

# ソフトウェアテスト

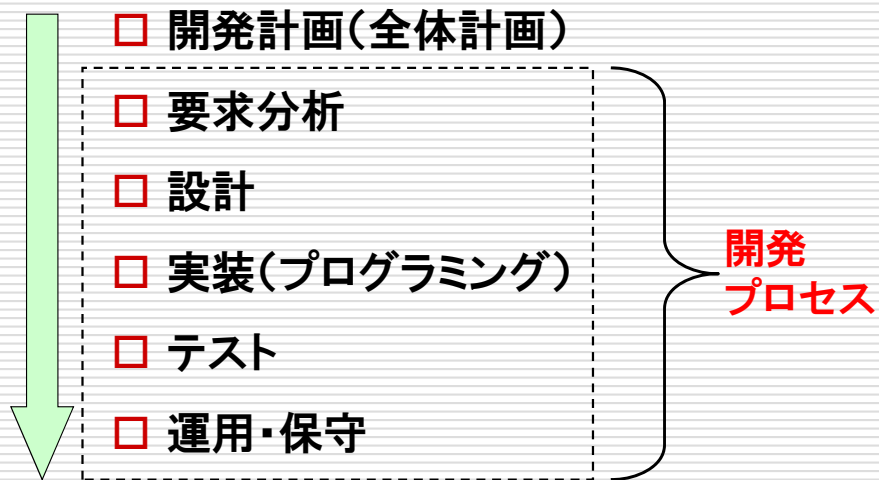
## [3] ソフトウェア開発プロセス

---

Software Testing  
[3] Software Development Process

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

# ソフトウェア開発の大まかな流れ



(C) 2007-2023 Hirohisa AMAN

2

開発計画(かいはつけいかく) : development planning

全体計画(ぜんたいけいかく) : total planning

要求分析(ようきゅうぶんせき) : requirements analysis

設計(せっけい) : design

実装(じっそう) : implementation

運用(うんよう) : operation

保守(ほしゅ) : maintenance

開発プロセス(かいはつぷろせす) : development process

## (0) 開発計画

---

☐ 開発計画(全体計画)

☐ 要求分析

☐ 設計

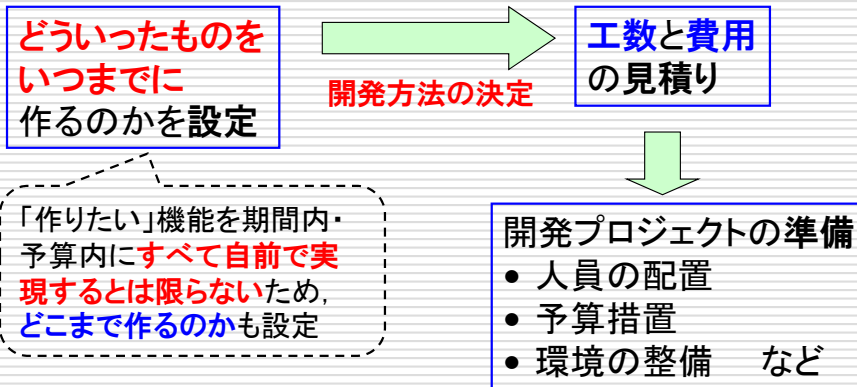
☐ 実装(プログラミング)

☐ テスト

☐ 運用・保守

## (0) 開発計画

### □ 設定 → 見積り → 準備



(C) 2007-2023 Hirohisa AMAN

4

設定(せってい) : setting

見積り(みつもり) : estimation

準備(じゅんび) : preparation

自前で実現(じまえでじつげん) : implementing by themselves

開発方法(かいはつほうほう) : development method

工数(こうすう) : effort

費用(ひよう) : cost

人員の配置(じんいんのはいち) : staff assignment

予算措置(よさんそち) : budgetary measures

環境の整備(かんきょうのせいび) : environmental arrangement

## (0) 開発計画：開発方法の決定

### 基本方針

- **新規に開発**するのか？
- **既存のシステムをベース**とするのか？



どれだけの規模の開発となるのかを算出  
(例)どれだけの量のコードを書く必要があるか？



工数や費用の見積りに効いてくる

他にも検討しておくべき課題はいろいろ  
人材の確保や体制は？ 開発環境は？ 予算は？

(C) 2007-2023 Hirohisa AMAN

5

新規に開発(しんきにかいはつ)する:newly develop

既存の(きぞんの):existing

規模(きぼ):size

算出(さんしゅつ)する:calculate, compute

コード = ソースコード

見積りに効いてくる:they influence the cost estimation

人材の確保(じんざいのかくほ):securing human resources

課題(かだい):issues, problems

体制(たいせい):structure (team/organization structure)

