

# ソフトウェアテスト

## [4] ブラックボックステスト

---

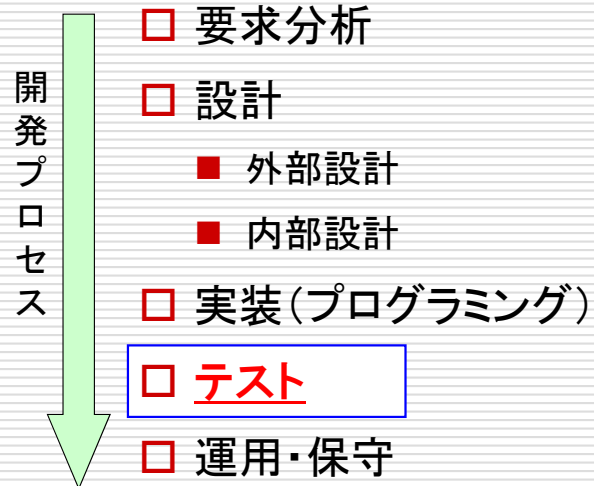
Software Testing  
[4] Black Box Testing Techniques

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

# テスト工程

## (ウォーターフォールモデルの場合)

---



テスト工程(テストこうてい): testing phase

# テストの目的と内容

---

## □ 目的

ソフトウェアに存在するかもしれないエラー  
(誤り, バグ)を見つけ出す



## □ 内容

**ソフトウェアを実行し, 仕様通りの正しい動き**  
**をするのかを確認する**

**また, 仕様にはない状況に対しても障害が起こ**  
**らないことを可能な限り確認する**

誤り(あやまり) : error

バグ : bug

仕様通りの正しい動き(しようどおりのただしいうごき) : correct behavior according to the specification

仕様にはない状況(しようにはないじょうきょう) : exceptional situations which were not specified in the specification

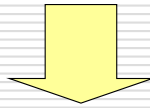
障害が起こらない(しょうがいがおこならい) : no failure occurs

可能な限り(かのうなかぎり) : as many as possible

## テストの難しさ

---

- 設計者・開発者は、もともと仕様通りのものを作ろうとした



- 仕様通りかどうかは、仕様書に沿って確認できる(ただし、それでも網羅は難しい)
- 仕様外については、そもそも想定外であったり、気付かなかったりするのでチェックは大変

難しさ(むづかしさ) : difficulty

網羅(もうら)する : encompass, cover the whole cases

仕様外(しょうがい) : unspecified

想定外(そうていがい) : unexpected

気付かない(きづかない) : unaware

大変(たいへん) : hard work

## テストの大事さ： テスト次第では防げたかもしれない事例

- AT&Tのネットワークが全面停止で**750万件の電話が不通**(1990年)  
原因：更新時にプログラムを1行追加しただけだった
- Ariane 5 **ロケットが空中で爆発**(1996年)  
原因：64ビットの数字を16ビットで扱っていた  
※開発費80億ドル以上(1兆円近い)
- 火星調査機が**火星面へ墜落**(1999年)  
原因：着陸のためのエンジン噴射の推力について、  
解析側と運用側で単位が違った：ポンドとニュートン

(C) 2007-2024 Hirohisa AMAN

5

電話(でんわ) : telephone

不通(ふつう) : disconnected

更新(こうしん) : update

空中(くうちゅう)で爆発(ばくはつ) : exploded in midair

開発費(かいはいはつひ) : development cost

80 億(おく)ドル : 80 billions of US dollars

1 兆円(ちょうえん) : 1 trillion(million million) Japanese yen

火星(かせい)調査機(ちょうさき) : Mars explorer

墜落(ついらく) : crash

着陸(ちゃくりく) : landing

エンジン噴射(エンジンふんしゃ) : engine injection

推力(すいりよく) : thrust force

解析側(かいせきがわ) : analysis side

運用側(うんようがわ) : operation side

単位(たんい) : unit

ポンド : pound

ニュートン : newton

用語:

## テストケース, テストスイート, テスト空間

---

□ **テストケース**(Test Case)

ある特定の入力データ・条件(と期待される出力)

□ **テストスイート**(Test Suite)

テストケースの集合で, 一連のテストを実行するもの

□ **テスト空間**(Testing Domain)

