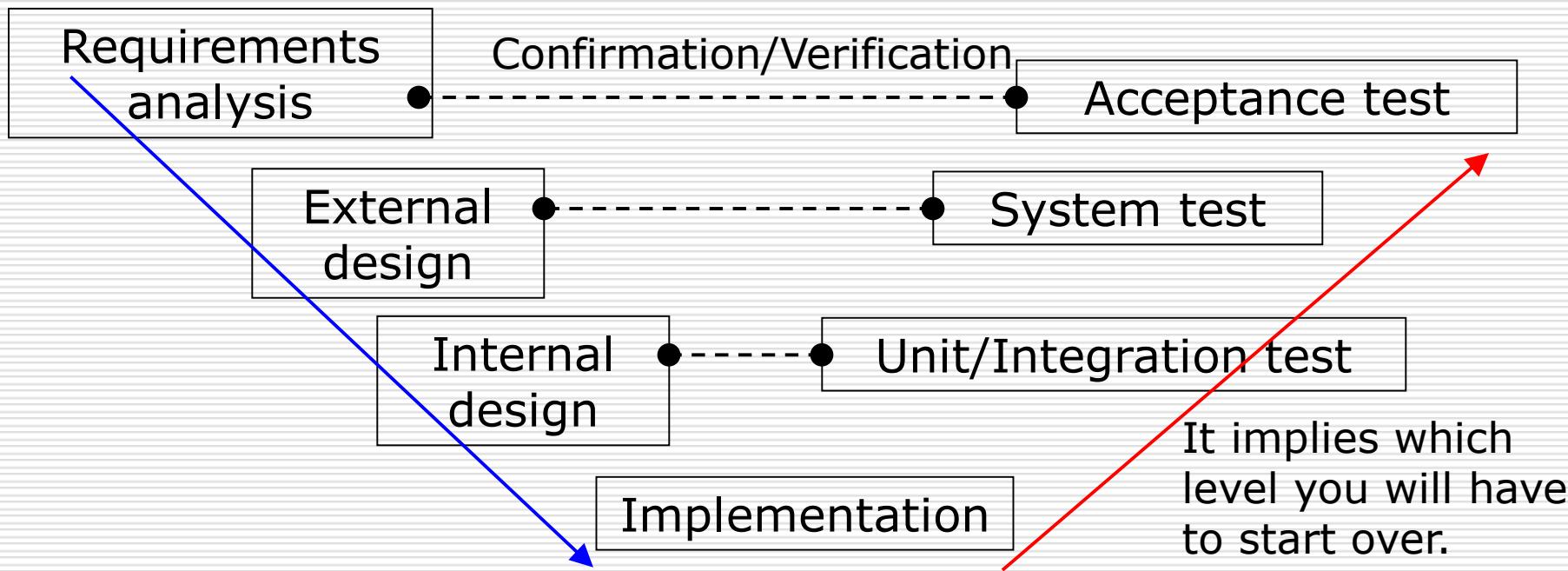
Even today, it is often employed in large-scale projects that require many people.

Waterfall Model: Problems

- 1. Requirements analysis is not always "perfectly" completed
 - Design, etc. based on incomplete Requirements Analysis → Processes are overturned and confused
- 2. Delays in upstream processes directly affect downstream
 - Failure to meet delivery deadlines or affecting engineer schedule adjustments
- 3. Not suitable for retrospective correction/redo
 - Go back upstream and start over → Increase costs
- 4. Tests will not be implemented until late
 - Testing is conducted only after "implementation" is complete, Requires major rework if defects are discovered

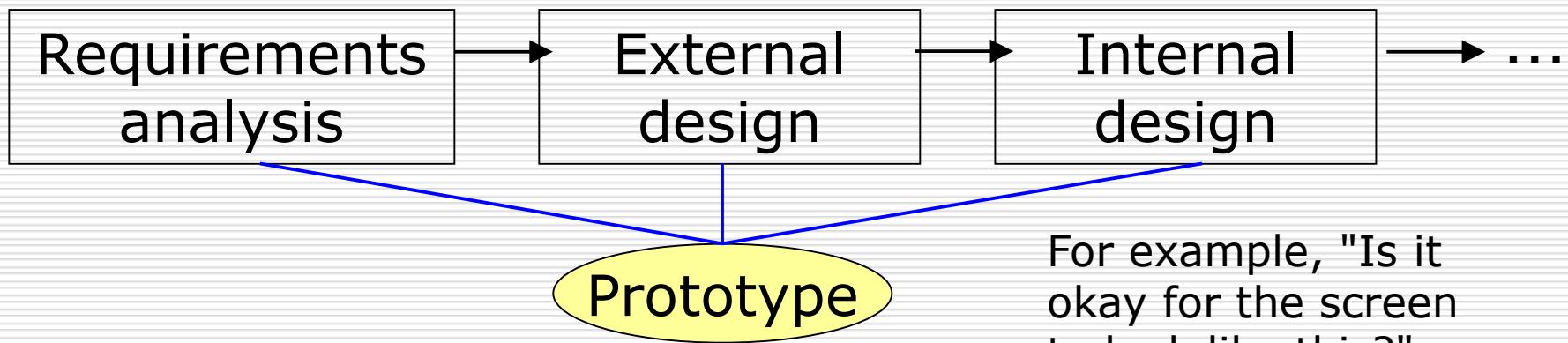
V Model

- The essence is the same as the waterfall model
- Detailed testing process and correspondence with upstream process



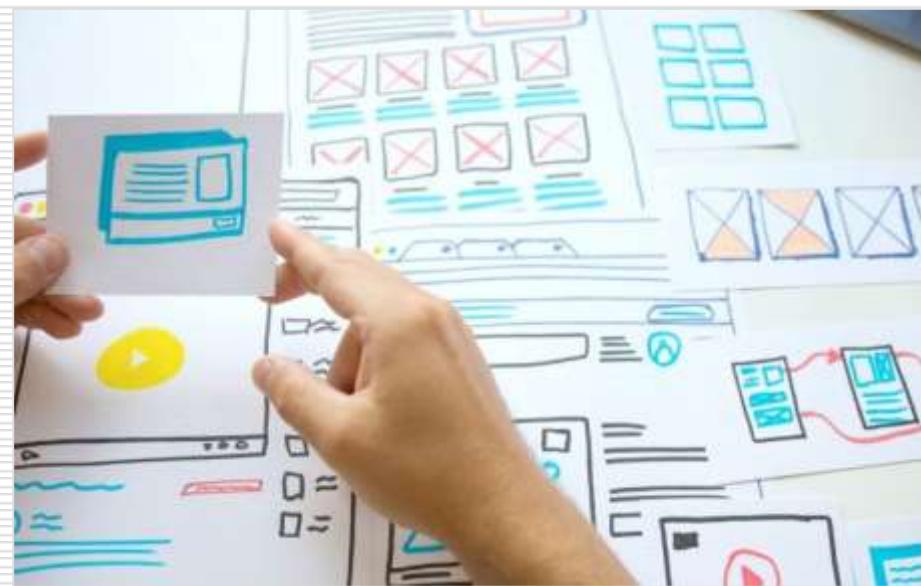
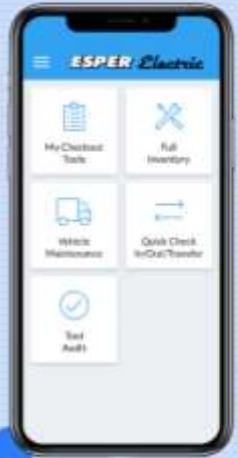
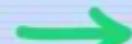
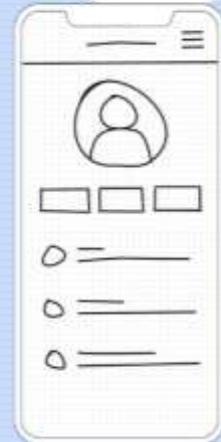
Prototyping

- A method of proceeding with the development process **while checking by creating a prototype**
- May be a mockup for confirmation
- Used to confirm and evaluate the specifications and designs



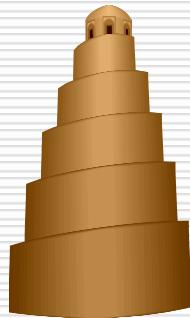
Prototyping (Example)

Prototyping



Spiral Model

- Include waterfall and prototyping



Develops by repeating the same process round and round

- Risk-driven management process

Suitable for large-scale projects

Basic cycle of spiral model

① Goal setting

- Set what you want to create (Goals)
 - Consider (AMAP) methods that match your goals
-

④ Next plan

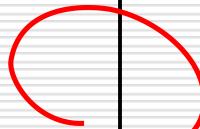
- Plan for the steps to be performed in the next iteration

② Evaluation

- Evaluate the merits and risks of each method
- Confirmation with prototype

③ Creation/execution

- Create documents and systems
- Run test



Intuitive image of the spiral model

- The main trend is the waterfall model, but it is not a matter of suddenly creating a finished product.
- Reduce the risk of failure by checking and assessing risks at each step.

Summary

Artifact

~ Software
Documentation

- Development planning & Requirements analysis are important first
- Then, external design (subsystem), internal design (module/function), implementation, testing, operation/maintenance
- Development process model
 - Waterfall: "From top to bottom"
 - Prototyping: "Based on the prototype"
 - Spiral: "Repeat Evaluation and confirmation"

Homework

Answer “[3] quiz”
by tomorrow 13:00pm

(Notes: Your quiz score will be a part of your final project evaluation)

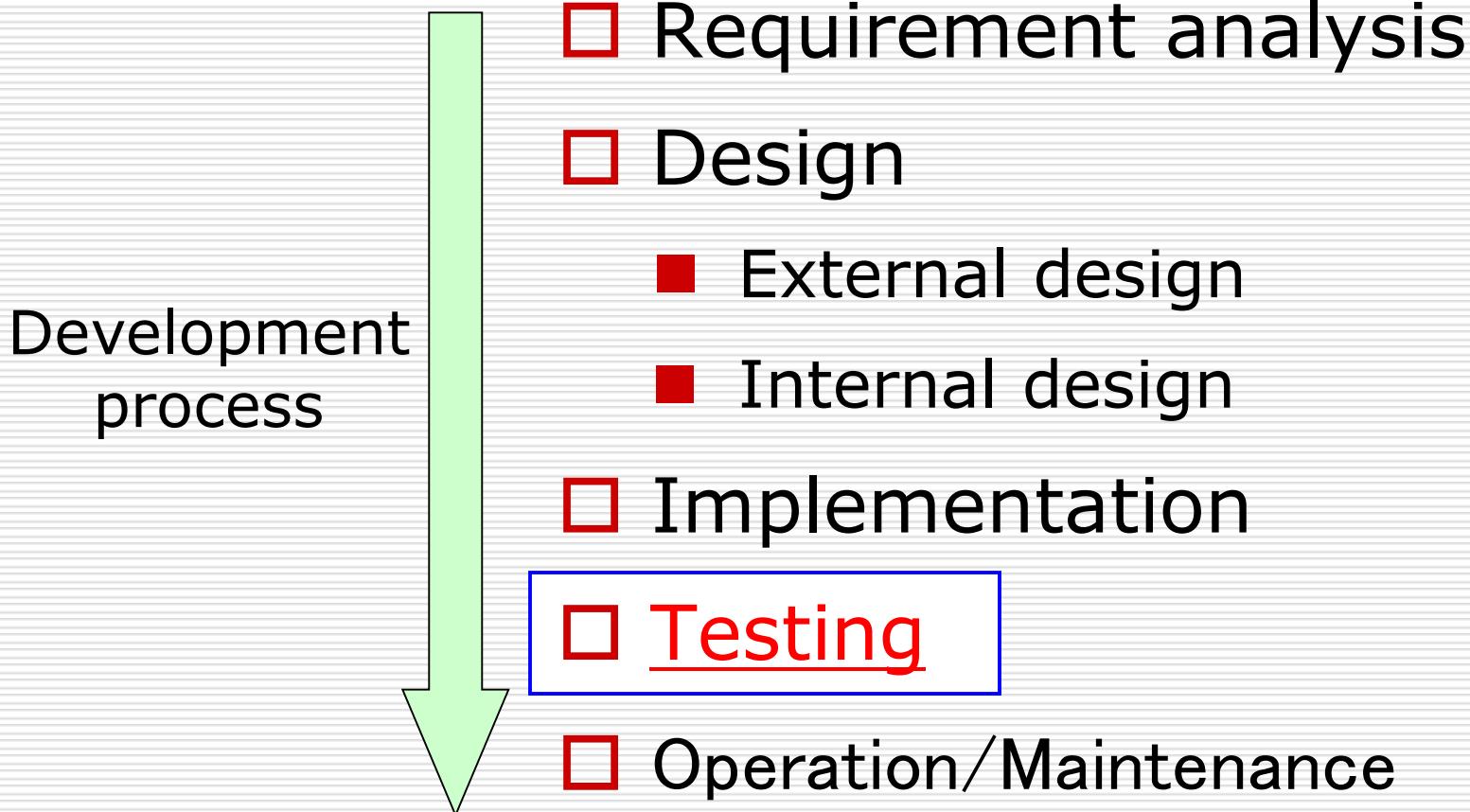
ソフトウェアテスト

[4] ブラックボックステスト

Software Testing
[4] Black Box Testing Techniques

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Testing phase (For waterfall model)



Purpose and content of testing

□ Purpose

Find errors (mistakes, bugs) that may exist in the software



□ Content

Run the software and check whether it works correctly according to specifications

Confirm AMAP that failures will not occur even in situations that are not in the specifications

Difficulty of testing

- Originally, designers and developers tried to make something according to specifications



- Check compliance by following specifications
→ It is difficult to be comprehensive
- If it is outside the specifications
→ Difficult to test as they may not be expected or may go unaware

The importance of testing: Cases that could have been prevented by testing

- AT&T's network was completely shut down, leaving **7.5 million telephone calls out of service** (1990)

Cause: Only one line of the program was added during update.

- Ariane 5 rocket **explodes in mid-air** (1996)

Cause: **64 bits numbers were handled as 16 bits**

*Development cost over 8 billion dollars (nearly 1 trillion yen)

- Mars explorer **crashes on Mars** (1999)

Cause: Regarding the thrust of the engine injection for landing, the analysis side and the operation side used **different units: pounds and newtons**

Terms:

Test case, Test suite, Test domain

Test case

Certain input data/conditions & expected output

Test suite

A collection of test cases that runs a series of tests

Testing Domain

A complete set of test cases that can guarantee that there are no errors in the program

Test example (1): Length conversion

“Kilometers → Miles” Converter

*1 kilometer
= 0.621371 miles

[Input] A real number between 0 and less than 10000, with two decimal places

[Output] Value converted to miles, rounded down to 3 decimal places

input	expected output	input	expected output
0	0.000	9.99	6.207
9999.99	6213.703	10	6.213
1	0.621	-1	error
1.001	0.621	10000	error

Other exceptions

Test example (2): Triangle problem

Consider a triangle with three integers as input and each as the length of the side. The resulting triangle is:

- Isosceles triangle
- Equilateral triangle
- Scalene triangle

Consider a test case for a program that determines which of the following.

Example test case for triangle problem

input	expected output
(2, 5, 5)	isosceles triangle
(5, 5, 5)	equilateral triangle
(3, 4, 5)	scalene triangle
(0, 0, 0)	error *become a point
(3, 4, 7)	error *become a line segment
(2, 5, 8)	error *longest side length > sum of the lengths of other sides
(1.3, 4, 5)	error *there are real numbers
(-2, 4, 5)	error *have a negative number

***Triangle Inequality:** Length of longest side < sum of the lengths of other sides

[Exercise 1]

Test an integer sorting program

- Now, We are given a program to sorts N integers
 - $0 \leq N \leq 10000$
 - Input is from a file (containing number sequences)
- Consider a test case (here only input is required) for this program

[Exercise 1] Answer

Test an integer sorting program

Random number sequence

{5, 1, 9, 2, 3, 7, ...}

Sorted sequence

{1, 2, 3, 4, ...} {100, 99, 98, 97, ...}

If there is only one or more than N

{ 2 } { 1, 2, 3, ..., N, N+1 }

0 pieces

{ }

It is also a good idea to consider other exceptions such as file not found or failure to open.

Important things in testing

□ Record the results

- What kind of defect was found in which test case, who discovered it and when
- → Important information when making corrections later

□ Communicate properly

- Ambiguous information is useless
- Reporting problems are important, but the way you report them is also important

Simply reporting that it doesn't work is **meaningless**.

You should not proudly present to the developer that you have found a defect. There are some hardships that only the person who made it can understand, so human relationships are also important.

Testability

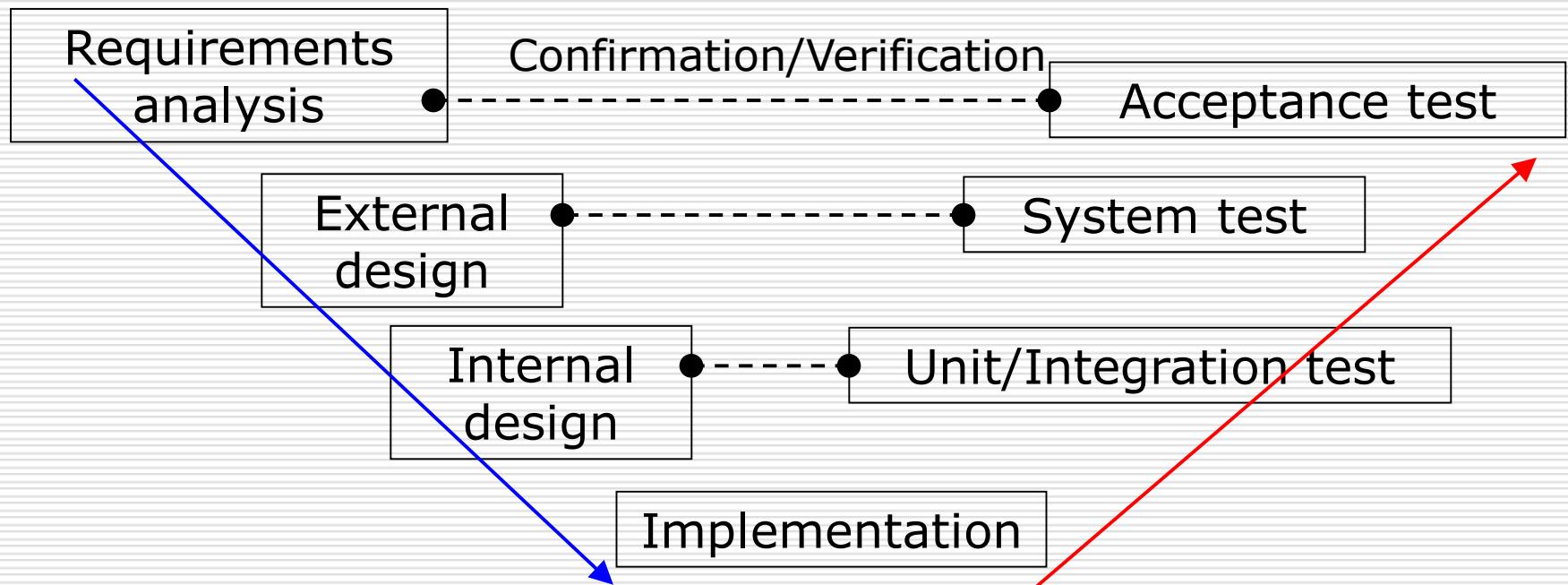
- Ease of testing
 - Reducing costs (man-hours) spent on testing
 - Shortened test period
- This leads to improved quality

Improved testability

- Design and implementation considerations are essential
 - Which function corresponds to which part of the program
 - Design for testing (Ex: Split it into functions
→ Test them independently)
- Keep testing in mind when making
 - Make it easy to understand what you're doing
 - If an error is discovered
 - Can be handled immediately

Supported by the V model

- Unit/integration test: Module operation check
- After system test: System operation check



Module

- A unit of program that can be separated
 - Group/Block of programs (**Software Parts**) → Can be replaced at that level
 - Ex: In C, this corresponds to a “**function**” or “**a collection of functions**”
 - Java, “**class**” corresponds to this (*A “method” corresponds to a “function”, but a method cannot exist by itself.)

Unit test, Integration test

Unit test



Give various inputs to **one module** and check whether appropriate output is obtained



Integration test

Combine multiple modules (unit tested) and check whether the modules can perform appropriate input/output while having a connection relationship

System test, Acceptance/Operation test

□ System test



Verify that the system **works according to specifications**



□ Acceptance test/Operation test

Similar to system test, but meaning of
test under actual operational environment and test by customer

Classification of tests

- Black box testing

Perform operational tests based on specifications without looking at the program

- White box testing

Perform operational tests based on internal structure (mainly flowcharts) of the program

- Random testing

Randomly create test cases and perform operation tests

Black Box Testing Techniques (1)

Equivalence partitioning

- To design test cases efficiently

Partition input space based on a kind of equivalence relation

- One partition is called an "equivalence class"
→ A collection of things that should be treated the same for each input condition

Two types of equivalence classes

- Valid equivalence class: Valid input
- Invalid equivalence class: Invalid input

(Reference) Equivalence relation

- A relationship between elements of a set (represented by R) that satisfies all of the following properties:
 - Reflexivity: xRx
 - Symmetry: xRy then yRx
 - Transitivity: xRy and yRz then xRz

Once one representative example is determined, a group (equivalence class) is formed of those that have an equivalence relationship with it.

Equivalence Partitioning Procedure

- (1) Set valid equivalence classes and invalid equivalence classes for each input condition
- (2) Create all **combinations of valid equivalence classes**
- (3) Create all combinations that **include only one invalid equivalence class**

*You can test by adding cases that include two or more, but first, just one. First, make sure that no problems occur by mixing invalid items.

Example of equivalence partitioning ① (1/2)

- Input: Subject grade (integer value)
 - Output: “Excellent, Good, Satisfactory, Passing, Failure, Not evaluated”

 - Specifications: 90-100 → Excellent,
80-89 → Good, 70-79 → Satisfactory,
60-69 → Passing, 0 to 59 → Failure,
-1 → Not evaluated
-

Example of equivalence partitioning ① (2/2)

□ Valid and invalid equivalence classes

valid equivalence class	invalid equivalence class
90-100	101 or more
80-89	-2 or less
70-79	
60-69	
0-59	
-1	

Select values from each equivalence class and test

Example of equivalence partitioning ② (1/4)

□ Parking fee calculator

[Input (1): Parking time (unit: minutes)]

- 1 minute – 30 minutes = 100 Yen
- 31 minutes - 60 minutes = 200 Yen
- After that = 100 Yen up per hour

[Input (2): Purchase amount at partner stores]

- 1 hour free for purchases over 2,000 Yen

*If your parking time exceeds 1 hour, you will receive a discount of 200 yen (for the first hour).

Example of equivalence partitioning ② (2/4)

input condition	valid equivalence class	invalid equivalence class
(1) Parking time	1~30	0 or less
	31~60	
	61 or more ←	
(2) Purchase amount	0~1999	-1 or less
	2000 or more	

They are grouped together because the calculation method is the same.

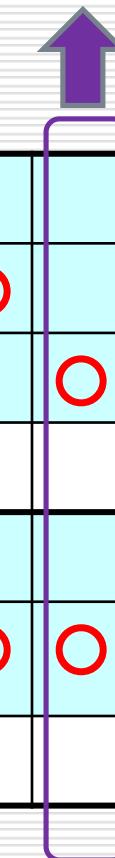
*Negative values for time and money are not possible, but here we assume that they can be entered.

Example of equivalence partitioning ② (3/4)

Test case ex: 70' parking,
& 2100 Yen shopping
⇒ Parking Fee: 100 yen

Create combinations of valid equivalence classes

(1) Parking time	valid	1~30	○		○			
		31~60		○		○		
		61~			○			○
	invalid	~0						
(2) Purchase amount	valid	0~1999	○	○	○			
		2000~				○	○	○
		negative						



Test with one vertical column

Example of equivalence partitioning ② (4/4)

Test case ex: 0' parking, &
2100 Yen shopping
→ Invalid input → Show error

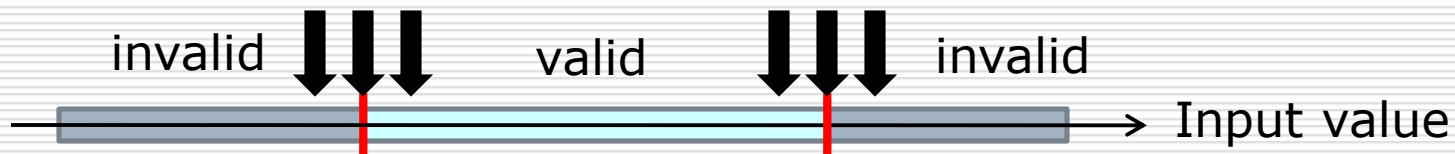
Create combinations of **only one** invalid equivalence class

(1) Parking time	valid	1~30	○					
		31~60		○				
		61~			○			
	invalid	~0				○	○	
(2) Purchase amount	valid	0~1999				○		
		2000~					○	
	invalid	negative	○	○	○			

Black Box Testing Techniques (2)

Boundary-value analysis

- Focus on the **boundaries of program conditions**
 - Test at the **change of** input and output conditions
 - Test at the **edges or boundaries** of valid and invalid equivalence classes, or **outside of them**



Why focus on changes in conditions?

- Surprisingly many mistakes are made in the program's condition checking part

Example

```
if ( score > 90 ){  
    printf("Excellent\n");  
}
```

- ✓ 90 → Is not "excellent"
- ✓ 101 → Become "excellent" instead of "error"



Correctly

```
if ( score >= 90 && score <= 100){  
    printf("Excellent\n");  
}
```

Boundary value analysis example

- Testing a program that **outputs decimals** (0 to less than 16) **in binary**

Change of conditions (Note the number of output bits)	Equivalence class edges and outside
(input) 0,1	1 bit
2,3	2 bits
4,7	3 bits
8,15	4 bits
0	valid min value
15	valid max value
-1	invalid
16	invalid

It is also good to have other **non-integer inputs as invalid equivalence classes.**

[Exercise 2]

Design test cases using boundary value analysis technique

□ Test object

Input: Two integers (0 to less than 10)

Output: Sum of them in hexadecimal

- Output is 1 digit for 0-15, 2 digits for 16-18
- Invalid input is displayed as ERROR

[Exercise 2] Answer

Change of conditions		Equivalence class edges & outside	
6+9	1 digit	0+0	valid min value
8+7		9+9	valid max value
7+9	2 digits	-1+0	invalid
8+8		10+0	invalid
9+7		10+10	invalid
5+5	Symbol	-1+10	invalid

It is also good to have other non-integer inputs as invalid equivalence classes.

*In the case of 1 digit, we consider not only 6+9 but also 8+7 because both patterns are considered based on the magnitude relationship between the two numbers. The same goes for 2 digits.

(Reference: Other methods)

Using orthogonal array

- When there are several functions to test, it is also important to cover **all their combinations**
- Select 2 functions from $n \rightarrow n(n-1)/2$ **combinations** → Enormous number of combinations

Ex: $n=10 \rightarrow 45$ ways, $n=30 \rightarrow 435$ ways

Orthogonal arrays allow you to create combinations efficiently.

For example, Character decoration in a Word processor
A = bold, B = italic, C = underline, D = shaded

Orthogonal array

- 2 functions
(A, B)

A	B
0	0
0	1
1	0
1	1

- 4 functions
(A,B,C,D)

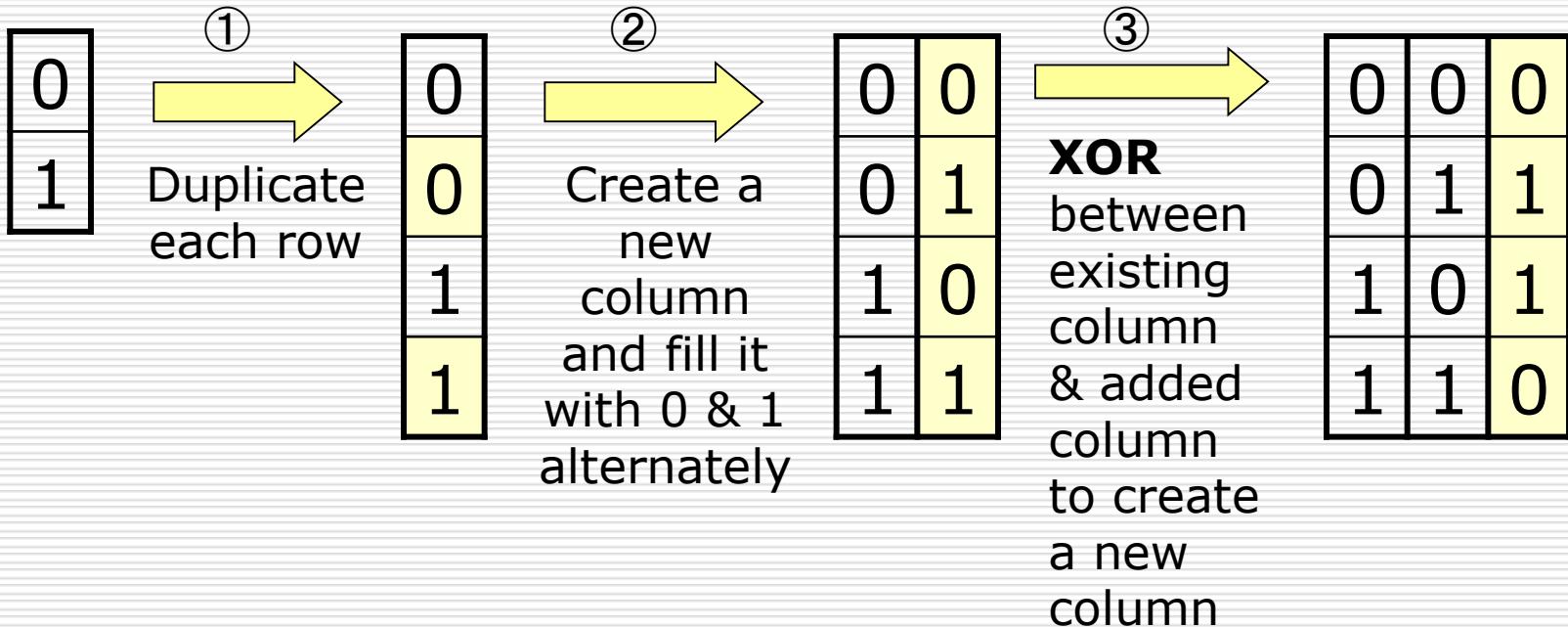
- The 8 options
on the right
are OK
- (0,0), (0,1), (1,0),
(1,1) appear the
same number of
times in any two
columns

A	B	C	D
0	0	0	0
0	0	0	1
0	1	1	0
0	1	1	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1

This type of table is called an **orthogonal array**,
and it is effective in combinatorial tests.