予算(よさん):budget

## (1) 要求分析

---

☐ 開発計画(全体計画)

---

☐ **要求分析**

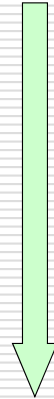
☐ 設計

☐ 実装(プログラミング)

☐ テスト

☐ 運用・保守

開発プロセスの  
はじまり



## (1) 要求分析

成果物

[要求仕様書](#)

ソフトウェアで実現しようとしている  
内容を明確にする

### □ 業務分析

ソフトウェアで実現しようとしている業務を分析・整理

### □ 要求内容の記述

- あいまいさ無く, 正確に記述
- 機能・コスト・納期の観点から実現可能性も検討

要求仕様書(ようきゅうしようしょ): requirements specification

業務分析(ぎょうむぶんせき): job analysis

あいまいさ: ambiguity

機能(きのう): functionality

納期(のうき): time to delivery

## 【例】掲示通知システム

### (1) 要求分析

- 自分に関係する掲示が出たら, スマホへ通知してくれるシステムが欲しい
  - リアルタイムなのか? 日ごとにまとめるのか?
  - どうかたちで通知して欲しいのか?
  - 通知先をどうやって登録・変更するのか?
    - なりすまし対策・セキュリティ保護はどうするか?
  - どのような形式で掲示の内容を入力するか?
  - そもそも「関係する掲示」の条件とは何か?
    - 受講者データベースとの連携が必要?

(C) 2007-2023 Hirohisa AMAN

8

掲示(けいじ): bulletin board, notice board

スマホ = スマートフォン

通知(つうち): notification

登録(とうろく): registration

変更(変更): change, modification

なりすまし: spoofing

セキュリティ保護(セキュリティほご): security protection

形式(けいしき): format

受講者(じゅこうしゃ): students who take the course

連携(れんけい): cooperation



## 【演習1】

### 要求分析に必要な検討事項を考えよ

---

□ 顧客からの要望

「食堂の混雑状況をスマホでチェックできるシステムを作って欲しい」

□ 要求仕様書を作成していく上で顧客に確認（あるいは検討）すべき事項を考え、列挙しなさい

検討事項(けんとうじこう) : items to be studied

顧客(こきやく) : customer

要望(ようぼう) : request

食堂(しょくどう) : cafeteria

混雑状況(こんざつじょうきょう) : congestion situation

列挙(れっきょ)する : enumerate

## (2) 設計

---

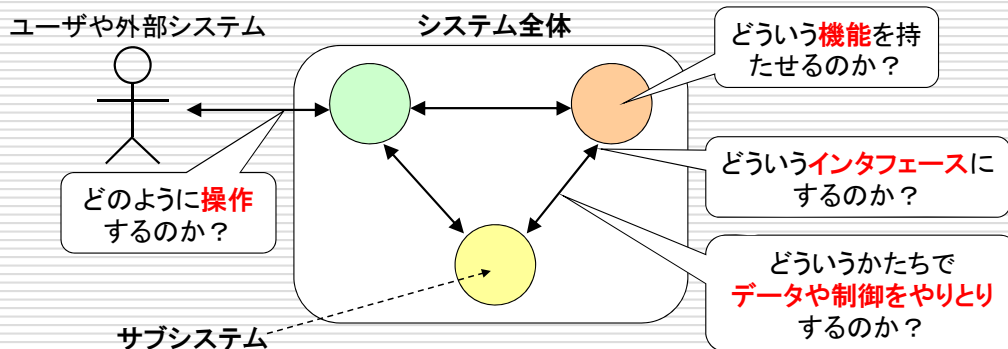
- ☐ 開発計画(全体計画)
- ☐ 要求分析
- ☐ **設計**
- ☐ 実装(プログラミング)
- ☐ テスト
- ☐ 運用・保守