プログラムに誤りが無いことを保証できるような  
**テストケースの全集合**

用語(ようご): technical term

ある特定の(あるとくていの): a certain

入力(にゅうりよく)データ: input data

条件(じょうけん): condition

期待(きたい)される出力(しゅつりよく): expected output

集合(しゅうごう): set

一連の(いちれんの): a series of

保証(ほしょう): assure

全集合(ぜんしゅうごう): entire set

## テストの例(1):長さの変換

※1キロメートル  
= 0.621371 マイル

### 「キロメートル → マイル」変換プログラム

【入力】0 以上 10000 未満の実数とし、

小数点以下は 2 桁まで有効(それより下の桁は無視)

【出力】マイルに変換した値で、小数点以下 3 桁に切り捨てたもの

入力	期待される出力	入力	期待される出力
0	0.000	9.99	6.207
9999.99	6213.703	10	6.213
1	0.621	-1	エラー
1.001	0.621	10000	エラー

他にも例外的な場合(数値以外を入力)もあるとよい

長さ(ながさ):length

変換(へんかん):conversion

キロメートル:kilometer

マイル:mile

未満(みまん):less than

実数(じっすう):real number

小数点(しょうすうてん):decimal point

小数点以下(しょうすうてんいか)2 桁(けた):two decimal places

有効(ゆうこう):valid

無視(むし):avoid

小数点以下 3 桁に切り捨て(きりすて):truncate the number to three decimal places

例外的(れいがいてき)な場合(ばあい):exceptional case

## テストの例(2) : 三角形問題 (※有名な問題)

---

3つの整数を入力とし、それぞれを辺の長さとした三角形を考える。そこで出来上がるのが

- 二等辺三角形
- 正三角形
- 不等辺三角形

のいずれなのかを判定するプログラムについて、テストケースを考えよ。

三角形(さんかくけい)問題(もんだい) : triangle problem

整数(せいすう) : integer

辺(へん)の長さ(ながさ) : length of side

二等辺三角形(にとうへんさんかくけい) : isosceles triangle = a triangle with two equal sides

正三角形(せいさんかくけい) : equilateral triangle = a triangle with three equal sides

不等辺三角形(ふとうへんさんかくけい) : scalene triangle = a triangle with no two sides of equal length



## 三角形問題に対するテストケース例

入力	期待される出力
(2, 5, 5)	二等辺三角形
(5, 5, 5)	正三角形
(3, 4, 5)	不等辺三角形
(0, 0, 0)	エラー ※点になってしまう
(3, 4, 7)	エラー ※線分になってしまう
(2, 5, 8)	エラー ※最長辺の長さ > 他の辺の長さの和
(1.3, 4, 5)	エラー ※実数がある(三角形としての不等式は成立)
(-2, 4, 5)	エラー ※負の数がある

※三角形としての不等式: 最長辺の長さ < 他の辺の長さの和

さらには数値の入力順序を入れ替えたものもあるとよい

点(てん): point

線分(せんぶん): segment

最長辺(さいちょうへん): the longest side

他(ほか)の辺(へん)の長さ(ながさ)の和(わ): sum of the other sides' lengths

実数(じっすう): real number

不等式(ふとうしき): inequality

成立(せいりつ): hold, satisfy

負の数(ふのすう): negative number

入力順序(にゅうりょくじゅんじょ): inputting order

## 【演習1】

### 整数のソートングプログラムをテストせよ

- いま,  $N$ 個の整数をソートングするプログラムが与えられている
  - $0 \leq N \leq 10000$
  - 入力は(数列が格納されている)ファイルから
  
- このプログラムのためのテストケース(ここでは入力だけでよい)を考えなさい

ソートング: sorting

数列(すうれつ): sequence of numbers

格納(かくのう): store

## テストで重要なこと

### □ 結果を記録する

- どのテストケースでこういった不具合が見つかったのか, それは誰がいつ見つけたのか
- 後で修正するときに重要な情報となる

### □ 適切に伝える

- 曖昧な情報では役に立たない
- 不具合を報告するのは大事だが, 伝え方も大事である  
不具合を見つけたことを得意げに開発者へ提示すべきではない  
作った本人しか分からない苦労もあるので人間関係も大切に