How to make an orthogonal array

□ 3 basic steps



How to make an orthogonal array (continued)

□ Second round

0	0	0
0	1	1
1	0	1
1	1	0

① Duplicate

0	0	0
0	0	0
0	1	1
0	1	1
1	0	1

② Add 0,1

0	0	0	0
0	0	0	1
0	1	1	0
0	1	1	1
1	0	1	0
1	0	1	1
1	0	1	0
1	1	0	0
1	1	0	1

③ XOR

0	0	0	0	0
0	0	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	1	0	1
1	0	1	1	0
1	0	1	0	1
1	1	0	0	1
1	1	0	1	0

[Exercise 3] Add two more columns to the orthogonal array

In the
orthogonal
array above

Add 2 more
columns?

0	0	0	0	0		
0	0	0	1	1		
0	1	1	0	0		
0	1	1	1	1		
1	0	1	0	1		
1	0	1	1	0		
1	1	0	0	1		
1	1	0	1	0		
0	0	0	0	0		
0	0	0	1	1		
0	1	1	0	0		
0	1	1	1	1		
1	0	1	0	1		
1	0	1	1	0		
1	0	1	1	0		
1	1	0	0	1		
1	1	0	1	0		

[Exercise 3] Add two more columns to the orthogonal array – Answer

0	0	0	0	0	0	
0	0	0	1	1	1	
0	1	1	0	0	1	
0	1	1	1	1	0	
1	0	1	0	1	0	
1	0	1	1	0	1	
1	1	0	0	1	1	
1	1	0	1	0	0	

0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	1	1	0	0	0	1
0	1	1	1	1	1	0
1	0	1	0	1	0	1
1	0	1	1	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	1	0
1	1	0	1	0	0	1

Effectiveness of combination of two functions

Relationship between number of combined functions and faults:
Bug detection rate (%)

Num. of combinations	Expert system			OS	Embedded medical device	Mozilla	Apache	DB
1	61	72	48	39	82	66	29	42
2	36	10	6	8	*	15	47	38
3	*	*	*	*	*	2	19	19
4	*	*	*	*	*	1	2	7
5	*	*	*	*	*	2	0	
6	*	*	*	*	*	1	4	

D.R. Kuhn, et al., "Software fault interactions and implications for software testing,"
IEEE Trans. Softw. Eng., vol.30, no.6, pp.418–421, June 2004.

Orthogonal array testing formula

□ OAT Formula

- Runs (N) – Number of rows → Number of test cases that will be generated.
- Factors (K) – Number of columns → Maximum number of variables that can be handled.
- Levels (V) – Maximum number of values that can be taken on any single factor.

$L_{\text{Runs}}(\text{Levels}^{\text{Factors}})$

Table of Taguchi Designs (Orthogonal Arrays)

- Table of Taguchi Designs
- https://www.york.ac.uk/depts/math/tables/taguchi_table.htm
- http://support.sas.com/techsup/technote/ts723_Designs.txt

n/k	3	4	5	6	7	10	11	12	13	15	21	22	26	27	31	40	63
4	L4 2																
8						L8 2											
9		L9 3															
12							L12 2										
16			L16b 4								L16 2						
18							L18 3.6										
25					L25 5												
27								L27 3.2									
32							L32b 4.2								L32 2		
36											L36 3		L36b 3.2				
50									L50 5.2								
54												L54 3.2					
64											L64b 2				L64 4		
81															L81 3		

Orthogonal array example ① (1/2)

□ Example 1

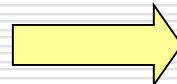
- A Web page has three distinct sections (Top, Middle, Bottom) that can be individually **shown** or **hidden** from a user
- → No of Factors = 3 (Top, Middle, Bottom)
- → No of Levels (Visibility) = 2 (Hidden or Shown)
- Search in **Taguchi Designs** → Array Type $L_4(2^3)$

L_{Runs} (Levels**^{Factors})**

Orthogonal array example ① (2/2)

□ Result of Example 1

Experiment Number	Column		
	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1



Test Cases	TOP	Middle	Bottom
Test #1	Hidden	Hidden	Hidden
Test #2	Hidden	Visible	Visible
Test #3	Visible	Hidden	Visible
Test #4	Visible	Visible	Hidden

Replace the value of each Factor **in order**

Orthogonal array example ② (1/2)

□ Example 2

■ Provide our personal information like below:

Age field	Highest Qualification
<ul style="list-style-type: none">• Less than 18• More than 18• More than 60	<ul style="list-style-type: none">• High School• Graduation• Post-Graduation
Gender field	Mother Tongue
<ul style="list-style-type: none">• Male• Female• NA	<ul style="list-style-type: none">• Hindi• English• Other

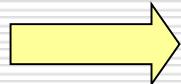
→ 4 Factors & 3 Levels

→ Array Type $L_9(3^4)$

Orthogonal array example ② (2/2)

□ Result of Example 2

Experiment Number	Column			
	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1



Run	Age	Gender	Higher Qualification	Mother Tongue
Run 1	Less than 18	Male	High School	Hindi
Run 2	Less than 18	Female	Post-Graduation	English
Run 3	Less than 18	NA	Graduation	Other
Run 4	More than 18	Male	Post-Graduation	Other
Run 5	More than 18	Female	Graduation	Hindi
Run 6	More than 18	NA	High School	English
Run 7	More than 60	Male	Graduation	English
Run 8	More than 60	Female	High School	Other
Run 9	More than 60	NA	Post-Graduation	Hindi

Orthogonal array example ② (1/2)

- Example 3
- 3 variables:
 - A – 2 values (A1, A2)
 - B – 3 values (B1, B2, B3)
 - C – 3 values (C1, C2, C3)
- 3 Factors & 3 Levels
- Array Type $L_9(3^4)$

Experiment Number	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

NOTE: $L_4(2^3)$ and $L_8(2^7)$ cannot be chosen because they have only 2 possible values (levels)

Orthogonal array example ② (2/2)

□ Result of Example 3

Test case #	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C3
4	A2	B1	C2
5	A2	B2	C3
6	A2	B3	C1
7	?	B1	C3
8	?	B2	C1
9	?	B3	C2

Test case #	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C3
4	A2	B1	C2
5	A2	B2	C3
6	A2	B3	C1
7	A1	B1	C3
8	A2	B2	C1
9	A1	B3	C2

Empty Orthogonal Array cell → Cycle values in them.

Summary

- Testing: Finding “Bugs”
 - Key to quality assurance
 - Flexible thinking is required

- Black Box Testing:

Operation confirmation seen from the outside

 - Equivalence partitioning:
Combination of valid and invalid equivalence classes
 - Boundary value analysis:
Focus on input/output conditions and boundaries
between equivalence classes

Homework

Answer “[4] quiz”
by tomorrow 13:00pm

(Notes: Your quiz score will be a part of your final project evaluation)

ソフトウェアテスト

[6] ホワイトボックステスト

Software Testing
[6] White Box Testing Techniques

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Classification of tests

□ Black box testing

Perform operational tests **based on specifications** without looking at the program contents (black box)

Explanation in the 4th session,
exercises in the 5th session

□ White box testing

Perform operational tests **based on the internal structure** of the program (**mainly flowcharts**)

□ Random testing

Create test cases **randomly** and perform operational tests

White box testing method

- How to design test cases by focusing on the **structure (contents) of the source program** rather than the specifications
 - Test not in line with required specifications
 - Dealing with structural complexity (combination of conditions, etc.) in the source program
 - Tests that supplement black box testing

White box testing method (1)

Statement coverage method

□ Execute all statements at least once

- It may not be possible with only one test case (execution path)
- With **different test cases**, and consider their collection of test cases → It is possible to cover them all
- The percentage of statements that were executed is called the **statement coverage rate**.
- This is also known as **C0**

Example to test target (1)

```
void foo(int x){  
    int sum, n;  
    sum = 0;  
    for ( n = 1; n < x; n++ ){  
        sum += n;  
    }  
    printf("%d\n", sum);  
}
```

[input]
Arguments x

[output]
Values printed by
printf

If $x > 1$, all statements can be executed
(statement coverage rate = 100%)

Example test case
Input x = 2, output sum = 3

Example to test target (2)

```
void foo(int x, int y){  
    int sum, n;  
    sum = 0;  
    for ( n = 0; n < x; n++ ){  
        sum += n;  
    }  
    if ( sum < y ){  
        printf("%d\n", sum);  
    }  
    else{  
        printf("%d\n", y);  
    }  
}
```

[input]
Arguments x, y

[output]
Values printed by
printf

Statement coverage example: x=1, y=0

Source code	Execution?
void foo(int x, int y){	---
int sum, n;	○
sum = 0;	○
for (n = 0; n < x; n++){	○
sum += n;	○
}	---
if (sum < y){	○
printf("%d\n", sum);	✗
}	---
else{	---
printf("%d\n", y);	○
}	---
}	---

Statement Coverage
(C0)

$$\frac{6}{7} \doteq 85.7\%$$

← Since $\text{sum} = 0$

[Note]

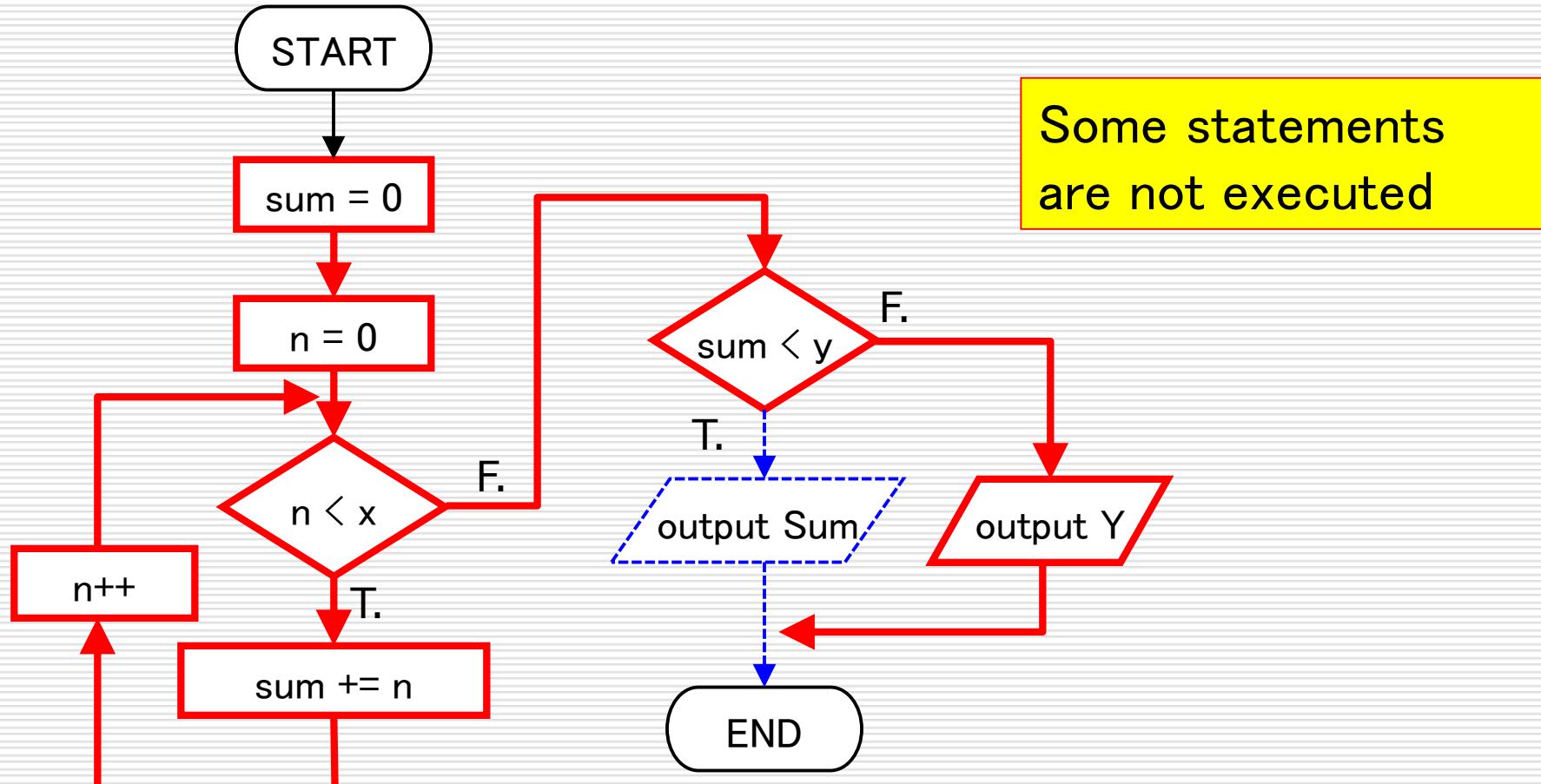
It is necessary to define in advance which statements are counted as executable.

Example of 100% statement coverage

Source code	x=1, y=0	x=1, y=1	total
void foo(int x, int y){	---	---	---
int sum, n;	○	○	○
sum = 0;	○	○	○
for (n = 0; n < x; n++){	○	○	○
sum += n;	○	○	○
}	---	---	---
if (sum < y){	○	○	○
printf("%d\n", sum);	✗	○	○
}	---	---	---
else{	---	---	---
printf("%d\n", y);	○	✗	○
}	---	---	---
}	---	---	---

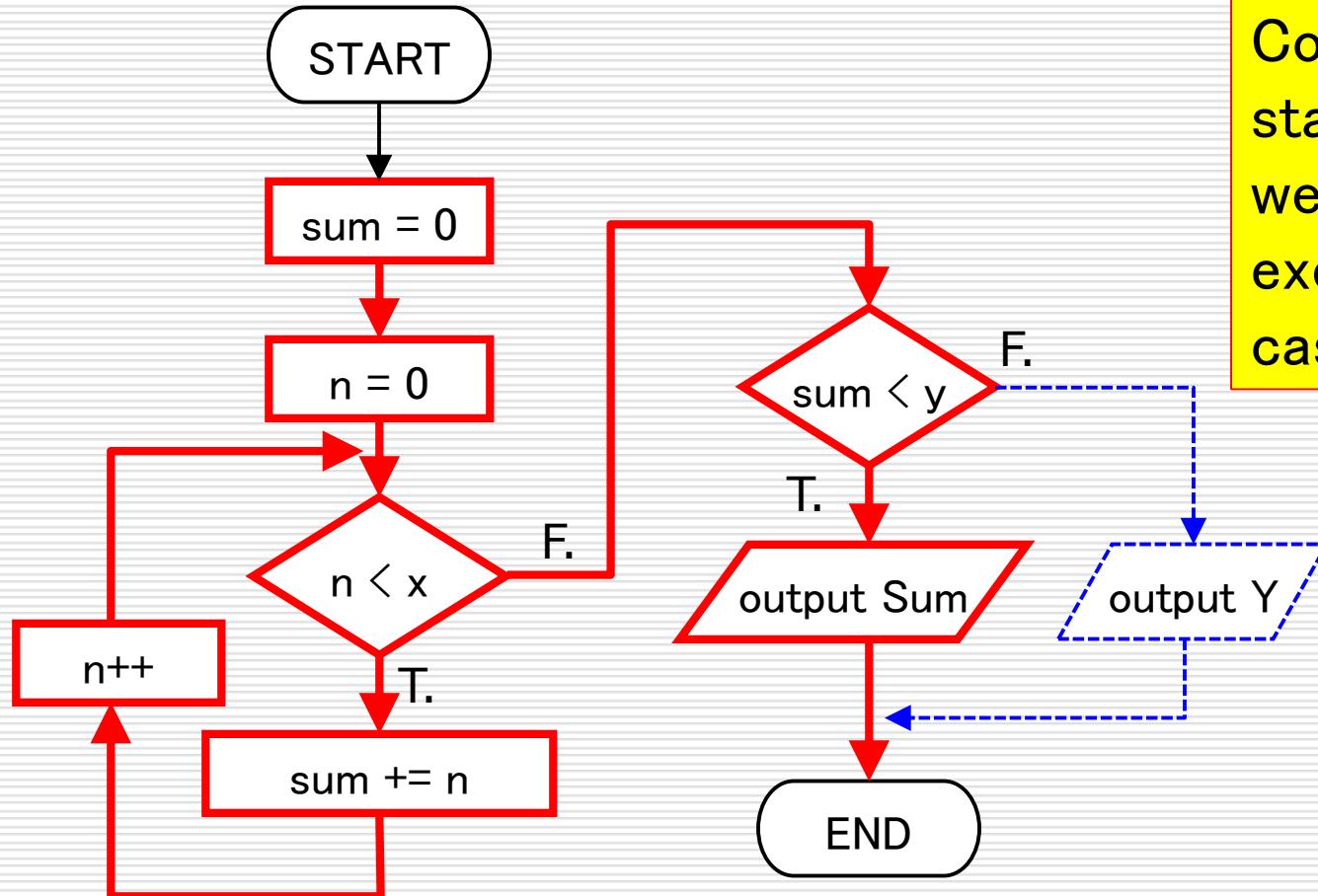
When drawing a flowchart

Test case ① $x=1, y=0$



When drawing a flowchart

Test case ② $x=1, y=1$



Covers statements that were not executed in test case ①

Automatic measurement of statement coverage rate

- Checking and counting the execution of instructions by **yourself** is not an easy task
- All you have to do is **monitor whether all commands have been executed.**
The compiler gcc has an option for this, and when used with a tool called gcov, it can be checked automatically.

Use this in next week's exercise

Important

- Next time, we will do a white box test exercise.
- Everyone should bring a PC. Also, **gcc** and **gcov** are required, so be sure to install them.

White box test method (2)

Branch coverage method

- In all **conditional branches** (if statement, while statement, for statement), **execute at least one** case when the result is **True** and once when the result is **False**.
 - It would be better if it could be covered by a **set of multiple test cases**
 - The rate of experiencing “T” and “F” for all **conditions** is called the **branch coverage rate**.
 - This is also known as **C1**

Branch coverage example: x=1, y=0

Source code	Exec?	T, F
void foo(int x, int y){	---	---
int sum, n;	O	---
sum = 0;	O	---
for (n = 0; n < x; n++){	O	T F
sum += n;	O	---
}	---	---
if (sum < y){	O	F
printf("%d\n", sum);	X	---
}	---	---
else{	---	---
printf("%d\n", y);	O	---
}	---	---
}	---	---

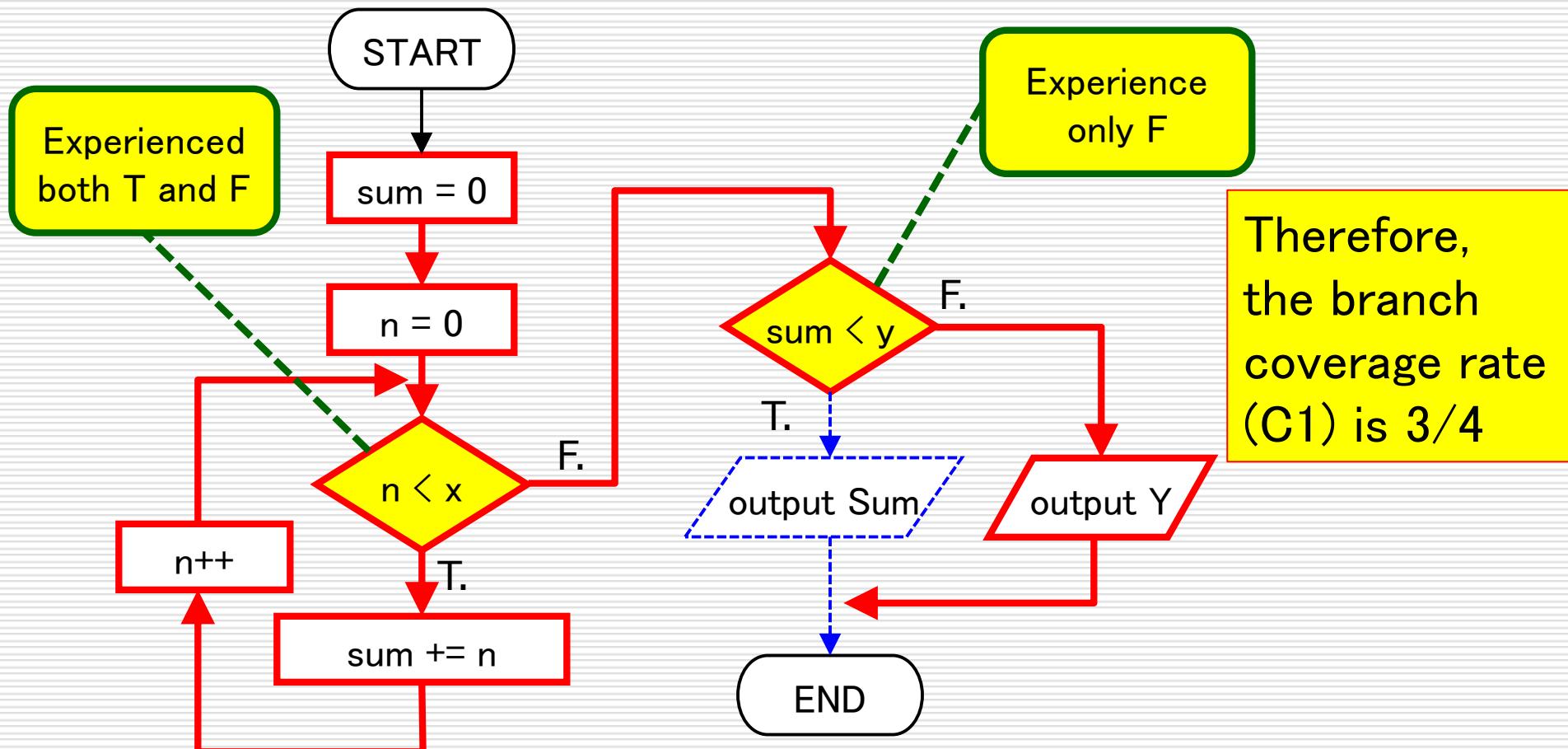
Branch
coverage rate
(C1)
 $\frac{3}{4} = 75\%$

Example of 100% branch coverage rate

Source code	x=1, y=0	x=1, y=1	total
void foo(int x, int y){	---	---	---
int sum, n;	---	---	---
sum = 0;	---	---	---
for (n = 0; n < x; n++){	T. F.	T. F.	T. F.
sum += n;	---	---	---
}	---	---	---
if (sum < y){	F.	T.	T. F.
printf("%d\n", sum);	---	---	---
}	---	---	---
else{	---	---	---
printf("%d\n", y);	---	---	---
}	---	---	---
}	---	---	---

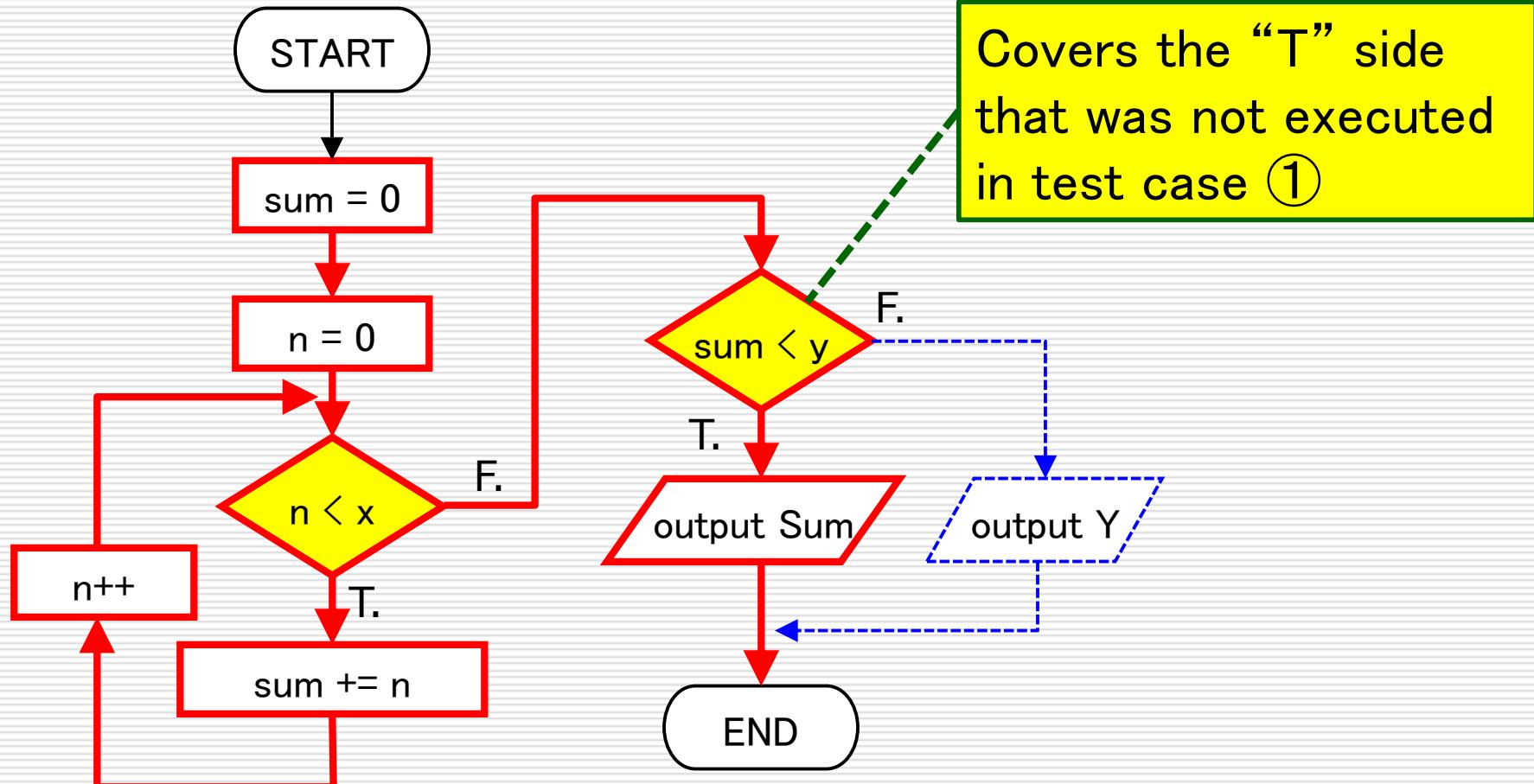
When drawing a flowchart

Test case ① $x=1, y=0$



When drawing a flowchart

Test case ② $x = 1, y = 1$



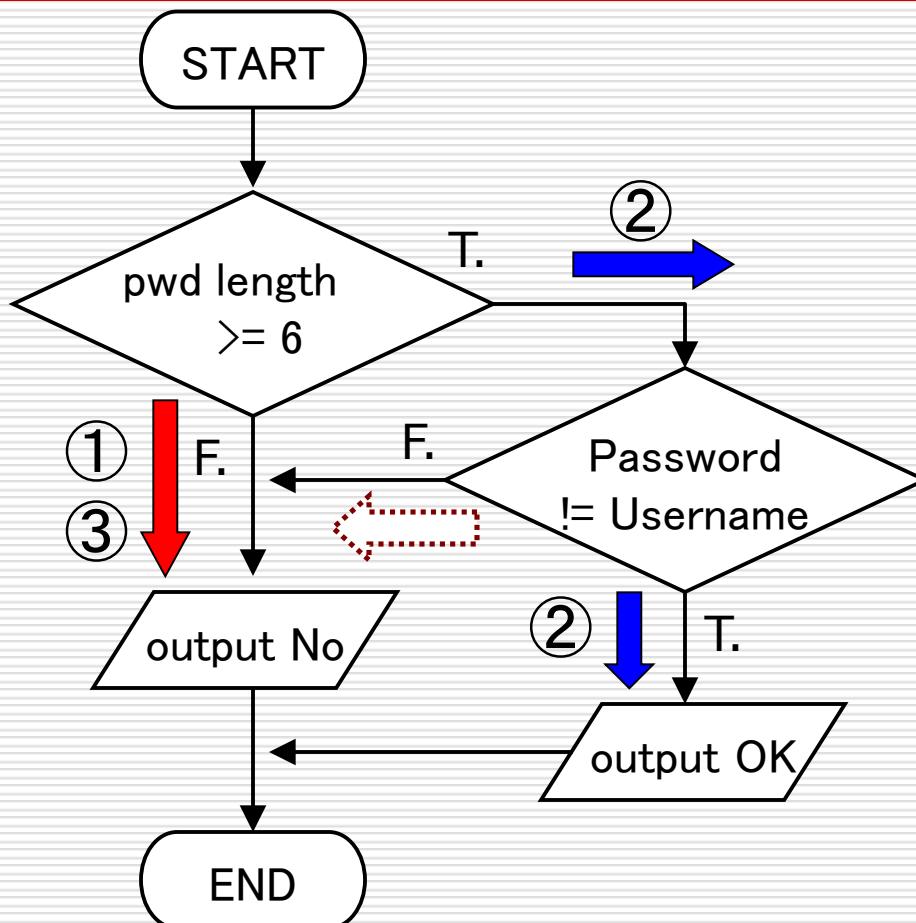
Let's consider another example

- Consider a program that performs a simple security check on passwords

[Check items]

- Is the password length at least 6 characters?
- Is the password the same as your username?