## (2-1) 外部設計

成果物

外部仕様書

目標のシステムをサブシステムへ分割し、  
各サブシステムの外部から見た仕様を設計



(C) 2007-2023 Hirohisa AMAN

12

外部設計(がいぶせつけい) : external design

外部仕様書(がいぶしょうしょ) : external specification

分割(ぶんかつ)する : divide

操作(そうさ) : manipulation

機能(きのう) : functionality

インタフェース : interface

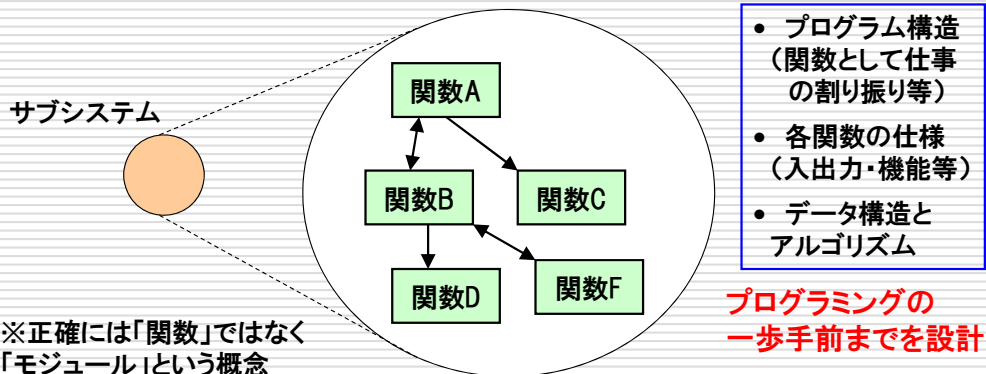
制御(せいぎょ) : control

## (2-2) 内部設計

成果物

内部仕様書

それぞれの外部設計の内容について  
詳細に**プログラムの仕様**を設計



(C) 2007-2023 Hirohisa AMAN

13

内部設計(ないぶせつけい) : internal design

内部仕様書(ないぶしょうしょ) : internal specification

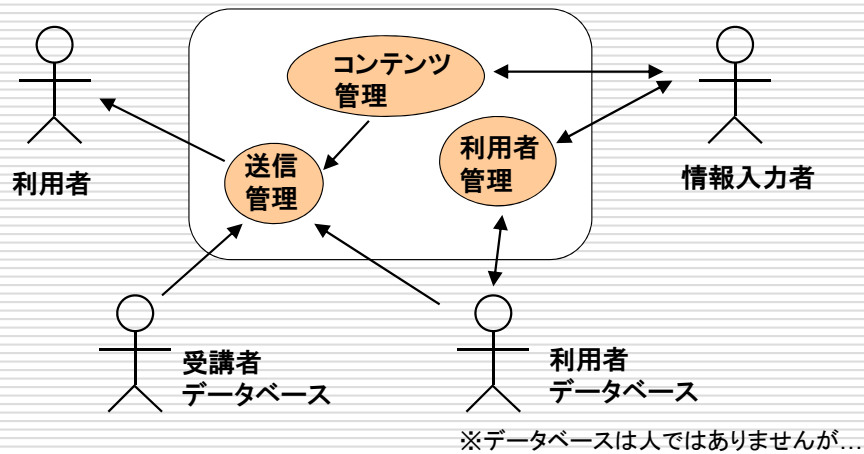
詳細に(しょうさいに) : in detail

関数(かんすう) : function

プログラム構造(プログラムこうぞう) : program structure

データ構造とアルゴリズム(データこうぞうとアルゴリズム) : data structure and algorithm

## 【例】掲示通知システム (2-1) 外部設計



(C) 2007-2023 Hirohisa AMAN

14

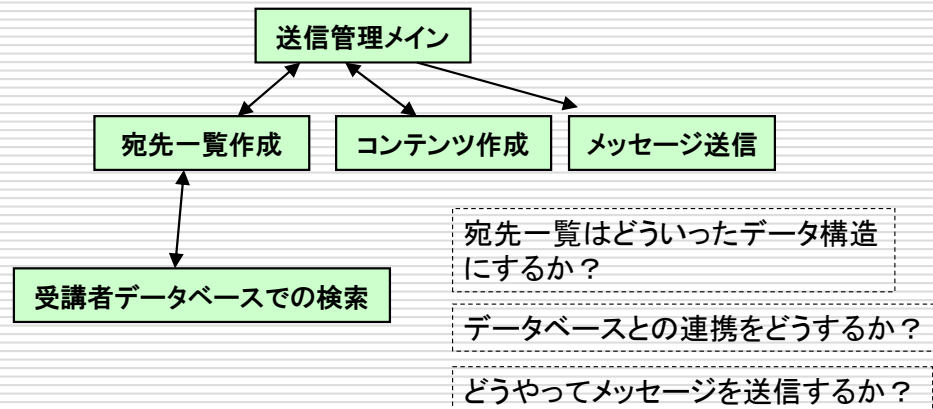
コンテンツ: contents

送信管理(そうしんかんり): sending management

利用者管理(りようしゃかんり): user management

## 【例】掲示通知システム (2-2) 内部設計

### □ 送信管理サブシステム



(C) 2007-2023 Hirohisa AMAN

15

宛先一覧(あてさきいちらん) : address list  
作成(さくせい) : making

### (3) 実装(プログラミング)

---

- ☐ 開発計画(全体計画)
- ☐ 要求分析
- ☐ 設計
- ☐ 実装(プログラミング)
- ☐ テスト
- ☐ 運用・保守

成果物

### (3) 実装(プログラミング) プログラム仕様書 ソースコード

#### 内部仕様に基づいてコーディング

- 言うまでもなく, プログラミング言語を使った  
コーディング(プログラミング)を行う
- 関連ドキュメントとしてプログラムの仕様書や  
フローチャート等の作成も

コーディング : coding

プログラミング言語(プログラミングげんご) : programming language

関連ドキュメント(かんれんどキュメント) : related documents

フローチャート : flow chart



## 【例】掲示通知システム

### (3) 実装(プログラミング)

---

- 目的の機能を実現するのに適したライブラリやプログラミング言語を用いる

例えば

- OS 独自の機能を活用 → C 言語
- データベースとの連携 → Java や Python
- 後で保守することも考慮し, プログラムの仕様やフローチャートといったドキュメントも積極的に作成しておく

