初級者向けテキスト

組込みソフトウェア管理者・技術者育成研究会
ーSESSAMEー
(http://www.sessame.jp/)

-*- 目　　　　　次 -*-

-*-

ソフトウェアテストの概要

西 康晴

１．SESSAMEの紹介およびコースの概要
２．開発課題と失敗事例の解説
３．組込み向け構造化分析の例・設計の概要(1)
４．組込み向け構造化分析の例・設計の概要(2)
　実習/回答と補足説明
５．組込み向け構造化設計(1)
６．組込み向け構造化設計(2) 実習/回答と補足説明

７．プログラミング　組込み用語基礎知識
８．ソフトウェアテストの概要
９．プログラミング実習への説明
１０．プログラミング　実習
１１．プログラミング　実習の回答と補足説明
１２．ソフトウェアテスト　実習
１３．ソフトウェアテスト　実習/回答と補足説明

付録．話題沸騰ポットのシミュレーション

目次

- テストは大事
  - バグの影響はとんでもなく大きいのだ
  - テストが上手になるとバグの無い開発ができる
- テストの進め方
  - テストのフェーズとプロセス
  - テストに必要な視点
- テストの設計
  - 単体テスト・結合テスト・機能テスト・システムテスト
- テストしやすい開発をしよう
  - テストが上手になるとバグの無い開発ができる

バグの影響はとんでもなく大きいのだ

- 300万行のうち1行余計だったせいで、AT&Tは11億ドルの損害
  - エラー回復コードがバイパス
  - 長距離電話が9時間に渡り話し中になった
- ブレーキシステムのバグで、GMにリコール？
  - 停車距離が15～20m伸びてしまった
  - 350万台がリコール、数百億ドルの損害
- ARIANE5ロケットは爆発
  - オーバーフローが原因
  - 4億ドルの損害

219

SESSAME CONTENTS 2004

テストは簡単？

- テストなんて、仕様を確認するだけじゃないか
- ちゃんと作れば、テストなんていらない
- テスターは、力の無い奴にやらせればいい
- テスターはあら探しをするからムカつく

220

SESSAME CONTENTS 2004

テストとデバッグは同じじゃないの？

- デバッガ上で動かしてみるのはテストではない
  - バグを「いぶり出す」ために知恵を捻って設計すべし
  - 動きやすいテストをしてはダメよ
  - バグが見つかって始めてテストは成功なのだ

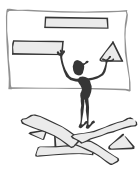
テストが上手になってくると
そもそもバグのない開発ができるようになる

221

テストはどうやって進めればいいの？

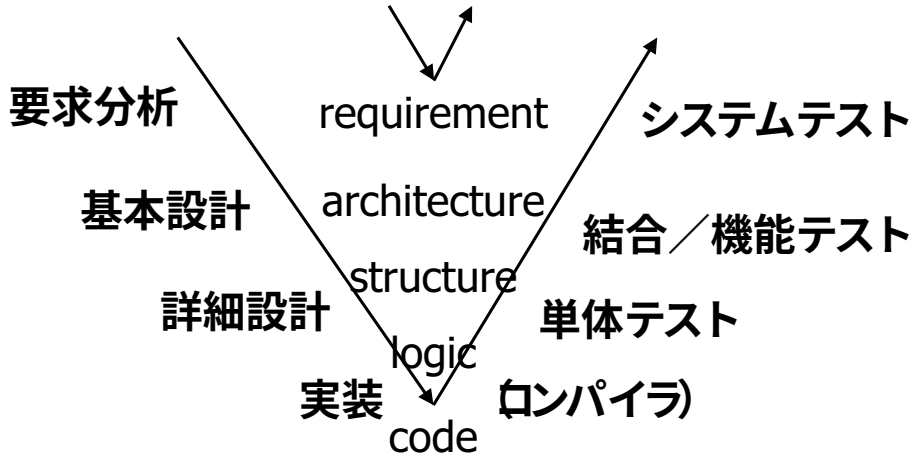
- 組込みソフトのテストのフェーズ
  - 下位Vモデル
    - 単体テスト／結合テスト／機能テスト／システムテスト
  - 上位Vモデル
    - シミュレータテスト／実機テスト／シミュレータテスト
  - リグレッションテスト（回帰テスト）
    - 修正や変更の副作用で入り込んだ新たなバグも
      きちんと見つけて取り除こう