Example of 75% branch coverage rate



test cases

- (username, password)
① ("taro", "")
② ("taro", "foobar")
③ ("taro", "taro")

There are password lengths < 6 and password lengths ≥ 6 , some have different usernames and passwords, and some have the same password, but this is not exhaustive.

[Exercise 1]

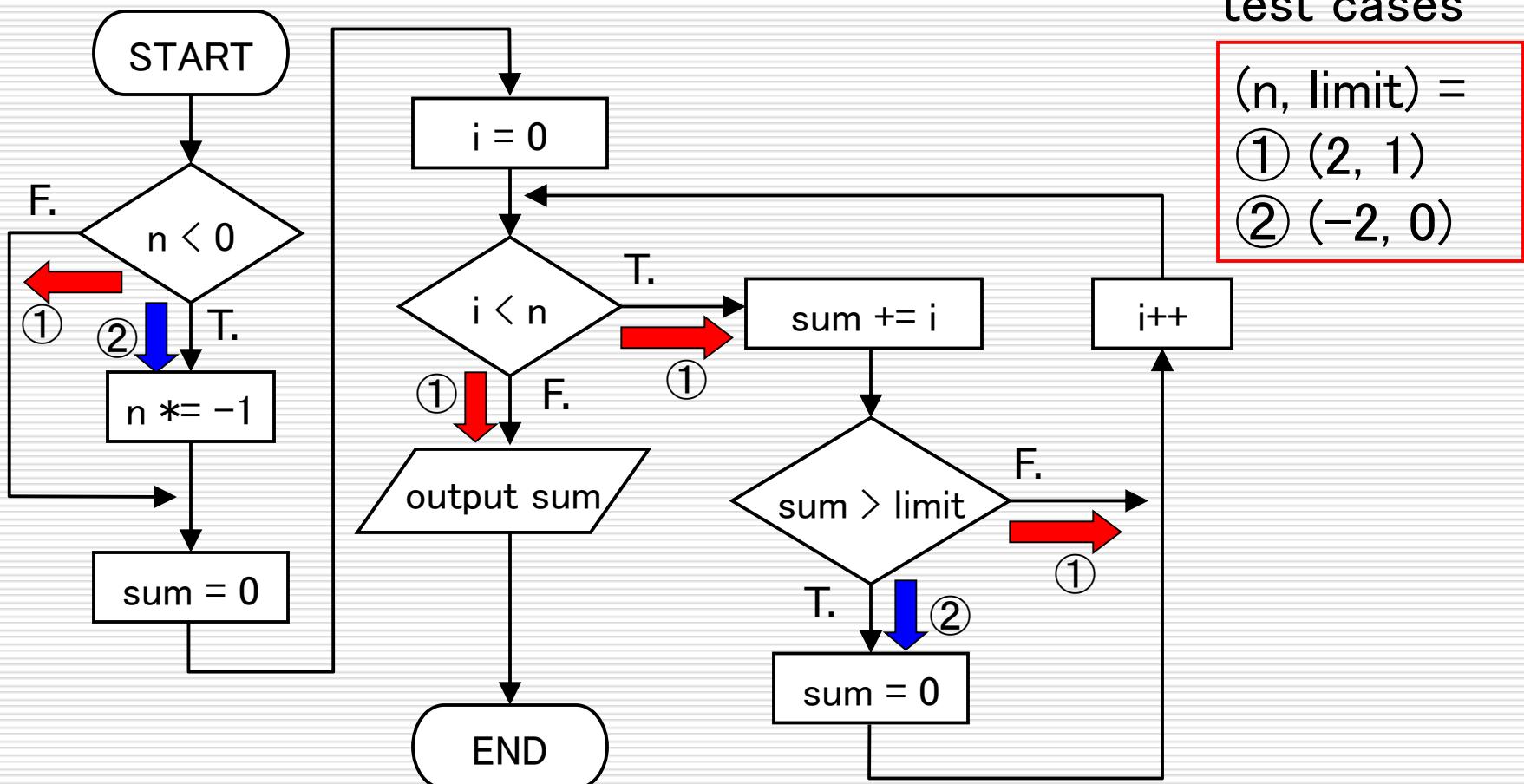
Calculate C0 and C1

(Note) “}” in the program is not counted as a statement.

- I ran two test cases for the code on the right
 - (n, limit)
= (2, 1), (-2, 0)
- In this case, calculate the C0 and C1

```
if ( n < 0 ){
    n *= -1;
}
sum = 0;
for ( i = 0; i < n; i++ ){
    sum += i;
    if ( sum > limit ){
        sum = 0;
    }
}
printf("%d\n", sum);
```

[Exercise 1] Answer: 100% for ① and ②



[Exercise 2]

Calculate branch coverage rate

- The function on the right
- Calculate the branch coverage rate (C1) when testing by calling

```
is_prime(1);
is_prime(2);
```

(Note) Please note that when you execute the return statement, the function ends at that point.

```
int is_prime(int n){
    int k;
    if ( n < 2 ){
        return 0;
    }
    for ( k = 2; k < n; k++ ){
        if ( n % k == 0 ){
            return 0;
        }
    }
    return 1;
}
```

It is a good idea to note down the conditions T and F for each of n=1 and n=2 while checking whether or not they are executed.

Notes for exercise 2

Source code	n=1		n=2		total
	実行	条件	実行	条件	
int k;		---		---	---
if (n < 2){					
return 0;		---		---	---
}		---		---	---
for (k = 2; k < n; k++){					
if (n % k == 0){					
return 0;		---		---	---
}		---		---	---
}		---		---	---
return 1;		---		---	---

[Exercise 2] Answer

$$(C1) \frac{3}{6} = 50\%$$

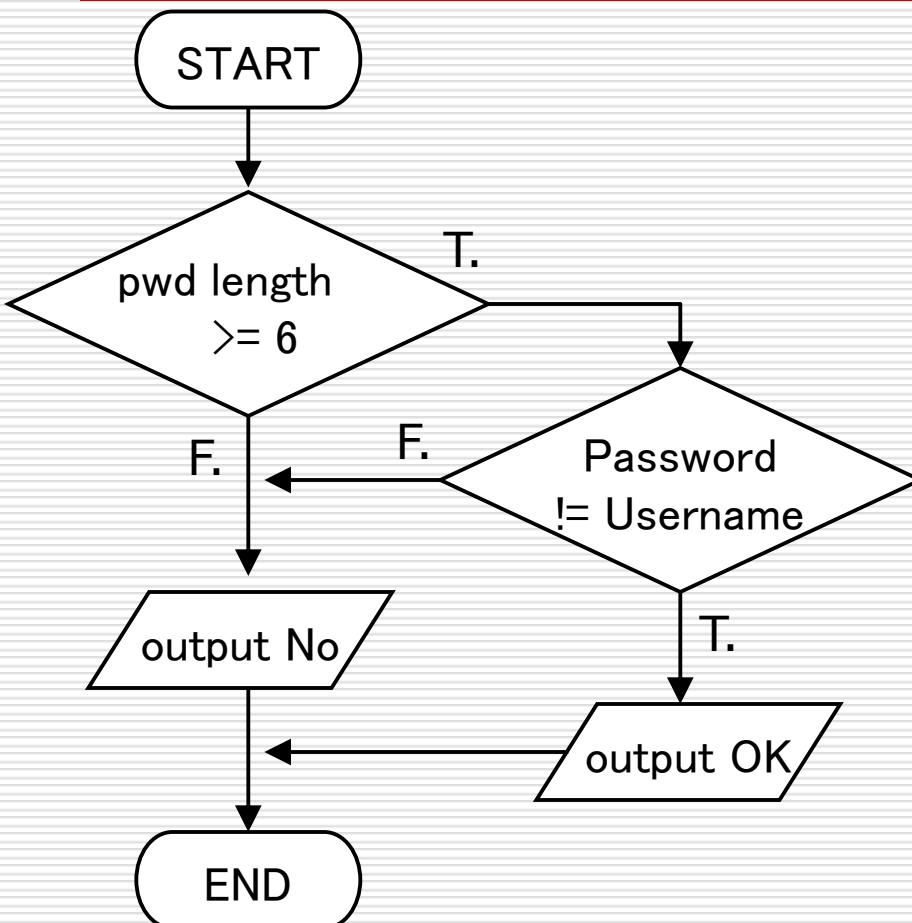
Source code	n=1 実行	条件	n=2 実行	条件	total
int k;	O	---	O	---	---
if (n < 2){	O	T	O	F	T F
return 0;	O	---	X	---	---
}		---		---	---
for (k = 2; k < n; k++){	X		O	F	F
if (n % k == 0){	X		X		
return 0;	X	---	X	---	---
}		---		---	---
}		---		---	---
return 1;	X	---	O	---	---

White box testing method (3)

Condition coverage method

- Execute all conditional branches to cover all combinations of T and F
 - Ex: If there are two conditional branches
 - Test all of
(F, F), (F, T), (T, F), (T, T)
 - In the worst case,
The complexity is $O(2^n)$
 - The rate at which these can be executed is called the **condition coverage rate**, also called **C2**.
 - Ignore infeasible combinations

Example of 100% condition coverage

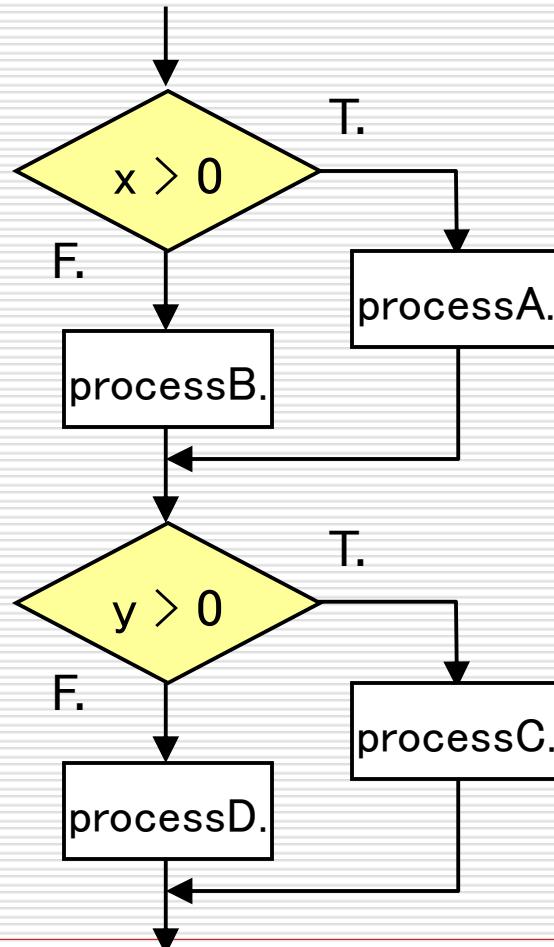


Combination		pwd length	
		< 6	≥ 6
Usr & Pwd	same	—	②
	difference	①	③

test cases

- (username, password)
- ① ("taro", "")
 - ② ("momotaro", "momotaro")
 - ③ ("taro", "foobar")

The difference between C1 and C2



test cases

① $x = 1, y = 1$ ② $x = 0, y = 0$

(2) C1=100%

(3) C2=50%

	T.	F.
x > 0	①	②
y > 0	①	②

	x > 0	y > 0
②	F.	F.
	F.	T.
	T.	F.
①	T.	T.

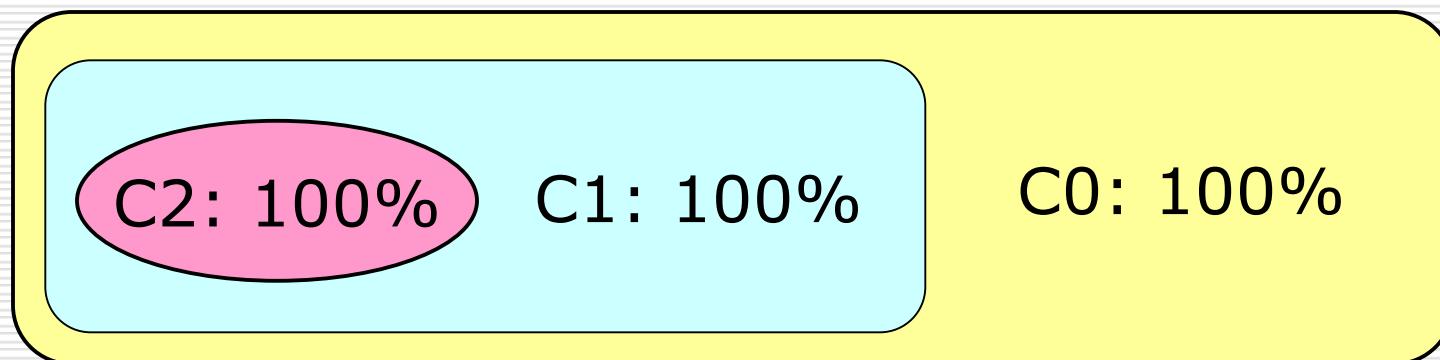
Three types of inclusion relations

□ C2: 100%

⇒ C1: 100%

□ C1: 100%

⇒ C0: 100%



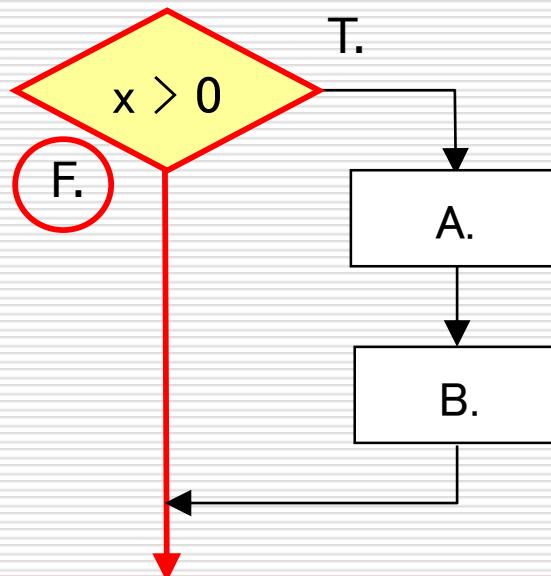
Common Misconceptions

- “C1: 100% \Rightarrow C0: 100%” means “ $C1 \leq C0$ ”, right?

(Example)

Error

This proposition only applies to the case of 100% coverage and is not guaranteed for other cases.



If the left condition is false (F.)

- ◆ $C1 = 1/2 = 50\%$
- ◆ $C0 = 1/3 = 33.3\%$

Three types of white box testing

- Statement coverage (C0) can be easily achieved
- Branch coverage (C1) is somewhat complicated, but unless the scale is large, it is possible to increase the coverage rate (however, the more conditional branches there are, the more difficult it becomes).
- The feasibility of condition coverage (C2) becomes much more difficult as the scale and complexity increases.
- For example, if we consider the state transition of a smartphone, there are as many branches as there are icons that can be tapped on each screen, so the total number of combinations increases explosively.

Relationship with black box testing

- White box testing is weaker than black box testing such as equivalence partitioning
 - In the first place, we did not test whether it complied with the required specifications.
 - However, it can check “complex combinations of conditions” that are often overlooked in black box testing
 - Also, there may be problems caused by the program (rather than the specifications)

The function is to reinforce black box testing

Realistically, you should perform a black box test and then pay attention to the coverage rate (C0, C1)

- It is of course necessary to check whether the specifications are met, so we execute several test cases using black box testing.
- At that time, by paying attention to the **statement coverage rate and branch coverage rate**, you can eliminate omissions in checks that **were not noticed** during black box testing.
 - The most basic is the statement coverage rate (C0), and if this is not 100%, that is, if there are statements that are not executed, it is no good.

Classification of tests

□ Black box testing

Perform operational tests **based on specifications** without looking at the program contents (black box)

□ White box testing

Perform operational tests **based on the internal structure** of the program (mainly flowcharts)

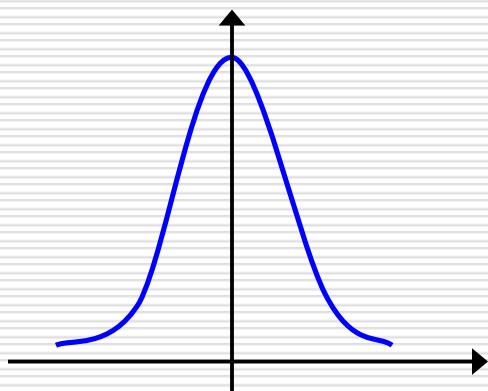
□ Random testing

Create test cases **randomly** and perform operational tests

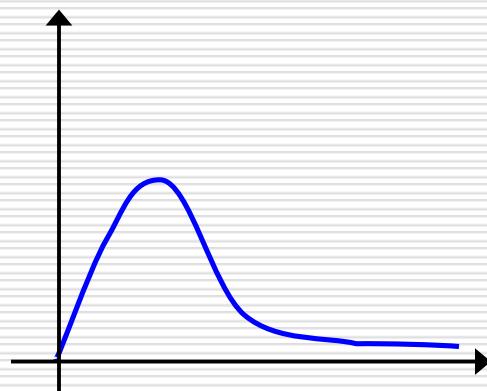
Random testing method

- How to **create input data randomly**
- It may be necessary to consider the **probability distribution** according to which random data is generated.

Normal distribution



Poisson distribution

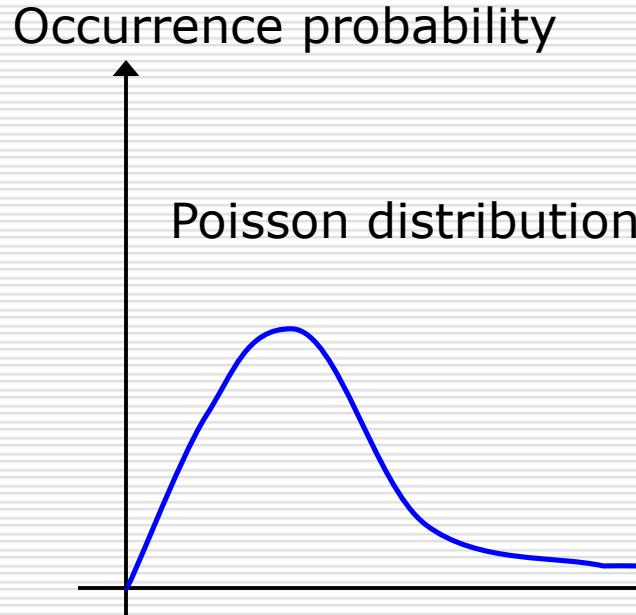


Uniform distribution



For example, consider the Poisson distribution

- When a request is sent to the server



When simulating such a situation, the number of requests per unit time can be modeled using a Poisson distribution.

Although it is random data,
→ Can perform tests
based on reality

Advantages and disadvantages of random testing methods

□ Advantage

- A large number of test cases can be automatically generated.
- Sometimes test cases are discovered that test designers and developers would not have thought of.

□ Disadvantage

- It is not possible to perform tests according to specifications.
- The coverage rate is low for the number of test cases.

Types of testing methods and stages of application

- Black box testing method
Can be used widely from unit tests to system tests
- White box testing method
Unit tests and (small) integration tests are limited (It is difficult to cover the control flow of the entire system)
- Random testing method
Try various input patterns to make sure there are no unexpected mistakes

Testing various requests

- The methods introduced so far mainly test the correctness of input and output.
- But, in reality, many other tests are required.

For example,

Various assumptions are required (Knowledge and experience are also important)

- Performance test (response time, processing speed)
- Storage test (required memory and hard disk capacity)
- Stress test (test under overload conditions)

Summary

- White Box Testing:
Execution Path Coverage
 - Statement coverage (C0): Execute all statements
 - Branch coverage (C1): Execute both True and False cases for all branches
 - Condition coverage (C2): Execute all combinations of True and False cases for all branches
- Random testing: Generate input data randomly
- It is also important to operate under load

Homework

Answer “[6] quiz”
by this Friday 23:59

(Note: Your quiz score will be a part of your final evaluation)

Important Notice

- Next week's lecture is an exercise.
You all must do exercises on your PC. Install gcc and gcov on your PC before the lesson.

ソフトウェアテスト

[8] テストの評価と信頼性

Software Testing
[8] Evaluation of testing and reliability

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Evaluation of the test

- Testing is the activity of quality assurance of software

Perfect Testing

⇒ There is no defect (fault) and no failure occurs.
There are no so-called “bugs”.

- Evaluation of “Testing Activities” is necessary
 - Evaluation based on Coverage rate
 - Evaluation based on Bug resolution rate
 - Evaluation based on Operability

Test evaluation (1)

Evaluation based on coverage rate

- Evaluation from the perspective of "how much was covered"
 - Focus on state transition
 - Focus on behavior or input/output according to specifications

*black box testing

- Focus on flow of program execution

*white box testing

Test evaluation (2)

Evaluation based on bug resolution rate

- Testing finds bugs and fixes them
- If the **total number of bugs** is known, testing activities can be evaluated based on the number of bugs resolved.

$$\text{Bug's resolution rate} = \frac{\text{number of bugs resolved}}{\text{total number of bugs}}$$

*Note: The term bug is not accurate; A phenomenon that does not work properly ("failure, defect") is found in the test, and "resolution" is to solve the "fault" that is the source of problem or the "error" that created it.

How do you know the “total number of bugs”?

- In conclusion, **the true number is unknown**
 - If we compare humans to systems, it would be like trying to figure out how many times we will get sick in our lifetime.
 - If we consider that “occurrence of a problem = symptoms of a disease”, then a fault can be considered to be the bacteria or virus that caused it, and an error can be considered to be the action that led to the infection.
- We can only estimate the total number of bugs (Statistically)

Bug number estimation method ①

Capture/recapture method (1/4)

[Example]

I would like to find out how many black bass there are in Lake Biwa.

*It would be nice if we could catch all the fish in Lake Biwa, but that is realistically impossible.



Bug number estimation method ①

Capture/recapture method (2/4)

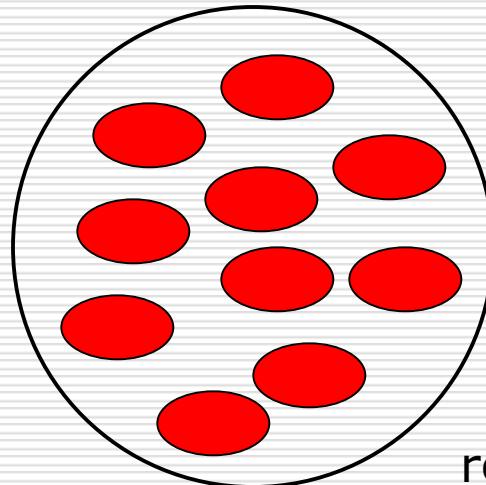
1. Catch an appropriate number of black bass.
2. Then **mark the black bass** and release it.
3. After a while, catch the black bass again. Estimate the total number based on how many marked ones are included.

Bug number estimation method ①

Capture/recapture method (3/4)

□ The image will like this

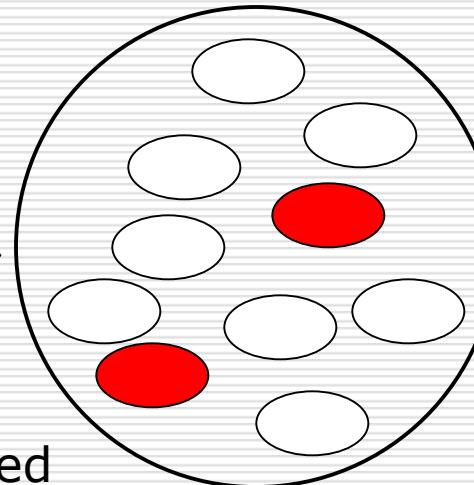
Marking
10 fishes
for the
first time



Then
recaptured



2nd time, 2
out of 10
fishes
confirmed
markings



Estimation of percentage of total This image diagram

$$\textcolor{red}{\bullet} : \textcolor{white}{\bullet} = 2 : 8$$

$$10 : ? = 2 : 8$$

So, there are 40 fishes left

Bug number estimation method ①

Capture/recapture method (4/4)

□ Applying this to software

1. Prepare n artificial bugs (marking)
2. Suppose that the test subsequently finds
 x artificial bugs and y real bugs.
3. The total number of real bugs is estimated by the following equation.

$n : ? = x : y$ Than

$$\frac{n \ y}{x}$$

Disadvantages of the capture / recapture method

- It's hard to make realistic artificial bugs
 - Inevitably, it becomes a bug that seems to be intentional, and it is found immediately
 - The psychology of engineers is that there is resistance to "creating bugs" (voices in the field)
- You end up releasing fish that are **easy to catch**, so the estimated number is lower than the actual number.

$$\frac{ny}{x}$$

* y is smaller and x is larger than it should be.

Bug number estimation method (2)

Two-step editing method (1/3)

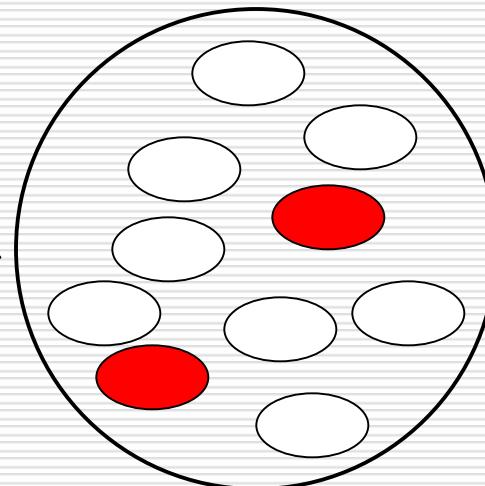
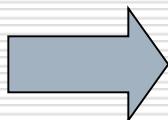
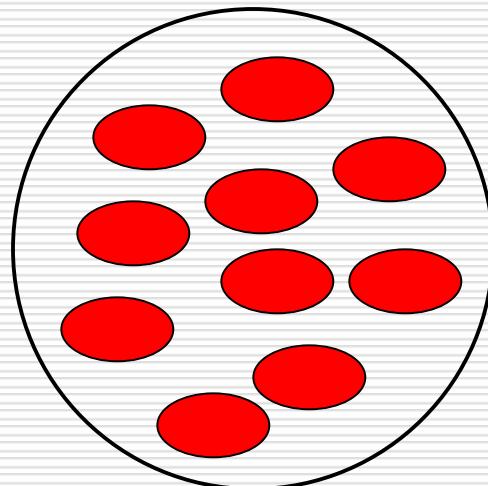
- Improved capture/recapture method without using artificial bugs
1. Prepare **two test groups A, B**
 2. **Group A tests** and records the bugs found (keeping them secret to **B**)
 3. **Group B tests** and estimates the number of remaining bugs based on **how many of the bugs found are the same as A.**

Bug number estimation method (2)

Two-step editing method (2/3)

□ Think like a capture-recapture method

Group A
found
10 bugs



2 out of 10 items
found in group B
included items
found in group A

Estimation of percentage of total **10 : ? = 2 : 8**

$$\text{Red oval} : \text{White oval} = 2 : 8$$

So there are 40 bugs that
have not been found by A.

Bug number estimation method (2)

Two-step editing method (3/3)

- Number of bugs found by group A = a
- Number of bugs found by group B = b
- Number of bugs found in both = c

$$a : ? = c : (b - c)$$

$$\text{Number of bugs A hasn't found} = \frac{a(b - c)}{c}$$

The total number of bugs (including those found) is:

$$\frac{a(b - c)}{c} + a = \frac{ab}{c} - \frac{ac}{c} + a = \boxed{\frac{ab}{c}}$$

Features of Two-step editing method

- No need to prepare artificial bugs
(Estimation can be made using only real bugs)
- Since the two teams need to test separately, which is more time consuming and costly
- Estimates tend to be smaller than actual values

[Exercise 1]

- ① Find an estimate of the total number of bugs

Group A finds **10** bugs,

Group B finds **50** bugs.

However, 5 were found in both groups.

- ② Find an estimate of the number of remaining bugs (not found yet).

Group A finds **100** bugs,

Group B finds **80** bugs.

However, 20 were found in both groups.