OS 独自の機能(OS 独自のきのう) : OS-specific function

積極的に(せっきよくてきに) : actively

## (4) テスト

---

- ☐ 開発計画(全体計画)
- ☐ 要求分析
- ☐ 設計
- ☐ 実装(プログラミング)
- ☐ テスト
- ☐ 運用・保守

## (4) テスト

成果物

[テスト仕様書](#)  
[テスト報告書](#)

各種の仕様に基づいて  
ソフトウェアが適切に動作するかを**テスト**

- 関数(モジュール)はプログラム仕様通りか？  
→ [単体テスト](#)
- サブシステムは内部仕様通りか？ → [結合テスト](#)
- システムは外部仕様通りか？ → [システムテスト](#)
- システムは要求仕様通りか？ → [受入れテスト](#)

(C) 2007-2023 Hirohisa AMAN

20

テスト仕様書(テストしようしょ) : test specification

テスト報告書(テストほうこくしょ) : test result report

単体テスト(たんたいテスト) : unit testing

結合テスト(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

## (4) テスト

---

### ☐ 単体テスト

一つの部品(関数等)レベルで正しく動くか？

### ☐ 結合テスト

いくつかの部品がつながったかたち(関数が別の関数を呼び出す等)で正しく動くか？

### ☐ システムテスト

システムは設計通り, 仕様通りに動くか？

### ☐ 受入れテスト

システムは顧客が納得いくかたちで動くか？

部品(ぶひん) : part

顧客(こきゃく) : customer, user

納得いく(なっとくいく)かたち : in a convincing manner / in a way satisfying the customers



10-minutes rest break

**10分休憩**

## (5) 運用・保守

---

- ☐ 開発計画(全体計画)
- ☐ 要求分析
- ☐ 設計
- ☐ 実装(プログラミング)
- ☐ テスト
- ☒ 運用・保守

## (5) 運用・保守

成果物

各種マニュアル  
障害・対応報告書

ユーザによる運用 &  
不具合の発見や修正要求に応じて保守

- 運用に先立って, 操作・運用マニュアルを作成
- 障害が発生すれば, その報告と対応を記録・保持

各種マニュアル(かくしゅマニュアル): manuals of every kind

障害・対応報告書(しょうがい・たいおうほうこくしょ): trouble shooting report

不具合の発見(ふぐあいのはっけん): finding bugs, finding failures

修正要求(しゅうせいようきゅう): request for modification

## (補足)用語の使い分け

### □ 障害, 不具合, 故障(failure)

ソフトウェアが適切に動作しない**現象**

### □ 欠陥, フォールト(defect, fault)

障害を引き起こしている直接の**原因**

### □ エラー, 誤り(error)

欠陥を作り出してしまった**ミス**

**バグ(bug)**というのは  
これら三つを含んだ  
曖昧な用語

(C) 2007-2023 Hirohisa AMAN

25

障害(しょうがい)