単に「動かない」という  
報告だけでは無意味

(C) 2007-2024 Hirohisa AMAN

12

記録(きろく) : record

曖昧(あいまい) : vague

役に立たない(やくにたたない) : does not work, cannot be useful

伝え方(つたえかた) : communication manner/style

得意げに(とくいげに) : in (with) a proud tone

提示(ていじ) : show, report

本人しか分からない苦労(ほんにんしかわかんならいくろう) : effort that others do not know

## テスト容易性(testability)

---

- テストの容易性(やりやすさ)は
  - テストに費やされる**コスト(工数)の削減**
  - テスト**期間の短縮**に直結する重要な特性
  
- 当然ながら品質の向上につながる

コストの削減(さくげん) : reduction in cost

テスト期間の短縮(テストきかんのたんしゅく) : shortening of testing period

直結する(ちょっけつする) : directly linked to

特性(とくせい) : characteristics

## テスト容易性の向上

- 設計や実装での考慮が不可欠
  - どの機能がプログラムのどの部分に対応しているのかが明確になるように作ることが大事
  - テストに配慮した設計にする  
例: うまく関数に分けることで独立してテストできる
- 作る時にはテストを念頭に置く
  - テストで誤りが見つかってもしすぐに対処できるよう、何をやっているのかが分かりやすいように作る
  - その場しのぎ(「とりあえず動けばOK」の感覚)で作るのは、はっきり言って時間と労力の無駄

機能(きのう) : function, functionality

対応する(たいおうする) : correspond to

明確(めいかく) : clear

配慮(はいりよ) : taking care, considering

うまく関数に分ける(うまくかんすうにわけける) : dividing the system into functions (subroutines) well

独立して(どくりつして) : independently

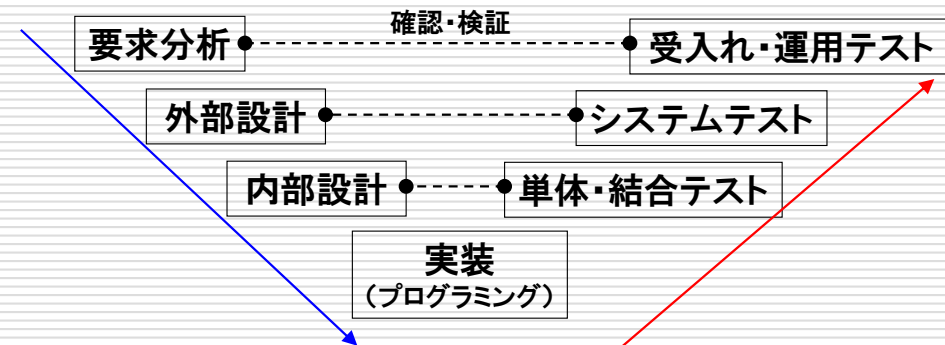
念頭に置く(ねんとうにおく) : bear it in mind

その場しのぎ(そのばしのぎ) : ad hoc

時間と労力の無駄(じかんとろうりょくのむだ) : waste of time and effort

## Vモデルでの対応

- 単体・結合テスト： モジュールの動作確認
- システムテスト以降： システムの動作確認



(C) 2007-2024 Hirohisa AMAN

15

単体(たんたい)テスト: unit testing

結合(けつごう)テスト: integration testing

システムテスト: system testing

受入れテスト(うけいれテスト): acceptance testing

## モジュール(module)

---

□ モジュールとは**分離可能なプログラムの単位**

- そのレベルで差し替えができるようなプログラムのかたまり(**ソフトウェア部品**)を意味する
- C 言語では「**関数**」または「**いくつかの関数をまとめたもの**(通常は1個のプログラム)」がこれに該当する
- Java のようなオブジェクト指向言語の場合は、「**クラス**」がこれに該当する(※「関数」には「メソッド」が相当するが、メソッドだけでは存在できない)