[Exercise 1] Answer

① $a = 10, b = 50, c = 5$

total number of bugs = $a \cdot b / c = 10 \times 50 / 5$
= **100**

② $a = 100, b = 80, c = 20$

total number of bugs = $a \cdot b / c = 100 \times 80 / 20$
= 400

therefore,

because what is found in both is
double subtracted

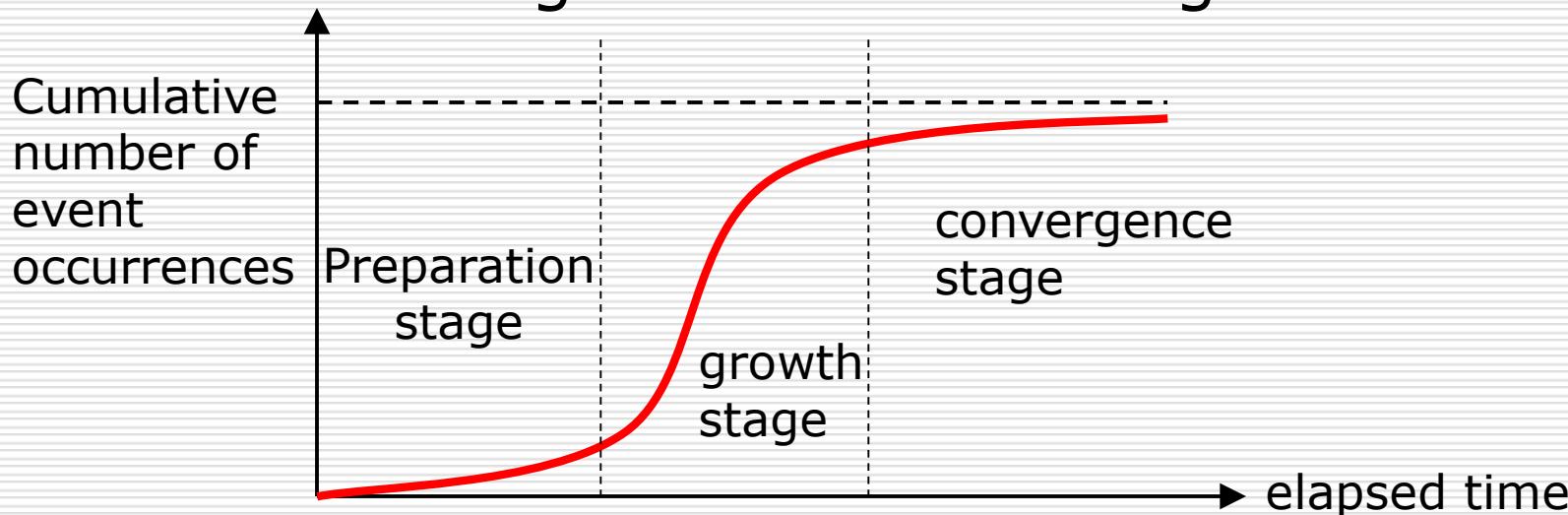
number of remaining bugs = $400 - a - b + c$
= $400 - 100 - 80 + 20 = \b{240}$

Bug number estimation method (3)

Gompertz curve model (1/4)

□ What is the Gompertz curve model?

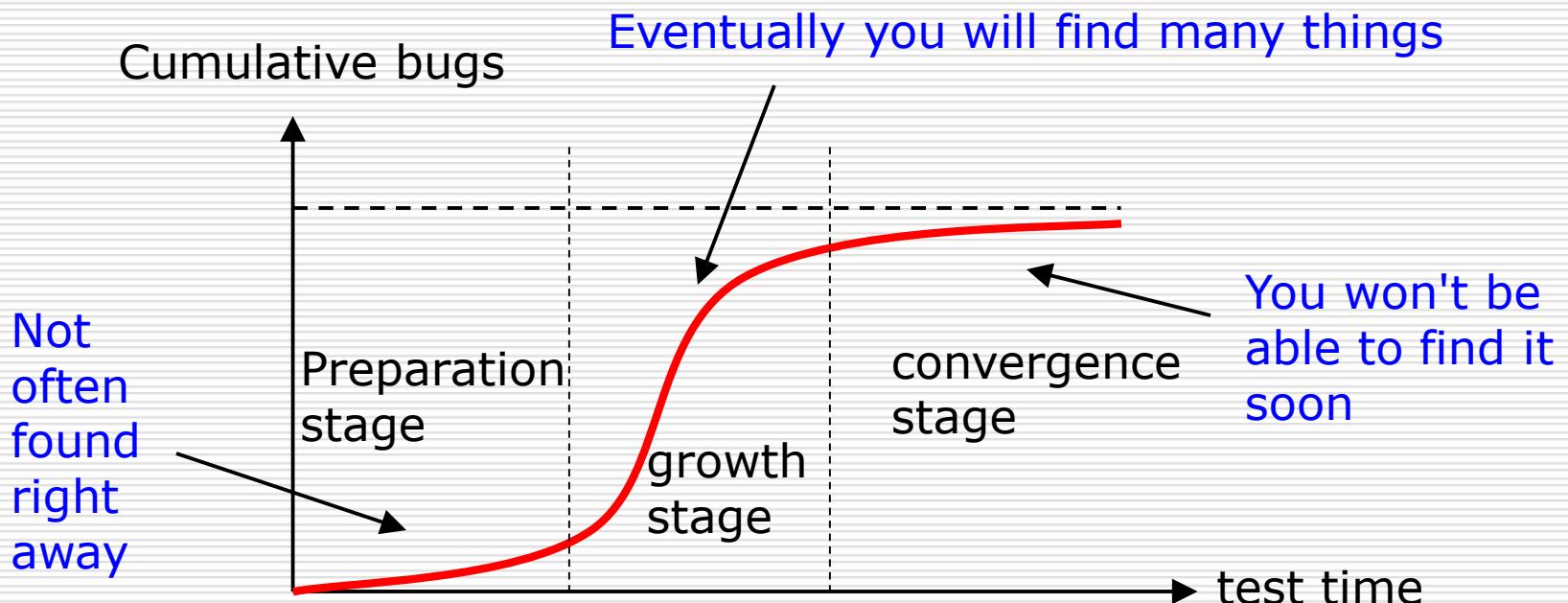
A type of growth curve model that represents the process from the beginning of various events to growth and convergence.



Bug number estimation method (3)

Gompertz curve model (2/4)

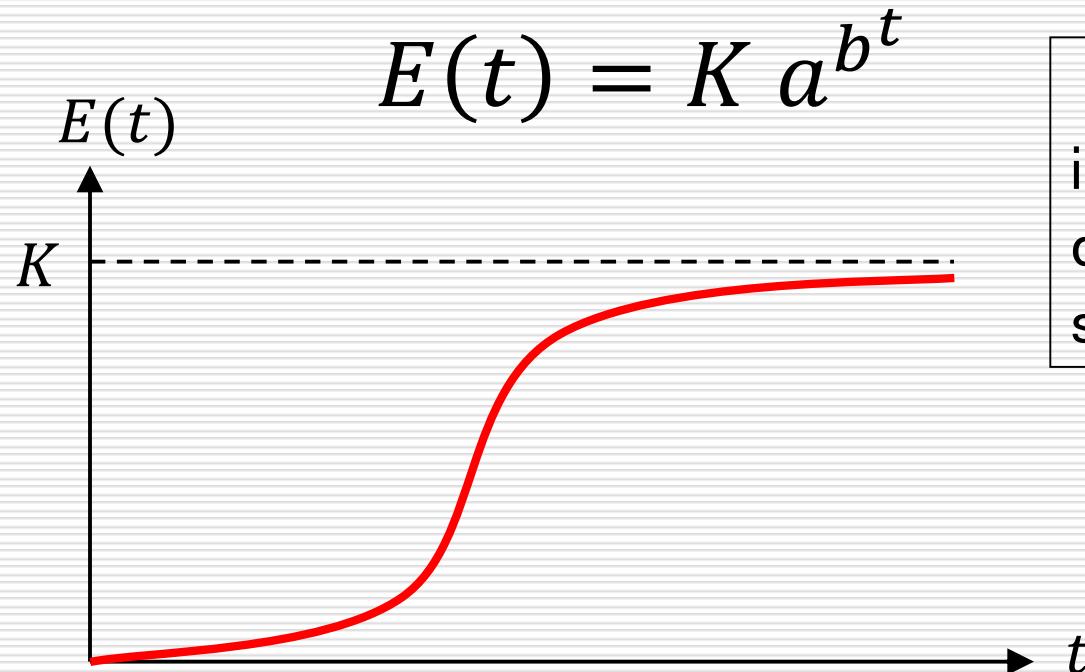
- Used for growth model of cumulative number of bugs



Bug number estimation method (3)

Gompertz curve model (3/4)

□ Mathematically



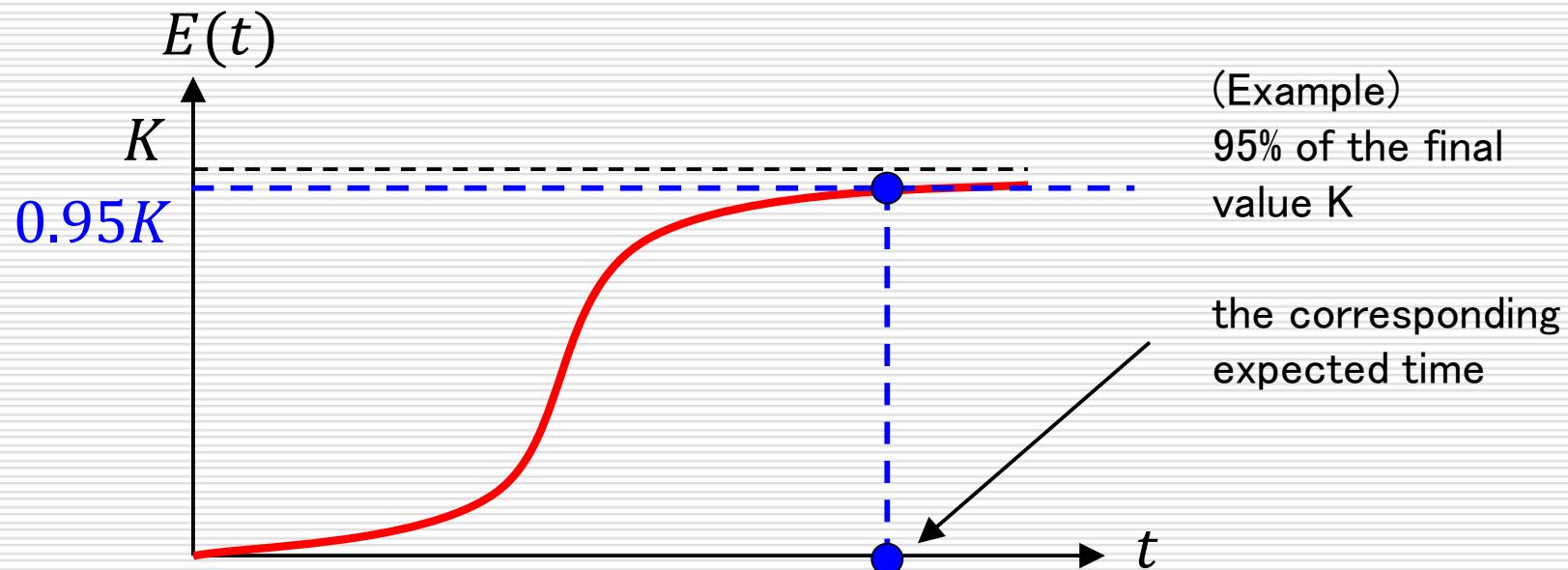
$a, b (< 1), K$
is a parameter that
determines the
shape of the curve

*This curve shows an empirical trend

Bug number estimation method (3)

Gompertz curve model (4/4)

- It is possible to estimate (predict) the test time required to obtain reliability at a specific point in time or target reliability.



Bug number estimation method (3)

Software reliability growth model

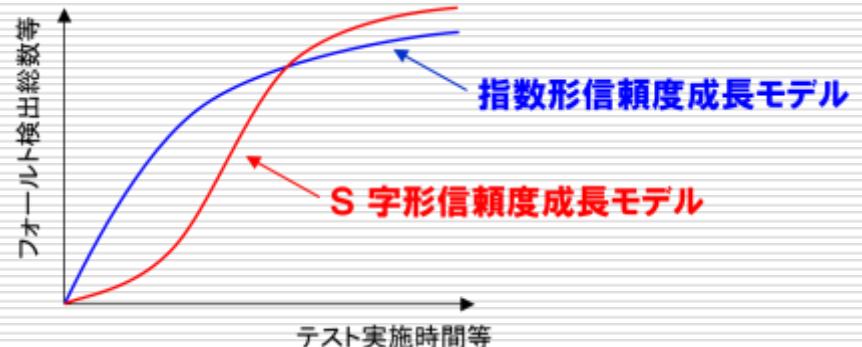
□ Exponential Software Reliability Growth Model

$$E(t) = a(1 - e^{-bt})$$

Mathematical model
expressing bug detection
probabilistically

□ Delayed S-shaped SRGM

$$E(t) = a \{ 1 - (1 + bt)e^{-bt} \}$$



Bug number estimation method ④

Method based on past statistical data

- Method using actual values from past projects
- For example, if the past bug density is 3.8 [bugs/KLOC] and the scale of this development is 100 [KLOC], then it is estimated that simply $3.8 \times 100 = 380$ bugs

*LOC = Lines Of Code : Number of code lines (excluding comments and blank lines) KLOC = 1000 LOCs

- Although it is a simple method, the prediction accuracy is not bad

[Exercise 2] Predict the number of remaining bugs

- When testing the deliverables of a project, we found a total of **38 bugs**.
- The bug density in a similar project in the past was **2.5 [bugs/KLOC]**.
- The scale of this development is **20 [KLOC]**.

How many more bugs are expected to remain undetected?

[Exercise 2] Answer

- First, based on past performance and the scale of this development,

$$2.5 \times 20 = 50$$

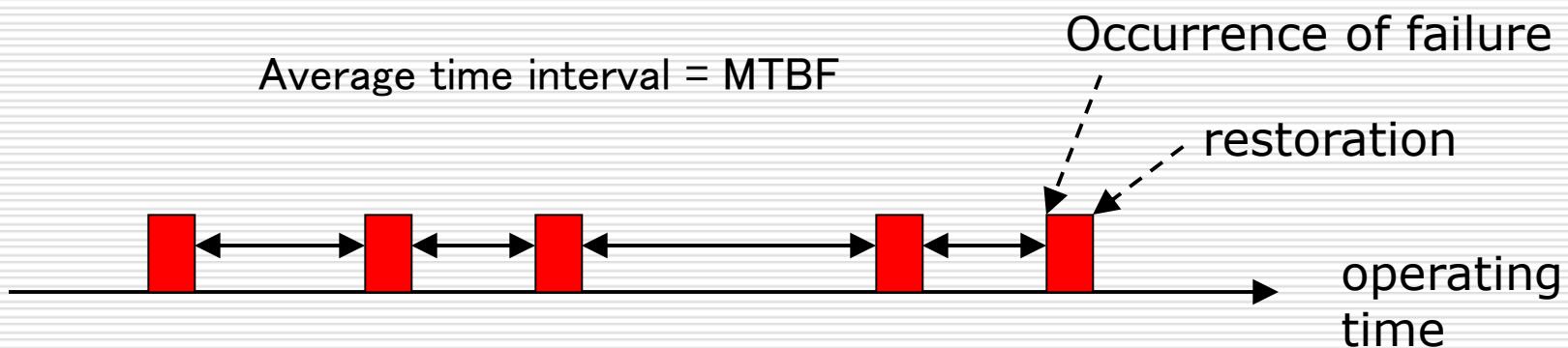
Than, it is predicted that there will be a total of **50 bugs**

- At the moment, 38 bugs found (76% = $38/50$), so **the rest 12 bugs** (24%).

Evaluation of the test (3)

Evaluation based on operability

- In system testing (imitating operations)
Average value of the time interval at which failures (malfunctions) occur
MTBF (Mean Time Between Failure)



Evaluation by MTBF

- MTBF length means high reliability
- However, it is meaningless unless the system test environment is close to the actual operational environment.

Random testing according to appropriate stochastic models is important

(Of course, human testing is also important)

Summary

- Test Evaluation:
 - coverage rate, bug resolution rate
 - Estimation of number of bugs → Two-step editing method, growth curve, bug density
- Operability Evaluation: MTBF

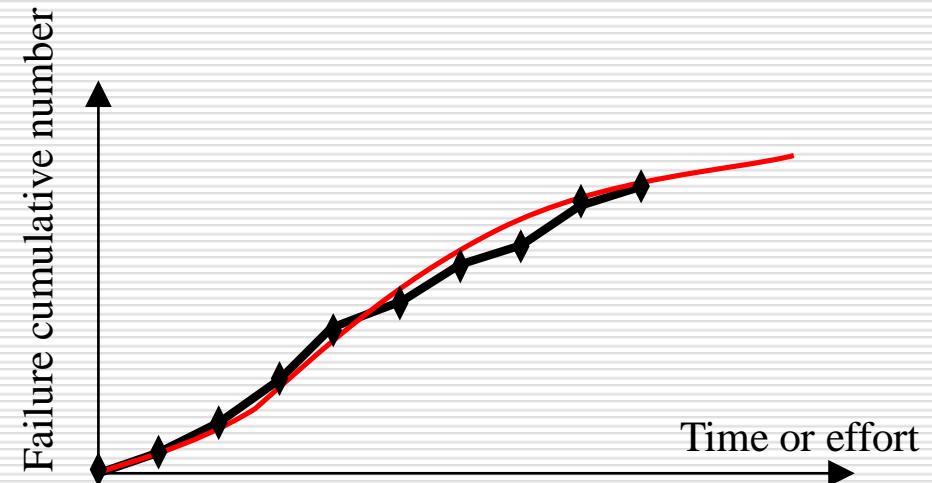
SOFTWARE RELIABILITY GROWTH MODEL (THEORY EXPLAINED AND R EXERCISES)

Download lecture08-materials.zip
(Includes data files srgm-data1.csv, srgm-data2.csv
and R script Rscript08.R)

Growth curve

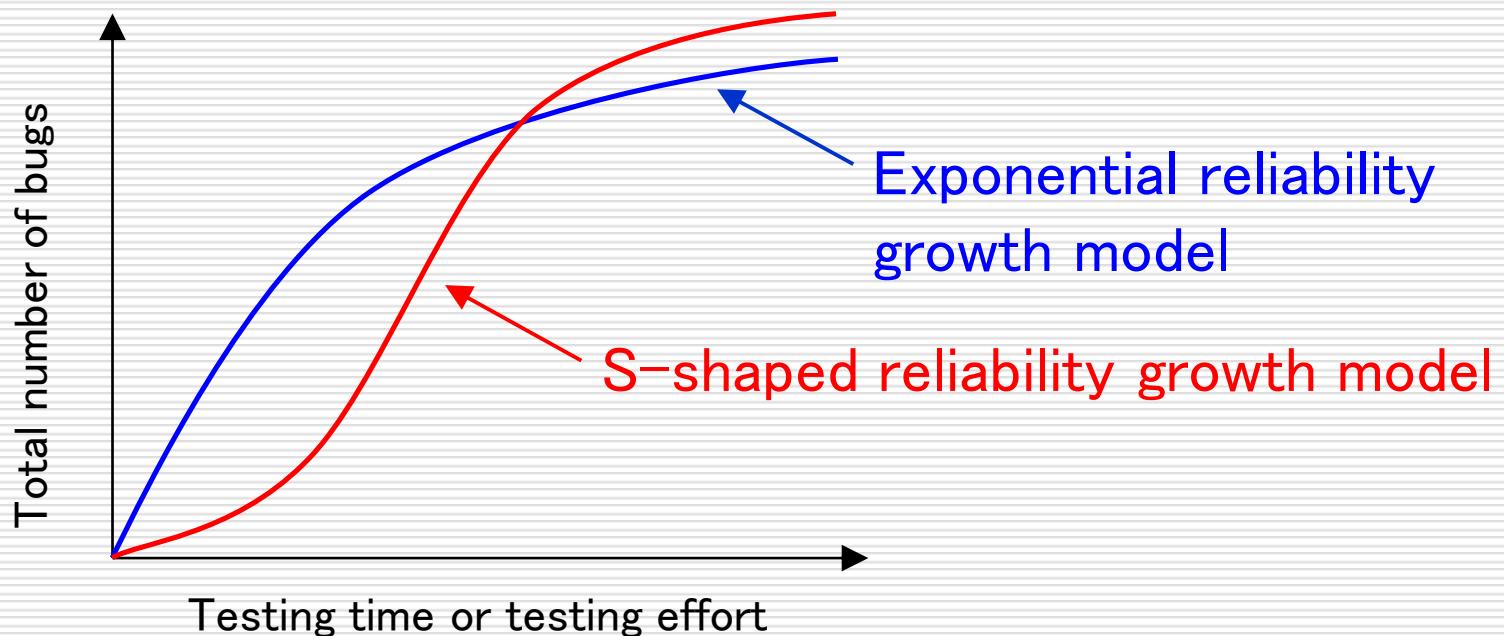
- A curve that models how the target data changes over time and input of man-hours
 - Using cumulative number to catch its growth
 - Estimation using models is also performed

The software reliability growth model that models the number of detected failures is famous.



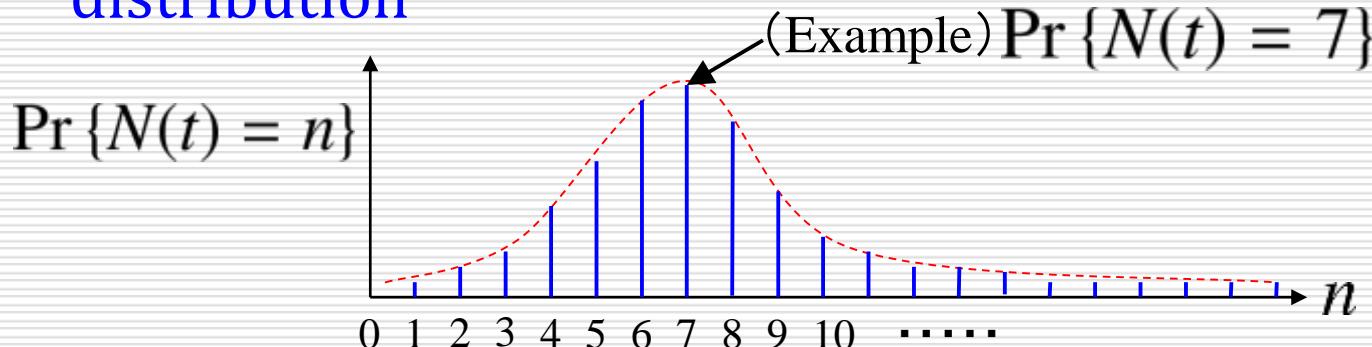
Software reliability growth model: SRGM

- Modeling the relationship between testing time and total number of bugs.



Stochastic process model for the number of bugs

- Modeling the relationship between testing time and total number of bugs.
- The **total number of bugs** at time t is represented as a **random variable** $N(t)$ (Consider a series of random variables over time)
- When the time t is fixed, $N(t)$ has **some probability distribution**



(Reference)

Binomial and Poisson distribution (1/3)

- Binomial distribution: If an event occurs with probability p , then the probability that it will actually occur k times in n trials is

$$\binom{n}{k} p^k (1 - p)^{n-k}$$

Ex: Probability of a traffic accident occurring in a city in one minute: $p = \frac{1}{1440}$

Probability of no traffic accidents in the next 24 hours

$$\binom{1440}{0} p^0 (1 - p)^{1440-0} = (1 - p)^{1440} = \left(\frac{1439}{1440}\right)^{1440}$$

(Reference)

Binomial and Poisson distribution (2/3)

- When we actually calculate

$$\left(\frac{1439}{1440}\right)^{1440} = \underline{\underline{0.3677517}}$$

- When n is big and p is small, it can be approximated by a Poisson distribution:

$(np < 10 \text{ and } n > 1500p)$

$$P(X = k) \frac{(np)^k e^{-np}}{k!}$$

$$(n = 1440, p = \frac{1}{1440}, k = 0) \\ = e^{-1} = \underline{\underline{0.3678794}}$$

(Reference)

Binomial and Poisson distribution (3/3)

- The approximation formula is (as $\lambda = np$)

$$P(X = k) = \frac{(np)^k e^{-np}}{k!} = \frac{\lambda^k e^{-\lambda}}{k!}$$

- This is generally known as **Poisson distribution**
- Both expected value (mean) and variance are $\lambda = np$

Poisson Process

- If we find bugs at a rate of λ per unit time ($\lambda > 0$)

Expected number of total bugs at time t is λt

Its probability distribution follows the Poisson distribution

$$\Pr\{ N(t) = n \} = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad (n = 0, 1, 2, \dots)$$

In other words, a series of random variables like this (following a Poisson distribution) is formed → Poisson process

Expected number of bugs (1/3)

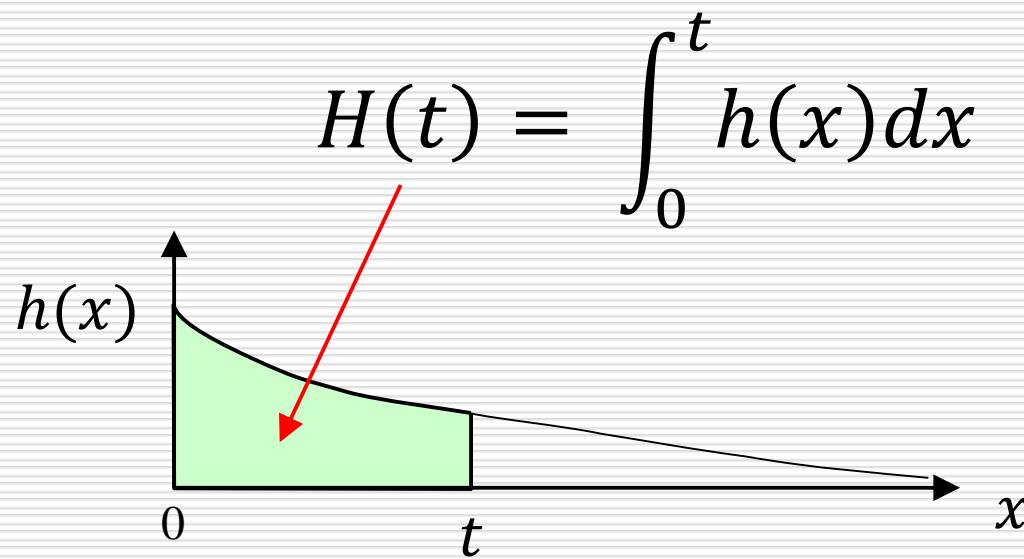
- In fact, the number of bugs found per unit of time is not constant (if it were, there would be an infinite number of bugs: $t \rightarrow \infty$, the expected value $\lambda t \rightarrow \infty$)
- Therefore, let the expected value at time t be $H(t)$

$$\Pr\{ N(t) = n \} = \frac{H(t)^n}{n!} e^{-H(t)}$$

$(n = 0, 1, 2, \dots)$

Expected number of bugs (2/3)

- The expected number of bugs at time t $H(t)$ is obtained from the bug detection rate $h(t)$ at each time up to that point



for example, $h(3)$ is the bug detection rate at time $t = 3$ and at that point, the probability of detecting $h(3)$ bugs per unit time is

Expected number of bugs (3/3)

- Of course, if the bug detection rate is always constant, $h(t) = \lambda$ (constant)

$$H(t) = \int_0^t h(x)dx = \int_0^t \lambda dx = \lambda t$$

$$\begin{aligned}\Pr\{ N(t) = n \} &= \frac{H(t)^n}{n!} e^{-H(t)} \\ &= \frac{(\lambda t)^n}{n!} e^{-\lambda t}\end{aligned}$$

A typical bug detection count model: NonHomogeneous Poisson process model

- Generalize bug detection rate as a function of time $h(t)$
- The Poisson process considers expected value as $H(t)$ instead of λt is called NHPPs

$$\Pr\{ N(t) = n \} = \frac{H(t)^n}{n!} e^{-H(t)} \quad (n = 0, 1, 2, \dots)$$

$$H(t) = \int_0^t h(x) dx$$

NHPPs mean value function and intensity function

$$\Pr\{ N(t) = n \} = \frac{H(t)^n}{n!} e^{-H(t)} \quad (n = 0, 1, 2, \dots)$$

$$H(t) = \int_0^t h(x) dx$$

$H(t)$: mean value function

Expected the total number of bugs $N(t)$ at time t

$h(t)$: intensity function

*Not a probability

Bug detection rate at time t

The current estimate of how many faults are likely to be found in a unit of time.

(Reference)

Homogeneous Poisson process

- If the intensity function is constant over time:

$$h(t) = \lambda \quad (\lambda > 0)$$

- Then the random variable $N(t)$ is a Homogeneous Poisson process (HPP)

Typical NHPPs model

- When actually evaluating and predicting reliability, it is necessary to specify the Mean Value Function **under some assumption**
- Example

Model name	Mean value function $H(t)$
Exponential	$a(1 - e^{-bt})$
Delayed S-shaped	$a\{1 - (1 + bt)e^{-bt}\}$

Assumptions set by many NHPP models

The number of bugs found per unit time
is proportional to the number of remaining
(undetected) bugs at that time

That is, the bug detection rate at time t is proportional to the
number of remaining bugs at that time

$$a - H(t) \quad h(t)$$

$$h(t) = b(t)\{a - H(t)\}$$

$b(t)$: Proportionality coefficient at time t
(Discovery rate per bug)

Assumptions set by many NHPP models

- Again, if we write it using the mean value function $H(t)$

$$\frac{dH(t)}{dt} = b(t)\{a - H(t)\}$$

$$H(t) = \int_0^t h(x)dx$$

- Solving this equation with the initial condition $H(0) = 0$

$$H(t) = a(1 - e^{-\int_0^t b(x) dx})$$

(1) Exponential Software Reliability Growth Model (Exponential SRGM)

- Assume that $b(t)$ used for proportionality coefficient is constant over time

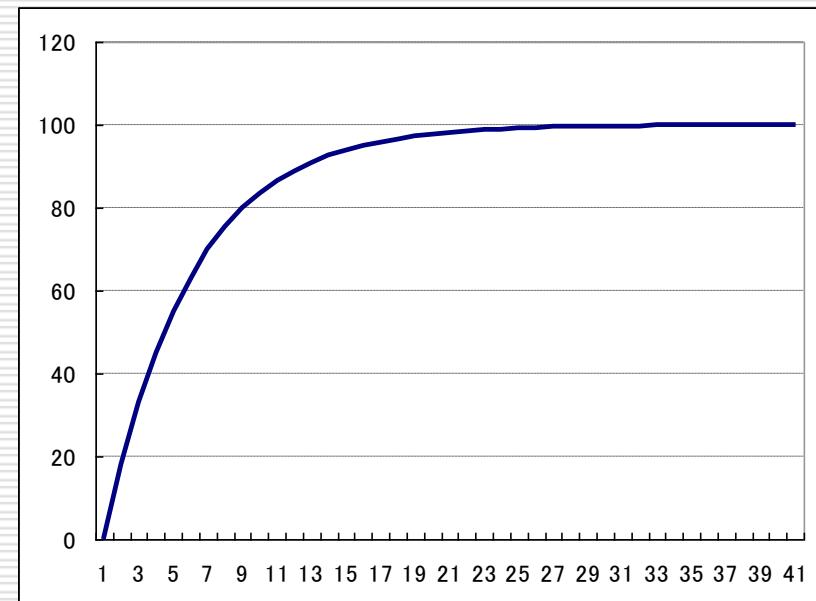
$$H(t) = a(1 - e^{-\int_0^t b(x) dx})$$

\downarrow

$b(t) = b$

$$H(t) = a(1 - e^{-bt})$$

(example) $a = 100, b = 0.2$



When actually using it

- Naturally, the curve parameters a, b are unknown and must be estimated in some way
- In reality, parameters are estimated so that the curve fits the actual data (available at that time) (maximum likelihood method or least-square method is used)

It is said that stable estimates are obtained after approximately 50 to 60% of the tests are completed.

