10年後のソフトウェアテスト技術
~日本発のブレークスルーを生み出そう！~

(株)日立製作所
ソフトウェア事業部
居駒幹夫

© Hitachi, Ltd. 2009. All rights reserved.

本講演を是非聞いていただきたい方々

活 推 研

- ソフトウェアテスト技術を日頃使用しているが、思ったような成果が出ない。 活
- ソフトウェアテスト技術を推進しているが、技術的に閉塞感がある。 推
- ソフトウェアテスト技術を研究しているが、現場での問題についてよく分かっていない。今後の研究の種に困っている。 研

等々、
「現在のソフトウェアテスト技術に満足されていない方」を主対象

© Hitachi, Ltd. 2009. All rights reserved.

講演の目標

- ソフトウェアテストの現在、2009年の技術的な壁を知る
- 技術的な壁をブレークスルーするために、
  - 技術的にどのような方向性が必要かを知る
  - テスト技術の研究者、推進者、実務者の立場からどのようにすればブレークスルーが実現するかのヒントを得る

© Hitachi, Ltd. 2009. All rights reserved.

講演の流れ

- ソフトウェアテスト技術の壁
  - ソフトウェアテストの取り巻く環境
  - 現状の主要テスト技術の限界、その理由
- テスト技術の壁を破るためには？
  - 技術的な観点でのブレークスルー案
  - 日本の強みを活かしてどのようにブレークスルーを実現するか

© Hitachi, Ltd. 2009. All rights reserved.

ソフトウェアテストの取り巻く環境

- ソフトウェアテストに対する認識は高まっている
  - 多くのテスト技術が多くの現場で使われ始め、成果を挙げている。(ASTER, JaSSTの貢献大)
- 社会において、ソフトウェアは信頼されているか？
  - ソフトウェアの信頼性は必ずしも高いとは言えない*1。
  - プロジェクトによって、差異が激しい*2
    - 「ソフトウェア開発データ白書」によると、ほとんどの統計データの散布図は極めて広い分散を持っている。
- 品質保証技術、ソフトウェアテストに対して、現状への厳しい評価、将来の改善方向が必要

＊1 西康晴 「ソフトウェア品質保証技術の動向」P10
http://www.jasst.jp/archives/jasst08t/pdf/2.pdf

＊2 IPA SEC 「ソフトウェア開発データ白書」 ISBN978-4-8222-6227-3

© Hitachi, Ltd. 2009. All rights reserved.

テストでは摘出困難なバグ

- 基本的なテストで摘出可能なバグは確実に摘出しているのが前提。
  - ホワイトボックステストのC0、C1
  - ブラックボックステストの直交法等
- しかし、現実に直面するトラブルの多くは基本的なテストでは解決できない。
- こういった、“しぶといバグ(die-hard bugs)”をしっかり認識することが必要
  - 現状のソフトウェアテストの有効度を知ることが可能
  - 新技術を生もうという動機づけになる
  - ソフトウェアテストに関する新技術に対する試金石になりうる。

© Hitachi, Ltd. 2009. All rights reserved.

ダイハード・バグの例

- 性能向上時に、バッファを初期化する処理が誤ってスキップされて、バッファが特定パターンのときのみ異常終了
- 動作しているプログラムに機能追加時、対応したフェイルセーフ用の機能の修正を漏らして異常発生時にダウン
- OSを変更したら、異常事象発生時、システムコールの返す値(errno)が異なり、異常終了
- あるパラメータは「0以上2の32乗-1」の仕様だったが、メモリ容量の関係でテスト不可能。時代は廻り、メモリ容量が増えた時点でバグが発覚
- 顧客が数バージョン前のプログラムを使って作成したDBを新バージョンで使ったら異常終了
- あるライブラリとの組み合わせで、グローバル名称に重なりがあり、誤動作する。

© Hitachi, Ltd. 2009. All rights reserved.