行き当たりばったりではなく
きちんと「設計」しよう



222

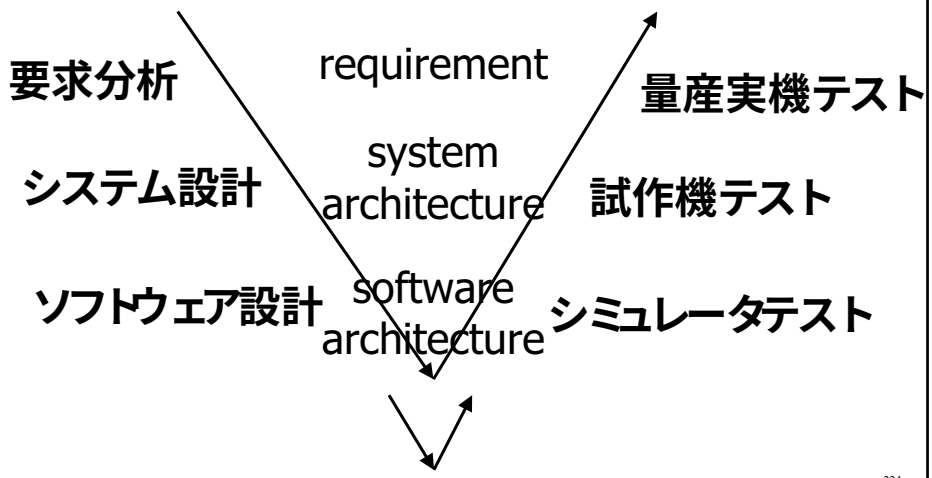
組込みソフトのテストのフェーズ：下位Vモデル



223

SESSAME CONTENTS 2004

組込みソフトのテストのフェーズ：上位Vモデル



224

SESSAME CONTENTS 2004

テストを設計する時には何を意識すればいいの？

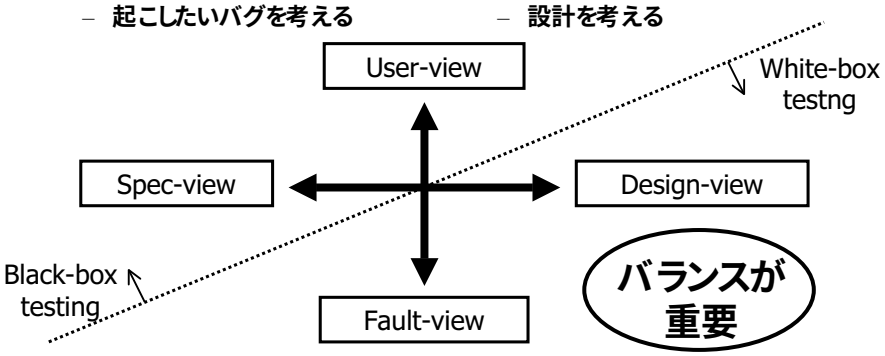
- 網羅
  - テスト漏れにはバグが潜んでいるかもしれない
    - どれくらい網羅したかを測る :カバレッジ
      - 制御パステスト
      - 機能網羅テスト
- ピンポイント
  - バグの多そうなところを狙う
    - 先人の知恵
      - 境界値テスト
      - ストレス系テスト
    - 過去の経験
      - バグの分析をして自分の弱いところを把握しよう

期待結果を
必ず記述すること

225

テスト設計に必要な視点

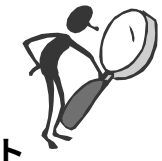
- User-view
  - ユーザが何をするかを考える
- Fault-view
  - 起こしたいバグを考える
- Spec-view
  - 仕様を考える
- Design-view
  - 設計を考える



226

モジュールをテストしよう:単体テスト

- どんなテスト?
  - 開発者が行うテスト
  - モジュールレベルのテスト
  - 詳細設計やプログラミングに着目したテスト
- どんな手法でテストするの?
  - 境界値テスト
    - モジュールの引数などに与えるテストデータを はじっこの値 境界値)にするテスト
  - 制御パステスト
    - モジュール内のロジックを全て通すように テストデータを与えるテスト



両方とも必要!

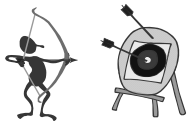
境界値にはバグが多い

- 境界値や限界値の周りにはバグがたくさん潜んでいる
  - 条件文の間違い:どんな間違いが多い?
    例) ○:if (a>0) then　　×:if (a≧0) then
    のように間違えることが多い。
    境界値であるa=0でテストすればバグが見つかる
    - if文などの条件文やwhile文などのループには、 境界ずれや一つ違いのバグが多い
  - リソースのリミット:テスト文字列の長さは?
    例) 10バイトの固定長文字列を入力するプログラムの場合 とても長い文字列を入力すれば落ちるかもしれない
    - 大きさを持ったデータ構造には、バッファオーバーフローなど 容量の限界値ギリギリの処理を忘れているバグが多い

バグが多いところを狙おう 境界値テスト

- 同値クラスを見落とすと、ゴソッとテストがモレる
  - 境界値や限界値を抽出する範囲を「同値クラス」と呼ぶ
    - モジュールの引数、返値の境界値
    - if文の条件式など内部の境界値
    - テンポラリファイルの容量などI/Oの境界値
  - 正常境界値だけでなく異常境界値もテストする
  - 境界値を考えるのをサボっちゃダメよ

どんなときでも
境界値を考えよう



229

境界値テストの練習をしてみましょう

- 以下のモジュールのテストを設計してみましょう
  - int型の引数aがあります
  - モジュール内にはif (a<0) とif (7<a)という2つの条件文があります

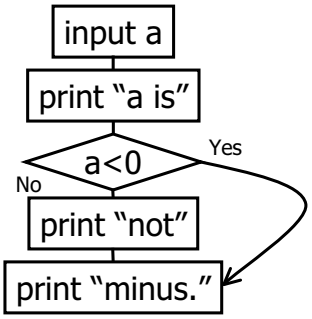
- まず同値クラスを挙げてみましょう。
- 次に同値クラスの境界値を挙げてみましょう
- 最後に、どんなバグが見つかるかを考えてみましょう

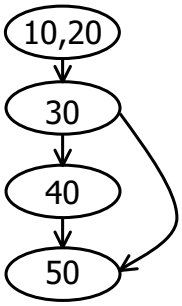
230

ロジックを網羅しよう:制御パステスト

- ロジックを網羅して全てきちんとモレなくテストしよう
  - フローグラフを描いてロジックを抜き出す
    - フローグラフ:ノードとリンクで書かれた図
    - フローチャートや状態遷移図はフローグラフ

```
10  input a
20  print "a is"
30  if (a<0) goto 50
40  print "not"
50  print "minus."
```