[R exercise] Numerical example

- Fault detection data of the real-time command and control program (217 KLOC)
[Goel(1985)]

srgm-data1.csv

- Read this as a dataframe `d1`

```
d1= read.csv( file.choose())
```

[Goel(1985)] Goel, AL : Software reliability models: Assumptions, limitations, and applicability, IEEE Trans. Software Eng., vol.SE-11, no.12, pp.1411–1423, Dec. 1985.

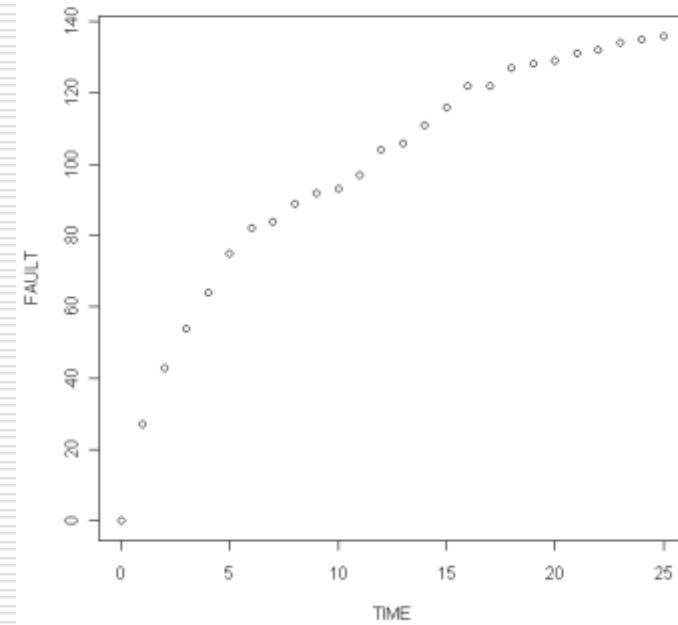
Numerical example : CSV Data Content

□ Elapsed time and accumulated number of bugs

```
> d1
```

	TIME	FAULT
1	0	0
2	1	27
3	2	43
.....

```
plot(FAULT ~ TIME, data=d1)
```



Time is the CPU time spent
running the test (in hours)

Growth curve fitting example

- For example, when using an exponential growth model

$$m(t) = a(1 - e^{-bt})$$

- $m(t)$: mean value function
(expected number of detected bugs at time t)
- a : Expected total number of bugs ($t \rightarrow \infty$)
- b : bug detection rate
- * This is a model in which bugs are detected with a constant detection rate (b).

Growth curve fitting example

- Estimating model parameters (nonlinear model)

```
model.e= nls( FAULT ~a*(1-exp(-b*TIME)),  
data=d1, start=list(a=200, b=0.2) )
```

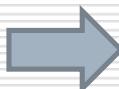


After execution

```
a=as.numeric(model.e$m$getPars()[1])  
b=as.numeric(model.e$m$getPars()[2])
```

Initial value for numerical calculation
(adjustment required if calculation does not go well)

a
b



135.9744
0.1382574

Note that the results of parameter estimation using the maximum likelihood method are different from the results of approximation using the least squares method.

Prepare the growth curve as a function

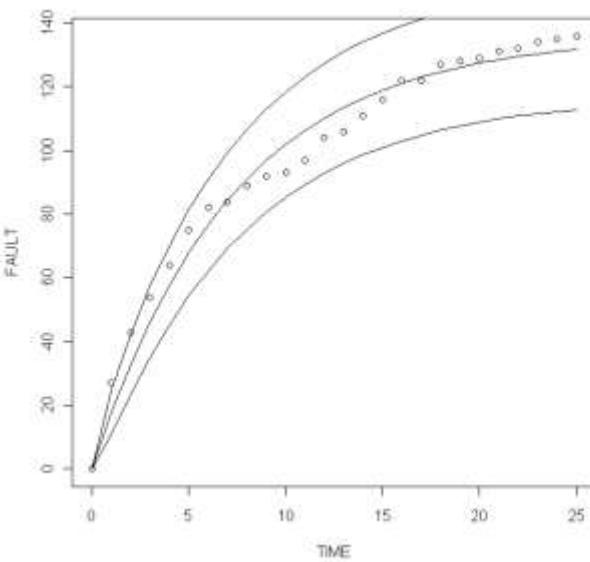
- Prepare the parameters of the exponential growth model, the mean value function as a function m

```
a.e = as.numeric(model.e$pars()[1])  
b.e = as.numeric(model.e$pars()[2])  
m = function(t){  
  a.e*(1-exp(-b.e*t))  
}
```

$$m(t) = a(1 - e^{-bt})$$

Growth curve fitting example (90% Confidence interval)

```
plot(FAULT ~ TIME, data=d1)
lines( m(TIME) ~ TIME, data=d1, col="red")
lines(m(TIME)+1.645*sqrt(m(TIME)) ~ TIME, data=d1)
lines(m(TIME)-1.645*sqrt(m(TIME)) ~ TIME, data=d1)
```



$$\begin{aligned} & a.e + 1.645 * \text{sqrt}(a.e) \\ & a.e - 1.645 * \text{sqrt}(a.e) \end{aligned}$$



$$\begin{aligned} & 155.1564 \\ & 116.7923 \end{aligned}$$

Total number of bugs is expected to be 117–155

Because $t = 25$ and 136 faults found
About 19 have not been found yet
(Reliability 90%)

(2) Delayed S-shaped SRGM (1/4)

- For complex software, bug detection is
 - Defect discovery
 - Investigation of cause
 - Bug (fault) detection
- Therefore, it can be interpreted that there is a difference between the expected number of bugs and the number of bugs at time t , and that difference influences the bug detection rate.

(2) Delayed S-shaped SRGM (2/4)

- Assuming that defect discovery is represented by an exponential model, and the mean value function is: $m(t) = a(1 - e^{-bt})$
- Then consider that the bug detection rate $h(t)$ at time t is proportional to the difference between the Number of defects found $m(t)$ and the Number of detected bugs $H(t)$

$$h(t) = b \{ m(t) - H(t) \}$$

(2) Delayed S-shaped SRGM (3/4)

- In short,

$$\begin{cases} h(t) = \frac{dH(t)}{dt} = b\{ m(t) - H(t) \} \\ m(t) = a(1 - e^{-bt}) \end{cases}$$

- Solving this differential equation

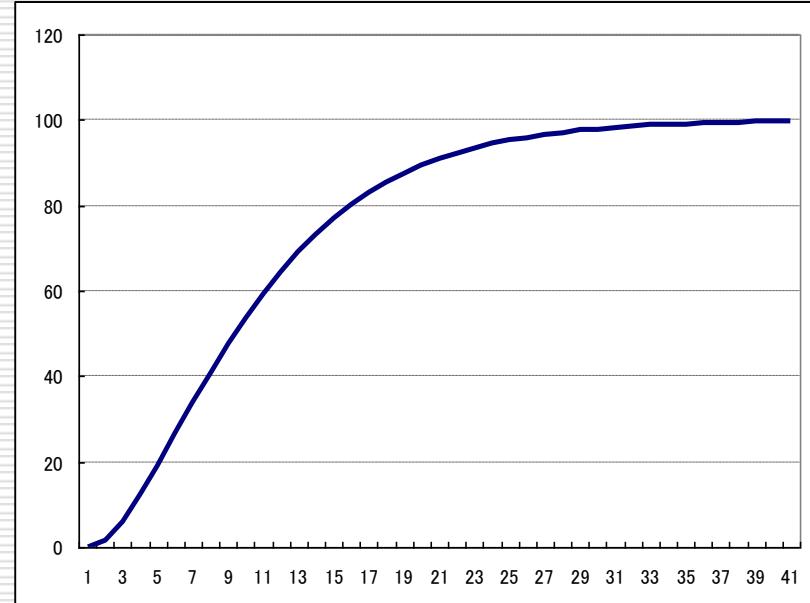
$$H(t) = a \{ 1 - (1 + bt)e^{-bt} \}$$

(2) Delayed S-shaped SRGM (4/4)

$$H(t) = a \{ 1 - (1 + bt)e^{-bt} \}$$

(example) $a = 100, b = 0.2$

- Intuitively, the inclusion of two exponential models causes a "delay" in growth



[R exercise] Numerical example

- Bug detection data for the online data entry software package (40 KLOC)
srgm-data2.csv [Ohba(1984)]

- Read this as a dataframe **d2**

```
d2 = read.csv( file.choose())
```

[Ohba(1984)] M. Ohba : Software reliability analysis models,
IBM J. Research and Development, vol.28, no.4, pp.428–443, July 1984.

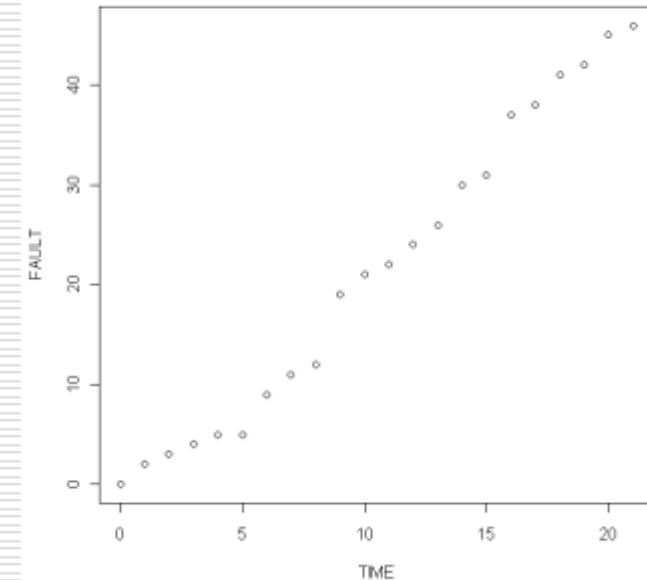
Numerical example : CSV Data Content

□ Observation time and Cumulative number of bugs

```
> d2
```

	TIME	FAULT
1	0	0
2	1	2
3	2	3
.....

```
plot(FAULT ~ TIME, data=d2)
```

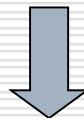


Time is the number of days spent on the test

Fitting a Delayed S-shaped

- Estimation model parameters (nonlinear model)

```
model.d= nls(FAULT ~a*(1-(1+b*TIME)*exp(-b*TIME)),  
data=d2, start=list(a=200,b=0.2) )
```



After execution

```
a=as.numeric(model.d$m$getPars()[1])  
b=as.numeric(model.d$m$getPars()[2])
```

a
b



78.69114
0.09493751

Initial value for numerical calculation (adjustment required if calculation does not go well)

Prepare the growth curve as a function

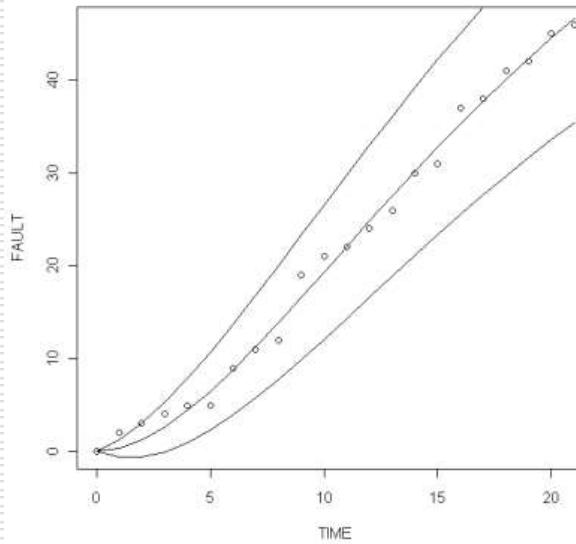
- Prepare a delay S-shaped long curve pattern, prepare the mean value function as H

```
a.d = as.numeric(model.d$m$getPars()[1])  
b.d = as.numeric(model.d$m$getPars()[2])  
H = function(t){  
  a.d*(1-(1+b.d*t))*exp(-b.d*t))  
}
```

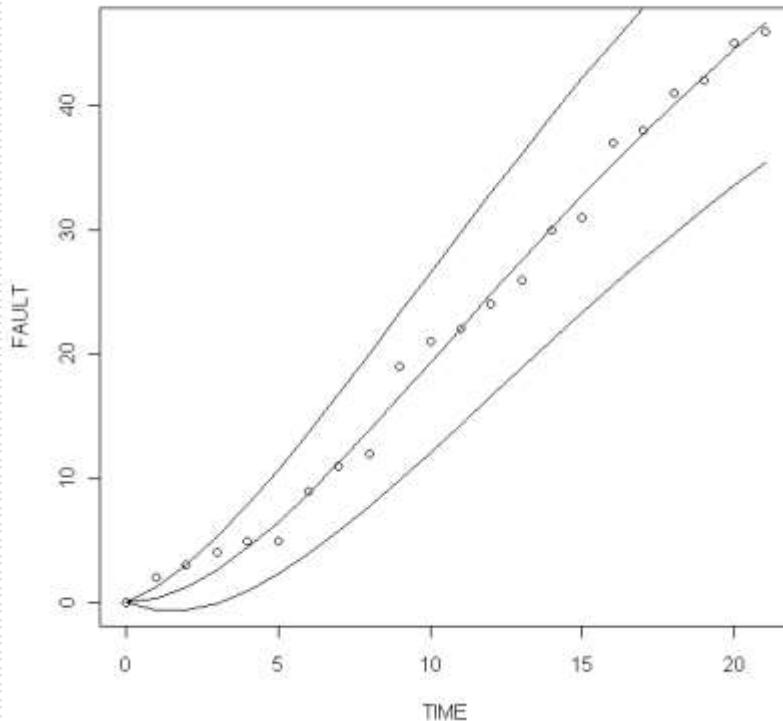
$$H(t) = a \{ 1 - (1 + bt)e^{-bt} \}$$

Growth curve fitting example (90% confidence interval) (1/2)

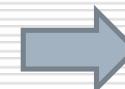
```
plot(FAULT ~ TIME, data=d2)
lines( H(TIME) ~ TIME, data=d2, col="red")
lines( H(TIME)+1.645*sqrt(H(TIME)) ~ TIME, data=d2)
lines( H(TIME)-1.645*sqrt(H(TIME)) ~ TIME, data=d2)
```



Growth curve fitting example (90% confidence interval) (2/2)



$$a.d+ 1.645*\sqrt{a.d}$$
$$a.d- 1.645*\sqrt{a.d}$$



$$93.28361$$
$$64.09867$$

Total number of bugs is expected to be 64–93
Because $t = 20$ and 46 faults found
About 18–47 have not been found yet (Reliability 90%)

Homework

Answer “[8] quiz”
by this Friday 23:59

(Note: Your quiz score will be a part of your final evaluation)

Software Testing

[8] Evaluation of testing and reliability

Một số kiến thức toán liên quan

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)

aman@ehime-u.ac.jp

Quá trình/biến ngẫu nhiên (Stochastic Process/Variable)

- Quá trình ngẫu nhiên là một quá trình trong đó **kết quả** của quá trình **tại một thời điểm cụ thể** là **không chắc chắn** và có thể được mô tả bằng một **biến ngẫu nhiên**.
- Ví dụ: **Pr{N(t) = n}**
- Nghĩa là, **Xác suất của một quá trình ngẫu nhiên** (được biểu diễn bởi biến ngẫu nhiên $N(t)$) nhận giá trị n tại thời điểm t .

Phân phối xác suất (Probability Distribution)

- Phân phối xác suất là **một hàm** mô tả khả năng thu được các kết quả khác nhau có thể xảy ra trong một **sự kiện** ngẫu nhiên.
- **Giá trị trung bình** của phân phối xác suất là kết quả trung bình của các **sự kiện** ngẫu nhiên → **Gọi là Kỳ vọng** (**Mean/Expected value**).
- **Kỳ vọng cho biết giá trị** điển hình hoặc **trung bình** mà người ta **mong đợi** quan sát được nếu **sự kiện** được lặp lại nhiều lần.

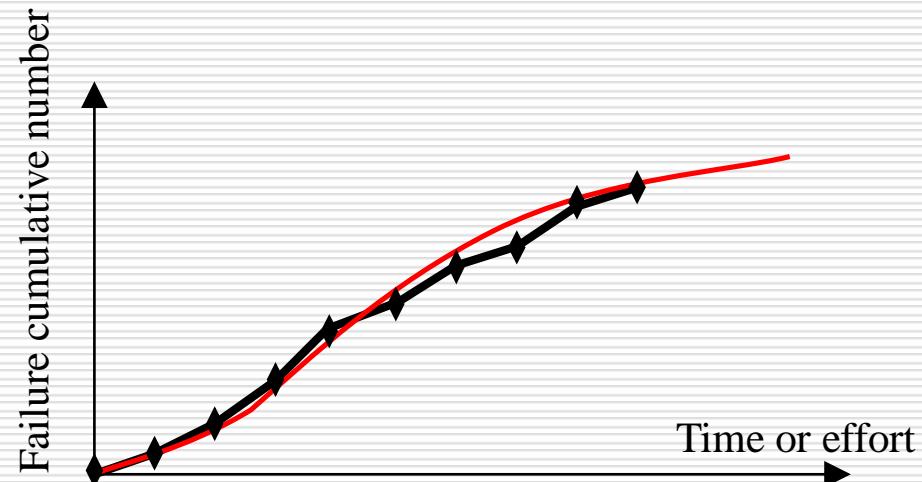
Kỳ vọng và Phương sai (Expected value & Variance)

- Phương sai là thước đo mức độ phân tán / trải rộng của phân phối xác suất
- Nó hàm ý các giá trị trong tập dữ liệu thường ở cách giá trị trung bình (Tức kỳ vọng) bao xa
- Phương sai cao chỉ ra các điểm dữ liệu nằm cách xa giá trị trung bình, nghĩa là tập dữ liệu được trải rộng hơn
- Phương sai thấp cho biết các điểm dữ liệu nằm ở gần giá trị trung bình

Đường cong tăng trưởng (Growth curve)

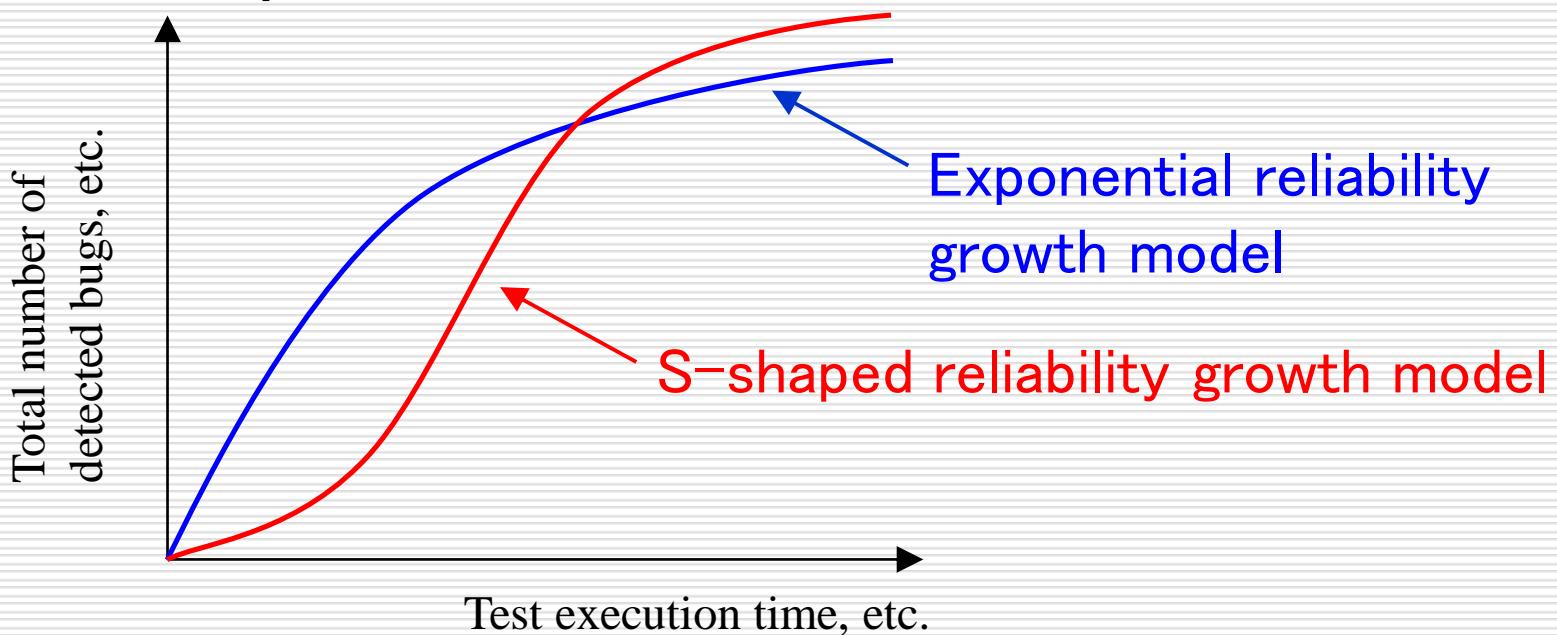
- ☐ Một đường cong mô hình cách dữ liệu mục tiêu thay đổi theo thời gian
- ☐ Sử dụng giá trị tích lũy để bắt kịp tốc độ tăng trưởng của dữ liệu mục tiêu (số bugs)

Trong đó, có 1 mô hình rất nổi tiếng là SRGM, dùng để mô hình hóa số lần phát hiện lỗi.



Software reliability growth model: SRGM

- ☐ Một mô hình điển hình về mối quan hệ giữa thời gian kiểm thử và tổng số bug được phát hiện.



Software reliability growth model: SRGM

- $a(1 - e^{-bt})$
- $a \{ 1 - (1 + bt)e^{-bt} \}$
- Mục tiêu: Ước tính các tham số của đường cong để khớp với dữ liệu thực tế nhất có thể
- → Từ đó có thể ước lượng tổng số bug trong tương lai

ソフトウェアテスト

[9] プログラミング技術

Software Testing
[9] Programming Tips and Techniques

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Position of programming

- The role of programming is to realize “program specifications” on a computer

Program specification



Make something like this

human world

programming

The key
is how
correctly
it can be
described

program

Formal execution instructions



computer world

The computer faithfully executes the contents of the program

Software development \neq Programming

- People tend to think that “Programming is software development”
 - In the case of small-scale development, requirements analysis & design are done only in the developer’s mind, **and programming is suddenly started.**
 - However, this is not enough for **Large-scale development**
 - Programming skills are important
 - Don’t underestimate design and management



For example, OS for smartphones: millions of lines or more

When printed → more than 100,000 sheets of A4 paper (over 10m in height)

Important things in programming

- Writing programs that correctly reflect the specifications (**error-free**)
 - Be careful not to make the mistake of creating bugs (errors)
- **Write easy to understand** programs
 - Idea: “I just need to know myself” → Is not good
 - **Maintenance is impossible unless the program is understood by others**
(Even the person who wrote it may not be able to understand it after 6 months)

How to write and make a program

- 1.** Easy to read and understand
- 2.** Prevent mistakes
- 3.** Simple structure
- 4.** Be environment independent
- 5.** Write comments appropriately
- 6.** Make it easy to change

(1) Easy to read and understand

- Add indentation to visually tell where the block starts and ends.

To prevent forgetting something, write "}" whenever you write "{" and insert the code in between.

- Intentionally insert blank lines to emphasize the change in execution content

```
sum = 0;  
for ( i = 0; i < n; i++ ){  
    if ( a[i] > 0 ){  
        sum += a[i];  
    }  
}  
  
printf("%d\n", sum);
```

Make variable names meaningful

- A name that explains **what the variables represent** is desirable

index, position, count
or idx, pos, cnt for short

A program with few lines and easily understandable content can be handled without excessive worry, like the variable i on the previous page.

- When asking Yes/No,
the **interrogative form** is also used

is_empty, hasNext

To separate words, use underscore (_) or use capital letters

Example: If the set is empty → 1, Otherwise 0

Example: In the list, the following node is 1 if present, otherwise 0

Variable name, function name snake case and camel case

snake case

Use underscore (_) to connect words

Ex: file_name, get_file_size

For C language,
snake case
is often used

camel case

use capital letters for word breaks

Ex: fileName, getFileSize

Be aware of function and return value in function names

- Use a name that makes it easy to understand what the function does and what the return value is.

(Example)

```
int get_length (char str[ ]){  
    ... 文字列の長さを調べる  
}
```

```
void move_to (int x, int y){  
    ... (x,y) へ移動  
}
```

```
void sort (int array[ ], int length){  
    ... array をソーティング  
}
```

To separate words, use underscore (_) or use capital letters

[Exercise 1] Think of a function name

□ get help

void ()

□ leap year

int (int year)

□ Copy the contents of an array to another array

void (int src[], int dest[], int length)

Original Copy to length

[Exercise 1] Think of a function name (Example answer)

□ get help

```
void print_help ()
```

(Another solution)

printHelp
isLeapYear
copyArray

□ leap year

```
int is_leap_year (int year)
```

□ Copy the contents of an array to another array

Or **array_copy**

```
void copy_array (int src[], int dest[], int length)
```

Original

Copy to

length

(2) Prevent mistakes

□ Write curly braces even in a single statement

```
if ( x > 0 )  
    foo(x);
```

```
if ( x > 0 )  
    printf("%d",x);  
    foo(x);
```

A little mistake

```
if ( x > 0 ) {  
    printf("%d",x);  
    foo(x);  
}
```

□ Avoid reuse of variables

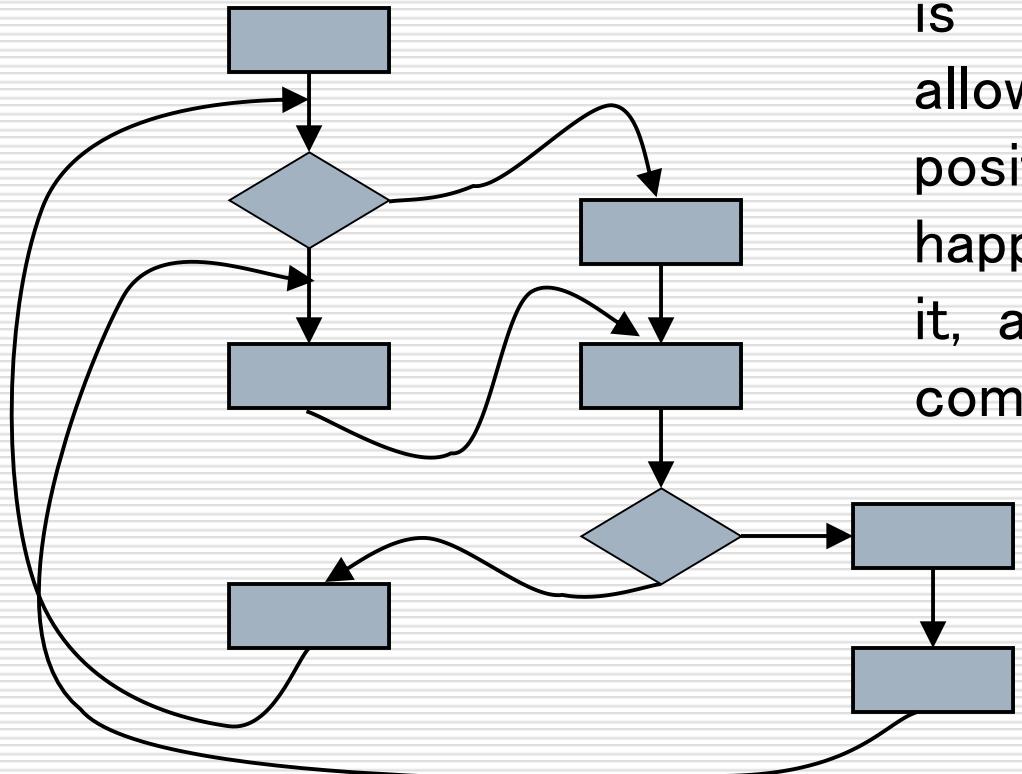
```
int x = 0;  
....(use x)  
x = 5;
```

....(x used for another purpose)

Someone asked me if there is some relationship between the variable x above and below. It is misunderstood.

(3) Simple structure

□ Do not use goto



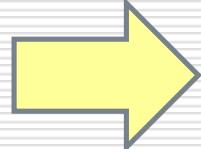
I feel like the goto statement is convenient because it allows you to jump to any position you want, but it only happens while you're creating it, and it just becomes more complicated in the end.