不具合(ふぐあい)

故障(こしょう)

現象(現象) : phenomenon

欠陥(けっかん)

原因(げんいん) : cause

曖昧な用語(あいまいなようご) : vague term



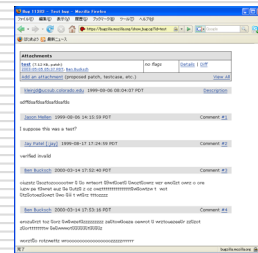
## 【例】掲示通知システム (5) 運用・保守

- 実際に使ってもらって
  - 不具合があれば報告してもらおう
  - 要望があれば出してもらおう

電子掲示板(BBS)や  
メーリングリストを活用

- 保守では
  - 不具合を修正
  - 要望に対応

Bugzilla や JIRA  
といった  
バグ管理システム  
もある



## 【演習2】 該当する工程を答えよ

□ 次の作業はどの工程(要求分析, 外部設計, 内部設計, 実装, テスト, 運用・保守)に該当するか答えなさい 「食堂の混雑チェックシステム」を想定

①混雑度データは, 測定時刻(日, 時, 分)と座席使用率(%)の構造体データとし, 3分間隔でファイルへ記録していくことにした

②本稼働前に同時に1000人分のアクセスを実行させたり, 営業時間外にアクセスさせたりして挙動を確認した

③システムの構成を  
(1)混雑度データの管理部  
(2)ユーザインタフェース部  
(3)制御(コントローラ)部の3つに分けて開発することとした

④利用者アンケートを実施し, 改善すべき案件を整理した

(C) 2007-2023 Hirohisa AMAN

27

工程(こうてい) : process, step

測定時刻(測定時刻) : measurement date

構造体(こうぞうたい) : struct data in C language

間隔(かんかく) : interval

本稼働前(ほんかどうまえ) : before the start of operations

営業時間外(えいぎょうじかんがい) : outside business hours

管理部(かんりぶ) : management/control part

利用者アンケート(りようしゃアンケート) : user survey

改善すべき案件(かいぜんすべきあんけん) : issues to be improved

# プロセスモデル

---

## □ ソフトウェア開発プロセス(基本的には, 次の5工程)

- 要求分析
- 設計
- 実装(プログラミング)
- テスト
- 運用・保守

をどのように進めていくのかを表したモデルのことをプロセスモデルという

## プロセスモデルの代表例

---

### □ ウォータフォールモデル

(また, その一つのバリエーションとして) **Vモデル**

### □ プロトタイピング

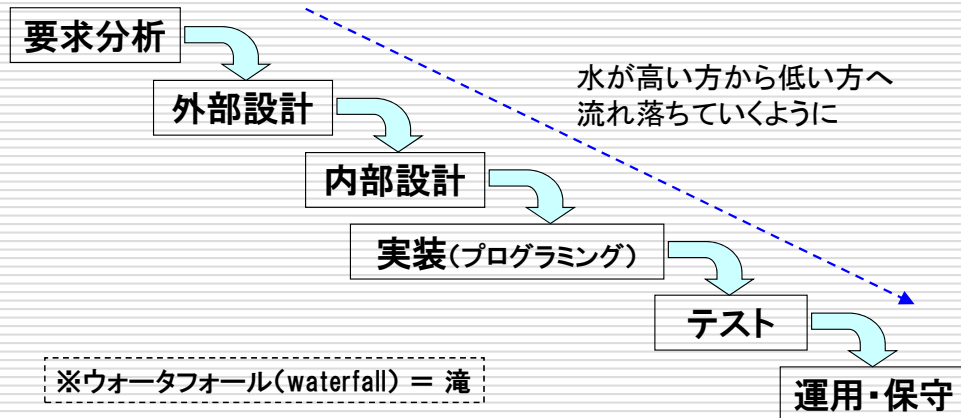
(これ自体はプロセスモデルというよりも開発手法)

### □ スパイラルモデル

開発手法(かいはつしゅほう) : development method

# ウォーターフォール(waterfall)モデル

□ 要求分析から運用・保守まで順々に進める



## ウォーターフォールモデル：特長

- 「要求分析」が完了したら次の「外部設計」へ、  
という具合に一つ一つの段階を順番にこなす
- プロジェクトの進捗(しんちょく)状況を把握しやすい
- 作業分担が明確である
- 伝統的な「定番」のモデルであり、教育しやすい  
(※最初の文献は1970年に発表)