Episode 1　割込み処理のテスト

- 四半世紀前、私が入社2,3年目だった頃の話。
- 通信制御プログラムの割込み処理で、多重割込みを可能にする仕様変更。外部仕様の変更無。
- 変更したところは、Open, Closeのみ。それだけの検査なら一日で終了。
- それで済むとは思わなかったが、何をどう検査すればよいか分からない。
  - 私:「何を検査したらよいか分からない」
  - 上司:「全てやれ！持ち弾を撃ち尽くせ。」
  - 仕方無しに、膨大なテストを実施したら…
- 大変だから、テストしないというのは本末転倒

© Hitachi, Ltd. 2009. All rights reserved.

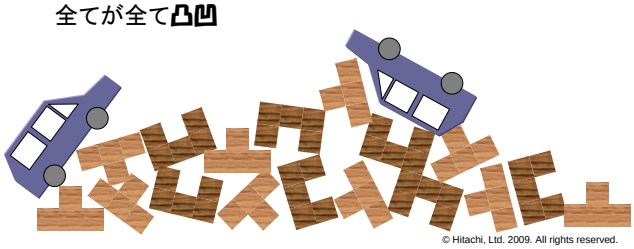
講演の流れ

- ソフトウェアテスト技術の壁
  - ソフトウェアテストの取り巻く環境
  - 現状の主要テスト技術の限界、その理由
- テスト技術の壁を破るためには？
  - 技術的な観点でのブレークスルー案
  - 日本の強みを活かしてどのようにブレークスルーを実現するか

© Hitachi, Ltd. 2009. All rights reserved.

ソフトウェアテストの問題の本質「凸凹」

- **凸凹**
  - 「でっこぼこ」と読んで下さい
  - 本講演では単に「バラツキがある」ということではなく、通常の統計的な知識では扱えないほどバラバラだということ。
  - ソフトウェアの設計、ソース、バグ数、生産性、信頼性等々、全てが全て**凸凹**



© Hitachi, Ltd. 2009. All rights reserved.

脱線:マンデルブローの「禁断の市場」

- (フラクタルで有名な)マンデルブローの研究 *1
  - 自然の営み、人間の営みともに、統計学が仮定しているような分布にはならない。
    - 統計学の分布は、計測する事象が独立、分散または標準偏差が一定であることが前提。
    - 自然の営み、人間の営みともに独立な事象など無い
  - 従って、
    - 千年に一度の大洪水は百年に一度起きるかも
    - 数百年に一度の金融恐慌は数十年に一度起きる かも
- 居駒が知る限り、
  - ソフトウェア関連の全ての統計は、「河川の危険水位」、「市場の株価変動」に比べても、はるかに**凸凹**。

*1 ベノワ・B・マンデルブロ「禁断の市場 フラクタルでみるリスクとリターン 」ISBN-10: 4492654178

© Hitachi, Ltd. 2009. All rights reserved.

凸凹の例:同値分割(1)

- 同値分割でも、本質的に等価なものと、主観的に等価に扱おうというものがある。
  - 「同じだ」
    - 外部仕様から同値であることが明らか
  - 「同じはずだ」
    - 外部仕様からおそらく同じであろう
  - 「同じことにしちゃおう」
    - 外部仕様上でも(少しは)違うことは分かっているのに同値にする。
- 主観的な同値分割はリスクが大きい
  - ソフトウェアテストにおいては、「～のはずだ」で同値にするのは危ない。(次ページの例)
  - さらに、テストケースを減らすためだけの目的に「同値分割」するのは(居駒個人的には)間違っていると考える。

© Hitachi, Ltd. 2009. All rights reserved.

凸凹の例：同値分割(2)

- `void *malloc(size_t size);`
  - 指定された長さのヒープ領域を動的に確保
  - 長さ*size*は、0以上$2^{31}-1$以下の整数とする。
- 上記の仕様の関数をテストする場合、外部仕様は、極めてフラットな仕様だが…実は、あちこちで凸凹
  - 初期確保、再確保
  - 確保する領域長
  - 動作環境（マルチスレッド、シグナル、forkなどでの動作）
  - 永続的な資源の有無（mallocの場合は勿論あり）
  - 実メモリ、カーネルパラメタとの関連
  - OSの実装による差異（OS実装によっては、シグナルが返って来るetc）

© Hitachi, Ltd. 2009. All rights reserved.