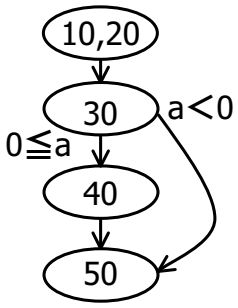




231

制御パステストの設計 :パスの網羅

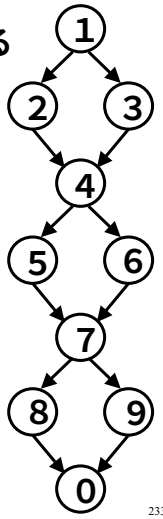
- パスとは？
  - フローグラフ上の経路
  - プログラムのロジック
  - パスの数がテストの数に比例する
- パスを全て網羅するように
  テストを設計する
  - まずパスの一覧表を作る
    - 10→20→30→40→50
    - 10→20→30→50
  - 次にパスを通るデータを実装する
    - 10→20→30→40→50: a=0
    - 10→20→30→50 : a=-1



232

どこまでテストすればいいのかな？

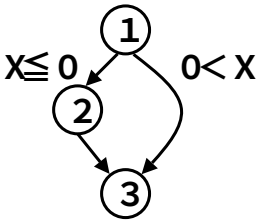
- 全てのパスを組み合わせようとすると膨大になる
  - if文の数の累乗で増えていく
  - if文に含まれるandやorの数の累乗でも増えていく
    - 右のグラフでは8本のパスが必要となる
- 最低でもif文の両側の分岐は1度テストしよう
  - 条件網羅（C1基準）
- 組み合わせが増えないように気を付けて開発する必要がある
  - KISS: Keep It Simple, Stupid!



参考：命令網羅（C0基準）

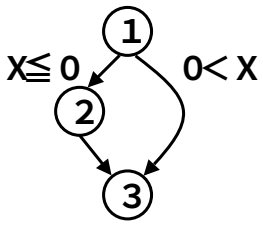
- どれだけ網羅できているか、を「カバレッジ」と呼ぶ
  - フローグラフのカバレッジ
  - 機能カバレッジ
- 全ての命令を一度実行すればいいような気がする
  - 命令網羅／ノード網羅／C0基準
- 右のフローグラフでは
  テストケースは1つでよい？

  ┌A：1→2→3
  └　　X＝0
- 明らかにテストが足りない
  - if (9<x) とタイプミスしていても分からない



参考 条件網羅（C1基準）

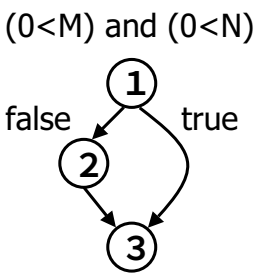
- 少なくとも条件文のtrueとfalseは網羅しなくてはならない
  - 条件網羅／リンク網羅／C1基準
  - 右のフローグラフでは、
    テストケースは以下の 2つになる
    - A：1→2→3
      X＝0
    - B：1→3
      X＝1



  - if (9<x) とタイプミスしていても分かる
    - 両方とも左に分岐するのでバグであると分かる

参考 複合条件網羅（C2基準）

- 組み合わせていない単一の条件式を
  すべて網羅しないとテストできたことにならない
  - 右のフローグラフでは、
    テストケースは以下の 4つになる
    - A：1→2→3　（falseの場合）
      (M＝-1, N＝-1)
      (M＝ 1, N＝-1)
      (M＝-1, N＝ 1)
    - B：1→3　　（trueの場合）
      (M＝ 1, N＝ 1)

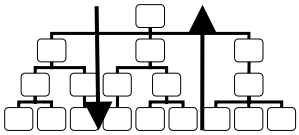


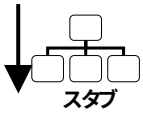
設計をテストしよう:結合テスト

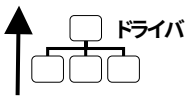
- どんなテスト?
  - 開発者が行うテスト
  - モジュールレベルのテスト
  - 概要設計や構造設計に着目したテスト
    - 状態遷移設計やモジュール間構造設計に着目したテスト
- どんな手法でテストするの?
  - トップダウンテスト/ビッグバンテスト
    - モジュール間の結合のミスを探すテスト
  - 状態遷移パステスト/状態遷移マトリクステスト
    - 状態遷移図や状態遷移マトリクスのミスを探すテスト
  - など

モジュール間のインターフェースのテスト:結合テスト

- モジュールをつなげる時のテスト
  - モジュールの結合順序を決める
  - ドライバやスタブを作成する
  - インターフェースをテストする
- トップダウンテスト
  - 上位のモジュールから先に作成し、結合していく
  - スタブが必要
- ボトムアップテスト
  - 下位のモジュールから先に作成し、結合していく
  - ドライバが必要
- ビッグバンテストは禁止である