今日でも、多くの人員を必要とした  
大規模プロジェクトで採用されることが多い

段階(だんかい) : stage, step

進捗(しんちょく) : progress

把握(はあく) : understanding

作業分担(さぎょうぶんたん) : task assignment

定番(ていばん) : standard, classic

文献(ぶんけん) : literature

大規模プロジェクト(だいきぼプロジェクト) : large-scale project

採用する(さいようする) : adopt

## ウォーターフォールモデル：問題点(1)

### □ 要求分析が「完全に」完了するとは限らない

- 理想的には、すべての要求が出し尽くされ、それら进行分析して、要求仕様書が出来上がる
- しかし、現実には難しい(気が付かないこともある)

要求分析が不完全なまま設計へ移行してしまいう可能性は高く、ウォーターフォールモデルの前提が覆され、混乱が生じる恐れもある

プログラムが出来上がった後になって仕様がかわってしまうなど

理想的(りそうてき) : ideal

出し尽くす(だしつくす) : output all of them

不完全(ふかんぜん) : imperfection

移行する(いこうする) : transition

前提(ぜんてい)が覆される(くつがえされる) : shatter the assumption

混乱(こんらん)が生じる(しょうじる) : cause much confusion among the developers

## ウォーターフォールモデル：問題点(2)

### □ 上流工程での遅延は下流へ直接影響する

- 例えば、要求仕様書の完成が1週間遅れたとする  
→ 外部設計はその後に開始となるので、当然1週間遅れる
- ちょうど一列で順番待ちしているようなもの  
前の人で時間がかかれば、後ろの人は待たされる

上流工程の完了を待つので、  
下流はその影響を受ける

納期に間に合わないとか、技術者のスケジュール調整に影響することも

上流(じょうりゅう) : upstream

下流(かりゅう) : downstream

遅延(ちえん) : delay

直接影響する(ちよくせつえいきょうする) : directly affect

順番待ち(じゅんばんまち) : waiting queue

完了(かんりょう) : completion



## ウォーターフォールモデル：問題点(3)

### □ 上流へさかのぼる修正・やり直しには不向き

- 各時点で「上流工程は完了している」のが前提
- しかし、現実には「上流へ戻ってやり直し」も多い  
(初めて作るシステムならば試行錯誤は必然的)

実際のソフトウェア開発にそぐわないモデル  
になってしまうことも → コスト増大の要因

例えば、「実装」段階だと、その時点では既に「要求分析」チームが解散していたり、他のプロジェクトに取りかかっている場合もありえる。  
もしもそれが外注だったら、追加契約が必要になるかもしれない。

さかのぼる: go back to

やり直し(やりなおし): redo

不向き(ふむき): not suitable

試行錯誤(しこうかくご): trials and errors

必然的(ひつぜんてき): inevitable

そぐわないモデル: improper model

増大(ぞうだい): increase

要因(よういん): cause

外注(がいちゅう): outsourcing

追加契約(ついかけいやく): additional contract

## ウォーターフォールモデル：問題点(4)

### □ テストが後半にならないと実施されない

- テストは「実装」が完了した後に初めて実施される
- テストで不具合が見つかり、当然「やり直し」が必要であり、前出の問題点(3)につながる

作った後で正常な動作を「確認する」ことに注目  
していて、不具合が見つかって大幅なやり直しと  
なることはモデルの本流にない

要求分析と設計が正しかったのかどうかは、実装が終わった後で初めて  
確認・検証されるというモデルである。  
そもそも各時点でその上流まではうまくいっているという思想が根底にある。