凸凹の例：直交表

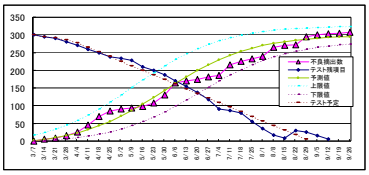
- 直交表 は各因子の関係に凸凹がないことが前提
  - 各因子に対応する機能モジュールが有る場合、各モジュール間の結合度が、すべて同様に低いことが必要。
- 結合度に大きな差があるソフトウェアに対して、無考慮に適用するリスクは大きい。
  - “我々は、多因子の組み合わせについては、（関連深い機能に対してのみ）部分的に多因子間網羅率を向上させることで対処している。” *1
- 余程きれいな設計で無い限り、結合度のばらつきは、かなり有るのでは？

*1 秋山浩一「直交表を活用したソフトウェアテストの効率化」
http://www.jasst.jp/archives/jasst05w/pdf/S4-1.pdf

© Hitachi, Ltd. 2009. All rights reserved.

凸凹の例：信頼性成長曲線(1)

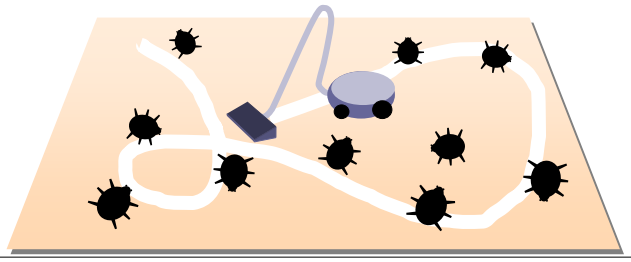
- 多くの信頼性モデルは、下記が前提
  - ソフトウェアのバグは、ソフトウェアの中にランダムに存在する(ポアソン分布)。
  - テストケースはテスト対象のプログラムをランダムに実行できる。
  - 上記におけるバグの発生確率は、その時点に残っているバグ数によって決まる。



© Hitachi, Ltd. 2009. All rights reserved.

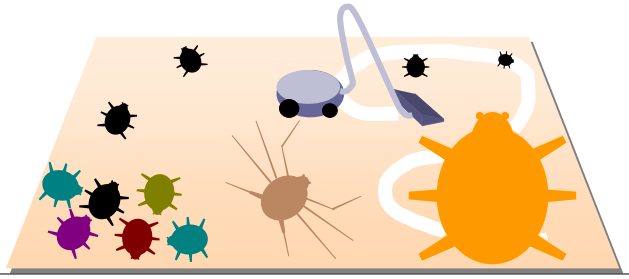
凸凹の例：信頼性成長曲線(2)

- 信頼性成長モデルが仮定しているソフトウェアテスト
  - バグの分布はランダム。発現頻度、影響度は均一
  - 満遍なくテストは実行可能
  - バグ数は単調減少



凸凹の例：信頼性成長曲線(3)

- ソフトウェアテストの実態
  - バグは偏在する。発現頻度、影響度もバラバラ(ie 凸凹)
  - テストできるところしかできない
  - 品質の良いと思っていた箇所にメガバグが登場



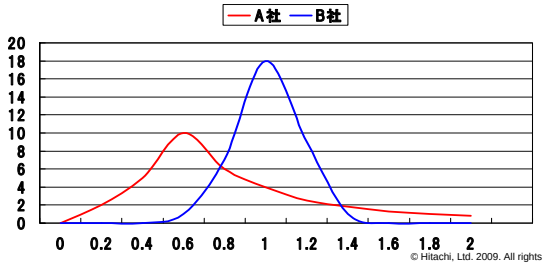
凸凹の例：受入れテストでのバグ密度(1)

- 最終テスト工程でのバグ数は工期遅延、納期遅延につながりやすい。
- 過去の受入れテスト工程でのバグ摘出数の平均
  - A社：　0.8件/KLOC
  - B社：　1.0件/KLOC
- どちらの会社を選ぶか？
  - A社を選ぶべきのように思えるが…

© Hitachi, Ltd. 2009. All rights reserved.