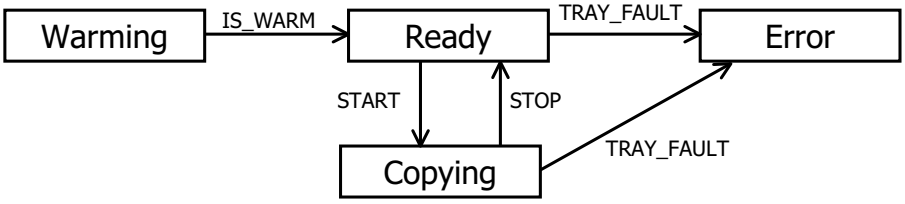


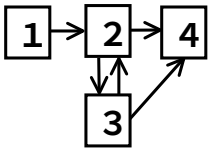




イベントを順番に起こそう :状態遷移パステスト

- イベントを順番に起こしてテストすることで
  状態遷移が正しく実現されているかをチェックしよう





パスの数は何本必要？

- 1→2→4
- 1→2→3→2→4
- 1→2→3→2→3→2→4
- 1→2→3→4
- 1→2→3→2→3→4
- 1→2→3→2→3→2→3→4

239

イベントと状態を網羅しよう :状態遷移マトリクステスト

- 状態遷移マトリクスをモレなくテストすることで
  状態遷移が正しく実現されているかをチェックしよう

何が起こるか分からない
↓
バグの可能性が高い

	S / E	起動処理中 (Warming)	スタンバイ中 (Ready)		
		0	1		
スタンバイ完了 (IS_WARM)	0	→Ready	×	×	×
コピーボタン押下 (START)	1	／	→Copying	?	／
コピー完了 (STOP)	2	×	×	→Ready	×
エラー発生 (TRAY_FAULT)	3	?	→Error	→Error	×

→ 遷移
／ 無視
× ありえない

- 状態遷移マトリクステストを設計すると、
  矛盾や抜けのある遷移もレビューできる

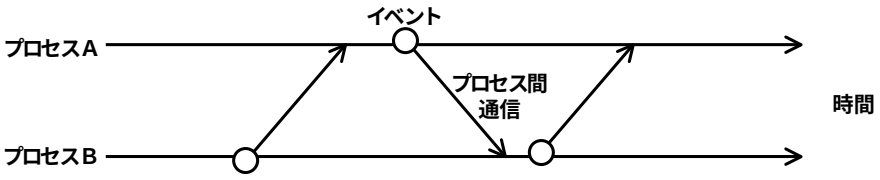
240

状態遷移テストの注意点

- 状態遷移パステストと状態遷移表テストは使い分ける
  - リンク網羅ならばカバレッジとしては等価
  - 状態遷移パステストは、ユーザの操作などと対応させてテストしやすい
  - 状態遷移表テストは、異常なイベントへの対応が網羅的にテストできる
- 状態遷移図そのものの間違いに気を付ける
  - 仕様書をよく読んで検討する
    - 状態や遷移のモレ／あいまいな遷移条件
- 時間に関する状態には気を付ける
  - ある一定時間だけ初期化処理の状態になる、など
  - その状態でいられる最小時間と最大時間を明確にする
- モデルと実装の差異に気を付ける
  - モデル上では独立な状態でも、履歴に依存するかもしれない
  - モデル上では瞬時に遷移することになっているが、実際にはほんの少しでも時間がかかるため、ありえないイベントが発生する

241

パステストの応用 :シーケンス図のテスト

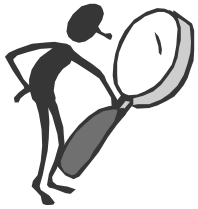


- C0基準 :イベント網羅
  - 一つ一つのイベントがちゃんと発生するか
- C1基準 :イベント間通信網羅
  - 一つ一つのプロセス間通信がちゃんと受け取れるか
- イベント発生可能性時刻の同定
  - 境界値テストで明らかにする
- フローグラフで表現できる設計は全てパステストが可能

242

コンパイル後に改めてテストしよう 機能テスト

- どんなテスト?
  - 開発者だけでなく、テスト担当者や品質保証部門も行うテスト
  - 組み上がったソフトウェアレベルのテスト
  - 機能仕様に着目したテスト
- どんな手法でテストするの？
  - 機能網羅テスト
    - 機能一覧を作って網羅するテスト
    - 簡単そうだが意外にやらない
  - 境界値テスト
    - 機能のパラメータに着目した境界値テスト
    - 仕様のバグが見つかる



243

機能網羅テスト

- ソフトウェアの持つ機能を漏れなくテストする
  - 表を作って機能を全て書き、OKになったらチェックする
    - 基本中の基本だが面倒がって行わないことが多い

機能	*チェック欄*
機能１	レ
機能２	
機能３	レ
機能４	
機能５	レ
機能６	

244

マトリクス網羅テスト

- 2次元の組み合わせを網羅する場合のテスト
  - 機能×データ)、状態×イベント)など
    - 行と列の組み合わせが網羅されるかをチェック

	条件A	条件B	条件C	条件D
条件1	レ			
条件2			レ	
条件3	レ			
条件4				レ
条件5		レ		
条件6				

    - ありうる組み合わせ」が網羅されるかをチェック

	条件A	条件B	条件C	条件D
条件1	○ レ			
条件2	○	○	○ レ	
条件3	○	○ レ		
条件4	○	○	○ レ	○
条件5		○	○	○ レ
条件6				○

245

境界値とバグ

- 境界値や限界値の周りにはバグがたくさん潜んでいる
- バグが多いところを狙ってテストしよう 境界値テスト
- 仕様の境界値をどんどんテストしよう
  - 境界値を考えると、仕様があいまいなことに気付く
    - 4月1日生まれは早生まれ？遅生まれ？
      - 民法第143条 :満年齢は起算日に応当する前日をもって満了する
      - 学校教育法第22条 :満6歳に達した日の翌日以降における最初の学年