後半(こうはん) : late stage

実施(じっし) : carry out, perform

大幅なやり直し(おおはばなやりなおし) : major revision

本流(ほんりゅう) : mainstream

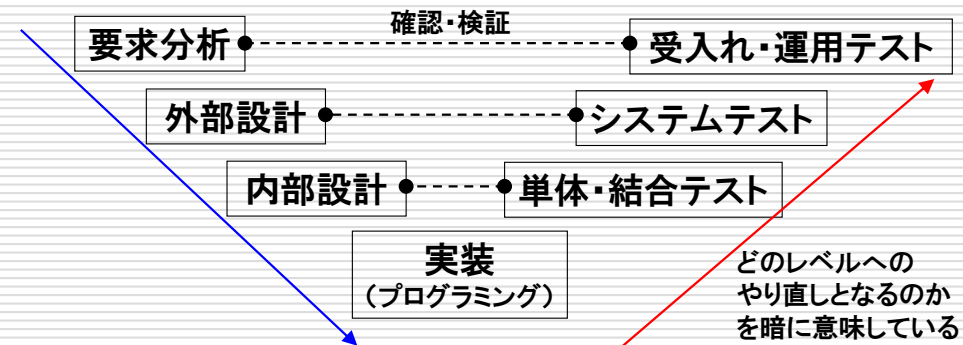
確認(かくにん) : check

検証(けんしょう) : verification

思想(しそう) : thought

# Vモデル

- 本質はウォーターフォールモデルと同じ
- テスト工程を詳細化し, 上流工程と対応付け



(C) 2007-2023 Hirohisa AMAN

37

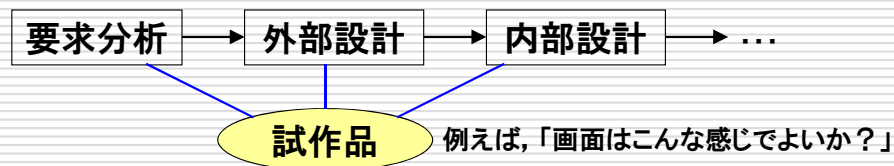
本質(ほんしつ) : essence

詳細化(しょうさいか) : elaboration, refinement

## プロトタイピング(prototyping)

□ 試作品(プロトタイプ)を作ることによって確認をしながら開発プロセスを進めていく手法

- 試作品は確認用のモックアップでもよい
- 仕様や設計の確認・評価に使う



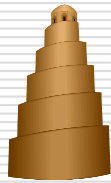
試作品(しさくひん) : prototype

モックアップ : mock-up

## スパイラル(spiral)モデル

---

- ウォータフォールモデルと  
プロトタイピングを包含したモデル



※spiral = らせん

ぐるぐると同様の工程を  
繰り返しながら発展していく

- リスク駆動型の管理プロセス

大規模プロジェクトに向いている

(C) 2007-2023 Hirohisa AMAN

39

包含する(ほうがんする) : include

リスク駆動型(リスクくどうがた) : risk-driven

# スパイラルモデルの基本サイクル

## ①目標設定

- その工程で作りたいもの(目標)を設定
- 目標に合った手段を(なるべく多く)検討

## ②評価

- 各手段の良さやリスクを評価
- プロトタイプによる確認

## ④次の計画

- 次の繰り返しで行うべき工程について計画

## ③作成・実行

- 実際に文書・システムを作成
- テストを実行

(C) 2007-2023 Hirohisa AMAN

40

目標設定(もくひょうせってい) : goal setting

評価(ひょうか) : evaluation





作成(さくせい) : making, developing

実行(じっこう) : carry out, perform

繰り返し(くりかえし) : iteration

## スパイラルモデルの直感的イメージ

---

- 大きな流れはウォーターフォールモデルであるが、いきなり完成品を作るのではない
- 各工程で確認・リスク評価を行いながら、失敗のリスクを下げていく
  - 第1サイクル:  要求仕様
  - 第2サイクル:  外部設計
  - 第3サイクル:  内部設計
  - 第4サイクル:  実装・テスト