凸凹の例：受入れテストでのバグ密度(2)

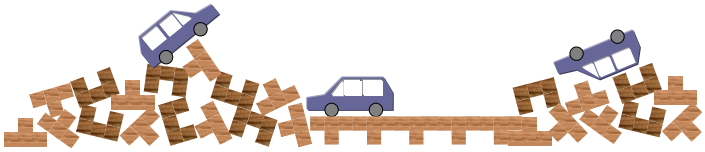
- 平均だけでは、リスクは判別不能。
- 下のような分散の場合、A社に発注するほうがギャンブル
- リスクが読めるほうが、平均よりも重要
  - 分散が小さい（本講演の用語では「凸凹が少ない」ということ）



© Hitachi, Ltd. 2009. All rights reserved.

現状のソフトウェアテスト技術の壁

- 凸凹の少ない部分で有効に活用可能
- 凸凹な部分で壁にぶちあたる
  - バグ密度、信頼性モデル、同値分割、直交表、、、
    統計に基づくような手法は何でも同じ！



© Hitachi, Ltd. 2009. All rights reserved.

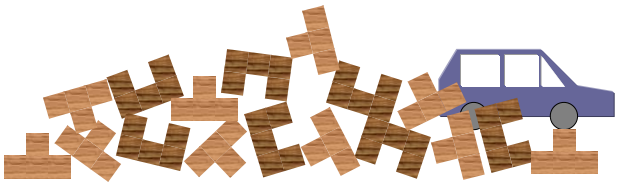
講演の流れ

- ソフトウェアテスト技術の壁
  - ソフトウェアテストの取り巻く環境
  - 現状の主要テスト技術の限界、その理由
- テスト技術の壁を破るためには？
  - 技術的な観点でのブレークスルー案
  - 日本の強みを活かしてどのようにブレークスルーを
    実現するか

© Hitachi, Ltd. 2009. All rights reserved.

よくない例

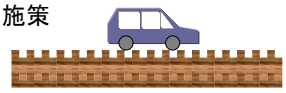
- 凸凹は分かっているけど、無視して突き進め！
  - 例えば、本当は同値で無いのに、同値とみなすというような
    方法
    - これをやってしまうと、一因子のテストすら漏れる危険性が大。



© Hitachi, Ltd. 2009. All rights reserved.

技術的な観点でのブレークスルーの方向

- 凸凹をある程度埋めて、既存のソフトウェアテスト技術
  をさらに活用できるようにする。
  - 多くの施策は、テスト工程以前の施策

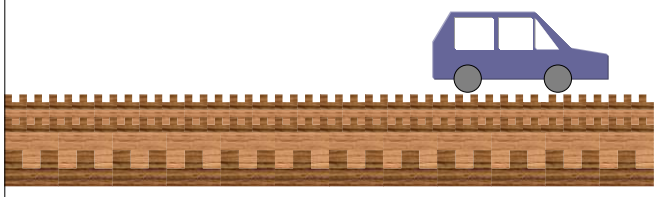


- 凸凹があっても適用可能なソフトウェアテスト技術を開
  発し、適用する。
  - 多くの施策は、テスト技術面のブレークスルーが必要

© Hitachi, Ltd. 2009. All rights reserved.

ブレークスルーの方向（1）

- 凸凹をある程度埋めて、既存のソフトウェアテスト技術
  をさらに活用できるようにする。

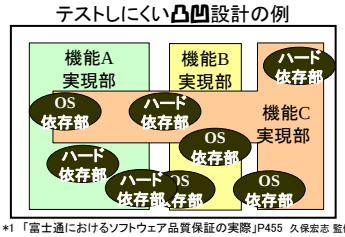


© Hitachi, Ltd. 2009. All rights reserved.