This kind of program is called
"spaghetti program"

Avoid too many conditional branches

- The more if, for, while statements, the more difficult to understand and the more mistakes

```
sum = 0;  
scanf("%d", &x);  
do{  
    if ( x > 0 ){  
        sum += x;  
        scanf("%d", &x);  
    }  
} while( x > 0 );
```



A better program (same movement)

```
sum = 0;  
scanf("%d", &x);  
while( x > 0 ){  
    sum += x;  
    scanf("%d", &x);  
}
```

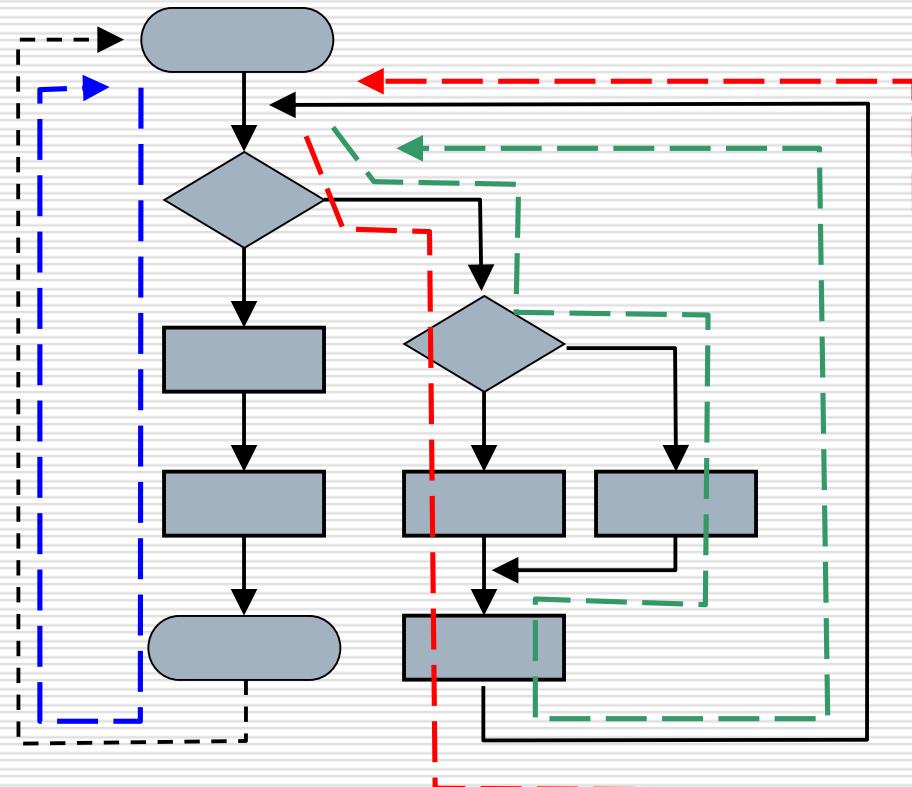
After trial and error, the program may have been like the one on the left, but it has become more complicated than necessary

Cyclomatic number

- Number of independent loops in the flow chart

It is known as a measure to evaluate the complexity of program structure.

There is a research report that says that maintainability is quite poor when this number exceeds 10.



(4) Be environment independent

- Avoid the influence of differences in computer environments
 - In the C language, the number of bytes allocated to a variable may vary depending on the processing system (for example, an int type may be 2 bytes or 4 bytes).

That is why we write **size of(int)**
 - If the OS is different, the header (library) read by #include may also be different.

A little technical, but switching using #ifdef etc.

(5) Write comments appropriately

- Functional description should be included as a comment for each meaningful chunk
- However, it is not enough to simply write a program

A program that cannot be understood without writing comments is also a problem

In many cases, it is sufficient to keep the program's structure simple by making the function and variable names easy to understand.

(6) Make it easy to change

□ Stop hard coding

- Writing numbers and strings as they are in the source code is called hard coding.
- Should be defined or read with symbols first (Ex) Number of data and file name

```
for ( i = 0; i < 256; i++ ){  
    for ( j = 0; j < 256; j++ ){  
        ....
```

Replace with a macro such as SIZE

```
FILE* fp = fopen("foo.txt", "r");
```

Read the file name at runtime or
specify it with a macro

[Exercise 2] About hard coding

- For example, it is recommended to give the length of an array and the name of a data file with macros like

```
#define SIZE 256
#define DATA_FILE "foo.txt"
```

- Consider the benefits of doing this

[Exercise 2] About hard coding (Sample answer)

① Resistant to change

If you want to change the length of the array, you can just rewrite 256 (replacing it every time will lead to mistakes).

② Easy to understand the meaning

In some cases, the meaning of a number or string alone may be misunderstood (the same “256” can also mean “there and here are different meanings”)

③ Easy to manage configurations

Unique configurations can be gathered in one place (at the beginning of the program), making it easy to change and check.

```
#define SIZE 256  
#define DATA_FILE "foo.txt"
```

If “255” was written to mean “SIZE-1”, the change would have been omitted.

Conventions for using macros

- When defining macros in the C language, it is common to have their names in all uppercase letters.
`#define SIZE 256`
- On the other hand, keep all Variable names lowercase
`int size;`
- That way, even if the two are mixed, you can immediately see whether they are variables or macros.

Refactoring

- Refactoring is the process of modifying source code to improve its **readability** and **maintainability** without changing its **functionality** or **meaning**.
 - Also includes improvements in performance (execution speed, memory usage)
- There are various refactoring patterns, so those who are interested should look into them.

XP (eXtreme Programming)

- A development style suitable for short-term development with a small number of people
 - Changes in the modern IT industry are dizzying (the so-called “dog year”)
 - Especially in the case of small-scale development, development must be completed in a few months at most.

Small-scale development requires agile development in a different style from large-scale development.

*Dog Year: A dog's lifespan is a fraction of that of a human, and time passes several times faster.

XP Practices

- ① Planning game
- ② Small release
- ③ Metaphor
- ④ Simple design
- ⑤ Testing
- ⑥ Refactoring
- ⑦ Pair programming
- ⑧ Collective ownership
- ⑨ Continuous integration
- ⑩ 40 hours work week
- ⑪ On-site customer
- ⑫ Coding standards

① Planning game

② Small release

□ Planning game

Coordinate plans with
customers/clients

Planning is not a one-time occurrence at the beginning but is **ongoing during development** as the situation change.

(Repeat development in a short span and adjust the plan)

□ Small release

We will **release every few months** and develop & improve the necessary functions in order by **getting feedback from customers**.

*Release: Delivering or disclosing software to customers/users.

*Feedback: Reflecting opinions and bug reports on the development site.

③ Metaphor

④ Simple design

□ Metaphor

- Metaphor means “simile” or “illustration”
- Comparing the structure and functions of the system to something and sharing the image with related parties
- Such examples are called metaphors, and sharing metaphor leads to improved development efficiency.

□ Simple design

- XP keep the design to a minimum necessary
- Eliminate extra complexity
- Don't think about future extensibility from the beginning
(No coding that may be used in the future)

⑤ Testing

- Test continuously
- Test-first
 - Create a test program first, then write the program
 - "A test program is a specification"
- Programmers should at least have unit tests (and run them every time they are rewritten)

Test-first example

- Create a convert function to convert a specified 1-byte hexadecimal number to a decimal number

Create a test program first

```
.....  
if ( convert("00") == 0 ){  
    printf("OK\n");  
}  
.....
```

```
convert("01") == 1  
convert("ff") == 255  
convert("FF") == 255  
convert("A0") == 160  
.....  
convert("000") == ERROR  
convert("fff") == ERROR  
convert("-1") == ERROR  
convert(NULL) == ERROR  
convert("xyz") == ERROR  
.....
```

⑥ Refactoring

Exercise in Week 10

- Change the program to make it better while keeping its functionality and meaning (**Refinement**)

- In specialized refactoring books, “**suspicious**” code that should be rewritten is called “**code smell**”

Example of code smell

□ "Duplicated code"

If you see similar code in two or more places, consider consolidating it into one place.

```
...  
for( i = 0; i < 10; i++ ){  
    printf("%d\n", a[i]);  
}  
...  
for( j = 0; j < 100; j++ ){  
    printf("%d\n", b[j]);  
}
```

```
print_array(int x[], int size){  
    int i;  
    for( i = 0; i < size; i++ ){  
        printf("%d\n", x[i]);  
    }  
}
```



```
...  
print_array(a, 10);  
...  
print_array(b, 100);
```

[Exercise 3] Think about refactoring

A 4-bit binary number was stored in a 4-length array bin, and a program was written to convert it to decimal

① was not good, so I tried ②,
but could it be better?

①

```
x = bin[3] * 8 + bin[2] * 4  
+ bin[1] * 2 + bin[0] * 1;
```

②

```
x = 0;  
for ( i = 0; i < 4; i++ ){  
    if ( i == 0 ) p = 1;  
    if ( i == 1 ) p = 2;  
    if ( i == 2 ) p = 4;  
    if ( i == 3 ) p = 8;  
    x += bin[i] * p;  
}
```

[Exercise 3] Think about refactoring (Sample answer)

```
x = 0;  
for ( i = 0; i < 4; i++ ){  
    if ( i == 0 ) p = 1;  
    if ( i == 1 ) p = 2;  
    if ( i == 2 ) p = 4;  
    if ( i == 3 ) p = 8;  
    x += bin[i] * p;  
}
```



```
x = 0;  
p = 1;  
for ( i = 0; i < 4; i++ ){  
    x += bin[i] * p;  
    p *= 2;  
}
```

p=1 is used as the initial value for the first time, and then double p

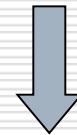
⑦ Pair programming

□ Programming in pairs

(Note: two people use one computer)

- Simple mistakes are reduced
- Even if you meet an obstacle, one person can code another part; the other person can think and research in the meantime.
- Reviews (evaluations/checks) are performed at the same time

Although it seems inefficient



As a result, it helps to maintain high code quality.

It is also important to change the partners in efficient pairs.

⑧ Collective ownership

⑨ Continuous integration

Collective ownership

- Allow all members to freely view and modify all code
- Quick improvements are possible because everyone understands the code and can modify it.

A system that centrally manages source code in one place is effective (e.g. version control system)

Continuous integration

- Do integration testing immediately after completing code creation/modification and unit testing for each pair.
- Perform integration multiple times a day at different timings.

⑩ 40 hours work week

⑪ On-site customer

40 hours work week

- Set a "target" of working hours of 8 hours a day x 5 days.
- Productivity does not increase if you are tired or work overtime.

On-site customer

- Include customer representatives on the development team or keep them nearby at all times
- In agile development , frequent meetings with customers are important

Fatigue of one person leads to the collapse of pair programming.

⑫ Coding standards

- The style of writing source code varies from person to person
- My own style of writing is difficult for others to understand.
- Set rules in your organization
 - how to name variables
 - how to write comments
 - how to put brackets
 - how to indent

```
while ( x > 0 ) {  
    ....
```

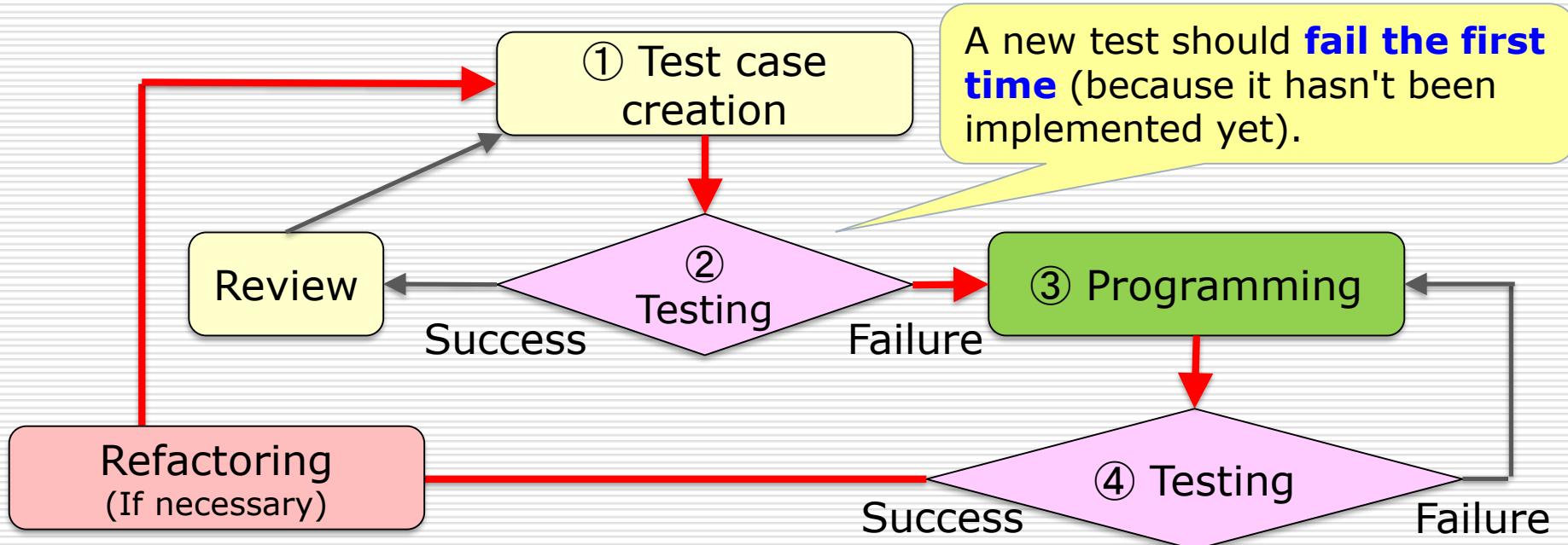
```
while( x > 0 )  
{  
    ....
```

Which
one to
choose?

(Test + Programming) Test driven development

Exercise in Week 11

A method of repeating “first create a test case, then write a program that passes it”



Features of test driven development

□ Clarify goals

- You will write a program so that it works correctly for the prepared test cases
- **A test case** plays the role of **a concrete specification**

□ Avoid overlooking bugs

- Since programming progresses while repeating the tests many times
So it is easy to notice even if bugs are introduced midway through

Summary

- Programming etiquette:
Easy to understand is important

 - Fundamentals of agile development
 - Test-first
 - Refactoring
 - Pair programming
 - Coding standards
- Even when it comes to programming, it is better to approach it as a **team play** rather than an individual play.

Homework

Answer “[9] quiz”
by this Friday 23:59

(Note: Your quiz score will be a part of your final evaluation)

ソフトウェアテスト

[12] 品質管理とメトリクス

Software Testing
[12] Software Quality Management
and Software Metrics

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Quality concept

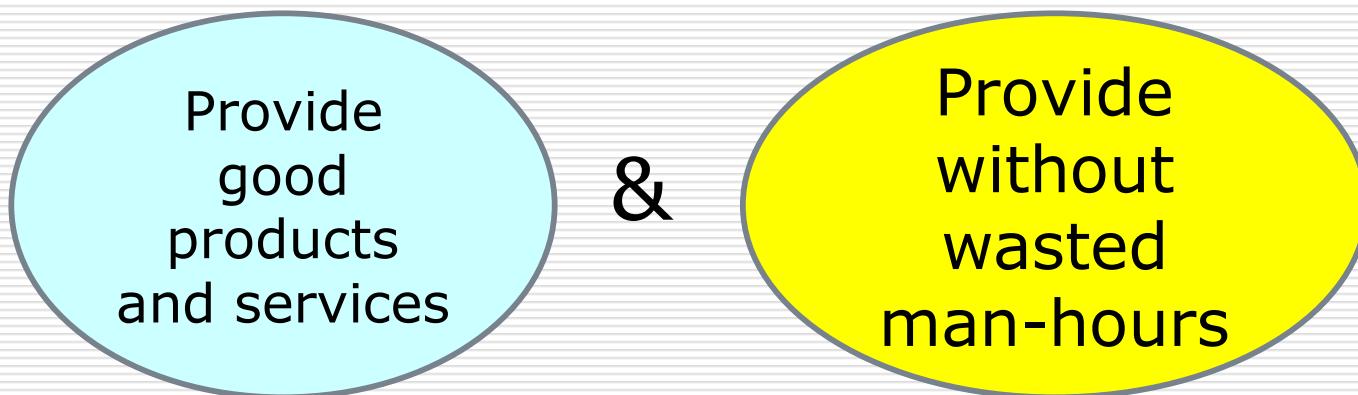
□ What is **quality**?

The extent to which the product or service meets the requirements

- Products and services have “**the things to be met**” or “**the things to be expected**:
How **satisfied** they are
- In addition, although it is not specified, it also includes items that are “**obvious**”

Quality Management

- Activities to provide products and services that *meet customer requirements*
- Activities that provide it *economically*



Example of mass production in a factory (1/2)

[Provide good products and services]

- Some defective products may be included in mass production
- Inspect products to detect and filter out defective products and ship only those that meet customer requirements

Establishing an appropriate inspection system is an important quality control activity

Example of mass production in a factory (2/2)

[Provide without wasted man-hours]

- Even if proper inspection is possible, if you **produce many defective products**, you will not be profitable.
- Monitor and improve processes** to avoid producing poor quality products

Process monitoring and improvement are also important quality control activities

If we compare this to creating a report

- Check defective products before submitting
 - System development: Making checklists and Get someone else to check
- Report creation process management
 - Cause: Creating in a hurry just before the deadline, not reading the texts and materials carefully, etc.
 - Improvement: Schedule management to start early , take notes on important things, etc.

For software

- It's not a physical entity like a factory, but the concept is the same.
- **Inspection:** Conduct sufficient testing before shipment and release, correct any defects.
- **Process monitoring and improvement:** Record, analyze, evaluate and improve various tasks in the development process

Maintenance is important as well as making

- Maintenance is the activity of maintaining software so that **it can be operated properly.**
 - Fixing of faults found after the start of operation
 - Modifications to meet changing requirements
 - Modifications to meet changing environmental
 - Continuous quality improvement
 - Failure prevention
-

Classification of maintenance

- Adaptive maintenance
 - Corrective maintenance
 - Emergency maintenance
 - Maintenance enhancement
 - Perfective maintenance
 - Preventive maintenance
-

Classification of maintenance (1/3)

Adaptive maintenance

Modifications to existing software to ensure continued use of the software product for **environmental changes**

For example, **changes in OS, libraries, hardware, etc.**

Corrective maintenance

Modifications to existing software to correct **problems that occur after the start of operation**

Bug fixes



Difficult to support legacy systems (e.g. COBOL code)

Classification of maintenance (2/3)

Emergency maintenance

A type of corrective maintenance, but **temporary modification** of existing software that is performed **unplanned** to ensure system operation

When a sudden trouble occurs, **you have to get out of the situation for now**

Maintenance enhancement

Modifying existing software to meet **new requirements** (including adding features)

Improvements such as adding **new functions** or **consolidating functions**

Classification of maintenance (3/3)

Perfective maintenance

Modifications to existing software **for the future**, although no problems have occurred

Make it easy to change, improve performance, etc

Preventive maintenance

Modifying existing software to **discover and correct problems before they become problems**

Instead of dealing with problems after they occur, try to **prevent them**

[Exercise 1] Corrective and preventive maintenance

- Both corrective maintenance and preventive maintenance involve fixing problems in software.
- Explain the difference between the two?

[Exercise 1] Corrective and preventive maintenance (Sample answer)

- Corrective maintenance is fixing problems that have occurred in the software.
- Preventive maintenance, on the other hand, is a fix before the problem becomes apparent.
- **Corrective maintenance** is to **deal with problems after it surface** and **Preventive maintenance** is to **deal with problems before it surface**

Tests required for maintenance : Regression Testing

- In maintenance work, another bug may be created at the same time as the fix is made
- It is important to **retest** not only the modified **module** but also the related modules: this is called **regression testing**.

Changing just one line of code can cause the system to crash or stop. Regression testing is extremely important.

It's helpful to keep a record of your tests

BACK TO THE TOPIC OF QUALITY CONTROL

Total Quality Control: TQC

- Efficiently implement quality control not at the individual level, but at the managers / leaders level, with the participation and cooperation of all companies and organizations

TQC will not work if the person at the top does not properly understand and practice quality control

- TQC is the norm for many Japanese companies
 - One of the reasons why Japanese products are praised for their high quality
-

Characteristics of TQC activities in Japan

1. Priority management
2. Upstream management
3. Prevention of recurrence
4. Understanding facts from data
5. Standardization
6. PDCA cycle

(1) Priority management

- Select items that are expected to have a **significant improvement** effect on problems that occur on-site.
 - Focused management
 - Prioritization
 - Main evaluation criteria: **quality, cost, delivery**(**OCD: Quality, Cost, Delivery**)

(2) Upstream management

- Identify and resolve the source (most upstream) of the problem
 - No impact on subsequent processes
-
- Root cause analysis is essential
 - In some cases, it may be necessary to review the system or make new investments

(3) Prevention of recurrence

- Learn lessons from past problems
- Take fundamental measures to prevent recurrence
 - Need to be objectively analyzed why the problem happened
 - Information sharing and checking system are also necessary

(4) Understanding facts from data

- A concept commonly called **visualization**
- **Digitize** what is actually happening, and process problems by judging them **logically, objectively, statistically**

- The contrast is KKD (experience, intuition, courage)
"I'm not sure, but this is working", "I think it's probably fine", "It'll work out"

(5) Standardization

- Set a **standard** for the work content and take advantage of it.

- Making so-called **manuals, rules**
 - By creating a successful “pattern” and following it → **Prevent unnecessary errors**

(6) PDCA cycle

- Driving TQC as a four-phase cycle
 - ① Make a plan (**P**lan)
 - ② Execute according to the plan (**D**o)
 - ③ Review, evaluate, and confirm the results of execution (**C**heck)
 - ④ Keep it good if it is good (standardization)
/ If not, conduct adjust or rework (**A**ction)

Examples of data analysis:

Pareto chart (1/4)

- **Pareto principle:** “Most of the income comes from a few people”
 - Originally about the economy, it was published in 1896 by Italian economist V.F.D. Pareto.
 - Also called **80 : 20 rule**
80% of product sales come from the sale of 20% of products.
80% of the company's profits come from the work of 20% of its employees.

Examples of data analysis:

Pareto chart (2/4)

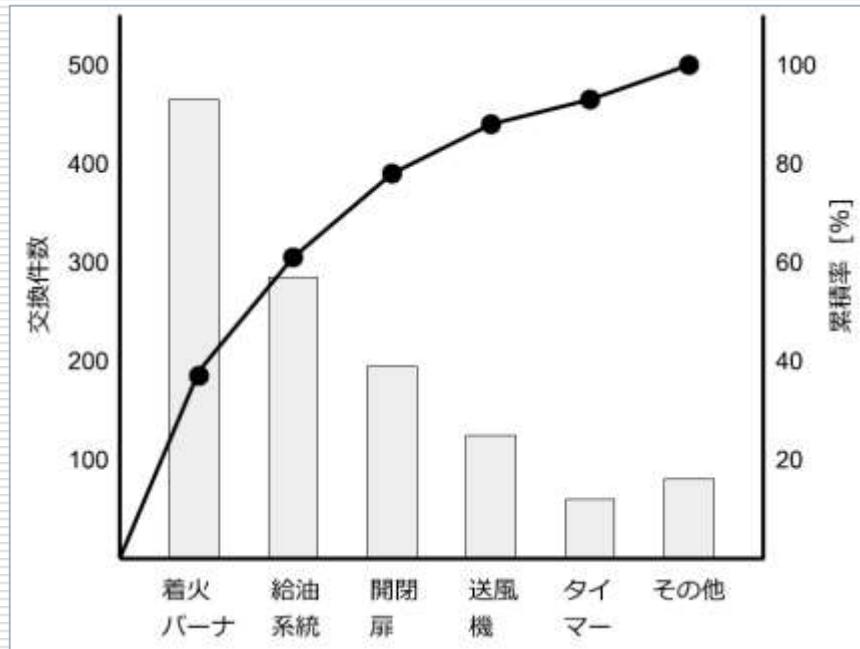
- The same thing is said for quality control (Example)
 - 80% of failures are caused by 20% of parts
 - 80% of bugs are found in 20% of modules

 - The numbers of 80% and 20% are not accurate, but they mean that there is such a tendency to concentrate
-

Examples of data analysis:

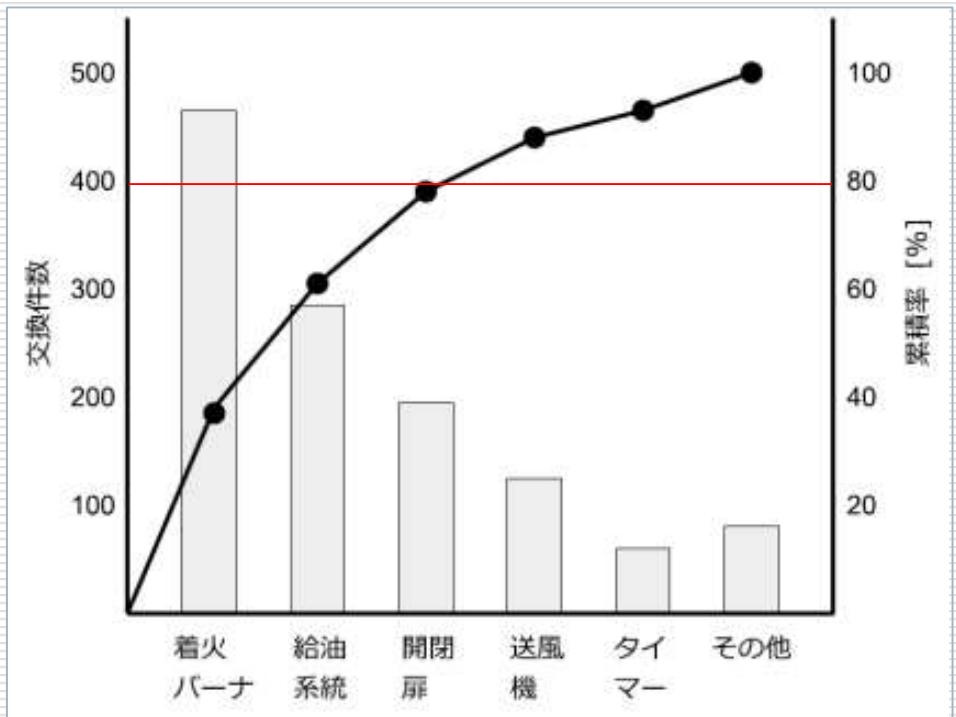
Pareto chart (3/4)

- Create a bar graph with the items on the horizontal axis in descending order of the number of cases, and also create a line graph



This will allow us to see **where it is most efficient to improve.**

Examples of data analysis: Pareto chart (4/4)



Top 3 parts:

- Ignition burner
- Oiling system
- Open/close door

Account for about 80% of the total, so we can see that these quality improvements are important

Examples of quality control

[Make a check sheet]

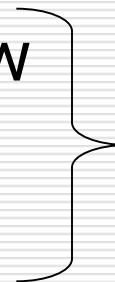
- Purpose: to prevent errors and oversights in work, Review requirements and standards
- Creation point:
 - List important (high priority) things that cannot be ignored
 - List common mistakes

Software quality assurance

□ Review

Incorporating evaluation into development activities

- Requirements specification review
- Design document review
- Source code review



Have it checked by an expert/advanced person in the field.

□ Test

We will focus on the criteria such as coverage rate

(Example) Quality control in programming

- Be careful not to make the program too long
- Be careful not to make the program too complicated
- Make sure that other people can understand the program

(Example 1) Program length

- Count the number of lines in the function you wrote to see how long it is
- However, do not count comments or blank lines.
- If you have a long function, consider whether it should be split up
 - Usually, a function is no longer than a few dozen lines.

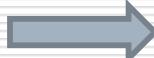
(Example 2) Program complexity

- The concept of "complexity" cannot be measured by a single standard, but to simplify it:
- Count the following items in one function
 - Number of conditional branches (if,while,for,etc)
 - Number of variables
 - Number of calls to other functions
 - Maximum nesting depth

(Example 3) Can other people understand it?

- Are the function and variable names easy for others to understand?
- Are the indentations correct?

```
for ( i = 0; i < 100; i++ ){  
    min = i;  
    for ( j = i+1; j < 100; j++ ){  
        if ( a[min] > a[j] ){ min = j;  
    }  
}
```



```
for ( i = 0; i < 100; i++ ){  
    min = i;  
    for ( j = i+1; j < 100; j++ ){  
        if ( a[min] > a[j] ){  
            min = j;  
        }  
    }  
}
```

- Did you have someone else check (review) it?

Software Metrics

- Software metrics are quantitative criteria for measuring software quality characteristics and feature data, (including the definition of the measurement method).
- Note that the purpose of software metrics (simply called metrics) is to measure characteristics, not to give evaluation values.

Typical Metrics (1)

□ Lines Of Code (LOC)

- The number of lines of code
- Measure the length (scale) of a program
- Blank lines and comment lines are generally not counted.
- Used to express the scale of a software system in units of **KLOC** (=1000LOC) or to express bug density as the average number of bugs per KLOC

Typical Metrics (2)

□ McCabe (**cyclomatic number**)

- Measure the complexity of the control flow in a flowchart
- Represent the complexity of a program
- **It is the number of independent paths in a flowchart, which is equal to the number of conditional branches + 1**
 - A cyclomatic number of 1 corresponds to a program with no branches, and this is defined as the program with the simplest structure

Cyclomatic number and testing

- Programs with large cyclomatic numbers have many execution paths
- Therefore, in order to increase the coverage rate of white box testing, many test cases are required.
- In other words, these programs are often difficult to test.