分離可能(ぶんりかのう) : separable

差し替え(さしかえ) : replace

関数(かんすう) : function

オブジェクト指向言語(オブジェクトしこうげんご) : object-oriented language

## 単体テスト, 結合テスト

---

### □ 単体テスト



1つのモジュールについて, さまざまな入力を与え, 適切な出力が得られるのかを確認

### □ 結合テスト



複数のモジュール(単体テスト済み)を組み合わせ, 結合関係を持った状態でモジュールが適切な入出力を行えるかを確認

単体(たんたい)テスト: unit testing

さまざまな入力(にゅうりょく): various inputs

適切な出力(てきせつなしゅつりょく): appropriate/correct outputs

結合(けつごう)テスト: integration testing

結合関係(けつごうかんけい): coupling



# システムテスト, 受入れ・運用テスト

---

## □ システムテスト



システムが仕様通りに動作するのを確認

## □ 受入れテスト・運用テスト



システムテストと同様であるが, 実際の運用環境下でのテスト, 顧客によるテストの意味  
実際に運用してもらう(仕事で使ってもらう等)

システムテスト: system testing

仕様通り(しようにおり) : according to the specification

受入れテスト(うけいれテスト) : acceptance testing

運用テスト(うんようテスト) : operational testing

実際の(じっさいの) : actual

運用環境下(うんようかんきょうか) : under the operational environment

顧客(こきやく)によるテスト: testing by the customers/users

## テストの分類

---

### □ ブラックボックステスト

プログラムの中身は見ないもの(ブラックボックス)として、仕様に基づいた動作テストを行う

### □ ホワイトボックステスト

プログラムの内部構造(主にフローチャート)に基づいた動作テストを行う

### □ ランダムテスト

テストケースをランダムに(無作為に)作成して動作テストを行う

ブラックボックステスト: black box testing

仕様に基づいた(しようにもとづいた): based on the specification

ホワイトボックステスト: white box testing

内部構造(ないぶこうぞう): internal structure

フローチャート: flowchart

ランダムテスト: random testing

無作為に(むさくいに): at random

## ブラックボックステスト手法(1) 同値分割法(equivalence partitioning)

---

- テストケースを効率的に設計するため  
一種の**同値関係**に基づいて**入力空間を分割**
  - 1つの分割を「**同値クラス**」という
  - 同値クラスは、**入力条件ごとに**プログラムから**同じ扱いを受けるはずのもの**をまとめたもの  
(同値クラスは2種類)
  - 有効同値クラス: 入力として有効なもの
  - 無効同値クラス: 入力として無効なもの

同値分割法(どうちぶんかつほう): equivalence partitioning

同値関係(どうちかんけい): equivalence relation

入力空間(にゅうりよくくうかん): input space

分割(ぶんかつ): divide, partition

同値クラス(どうちクラス): equivalence class

入力条件(にゅうりよくじょうけん): input condition

同じ扱いを受けるはず(おなじあつかいをうけるはず): expected to get the same treatment

有効(ゆうこう): valid

無効(むこう): invalid

## 同値関係(equivalence relation)

□ 集合の要素間の関係( $R$ で表す)で, 次の性質をすべて満たすもの

- 反射律(reflexivity):  $xRx$
- 対称律(symmetry):  $xRy$  ならば  $yRx$
- 推移律(transitivity):  $xRy$  かつ  $yRz$  ならば  $xRz$

【 $R$  に該当するものの例: 同値関係】

= (等しい),  $\parallel$  (直線が平行),  
「10で割った余りが同じ」等

【 $R$  に該当しないものの例】

< (小さい), 「(図形が)重なる」,  
「 $x$  が  $y$  を)知っている」等

代表例を1つ決めると, それと同値関係にあるものでグループ(同値クラス)ができる

(C) 2007-2024 Hirohisa AMAN

22

集合(しゅうごう): set

要素(ようそ): element

関係(かんけい): relation

反射律(はんしゃりつ): reflexivity

対称律(たいしょうりつ): symmetry

推移律(すいいりつ): transitivity

直線が平行(ちよくせんがへいこう): lines are parallel

10で割った余り(10でわったあまり): remainder when divided by 10

重なる(かさなる): overlap

代表例(だいひょうれい): representative example



10-minutes rest break

**10分休憩**

## 同値分割の例① (1/2)

---

□ 入力: 科目の成績(整数値)

□ 出力: 「S, A, B, C, D, NA」  
のいずれか1つ

□ 仕様: 90~100 → S,    80~89 → A,  
70~79 → B,    60~69 → C,  
0~59 → D,    -1 → NA