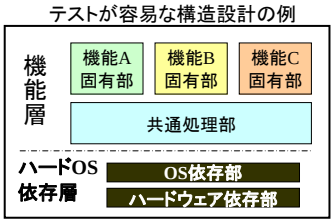
構造設計レベルの凸凹の解消

- 高い凝集度。低い結合度。
- 単に「結合度が低い」ではなく、同じ様な結合度で多くのモジュールが構成されていること
  - 層毎のモジュール構成、同じ層では、短冊形のモジュール構成
- 七つの設計原理　*1
  - 単純原理、同型原理、対称原理、階層原理、線形原理、明証原理、安全原理

テストしにくい凸凹設計の例



テストが容易な構造設計の例



*1「富士通におけるソフトウェア品質保証の実際」P455　久保宏志 監修ISBN4-8171-6022-5

© Hitachi, Ltd. 2009. All rights reserved.

「七つの設計原理」(1)

- 単純原理
  - 自然であれという原理。たとえば、「全体的な関連性を意識しながら」プログラムを作ったりせず、局所完全的なプログラムで満足するなど、単純に素人っぽくプログラミングするということ。
- 同型原理
  - 形にこだわるという原理。たとえば、同じことは同じようにやることにこだわること。また、例外は設けないように固く決意したりすること。一様性を重んじることも、この原理の一部。
- 対称原理
  - 形の対称性にこだわるという原理。たとえば、上があれば下、右があれば左、アクティブがあればインアクティブというような対称性にこだわるということ。

出典:「富士通におけるソフトウェア品質保証の実際」P455　1989久保宏志 監修ISBN4-8171-6022-5

© Hitachi, Ltd. 2009. All rights reserved.

「七つの設計原理」(2)

- 階層原理
  - 形の階層的美しさにこだわるという原理。たとえば、ものごとの主従関係、前後関係、本末関係などの階層関係を常に意識し、あるべき姿を求めるということ。
- 線形原理
  - 形は直線だけによって描かれており、さらに短形であるのがよいとする原理。たとえば、ある機能はいくつかの機能の重ね合わせによって実現されている(線形結合的である)のがよいとすること。
- 明証原理
  - ロジックの明証性にこだわるという原理。たとえば、少しでも不透明なロジックは証明しておくように努めること。
- 安全原理
  - 必然性を意識するという原理。たとえば、必然性のないところやあいまいなところは、ちょっとルーズに安全サイドで設計しておくということ。

出典:「富士通におけるソフトウェア品質保証の実際」P455 1989久保宏志 監修ISBN4-8171-6022-5

© Hitachi, Ltd. 2009. All rights reserved.

Ruby on Rails(RoR)

活

- RubyとRoR
  - Rubyは、例えれば蒸気機関
    - 動力にはなるが、それだけであちこち行ける訳ではない。
    - 何でも書けるが、テストしずらいコードを書くことも朝飯前。
  - On Railにすることにより、
    - 凸凹の少ないレールの上を走ることによって、見たことのない世界に連れて行ってくれる。
    - 自然とMVCモデルやデザインパターンに従った設計が可能。
- フレームワークを利用したテスト
  - RoRに限らず、アプリケーションフレームワークとテスト(フレームワーク)を結合することにより、結合度の低い設計→テスタビリティの高い設計が可能。

© Hitachi, Ltd. 2009. All rights reserved.

コードレベルの凸凹の解消

活

- インデンテーションの無いコードで実現できれば理想
  - だが、有りえない。
- テストケース数を爆発させるのは、条件式よりもループ
  - ループにもテストケースを爆発させるものと、させないものがある。
- 非同期処理のテストケースは、ループよりもさらに問題になりうる。
  - 非同期処理での同値分割を意識して設計する。

© Hitachi, Ltd. 2009. All rights reserved.

Loopレスプログラミング

活

- GOTOレスではない。
- 本質的にループが必要な処理と、必要でない処理を分けて考える。
- ループの順番に関係のある処理と無い処理。
  - テストケースを爆発させるのは、前者
  - C言語では区別がつかないが、区別のつく言語では、明示的に分けて記述する。

```
int aResult = 1;
for(int i=0;i<V_MAX;i++){
      aResult *= 2;
}
```

```
int anArray[A_MAX];
for(int i=0;i<A_MAX;i++){
      anArray[i] = 0;
}
```

© Hitachi, Ltd. 2009. All rights reserved.