Download lecture12-materials.zip
(sample1202.c is in it)

[Exercise 2]

- For the C program **sample1202.c** in Teams, for each function

Answer:

- LOC value
- Cyclomatic number

[Exercise 2] (Sample answer)

- **find_max**
 - LOC = 10, cyclomatic number = 3
- **selection_sort**
 - LOC = 9, cyclomatic number = 2
- **main**
 - LOC = 21, cyclomatic number = 4

Data Analysis Exercises using R

Download lecture12-materials.zip
(contains data file data12.csv, R
script Rscript12.R)

Take a look at the real data

□ Raw data

This time, only a part of the data is used

NASA IV&V Facility Metrics Data Program
Project CM1

<https://zenodo.org/>

- 20 KLOC of scientific measurement software written in C
- Number of modules: 505
- 40 metric data types (including whether bugs were detected in each module)

[R] Loading CSV data

- We have **data12.csv** available here
- Load this content as a dataframe named **cm1**

```
cm1 = read.csv( file.choose() )
```

[R] CSV data loading and results

> **cm1**

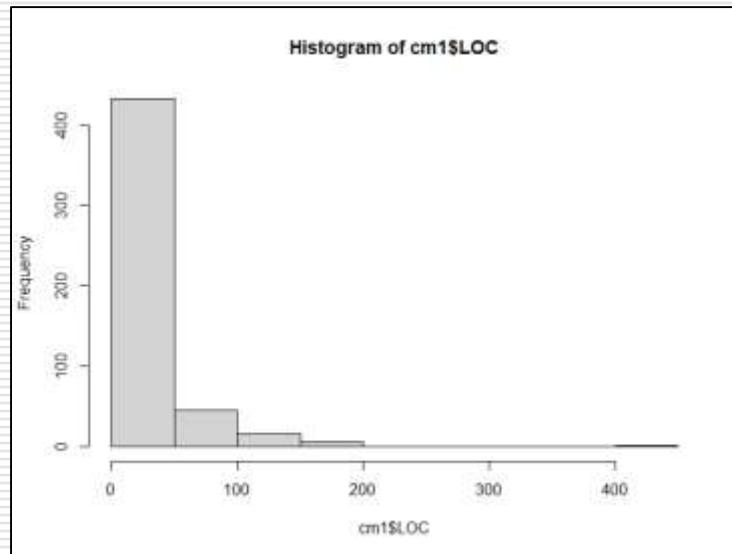
	MODULE	LOC	CC	BUG
1	24972	24	5	0
2	24973	31	4	1
3	24974	29	5	1
4	24975	7	2	0
5	24976	12	2	0
6	24977	106	13	0
7	24978	15	3	0
.
.
.

Column name	meaning
MODULE	Module ID
LOC	Lines Of Code (scale)
CC	Cyclomatic number (complexity)
BUG	Presence of bugs *Yes: 1, None: 0

[R] LOC Histogram and Kernel Density Estimation

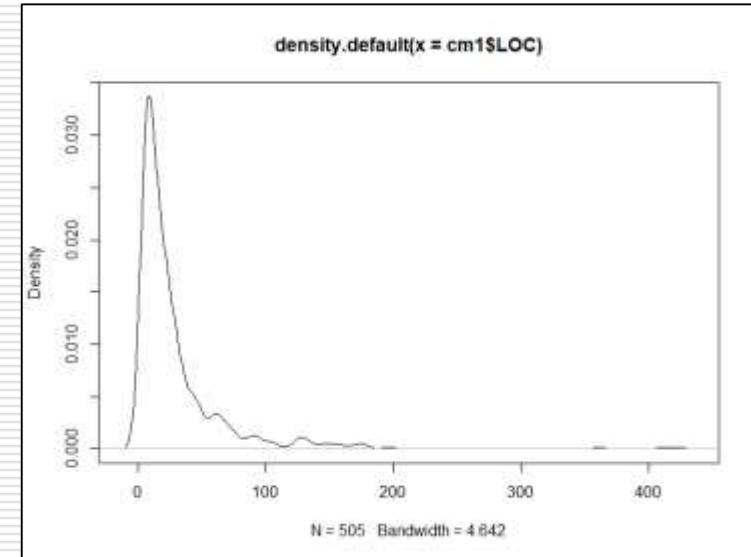
Histogram

```
hist(cm1$LOC)
```



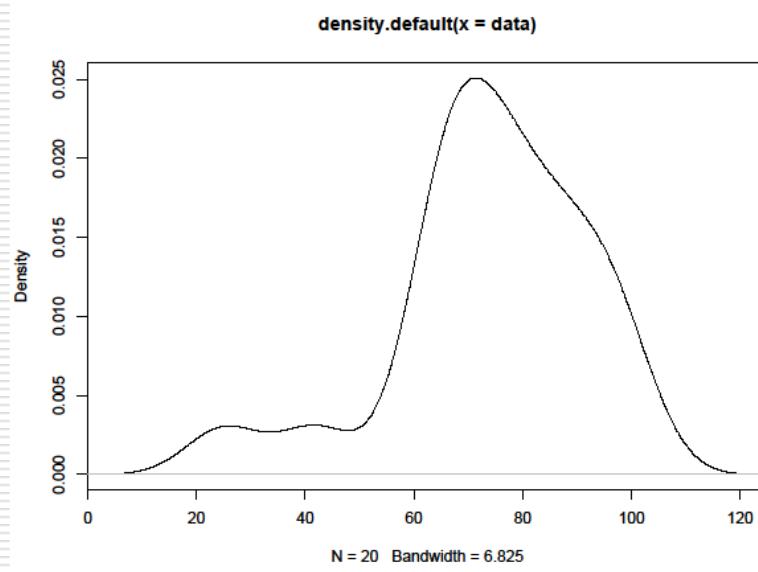
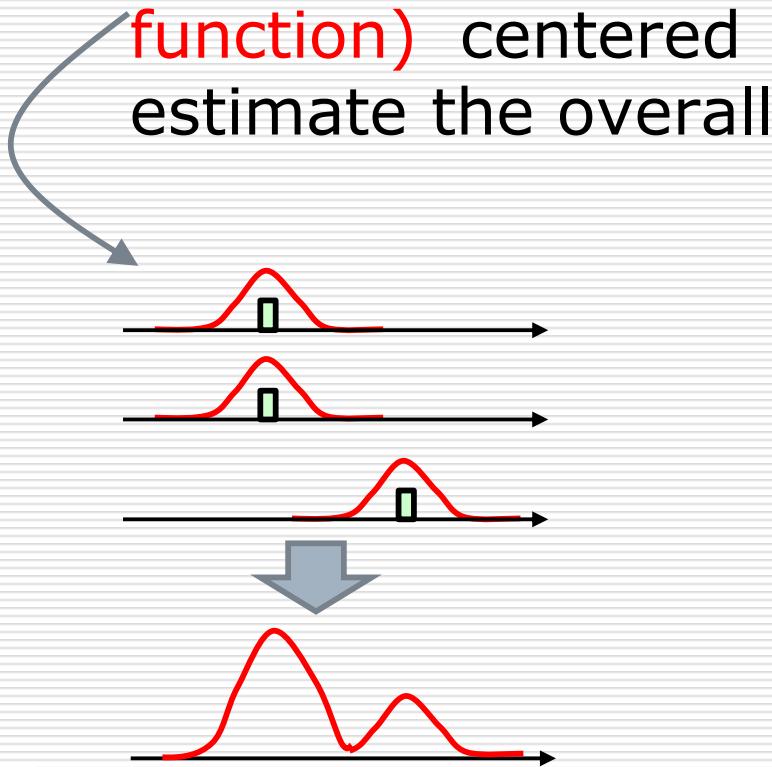
Kernel density estimation

```
plot(density(cm1$LOC))
```



(Reference) Kernel density estimation

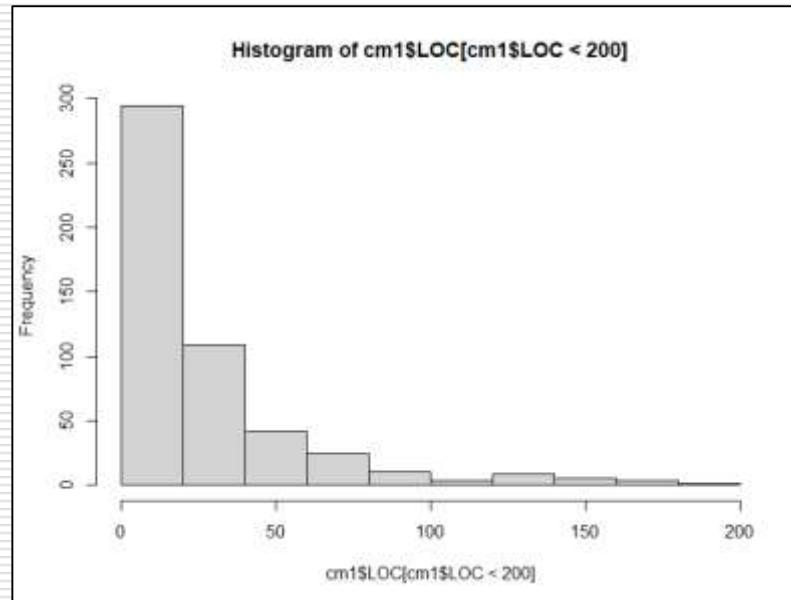
- For each piece of data, consider a probability density function (more precisely, a kernel function) centered on it, and combine them to estimate the overall probability density.



[R] LOC histogram (continued)

- If you redraw it only to $\text{LOC} < 200$

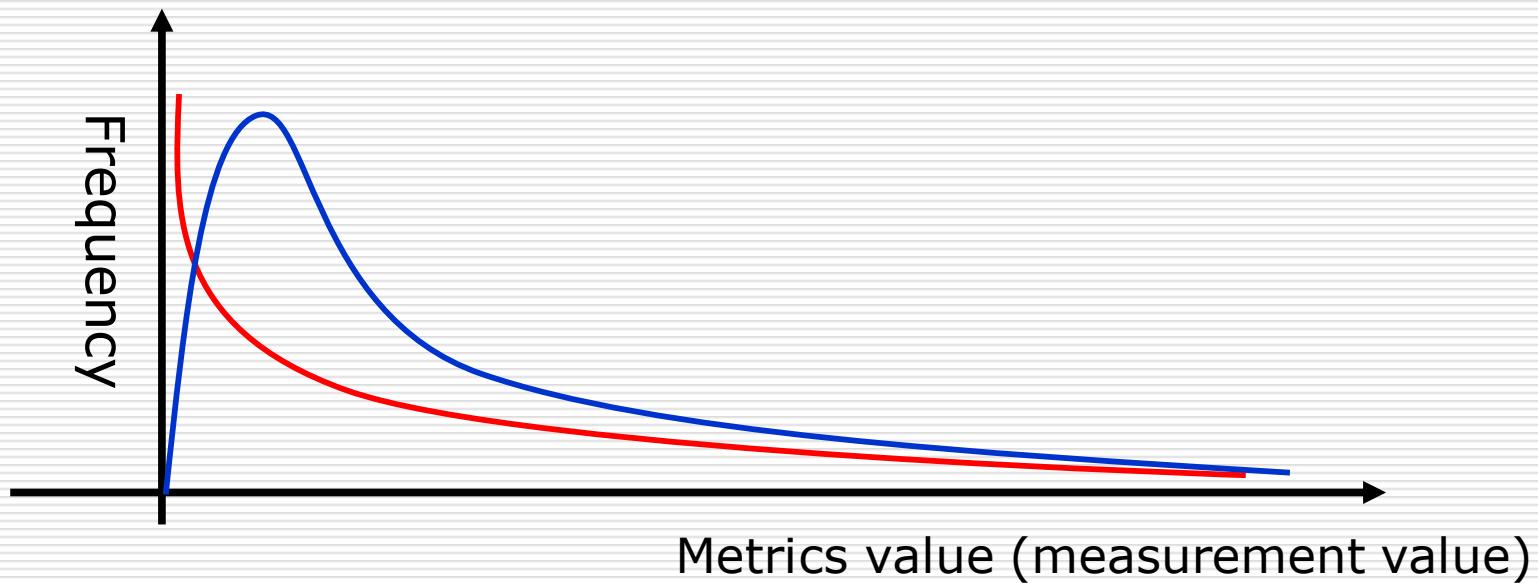
```
hist(cm1$LOC[cm1$LOC < 200])
```



As can be predicted from the kernel density estimation on the previous page, it can be seen that there tends to be a **very large number of ones with less than 200 lines**.

Common patterns in the distribution of metric values

- "Right-skewed" distribution
 - * Also known as a "**right -tailed**" distribution

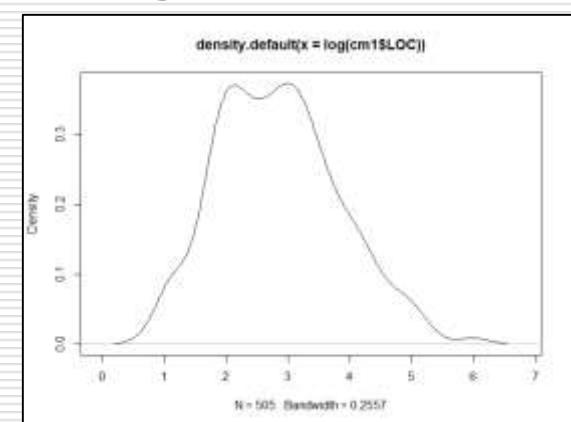
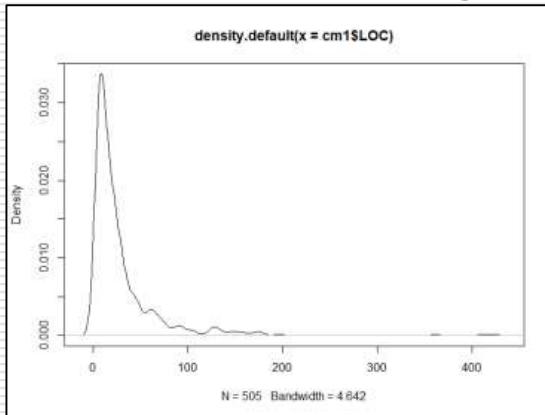


Logarithmic transformation:

$$x \rightarrow \log x$$

* If x contains 0, add 1 and use $\log(x+1)$

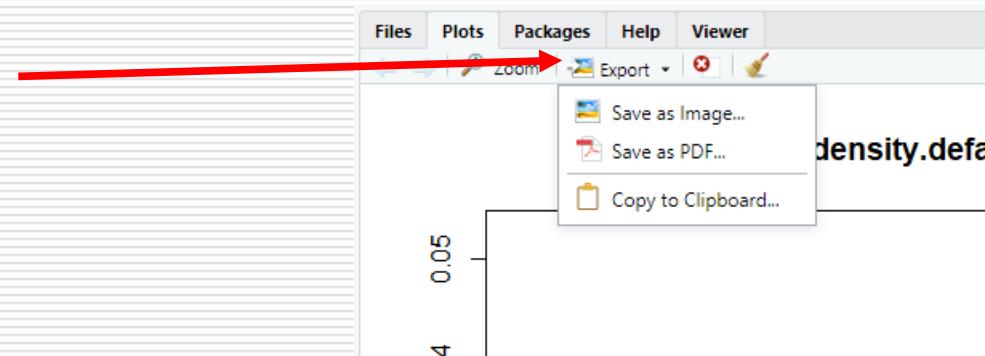
- Right-skewed distributions may be easier to analyze by applying a logarithmic transformation.
- Ex: Many data are concentrated in a small range before transformation, but can be distributed by taking $\log(\text{LOC})$.



[Exercise 3] (Homework)

- For cyclomatic numbers (column names: CC), create a histogram and perform kernel density estimation in the same way as for LOC.
- However, in both cases, perform a **logarithmic transformation**.

"Save as PDF" appears in the "Export" menu in the "Plots" area. Please use this to save the graph as a PDF and submit it

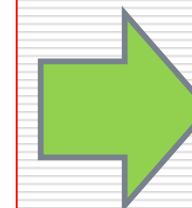


[R] Mean and median

Both distributions are right-skewed, **but the mean is greater than the median.**

□ LOC's Mean and Median

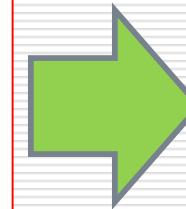
```
mean(cm1$LOC)  
median(cm1$LOC)
```



```
29.6099  
16
```

□ Cyclomatic number's Mean and Median

```
mean(cm1$CC)  
median(cm1$CC)
```



```
5.182178  
3
```

[R] See the difference between bugs and non-bugs

- To extract only the LOC values of modules with bugs

```
cm1$LOC[cm1$BUG==1]
```

Extract only the LOC values when the value of the BUG column is 1
※ cm1\$BUG==1 is a vector of TRUE and FALSE

- Similarly, to extract only non-bugs modules, write cm1\$BUG==0

[R] See the difference between bugs and non-bugs (LOC)

□ Mean and median LOC values in buggy

```
mean(cm1$LOC[ cm1$BUG==1 ])
```

62.77083

```
median(cm1$LOC[ cm1$BUG==1 ])
```

42.5

There is a clear difference

```
mean(cm1$LOC[ cm1$BUG==0 ])
```

26.12691

```
median(cm1$LOC[ cm1$BUG==0 ])
```

15

[Exercise 4] (Homework)

- Focusing on the cyclomatic numbers
in buggy and non-buggy modules,
Find its mean and median

[12] Exercise-4

Deadline: Tomorrow at 13:00

Summary

- Type of maintenance:
 - adaptive / corrective / emergency / enhancement / perfect / preventive
 - **Regression testing** is also important
- Quality control
 - Checklists are a method that even individuals can easily put into practice
 - The importance of **review and testing**
- Software metrics
 - **Quantifying the characteristics of software**
 - It can also be used for evaluation and is closely related to testing

Homework

Submit the PDF file for “[12] Exercise-3” and answer “[12] Exercise-4”

Finish “[12] Exercise-3,Exercise-4” by tomorrow 13:00 (1pm)

(Notes: Your quiz score will be a part of your final project evaluation)

ソフトウェアテスト

[13] バグ予測とテスト計画

Software Testing
[13] Bug Prediction and Test Plan

あまん ひろひさ
阿萬 裕久 (AMAN Hirohisa)
aman@ehime-u.ac.jp

Fault-Prone module analysis

- **Bug** is often called **fault** in software engineering
 - A programmer's **error** introduces a **fault** into the program. As a result, **faults, failures, defects** occur during program execution.
- Find features of modules **likely to be faulted** by metrics (**Fault-Prone**)

Significance of Fault-Prone module analysis

- Activities required for software quality assurance:
 - **Testing**
 - **Reviewing**
- Help plan these activities:
Identify where the fault is likely to be

*Intuitively, The idea is "Where is the intersection where accidents are likely to occur?"

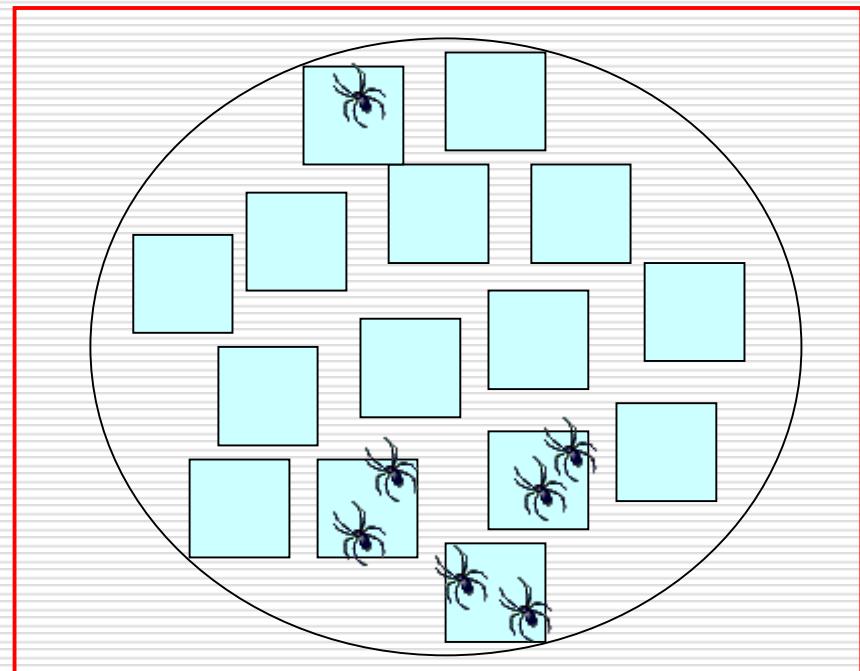
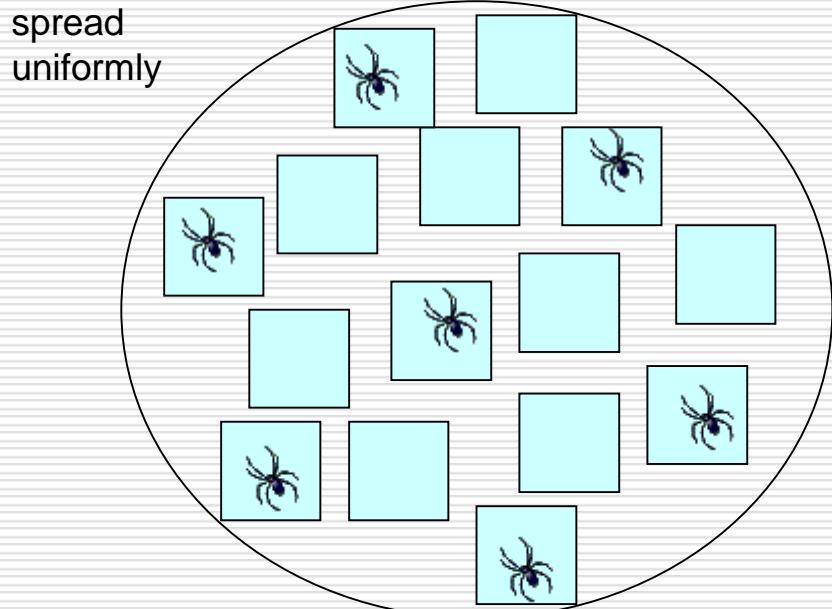
Focus on the distribution of faults (bugs)

- Faults (bugs) are **made** incorrectly **by humans**
 - Even if you focus only on each bug, **the cause is different**. Therefore, it is difficult to analyze it by itself.
 - **A statistical approach** is important to capture overall trends
- Let's look at **how faults (bugs) have been distributed so far.**

(Reappears) Pareto principle

- About 80% of bugs are present in about 20% of modules

concentrated in fewer parts



Launch RStudio

- Let's analyze the data using RStudio

- Download [Rscript13.R](#) and [data13.csv](#) from Teams

- Open [Rscript13.R](#) in RStudio

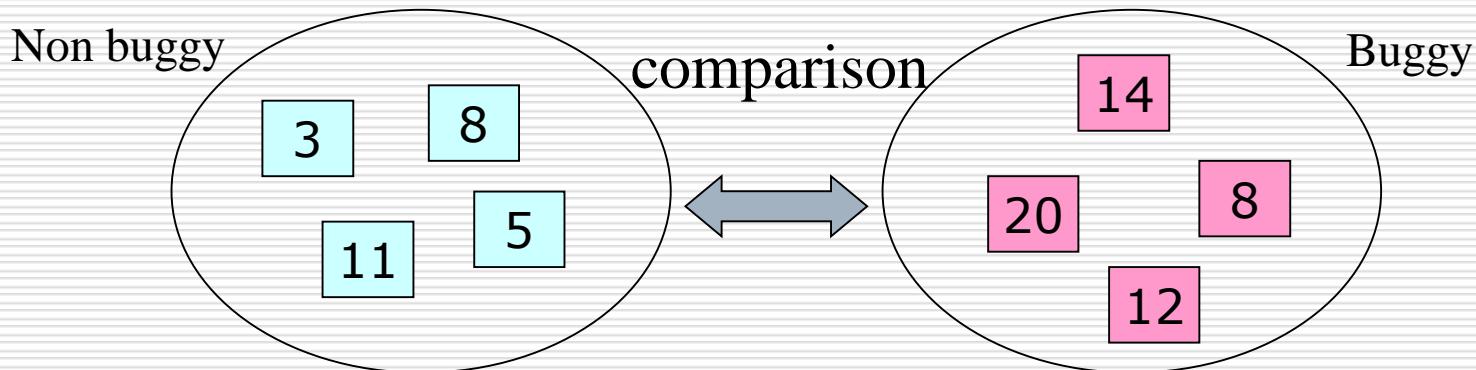
Use the same metric data as last time

- First, use the same NASA-published data as last time
 - Data file name **data13.csv**
 - Load this content as a dataframe named data

```
data = read.csv( file.choose() )
```

Start with a simple analysis

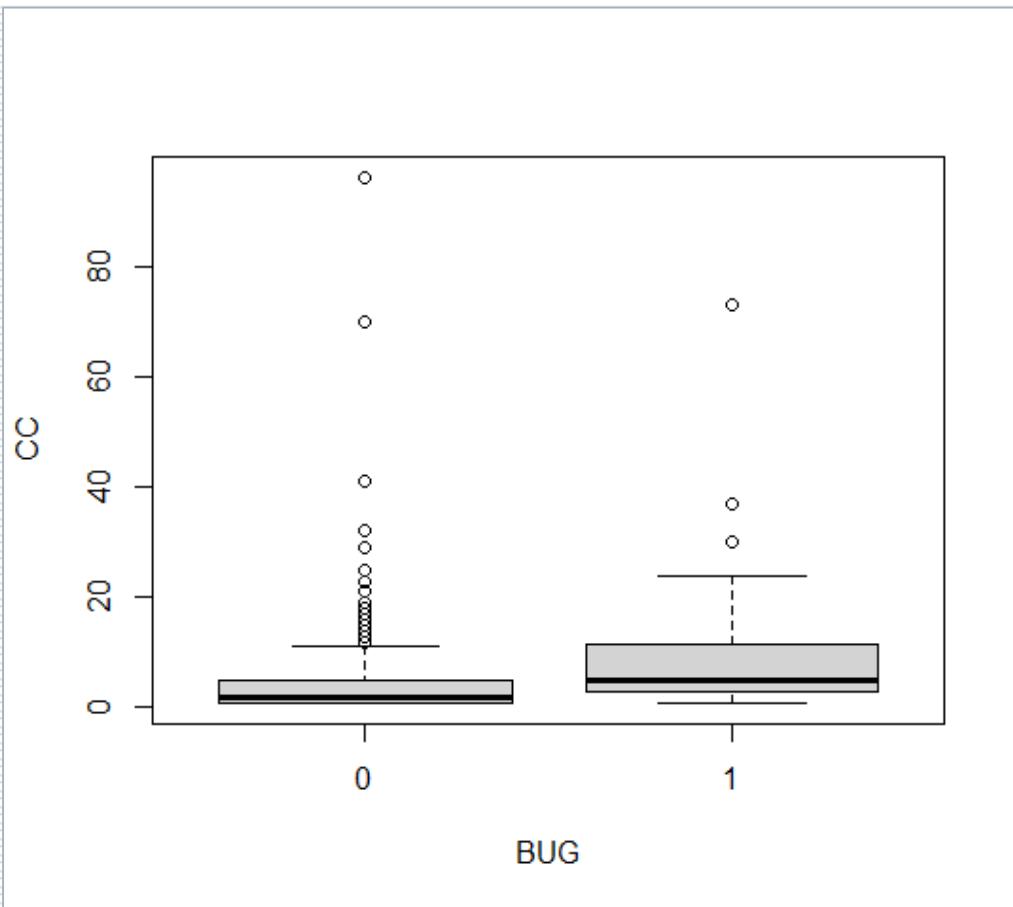
- The module set is divided into two types, “**non buggy**” and “**buggy**”,
 - See the **difference in cyclomatic numbers**
 - Find the **threshold** for bug prediction (of cyclomatic number)



Box plot

```
boxplot(CC~BUG, data=data)
```

Column name Column name DataFrame



Although it is rough, you can see the difference in distribution

Vertical axis: cyclomatic number

Horizontal axis: presence or absence of bugs
(0 = none, 1 = yes)

Compare by summary statistics

- Separate the cyclomatic numbers with **non buggy** and **buggy** and check with the summary function

```
cc0 = data$CC[data$BUG==0]
```

```
cc1 = data$CC[data$BUG==1]
```

```
summary(cc0)
```

```
summary(cc1)
```

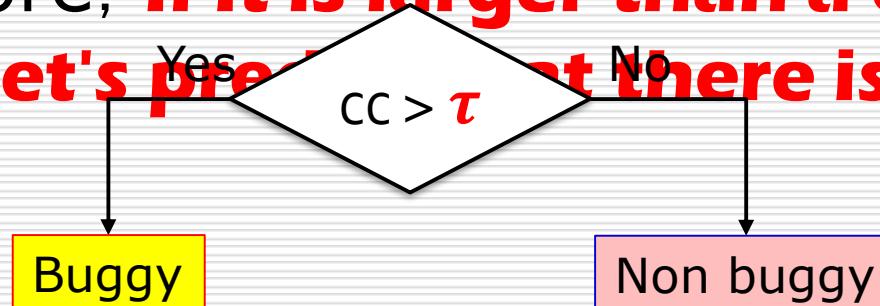


Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	1.000	2.000	4.705	5.000	96.000

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	3.000	5.000	9.729	11.250	73.000

Consider Metric Thresholds

- Looking at data, **the cyclomatic number seems to be related to the presence or absence of bugs**
 - **A larger cyclomatic number is more suspicious**
- Therefore, **if it is larger than a certain value, let's presume there is a bug.**



Split dataset

- Consider a simple predictive model
 - **Training data** is needed to build this **predictive model**
 - Furthermore, to **evaluate** the capabilities of the **predictive model**, **test data** is also required
- So let's split the data into two and prepare them.