## 同値分割の例①（2／2）

- 有効同値クラスと無効同値クラスは次のようになる

有効同値クラス	無効同値クラス
90 ～ 100	101以上
80 ～ 89	－2以下
70 ～ 79	
60 ～ 69	
0 ～ 59	
－1	

各同値クラス  
から値を選んで  
テストを行う

## 同値分割の例②（1／4）

---

### □ 駐車料金計算プログラム

【入力(1):駐車時間(単位:分)】

- 1分～30分 = 100円
- 31分～60分 = 200円
- それ以降 = 1 時間単位で 100 円アップ

【入力(2):提携店での買い物額】

- 2,000円以上で 1 時間無料

※駐車時間が1時間を超える場合は、  
200円(最初の1時間分)だけ割引

駐車料金(ちゅうしゃりょうきん) : parking fee

駐車時間(ちゅうしゃじかん) : parking time

提携店(ていけいてん) : partner shop

買い物額(かいものがく) : purchased amount



## 同値分割の例②（2／4）

入力条件	有効同値クラス	無効同値クラス
(1) 駐車時間	1～30	0 以下
	31～60	
	61 以上 ←	
(2) 買い物額	0～1999	-1 以下
	2000 以上	

→ 計算法が  
同じという  
ことでまと  
めている

※時間や金額でマイナスの値はあり得ないが、  
ここでは「入力できてしまう」という想定にしている

## 同値分割の例② (3/4)

例えば, テストケース:  
70分駐車, 2100円買い物  
⇒ 駐車料金100円  
(= 300円 - 200円)

まず, 2つの入力条件について,  
**有効同値クラスの組合せ**を作る

条件(1) 駐車時間	有効	1~30	○			○		
		31~60		○			○	
		61~			○			○
	無効	~0						
条件(2) 買い物額	有効	0~1999	○	○	○			
		2000~				○	○	○
	無効	マイナス						

縦方向の  
列が1個  
のテスト

## 同値分割の例② (4/4)

次に、無効同値クラスを1個だけ  
含むような組合せも作る

例えば、テストケース:  
0分駐車, 2100円買い物  
⇒ 不正な入力として  
エラーを表示

条件(1) 駐車時間	有効	1~30	○				
		31~60		○			
		61~			○		
	無効	~0				○	○
条件(2) 買い物額	有効	0~1999				○	
		2000~					○
	無効	マイナス	○	○	○		

不正な入力(ふせいなにゅうりょく): invalid input

## 同値分割法の手順

---

(1) 有効同値クラス, 無効同値クラスを入力条件ごとに設定する

(2) 有効同値クラスの組合せをすべて作る

(3) 無効同値クラスを1個だけ含むような組合せをすべて作る

※もちろん, 2個以上含むかたちも加えてテストしてよいが, まずは1個だけ無効なものを混ぜて不具合が出ないことを確認するのが先

手順(てじゅん) : steps, procedure

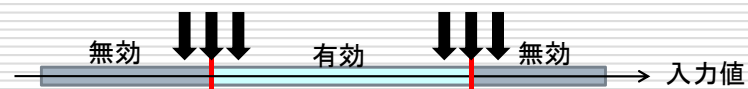
組合せ(くみあわせ) : combination

混ぜる(まぜる) : mix

## ブラックボックステスト法(2)

### 境界値分析法(boundary-value analysis)

- プログラムの**条件の変わり目(境目)**に着目
  - 入力, 出力の**条件の変わり目**でテストする
  - 有効同値クラス及び無効同値クラスの**端**や**境目**, あるいはそれから**外に出たところ**でテスト



変わり目(かわりめ), 境目(さかいめ) : boundary

端(はし) : edge, end

外に出たところ(そとにでたところ) : outside

## なぜ条件の変わり目に着目するのか

- プログラム中の条件チェックの部分で間違いを犯していることが意外と多い

(例) 成績評価プログラムの if 文の条件式でミス

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

- ✓ 90 点が「S」にならない
- ✓ 101 点が「エラー」にならずに「S」になる



正しくは

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

成績評価(せいせきひょうか) : assessment of academic achievement

## 境界値分析法の例

- 10進数(0以上16未満)を2進数で出力する  
プログラムをテスト対象とした場合

条件の変わり目 (出力ビット数に注目)		同値クラスの端 とその外	
(入力)0, 1	1ビット	0	有効最小値
2, 3	2ビット	15	有効最大値
4, 7	3ビット	-1	無効
8, 15	4ビット	16	無効