**境界値テストは
なるべく上流で設計して
仕様のあいまいな部分の
レビューに使おう**

246

色々いじめてみよう:システムテスト

- どんなテスト?
  - 開発者だけでなく、テスト担当者や品質保証部門も行うテスト
  - ソフトウェアを使うユーザレベルのテスト
  - 要求仕様に着目したテスト
- どんな手法でテストするの？
  - ストレス系のシステムテスト
    - ソフトウェアに負荷をかけるテスト
    - よくバグが見つかるので手抜きしないでテストする
  - 環境系のシステムテスト
    - 相性や互換性の問題を見つけるテスト
    - データだけでなく、電気や熱の流れなどにも気を付ける
  - 評価系のシステムテスト
    - どれくらい堅牢か、どれくらいユーザが満足するかの評価
    - ユーザに使わせるだけでなく、きちんと考えて設計する

参考 :ストレス系のシステムテスト

- ボリュームテスト
  - 大きなデータ、たくさんのデータを与えるテスト
    - 巨大な表のあるページで某Webブラウザがクラッシュ
- ストレージテスト
  - ディスクやメモリなどを残り少ない状態で使うテスト
    - メモリ12Mで動かしたら某OSがブルーサンダー
- 高頻度テスト
  - 短時間にたくさん処理させたり同時に処理させるテスト
    - ある種のパケットを大量に投げると某Webサイトが停止
- ロングランテスト
  - 長時間実行させるテスト
    - 某ターミナルエミュレータは長時間の使用でブルーサンダー

参考 環境系のシステムテスト

- 構成テスト
  - 一緒に利用している他のソフトやハード「から」悪影響を与えられる問題を検出するテスト
    - 某ビデオドライバをインストールするとテスト対象であるWindowsが起動しなくなる
- 両立性テスト
  - 一緒に利用している他のソフトやハード「に」悪影響を与える問題を検出するテスト
    - テスト対象である某Webブラウザをインストールすると某ワープロが起動しなくなる
- 互換性テスト
  - 他のソフトやハードとデータなどの交換をさせるテスト
    - 某ワープロで枠を作ると、異なるバージョンでは位置ズレを起こす

249

参考 評価系のシステムテスト

- 障害対応性テスト
  - 電源を抜くなどの障害を起こして復旧性などを評価するためのテスト
    - 某PCは電源を抜いてスリープさせるとHDDがクラッシュ
- セキュリティテスト
  - 機密保護などの穴を突いてセキュリティを評価するためのテスト
    - 某ウィルスチェッカは、特定のURLで管理機能を奪取できる
- ユーザビリティテスト
  - 操作性や視認性などを評価するためのテスト
    - 某統計ソフトのインストーラは、OKとCancelが逆の場合があり、インストール不能だとユーザに判断されてしまう
- ユーザ操作テスト
  - ユーザが満足しているかを評価するためのテスト

250

Testability の高い開発を意識しよう

- テストが上手になると、そもそもバグのない開発ができるようになる
  - テストを設計すると、構造を見直したり曖昧なところを明確にすることになるので、気付かなかったバグに気付くようになる
  - テスト設計が簡単なプログラムは、設計や実装もシンプルなので、バグが入り込みにくくなる
  - テストが実施しやすいプログラムは、実施できるテストの量も増えるので、バグが見つかりやすい
- テストが容易な製品設計を、テスト容易化設計と呼ぶ
  - Testability DesignやDFT (Design For Test)とも呼ぶ
  - プログラムの内部に依存関係があると、その分だけ組み合わせが必要になるので、きちんとモジュール化を行う
  - プログラムの出力や内部状態の変化が分からないとバグを見逃してしまうので、なるべく簡単に分かるようにしておく

251

まとめ

- テストをする時に意識すべきこと
  - 網羅：モレなくテストする
  - ピンポイント：バグが多いところを狙ってテストする
- テストとは
  - 知恵を絞って設計すべし
  - バグが見つかって初めてテストは成功なのだ
  - いろいろなテスト手法がある
- テストが上手になってくると、そもそもバグのない開発ができるようになる

テストを意識して開発することで
はじめからバグの無いソフトを作ろう

252

ソフトウェアテストの概要
～話題沸騰ポットに対するテストの実践～

大野 晋

１．SESSAMEの紹介およびコースの概要
２．開発課題と失敗事例の解説
３．組込み向け構造化分析の例・設計の概要(1)
４．組込み向け構造化分析の例・設計の概要(2)
　実習/回答と補足説明
５．組込み向け構造化設計(1)
６．組込み向け構造化設計(2) 実習/回答と補足説明

７．プログラミング　組込み用語基礎知識
８．ソフトウェアテストの概要
９．プログラミング実習への説明
１０．プログラミング　実習
１１．プログラミング　実習の回答と補足説明
１２．ソフトウェアテスト　実習
１３．ソフトウェアテスト　実習/回答と補足説明

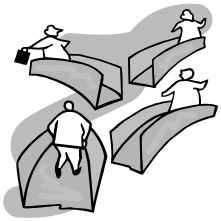
付録．話題沸騰ポットのシミュレーション

テストは難しい

- 通常、プログラム開発の工数の半分以上をテストに費やしている
- しかもその大半をテストの実施ではなく、デバッグに費やす
- テスト時間のほとんどは様々な原因のテストのやり直し
- 体系的にテストを計画し、実践することで工数の大半を削減できる
- しかし、実際には行き当たりばったりにテストが実施され、時間が浪費される