非同期処理のテストにおける凸凹

活

- 同値分割の原語（equivalence partitioning）には、変数の値という制限は無い。
- 非同期処理のテストの場合、処理実行処理を同値分割。（等価分割といったほうが正確かも）
  - 同期、非同期で、共有する資源の状態が同じ処理間では、どこで割り込まれても同値（等価）とする。
- 凸凹を無くすためには、同期、非同期で、共有する資源を最小限にする。
  - 共有する資源がなければ、そもそもテスト不要
  - 共有する資源が多いと、テストケースが発散

© Hitachi, Ltd. 2009. All rights reserved.

非同期実行ルーチンのテスト

活

悪い例

```
// 割込み時の処理
void interrupt_proc(){
        // 割込み時の処理
}
// 通常のイベント処理
void normal_proc(struct foo aEv){
        // 通常のイベント処理
}
```

多くの資源を共有

```
// 非同期の割り込みルーチン
void async_handler(){
        interrupt_proc();
}
// 通常処理用のメインループ
void main_loop(){
        struct foo aEv;
        while(read_event(&aEv)){
                normal_proc(aEv);
        }
}
```

良い例

```
bool interrupt_flag = false;

// 非同期の割り込みルーチン
void async_handler(){
    // フラグを立てるだけ
    interrupt_flag = true;
}

// 通常処理用のメインループ
void main_loop(){
   struct foo aEv;
   while(read_event(&aEv)){
      // フラグをみて同期的に割込み処理
      if(interrupt_flag){
        interrupt_proc();
      }
      normal_proc(aEv);
   }
}
```

© Hitachi, Ltd. 2009. All rights reserved.

開発プロセスの平準化

活 推

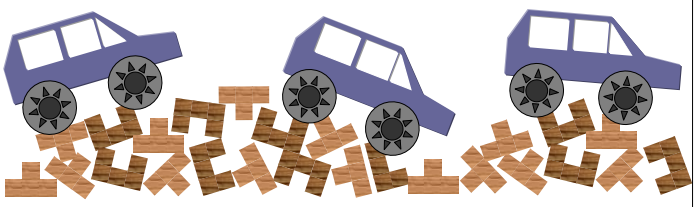
- 開発するソフトウェアの特性に従った開発プロセスを採用するのは当然
  - 必然的に開発プロジェクト毎に違ったプロセスを経る。
- ただし、ソフトウェアテストの観点で、開発プロセスの基盤技術は平準化するべき
  - 要求管理
  - 品質管理
  - 構成管理（変更点管理）
- 平準化した基盤技術に対するテスト技術の確立
  - 要求や、要件、機能レベルの変更に対するテスト技法

© Hitachi, Ltd. 2009. All rights reserved.

ブレークスルーの方向（2）

推 研

- 凸凹があっても適用可能なソフトウェアテスト技術を開発し、適用する。



© Hitachi, Ltd. 2009. All rights reserved.

タグチメソッドのソフトウェアへの適用（1）

推 研

- ハードウェアでのタグチメソッド
  - 凸凹でないところを見つけ、そこを使って最適な機能設計を可能にする手法
  - 凸凹のところ、凸凹有無の理由は無視。
- ソフトウェアの直交表
  - 「凸凹がひどくない」と仮定してテストケースを作成
  - 凸凹があまり無く、一様な部分では効率的にテスト可能
  - 凸凹が大きいと必要なテストが漏れる危険性大

© Hitachi, Ltd. 2009. All rights reserved.

タグチメソッドのソフトウェアへの適用（2）

推 研

- ソフトウェアでもハードウェア的、さらには、もっと本質的なタグチメソッドの使い方は無いか
  - ハードと同様に凸凹でないところを見つける方法
  - ソフトウェアでは、凸凹を無視することはできないので、ソフトウェアの凸凹を無くすために使用できればさらに良い

© Hitachi, Ltd. 2009. All rights reserved.