他にも無効  
同値クラスと  
して、整数  
以外を入力  
もあるとよい

10進数(10しんすう) : decimal number

0以上16未満(0いじょう16みまん) : greater than 0 and less than 16

2進数(2しんすう) : binary number

有効最大値(ゆうこうさいだいち) : valid maximum value

有効最小値(ゆうこうさいしょうち) : valid minimum value

無効(むこう) : invalid

## 【演習2】

### 境界値分析法でテストケースを設計せよ

---

#### □ テスト対象

2つの整数(0以上10未満)を入力とし, それらの和を16進数で出力するプログラム

- 出力は0ー15ならば1桁, 16ー18は2桁
- 無効な入力に対しては ERROR と表示

桁(けた): digit



## (参考:その他の手法)直交表を利用

- いくつかのテストすべき機能があったとき, それらの**組合せを網羅**することも大事
- 単純に  $n$  個の機能の中から 2 個を選ぶと  $n(n-1)/2$  **通り**となり, 組合せ数が膨大になる恐れがある  
例えば  $n=10$  で 45 通り,  $n=20$  で 190 通り,  
 $n=30$  で 435 通り

直交表では効率的に組合せを作ることができる

直交表(ちょっこうひょう) : orthogonal array

組合せを網羅(くみあわせをもうら) : cover possible combinations

xxx 通り(とおり) : xxx cases

組合せ数(くみあわせすう) : number of combinations

膨大(ぼうだい) : enormous

## 直交表

例えば, ワープロでの文字修飾で  
A = 太字, B = 斜体, C = 下線, D = 影付き

□ 2機能(A,B)  
について網羅

A	B
0	0
0	1
1	0
1	1

□ 4機能(A,B,C,D)  
では, 結果的には  
右の8通りでOK

どの2列をとっても

(0,0)

(0,1)

(1,0)

(1,1)