立場によってもテストの内容が違ってくる

- プログラム作成者のテスト:意図した通りかどうかの確認
- テスターのテスト:バグのたたき出し
- 品質保証（QA)のテスト:品質の確認、完成の最終確認



255

プログラム作成者のテスト

設計者としてできたものの動作を保証するためにしていたテストは。

⑴ 共用ルーチン、マクロなど共有物（部品）のテスト
⑵ モジュールレベルの単体テスト（C0,C1：100%はマナー！）
⑶ モジュールを組みたてた組み合わせテスト（機能仕様書、詳細仕様書レベル）
⑷ 全部組み立てた機能テスト（機能の動作確認）
⑸ パラメタを組み合わせた組み合わせ条件のテスト
⑹ 境界条件テスト
⑺ エラーケースのテスト
⑻ 限界テスト
⑼ 他のプログラムなどとの組み合わせテスト
⑽ 使用条件を考えたユーザイメージの機能評価

256

テスターのテスト

デバッグを目的としたテスト
⑴ 全体の機能に対する機能テスト
⑵ パラメタを組み合わせた組み合わせ条件のテスト
⑶ 境界条件テスト
⑷ エラーケースのテスト
⑸ 限界テスト
⑹ 他のプログラムなどとの組み合わせテスト
⑺ 使用条件を考えたユーザイメージの機能評価

257

品質保証（QA）としてのテスト

限られた時間内でプログラムの品質を保証するための検査
⑴ 主力機能をサンプリングした機能テスト（機能の確認）
⑵ パラメタを組み合わせた組み合わせ条件のテスト
⑶ 境界条件テスト
⑷ エラーケースのテスト
⑸ 限界テスト
⑹ 他のプログラムなどとの組み合わせテスト
⑺ 使用条件を考えたユーザイメージの機能評価
で、特に ⑶ ⑷ ⑸ ⑹ ⑺ は大切。
しかし、本当に大切なのは、バグが出た場合の要因分析！

258

さて、テストを始めよう:テストの準備

•まず、計画（戦略）を立てる

•そして、テストの準備をする
⇒例えば、ポットのテストにはこんなものが必要！

1. ポットのシミュレータ
2. ポットへの温度計設置
3. センサの作動状態がわかるような仕掛け
4. ポットのデバッグ環境

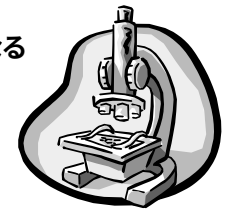
259

テストケースを作る（1）

- 機能の一覧を作成する
- それぞれの機能の同値クラス分析と境界値分析を行う

同値クラスとは…
「2つのテストを実行して同じ結果を期待するとき、2つは同値であるという。」（Kaner他）

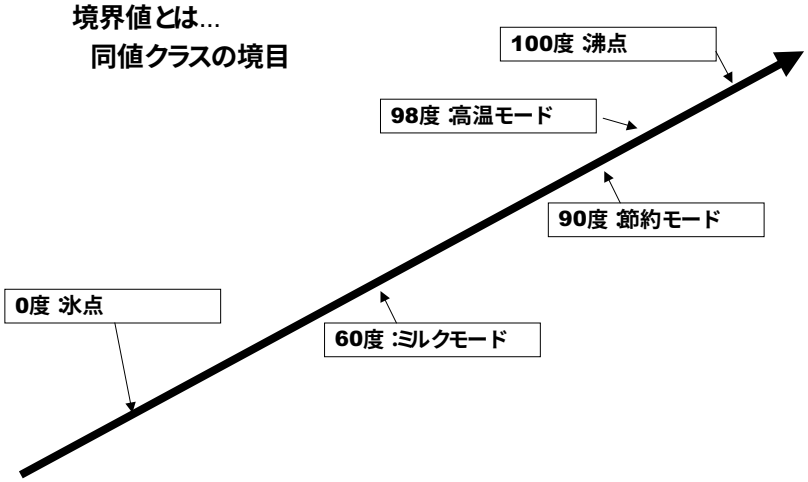
例えば、温度制御のテストを考える場合
•水温：10度、20度、36度、50度は同値クラスになる



260

テストケースを作る (2)

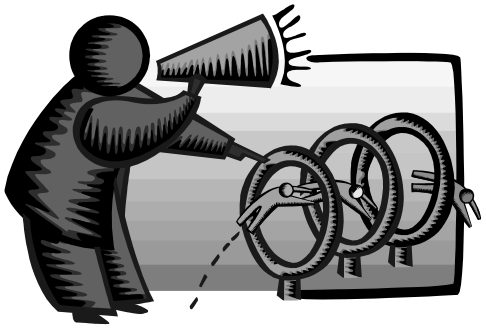
境界値とは...
同値クラスの境目



261

SESSAME CONTENTS 2004

さあ、各モードのテストケースを挙げてみよう



262

SESSAME CONTENTS 2004

テストケースを作る（3）

- 保温モード:高温
  - 100度
  - 98度
  - 95度
  - 80度
  - 0度
- 保温モード:節約
  - 100度
  - 98度
  - 90度
  - 80度
  - 0度
- 保温モード:ミルク
  - 100度
  - 98度
  - 60度
  - 30度
  - 0度

ほんの1例。

どこまで、確認するかでリストアップする温度も増えていく!