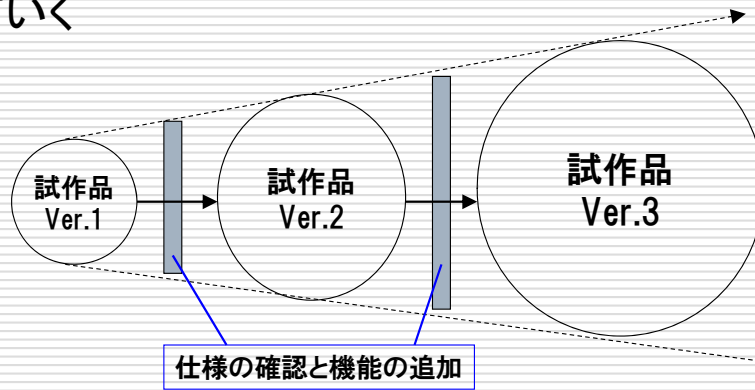
直感的(ちょっかんてき) : intuitive

イメージ : image

## その他の手法・モデル

### ①進化型プロトタイピング

□ 試作品を徐々に発展させて完成品へと仕上げていく



(C) 2007-2023 Hirohisa AMAN

42

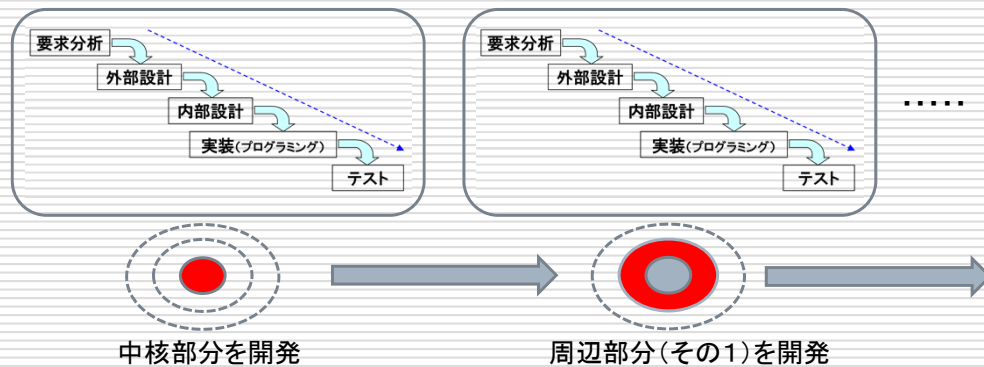
徐々に発展(じょじょにはってん) : gradually evolve



## その他の手法・モデル

### ②インクリメンタル開発モデル

- 対象システムをいくつかの部分に分け, 主要部分から順に作り上げていく



(C) 2007-2023 Hirohisa AMAN

43

主要部分(しゅようぶぶん) : major part

中核部分(ちゅうかくぶぶん) : core part

周辺部分(しゅうへんぶぶん) : surrounding part

## まとめ

---

- まずは**開発計画**と**要求分析**が重要
- そして、**外部設計**(サブシステム設計), **内部設計**(モジュール・関数設計), **実装**, **テスト**, **運用・保守**と進んでいく
- 開発プロセスのモデル
  - **ウォーターフォールモデル**:「上から順番に」
  - **プロトタイピング**:「プロトタイプを投げ所」に」
  - **スパイラルモデル**:「評価・確認を繰り返す」

## 宿題(homework)

---

**“[03] quiz” に答えなさい  
(明日の 13 時まで)**

Answer “[03] quiz”  
by tomorrow 13:00pm

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

## 【演習1】（解答例） 食堂の混雑チェックシステムについて

---

- ☐ どういうやり方でチェックするのか？
  - Web ブラウザでサーバへアクセスする？
  - アプリにサーバと通信させて状況を表示させる？
- ☐ 「混雑」の状態をどう表現するのか？
  - 座席の空き状況？（席の占有率やマップを表示）
  - 直近20分間のレジ通過者数？
- ☐ 使用可能なスマホに制約はあるか？
- ☐ 同時にサービスを利用できる人数の上限は？

---

(C) 2007-2023 Hirohisa AMAN

10

座席(ざせき) : seat

占有率(せんゆうりつ) : occupancy

レジ = cash register

制約(せいやく) : constraint

## 【演習2】（解答） 該当する工程を答えよ

①混雑度データは、測定時刻（日、時、分）と座席使用率（％）の構造体データとし、3分間隔でファイルへ記録していくことにした

【内部設計】システムの設計で、データ構造にまで言及している

②本稼動前に同時に1000人分のアクセスを実行させたり、営業時間外にアクセスさせたりして挙動を確認した

【テスト】システムの事前の動作確認で、高負荷状態や例外的な使い方を試している

③システムの構成を  
（1）混雑度データの管理部  
（2）ユーザインタフェース部  
（3）制御（コントローラ）部  
の3つに分けて開発することとした

【外部設計】システムの設計で、大まかなサブシステム構成を決めている

④利用者アンケートを実施し、改善すべき案件を整理した

【運用・保守】運用後にシステムの発展・向上を目指した活動を行っている

(C) 2007-2023 Hirohisa AMAN

28

高負荷（こうふか）: high load

例外的な使い方（れいがいてきなつかいかた）: exceptional use case

発展（はってん）: evolution

向上（こうじょう）: improvement

活動（かつどう）: activity