が**同じ個数だけ登場**

この種の表を直交表といい,  
組合せテストで効果を発揮している

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

(C) 2007-2024 Hirohisa AMAN

36

ワープロ = ワードプロセッサ : word processor

文字修飾(もじしゅうしょく) : character decoration

太字(ふとじ) : bold

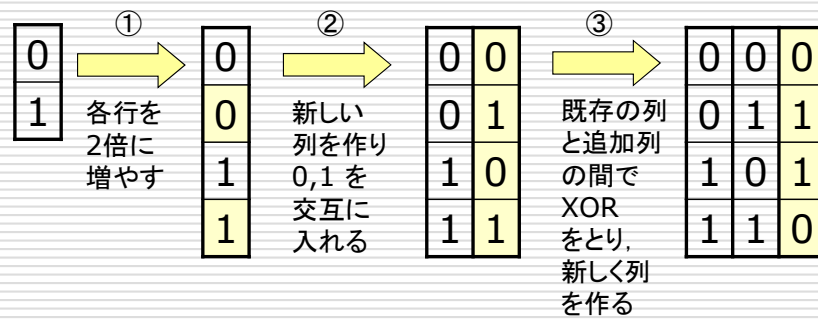
斜体(しゃたい) : italic

下線(かせん) : underlined

影付き(かげつき) : shaded

# 直交表の作り方

## □ 基本は3ステップ



(C) 2007-2024 Hirohisa AMAN

37

XOR = exclusive OR

2倍(2ばい)に増やす(ふやす) : duplicate

交互に(こうごに) : alternately

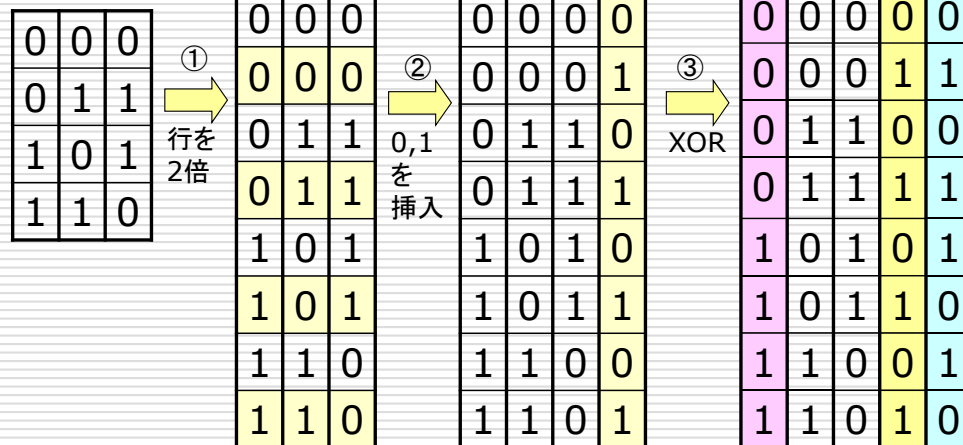
既存の列(きぞんのれつ) : existing column

追加列(ついかれつ) : added column

## 直交表の作り方(つづき)

※あと2列追加できる

□ (2周目)



(C) 2007-2024 Hirohisa AMAN

38

### 【演習3】

#### 直交表へさらに2列追加せよ

先の直交表  
では, さらに  
あと2列  
追加できる

これを  
求めなさい

	↓	↓				
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	1	0	0	1		
1	1	0	1	0		

## 2機能の組合せに関する有効性

組合せ機能数とフォールトの関係: バグ検出率(%)

組合せ数	エキスパートシステム					OS	組込み医療機器	Mozilla	Apache	DB
1	61	72	48	39	82		66	29	42	68
2	36	10	6	8	*		15	47	38	25
3	*	*	*	*	*		2	19	19	5
4	*	*	*	*	*		1	2	7	2
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.

(C) 2007-2024 Hirohisa AMAN

41

エキスパートシステム: expert system

組込み医療機器(くみこみいりようきき): embedded medical device

バグ検出率(バグけんしゅつりつ): bug detection rate

フォールト: fault = software bug

## まとめ

---

- テスト:「バグ」を見つけ出す作業
  - 品質保証の要である
  - **柔軟な発想**が要求される
  - 例: **三角形問題**
- ブラックボックステスト:外部から見た動作確認
  - **同値分割法**:  
有効同値クラスと無効同値クラスの**組合せ**
  - **境界値分析法**:  
入出力条件や同値クラスの**境目に注目**

---

(C) 2007-2024 Hirohisa AMAN

42

品質保証(ひんしつほしょう): quality assurance

要(かなめ): key

柔軟(じゅうなん)な発想(はっそう): flexible mindset

例(れい): example

三角形問題(さんかくけいもんだい): triangle problem

## 予告(Notice)

---

- 次回は自分の PC で C プログラムの作成とテストを行うという演習を行う

The next class is an **exercise** class;  
All students have to **develop and test C programs on your PCs**

- C コンパイラをインストールしておくこと  
(推奨: gcc)

You must finish installing a **C compiler**  
on your PC (**recommendation: gcc**)



## 宿題(homework)

---

**“[04] quiz” に答えなさい  
(今週の金曜日まで)**

Answer “[04] quiz”  
by this Friday 23:59

注意: quiz のスコアは成績の一部となります  
(Note: Your quiz score will be a part of your final evaluation)

## 【演習1】 解答例

---

□ ランダムな数列

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

□ 整列済み数列

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

□ 1個だけの場合, N個を超える場合

{ 2 }      { 1, 2, 3, ..., N, N+1 }

□ 0個

{ }

他にもファイルが見つからない, オープンに失敗する等の例外も考えるとよい

整列済み(せいれつずみ): sorted

失敗(しっぱい): fail

## 【演習2】 解答例

条件の変わり目		同値クラスの端とその外	
6+9	1桁	0+0	有効最小値
8+7		9+9	有効最大値
7+9	2桁	-1+0	無効
8+8		10+0	無効
9+7		10+10	無効
5+5	記号	-1+10	無効

他にも無効同値クラスとして、整数以外の入力もあるとよい

※1桁の場合に6+9だけでなく8+7も考えているのは、2つの数の大小関係で両パターンを考慮しているため。(必須ではない)  
2桁の場合も同様。

### 【演習3】 解答

直交表へさらに2列追加せよ

	↓		↓				
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	1	1		1	
0	1	1	0	0		1	
0	1	1	1	1		0	
1	0	1	0	1		1	
1	0	1	1	0		0	
1	1	0	0	1		0	
1	1	0	1	0		1	

(C) 2007-2024 Hirohisa AMAN

40

桁(けた):digit