テストケースを作る（4）

テストケースを作るには

- 機能を同値分析、境界分析する
- 抽出した機能のテスト値を組み合わせてテストケースを作成する
- 実験計画法の直交表を用いることで、テストケースを最適化できることもある

テストケースを作る（5）

機能から視点を変えてテストケースを考える

- エラー処理
- 性能
- 割り込み
- Ｉ／Ｏ
- 温度
- スイッチの種類とタイミング
- 過去の失敗

など

これらを同値分析、境界値分析などを行ってテストケースを作成する

テストを測る

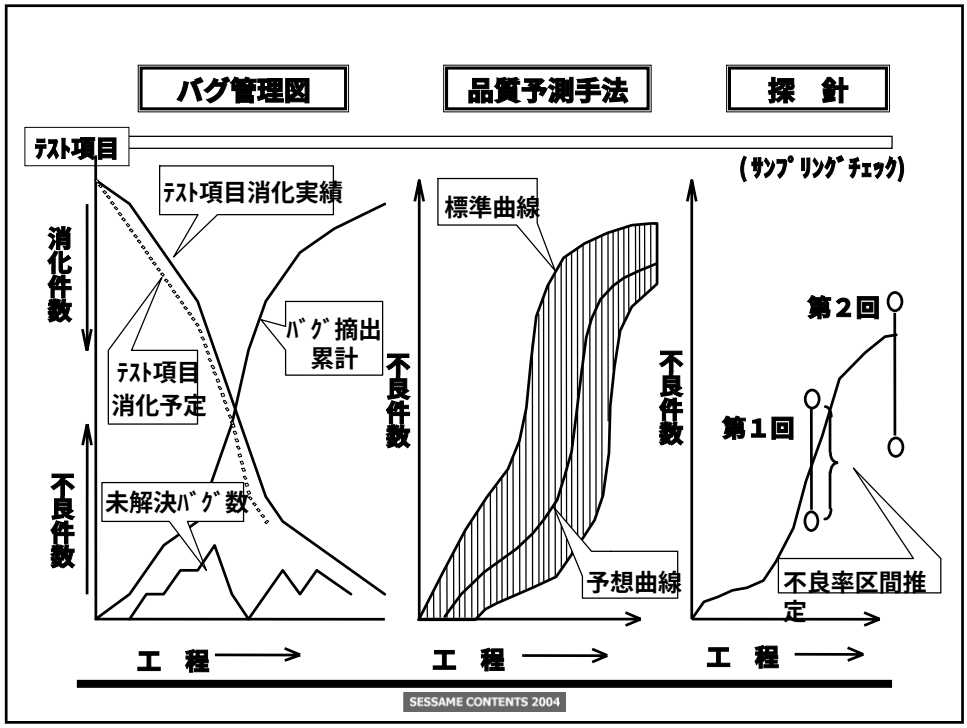
- テストの結果をまとめることでいろいろなことが見えてくる
- テストのまとめ方には

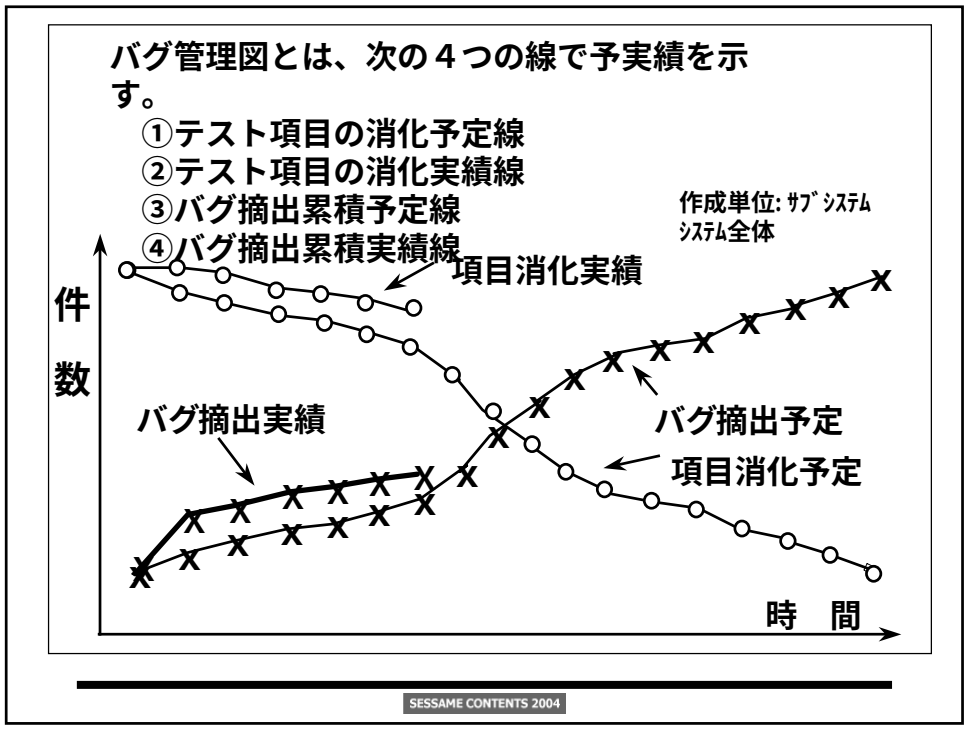
✧ バグを1件ずつ分析するためのレポートを書く

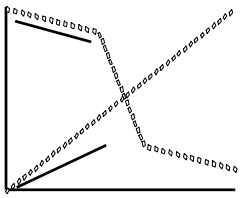
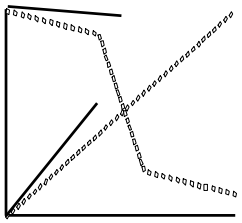
✧ レポートの一覧をもとにバグの傾向を分析する