Random Distribution

- First shuffle row number $1 \sim \text{nrow}(\text{data})$:
get a list of shuffled row numbers with the
following function sample:

```
set.seed(1234)
idx = sample(nrow(data))
```

Split data after shuffling

- training data (`d.train`): first 300

```
d.train = data[idx[1:300], ]
```

- test data (`d.test`): remaining 205

```
d.test = data[idx[301:nrow(data)], ]
```

(Ex) When the threshold of the cyclomatic number is set to 10(1/6)

- As a simple example, **If the cyclomatic number(CC) exceeds 10**, let's predict that there is a bug
- Check the training data for bugs in this module

```
d.train$BUG[d.train$CC>10]
```



```
[1] 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0
```

(Ex) When the threshold of the cyclomatic number is set to 10(2/6)

- The result of "predicting that there is a bug" as the result

```
result = d.train$BUG[d.train$CC>10]
```

```
[1] 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0
```

length(result)

sum(result)



Predicted number

```
[1] 41
```



Number of correct answers in the prediction

```
[1] 10
```

(Ex) When the threshold of the cyclomatic number is set to 10(3/6)

- If you organize the results

`length(result)`

`sum(result)`

Predicted number

[1] 41

[1] 10

Number of correct answers in the prediction

Results of predictions using training data		Actual		total
		Buggy	Non Buggy	
Prediction	Buggy	10	31	41
	Non Buggy			
total				

(Ex) When the threshold of the cyclomatic number is set to 10(4/6)

- The actual number of bugs was

```
sum(d.train$BUG)
```

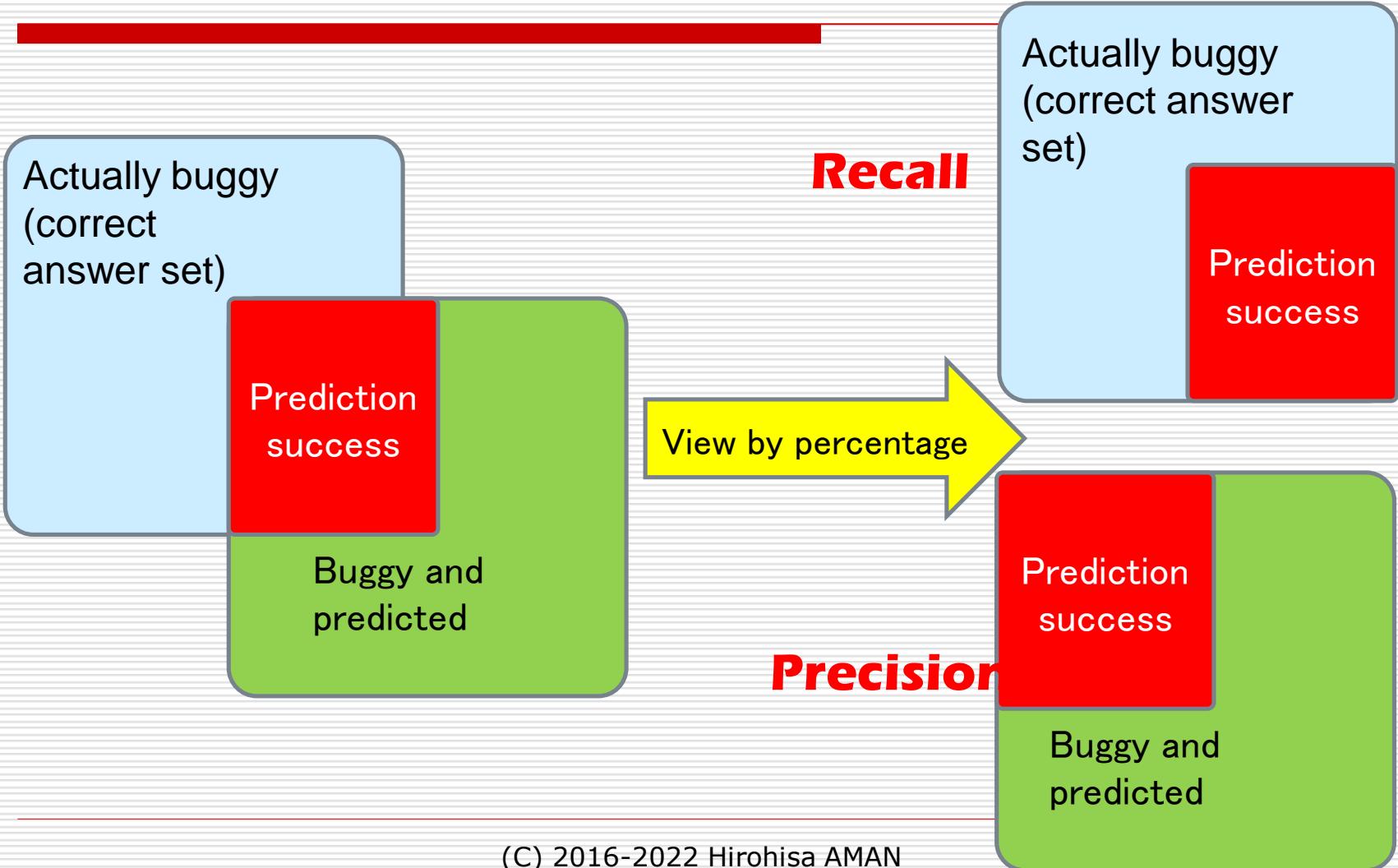


Number of actual bugs

```
[1] 30
```

Results of predictions using training data		Actual		total
		Buggy	Non Buggy	
Prediction	Buggy	10	31	41
	Non Buggy	20		
total		30		

Recall and Precision



(Ex) When the threshold of the cyclomatic number is set to 10(5/6)

Đoán đúng 10 bug

Tổng dự đoán 41 bug

Results of predictions using training data		Actual		total
		Buggy	Non Buggy	
Prediction	Buggy	10		41
	Non Buggy			
total		30		

Tổng số bug thực tế 30 bug

$$Recall = \frac{10}{30} \cong 0.333$$

Recall = Đoán đúng bug / Tổng số bug thực tế

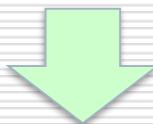
$$Precision = \frac{10}{41} \cong 0.244$$

Precision = Đoán đúng bug / Tổng dự đoán

F-score

□ **Harmonic mean of recall and precision**

$$\frac{1}{F-score} = \frac{\frac{1}{recall} + \frac{1}{precision}}{2}$$



$$\begin{aligned} F-score &= \frac{1}{\frac{1}{2}\left(\frac{1}{recall} + \frac{1}{precision}\right)} \\ &= \frac{2 \cdot recall \cdot precision}{recall + precision} \end{aligned}$$

(Reference)

Example of harmonic mean

Elementary school math problems

- I drove back and forth between my house and the station. The travel speed was 30km/h for the way and 50km/h for the return. What is the average travel speed?

$$\frac{\text{Distance traveled}}{\text{Time to go}} = 30 \quad \rightarrow \quad \text{Time to go} = \frac{\text{Distance traveled}}{30}$$

$$\frac{\text{Distance traveled}}{\text{Time to return}} = 50 \quad \rightarrow \quad \text{Time to return} = \frac{\text{Distance traveled}}{50}$$

$$\text{Average} = \frac{2 \cdot \text{Distance traveled}}{\text{Time to go} + \text{Time to return}} = \frac{1}{\frac{1}{30} + \frac{1}{50}} = 37.5$$

(Ex) When the threshold of the cyclomatic number is set to 10(6/6)

$$Recall = \frac{10}{30} \cong 0.333$$

$$Precision = \frac{10}{41} \cong 0.244$$

- The F-score in this case is

$$F-score = \frac{2 \cdot \frac{10}{30} \cdot \frac{10}{41}}{\frac{10}{30} + \frac{10}{41}} \cong 0.282$$

[Exercise 1]

[13] Exercise-1
(by tomorrow, 1pm)

- Use **LOC** instead of cyclomatic number (**CC**) as a metric

- For the training data, answer the recall, precision, and F-score when the **LOC** threshold is 30.

p15 -- p23

Function for calculating the F-score(1/5)

- Create a function to calculate the F-score
- First, calculate the **Recall**

$$\text{recall} = \frac{\text{Number of correct answers predicted as buggy}}{\text{Number of actual bugs}}$$

```
result = d.train$BUG[d.train$CC>10]
recall = sum(result)/sum(d.train$BUG)
```

Function for calculating the F-score(2/5)

- Next, the **precision** is

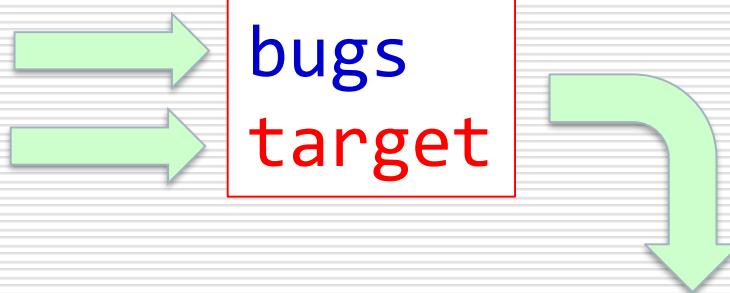
$$\text{precision} = \frac{\text{Number of correct answers predicted as buggy}}{\text{Number of bugs predicted to be}}$$

```
result = d.train$BUG[d.train$CC>10]
recall = sum(result)/sum(d.train$BUG)
precision = sum(result)/sum(d.train$CC>10)
```

Function for calculating the F-score(3/5)

- For simplicity, we will replace it with the following:

```
d.train$BUG  
d.train$CC>10
```



```
result = bug[target]  
recall = sum(result)/sum(bug)  
precision = sum(result)/sum(target)
```

Function for calculating the F-score(4/5)

$$F-score = \frac{2 \cdot recall \cdot precision}{recall + precision}$$

- The F-score is obtained as follows

```
result = bug[target]
recall = sum(result)/sum(bug)
precision = sum(result)/sum(target)
2*recall*precision/(recall + precision)
```

Function for calculating the F-score(5/5)

- Using the symbol replaced earlier as an argument, defining a function that can be used in R is as follows

```
f.value = function(bug, target){  
  result = bug[target]  
  recall = sum(result)/sum(bug)  
  precision = sum(result)/sum(target)  
  2*recall*precision/(recall + precision)  
}
```

Use the function you created

- If you call it like this, it will calculate the F-score.

The diagram illustrates the use of a function to calculate an F-score. A red box contains the R code: `f.value(d.train$BUG, d.train$CC>10)`. Two green speech bubbles point to the arguments: one points to `d.train$BUG` with the text "Presence of bugs", and another points to `d.train$CC>10` with the text "Predict by metric value". A large green arrow points downwards from the code box to a white box containing the output: `[1] 0.2816901`.

```
f.value(d.train$BUG, d.train$CC>10)
```

Presence of bugs

Predict by metric value

[1] 0.2816901

Verification with test data

- Not training data (d.train)
Try to predict test data (d.test)

- Evaluate by F-score

```
f.value(d.test$BUG, d.test$CC>10)
```



```
[1] 0.2051282
```

Because it is judged only by the cyclomatic numbers, the performance is low ...



10-minutes rest break

Test prioritization

- Consider testing modules that may have bugs first
- It's not perfect, but you can prioritize using metric values

Method of testing in descending order of metric values

Test in descending order of cyclomatic numbers

- Arrange modules of test data (**d.test**) in descending order of cyclomatic number (**d.test\$CC**)

```
order(d.test$CC, decreasing=T)
```

- Record the presence or absence of bugs in that order

```
found.bugs =  
d.test$BUG[order(d.test$CC, decreasing=T)]
```

Accumulate bug finds

- The presence or absence of bugs (**1** or **0**) is recorded in order in the variable found.bugs

```
0 0 0 0 1 0 1 0 0 0 0 0 .....
```

- By accumulating (adding) this, Find out how many bugs have been found so far

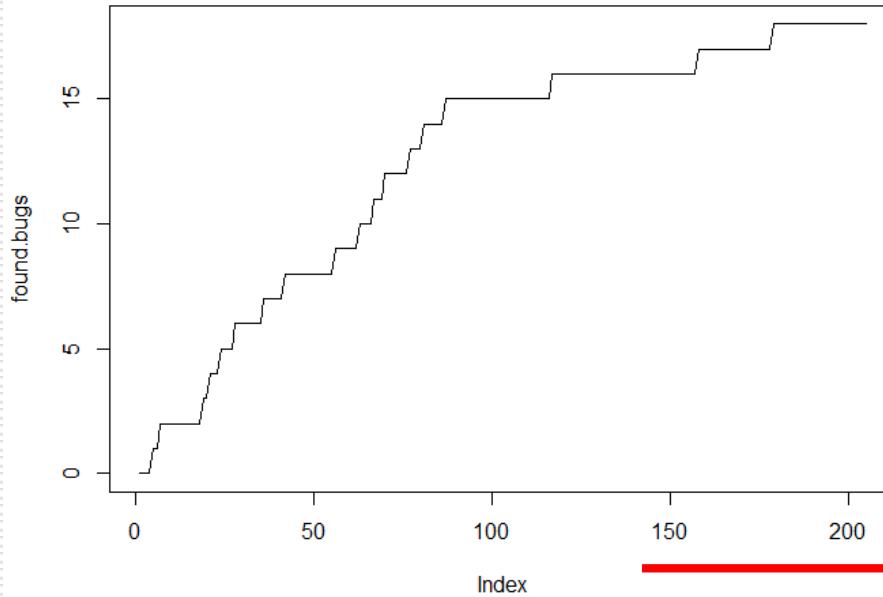
```
for ( i in 2:length(found.bugs) ){
  found.bugs[i] = found.bugs[i-1] + found.bugs[i]
}
```

Graph cumulative bug finds

- Graph the accumulated found.bugs

```
plot(found.bugs, type='l')
```

Total
number
of bugs
found



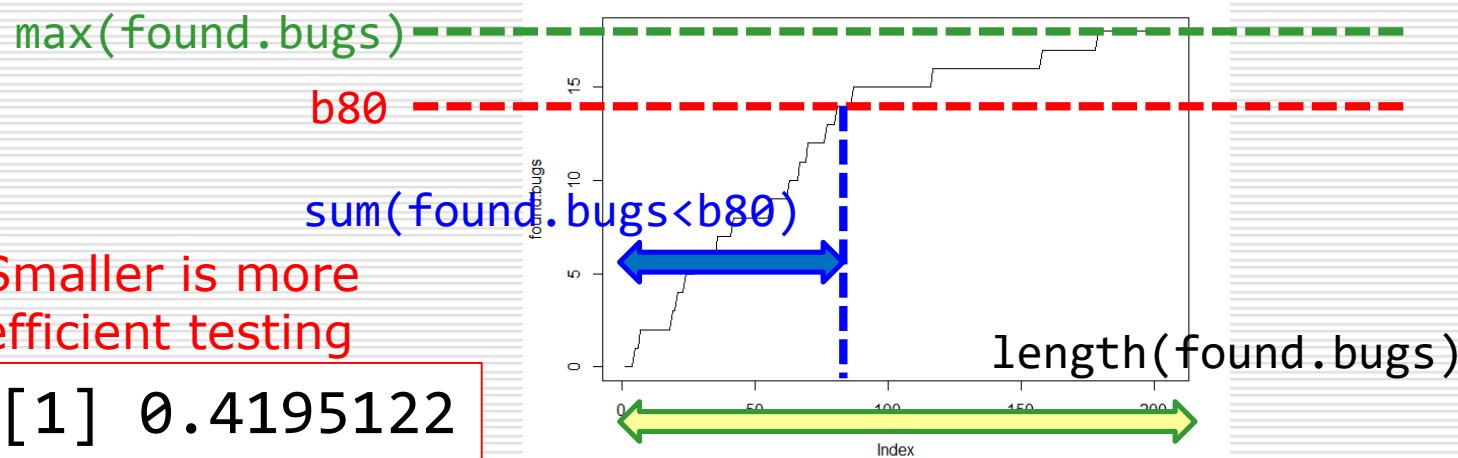
Number of
modules
tested

How quickly did you find 80% of bugs?

- How few tests were able to find 80% of the total bugs?

$b80 = \max(\text{found.bugs}) * 0.8$

$\sum(\text{found.bugs} < b80) / \text{length}(\text{found.bugs})$



[Exercise 2]

[13] Exercise-2
(by tomorrow, 1pm)

- Suppose you use **LOC** as a metric and test in descending order.
For test data, what percentage of tests are needed to find 80% of bugs?
(For cyclomatic number, it was 0.4195122)
- Also answer when you use **CALL** as a metric

p34 -- p37

Forecasting with multiple metrics

- Isn't it possible to make more accurate predictions by using **multiple metrics** than by predicting bugs with only a single metric?
- Is it possible to use the following **linear multiple regression model** with each metric as an explanatory variable (prediction material)?

$$y = a_1x_1 + a_2x_2 + a_3x_3 + b$$

CC

CALL

LOC

(where, a_1, a_2, a_3, b are all constants)

There is one problem...

- If this continues, the domain of the objective variable **y will be $(-\infty, +\infty)$**

$$y = a_1x_1 + a_2x_2 + a_3x_3 + b$$

- What we want to predict is the possibility of the presence or absence of bugs, and if we put this as **p** (as with probability), **the value range of p must be [0, 1]**.

Something appropriate conversion
needs to be done between y and p

Logit Transformation

- Replacing p with $\log \frac{p}{1-p}$ is called a **logit transformation** (where $0 < p < 1$)

- $p = 0.5$: $\log \frac{p}{1-p} = \log \frac{0.5}{1-0.5} = \log 1 = 0$
- $p \cong 1$: $\log \frac{p}{1-p} \cong \log \frac{1}{0} = \log \infty = +\infty$
- $p \cong 0$: $\log \frac{p}{1-p} \cong \log \frac{0}{1} = \log 0 = -\infty$

When the probability p is 50%, it becomes 0, above it is positive, and below it is negative. Moreover, the value range is $(-\infty, +\infty)$.

Transform the objective variable y into a logit transformation of p

$$y = a_1x_1 + a_2x_2 + a_3x_3 + b$$



$$\log \frac{p}{1-p} = a_1x_1 + a_2x_2 + a_3x_3 + b$$

- This is called a **logistic regression**

model.

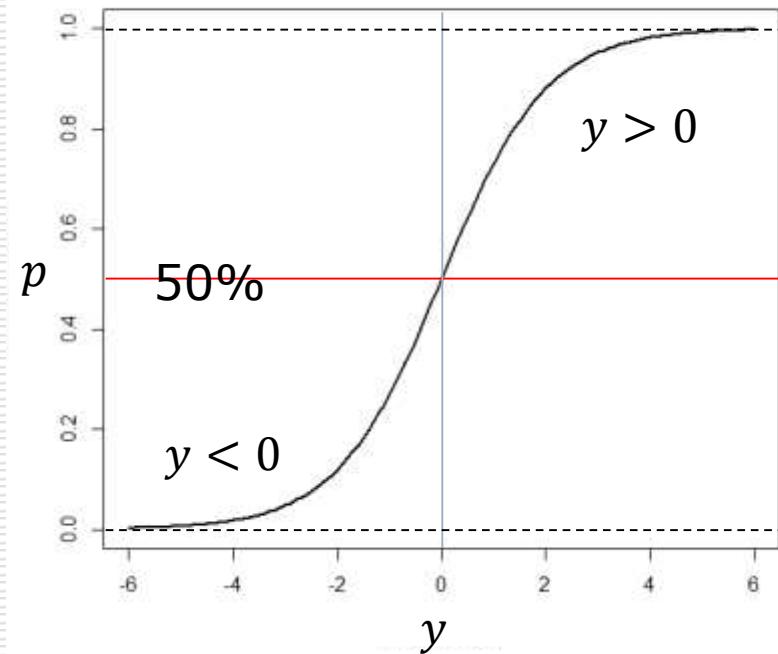
Alternative representations of logistic regression models

- Transform the previous formula so that the left side is p

$$p = \frac{1}{1 + e^{-y}}$$

$$y = a_1x_1 + a_2x_2 + a_3x_3 + b$$

Same as the sigmoid function often used in neural network activation functions



Create a logistic regression model in R

□ Using the **glm** function

```
model = glm(BUG ~ CC+CALL+LOC, d.train,  
           family="binomial")  
summary(model)
```



```
Call:  
glm(formula = BUG ~ CC + CALL + LOC, family = "binomial", data = d.train)  
  
Deviance Residuals:  
    Min      1Q  Median      3Q      Max  
-1.6280 -0.4119 -0.3388 -0.3103  2.4587  
  
Coefficients:  
            Estimate Std. Error z value Pr(>|z|)  
(Intercept) -3.00685   0.29626 -10.149 < 2e-16 ***  
CC          -0.17097   0.06649  -2.571  0.01013 *  
CALL         0.14645   0.06581   2.225  0.02607 *  
LOC          0.03375   0.01202   2.808  0.00499 **  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '  
  
(Dispersion parameter for binomial family taken to be 1)  
  
Null deviance: 195.05 on 299 degrees of freedom  
Residual deviance: 168.69 on 296 degrees of freedom  
AIC: 176.69  
  
Number of Fisher Scoring iterations: 5
```

$$\log \frac{p}{1-p} = -0.17x_1 + 0.15x_2 + 0.03x_3 - 3.01$$

Get prediction results

- Get predictions on test data

```
predict(model, type="response", d.test)
```



99	505	124	97
0.053450137	0.067460143	0.148106912	0.050136245
.....				

Extract only the output probability from this

```
result = predict(model, type="response", d.test)
result = as.vector(result)
```

Tests in order of highest probability

- Test data (d.test) module
Probability output of logistic regression model
- Arrange (result) in descending order**
- **Record the presence** or absence of **bugs** in that order

```
found.bugs =  
  d.test$BUG[order(result, decreasing=T)]
```

Accumulate bug finds

- The presence or absence of bugs (**1** or **0**) is recorded in order in the variable `found.bugs`

```
0 0 0 0 1 0 1 0 0 0 0 0 .....
```

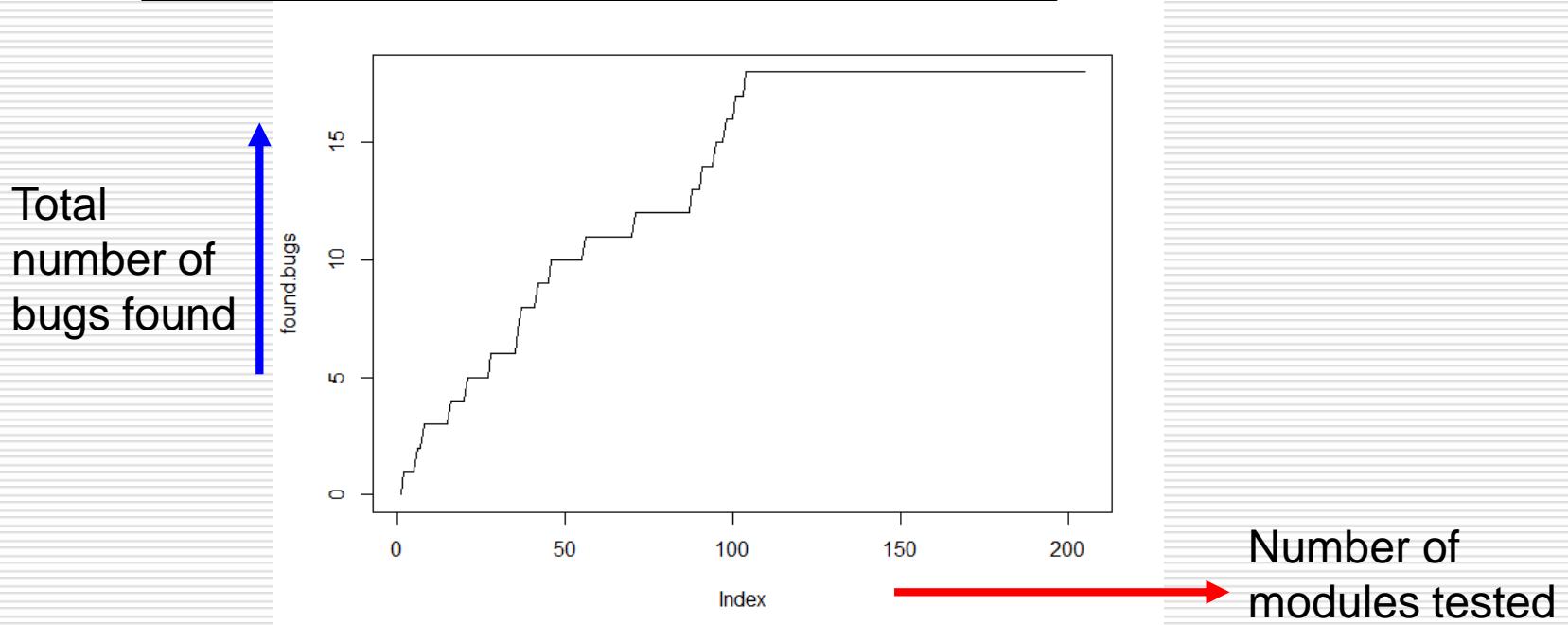
- By accumulating (adding) this, Find out how many bugs have been found so far

```
for ( i in 2:length(found.bugs) ){
  found.bugs[i] = found.bugs[i-1] + found.bugs[i]
}
```

Graph cumulative bug finds

- Graph the accumulated found.bugs

```
plot(found.bugs, type='l')
```



How quickly did you find 80% of bugs?

- How few tests were able to find **80% of the total bugs**?

```
b80 = max(found.bugs)*0.8
```

```
sum(found.bugs < b80) / length(found.bugs)
```

Smaller is more efficient testing

```
[1] 0.4585366
```

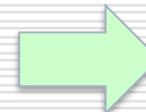
For only the cyclomatic numbers

```
[1] 0.4195122
```

Why didn't the performance improve?

- Affected by uneven number of buggy and non buggy samples.
 - **Overwhelmingly non buggy samples**

```
sum(d.train$BUG==1)  
sum(d.train$BUG==0)
```

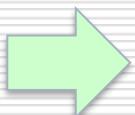


```
[1] 30  
[1] 270
```

- Since the parameters are adjusted so that the **overall error is minimized**, the result is a model that is easy to **predict without bugs**.

How to deal with it?

- There are two main types of basic measures
 - (1) **Change the weight in training**: make the error heavier for the smaller (buggy) sample;
Or reduce the error for larger samples
 - (2) **Give uniform training data**: Make the number of buggy and non buggy samples in the training data roughly the same (50:50)



We will take the (2) approach here.

Undersampling and Oversampling

Undersampling

Instead of using all the majority (non buggy), use only a partial sample obtained by random sampling

If we simply undersample, the training data becomes extremely small

Oversampling

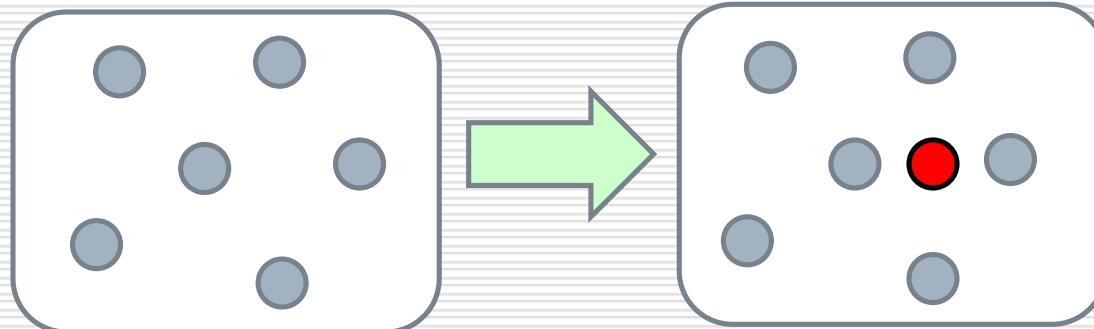
Use the minority (buggy) padding

Note the method of padding: I doubt whether it is possible to learn by simply choosing the same sample repeatedly.

SMOTE algorithm

Synthetic Minority Over-sampling Technique

- Oversampling by **artificially creating minority samples** (randomly using neighboring data)



- At the same time, **undersampling** is also performed for the majority.

Prepare training data with SMOTE algorithm (1/3)

- First, install the dedicated package

```
install.packages("smotefamily")
```

- And, load it

```
library(smotefamily)
```

Prepare training data with SMOTE algorithm (2/3)

- Create a dataset with the **SMOTE** algorithm
(The seed of the **random number** is set here)

```
set.seed(1234)  
d.train.smote  
= SMOTE(d.train[,c(2,3,4)],  
        d.train$BUG, K = 10)$data
```

Specify only columns
with metric values

Objective variable
(bug or not)

Number of neighbors to
use in synthesis

Prepare training data with SMOTE algorithm (3/3)

- The format of the data produced by the SMOTE algorithm is slightly different from that used in the logistic regression model, so adjust

```
names(d.train.smote)[4] = "BUG"  
d.train.smote$BUG = as.numeric(d.train.smote$BUG)
```

Rebuild model with training data by SMOTE algorithm

- Rebuild the logistic regression model and make predictions on the test data

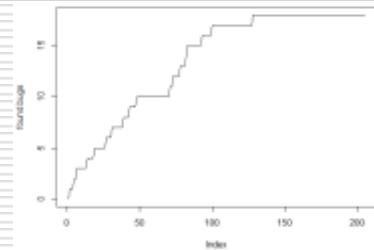
```
model = glm(BUG ~ CC+CALL+LOC,  
            d.train.smote, family="binomial")
```

```
result = predict(model, type="response", d.test)  
result = as.vector(result)
```

Evaluate test efficiency

□ Evaluate test efficiency again

```
found.bugs = d.test$BUG[order(result, decreasing = T)]  
for ( i in 2:length(found.bugs) ){  
  found.bugs[i] = found.bugs[i-1] + found.bugs[i]  
}  
plot(found.bugs, type='l')  
  
b80 = max(found.bugs)*0.8  
sum(found.bugs<b80)/length(found.bugs)
```



[1] 0.4

Slightly improved results

There are many other predictive models

- The logistic regression model is a basic model, but not necessarily the best model
- **There are many other machine learning models**, so there is a possibility that testing can be made **more efficient** by using other models.

[Exercise 3]

[13] Exercise-3
(by tomorrow, 1pm)

- Reset the random number seed to 9876 and redo the split into training and test data

```
set.seed(9876)
```

- Then, for the training data, answer the recall, precision, and F value when the cyclomatic number threshold is 10.

p13 -- p23

[Exercise 4]

[13] Exercise-4
(by tomorrow, 1pm)

- As a continuation of exercise 3, create a logistic regression model using the SMOTE algorithm (specify in the R script on p54 as well)

```
set.seed(9876)
```

- And answer the efficiency of the test for the test data

p54 -- p58