✧ バグの摘出とテストの進捗からプログラムの品質を分析する

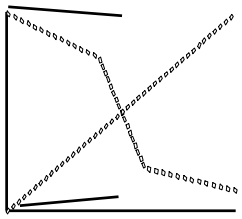
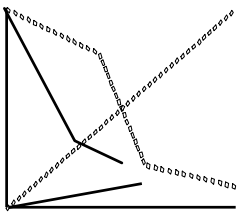
⇒次にテストの進捗傾向の見方の例を示そう


バグ管理図
品質予測手法
探　針
ﾃｽﾄ項目
(ｻﾝﾌﾟﾘﾝｸﾞﾁｪｯｸ)
ﾃｽﾄ項目消化実績
消化件数
ﾊﾞｸﾞ摘出累計
ﾃｽﾄ項目消化予定
不良件数
未解決ﾊﾞｸﾞ数
標準曲線
不良件数
予想曲線
不良件数
第２回
第１回
不良率区間推定
工　程
工　程
工　程
SESSAME CONTENTS 2004


バグ管理図とは、次の４つの線で予実績を示す。
①テスト項目の消化予定線
②テスト項目の消化実績線
③バグ摘出累積予定線
④バグ摘出累積実績線
作成単位: ｻﾌﾞｼｽﾃﾑ
ｼｽﾃﾑ全体
項目消化実績
件数
バグ摘出実績
バグ摘出予定
項目消化予定
時　間
SESSAME CONTENTS 2004

項番	組合せ		バグ管理図	考えられる問題要因
	項目消化	バグ摘出		
1	Ｐ１ 正常	Ｂ１ 正常		特になし ＊架空の報告の可能性あり
2	Ｐ２ 消化停滞	Ｂ３ バグ多発		①バグの作りこみ量が多い ②仕様変更、中途追加でバグを作り込み

バグ管理図の見方（その１）

項番	組合せ		バグ管理図	考えられる問題要因
	項目消化	バグ摘出		
3	Ｐ２ 消化停滞	Ｂ２ 摘出不足		①デバッグ／テスト環境の不備 ②開発要員不足 ③同一テストをしている
4	Ｐ３ 急激消化	Ｂ２ 摘出不足		①テスト項目の質が低い ②テストの実施が一部の機能に偏っている ③品質が良い（？）

バグ管理図の見方（その２）

テスターの視点 (1)

普通のテストはこのように進んでいく。
⇒ただ、バグを良く見つける人と見つけられない人がいる

- 良いテストができるひと
  - 気のつく人？
  - 気配りのできる人？
  - よく仕様を知っている人？
  - 品質保証部の人？
  - お客さん？
  - 心配性の人？

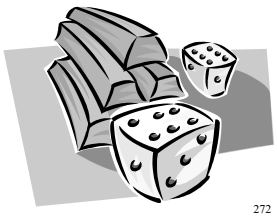
271

テスターの視点 (2)

単純に「目に見える仕様」をテストしてもバグはなかなか見つからない！

⇒例えば、「話題沸騰ポット」をテストするとしたら、どのようなポイントをテストするだろうか？

仕様書からこんな点をテストしたいと思うポイントを挙げてみよう！



272

テスターの視点 (3)

(1)気圧が低く、沸点が下がっている場合
(2)高温の水を沸かす
(3)低温の水 (氷)を沸かす
(4)水量センサーの誤作動 :空瓶センサーがoff
(5)センサー面での水のゆれ
(6)蓋を開けるタイミング :ヒーターの切り替えと蓋の開け閉め
(7)加熱中の水量変化
(8)水以外の液体
(9)外気温の影響
(10)ボタンの同時押し
(11)周波数による違い
⇒本来、考えられるべき事項 (文章にならない仕様)
⇒良いテストとは、文章以外の仕様を見つけ、それを確かめること
⇒特に繊細な制御をするプログラムの場合、制御に抜けが生じやすい

273

テスターの視点 (4)

- 過去のバグを見直すことで、考えておくべき視点を知ることができる
- バグレポートは知識の宝庫！
- 蓄積がない場合には、書籍の付録のバグ一覧が有効！
- 「書いてある仕様」でプログラムが間違っていることは本当に少ない！
- 「書いていない仕様」で問題を抱えていることは多いが、バグなのか、その通りで良いのか、の判断は難しい！

274

まとめ

- ソフトウェアのテストのやり方は単一のパターンではなく、目的によってやり方が異なる
- テストの視点とは、設計者と同じ視点である

テスターの言い分：

- 良いテスターの視点を設計の早い時期に役立てることで効率よく、品質の良いプログラムが出来上がる
- だから、テストでバグ出しをするよりもレビューや机上見直しが効率的！

275

SESSAME CONTENTS 2004

余白）

276

SESSAME CONTENTS 2004

OpenSESSAME Seminor

組込みソフトウェア技術者・管理者向けセミナー

初級者向けテキスト

2002 **年** 10 **月** 15 **日 初版 第** 1 **刷発行**
2003 **年** 10 **月** 29 **日 初版 第** 2 **刷発行**
2004 **年** 3 **月** 19 **日 第** 2 **版 第** 1 **刷発行**
2004 **年** 4 **月** 30 **日 第** 3 **版 第** 1 **刷発行**
2004 **年** 6 **月** 17 **日 第** 4 **版 第** 1 **刷発行**

著 者 上原慶子、大野晋、坂本直史、鈴木圭一、須田泉、西康晴、
二上貴夫、三浦元、三宅貴章、森孝夫、山田大介、山崎辰雄

編集・発行 組込みソフトウェア管理者・技術者育成研究会
（SESSAME）
http://www.sessame.jp