タグチメソッドのソフトウェアへの適用(3)

推 研

テストツールとしての活用

1. 直交表によってソフトウェアを実行
   - テストが目的ではない。
2. テスト対象プログラムをプロファイリング
3. 各因子間の凸凹度を測る
   - 例えば、モジュールの結合度(Coupling)と同様な方法が使用できるかも
4. 凸凹度によってテスト方針を変える
   - 結合度が0であれば組み合わせテスト不要
   - 結合度が低い因子間は直交表を活用しテスト効率化
   - 結合度が高い(凸凹な)因子間は入念に組合わせテスト実施
- 同値分割の妥当性分析にも使えるかも

© Hitachi, Ltd. 2009. All rights reserved.

タグチメソッドのソフトウェアへの適用(4)

推 研

設計ツールとしての活用(1~3までは前ページと同じ)

1. 直交表によってソフトウェアを実行
   - テストが目的ではない。
2. テスト対象プログラムをプロファイリング
3. 各因子間の凸凹度を測る
   - 例えば、モジュールの結合度(Coupling)と同様な方法が使用できるかも
4. 本来、凸凹の差が無いはずの因子間で大きな差が出てくる可能性がある。
   - その場合、凸凹度が低く、平準化できるようにソフトウェアの設計を改善(リファクタリング)する。

© Hitachi, Ltd. 2009. All rights reserved.

反復的なテストプロセス

活 研

- ほとんどのソフトウェアテスト技術はウォータフォールライク
  - 仕様→テスト設計→テストケース作成→実行
  - しかし、「テストを始めて、初めて分かったこと」も重要。
- テストの結果によって、テスト方針をエボリューショナルに改善していくような方法論
  - テストケースの結果はそれぞれ独立ではない。あるテストケースの結果は他のテストケースに影響がある。
  - 通常、やられていることではあるが、習慣や精神論ではなく、システマチックな方法論があると良い。

© Hitachi, Ltd. 2009. All rights reserved.

講演の流れ

- ソフトウェアテスト技術の壁
  - ソフトウェアテストの取り巻く環境
  - 現状の主要テスト技術の限界、その理由
- テスト技術の壁を破るためには?
  - 技術的な観点でのブレークスルー案
  - 日本の強みを活かしてどのようにブレークスルーを実現するか

© Hitachi, Ltd. 2009. All rights reserved.

日本的品質管理の強みとテスト

- 日本的品質管理の特徴 *1
  - 要求充足ではなく結果満足
  - 改善指向
  - 品質の作りこみ
  - 現場、現物主義
  - 全員参加
- 日本の強みである品質管理の特徴を活かした、ソフトウェアテスト技術のブレークスルーが可能ではないか。

*1 西康晴「ソフトウェア品質保証技術の動向」を参考 http://www.jasst.jp/archives/jasst08t/pdf/2.pdf

© Hitachi, Ltd. 2009. All rights reserved.

要求充足ではなく結果満足

活

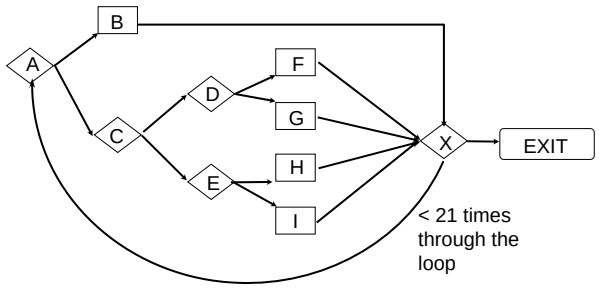
- ソフトウェアテストを必要以上に難しくしていないか?
  - 百兆個のテストケース(Myersの有名な例)
  - テストでバグが無いことを示すことはできない(Dijkstra)

  →上記は典型的なプロダクト指向な考え方
- 要求や仕様のほうに目が向いて、お客様の環境で動く組み合わせが漏れてしまったら本末転倒
  - 市場で使われるパスが、百兆通りあるのか?
  - お客様が使う有限のパスでバグが無いことを本当にテストで示せないのか?

© Hitachi, Ltd. 2009. All rights reserved.

100,000,000,000,000パスって本当？

- Myersさん、実は百兆という数字を出したかったのでは？
- 百万だったら、(自動的に)やろうという話も出たはず。



出典： Myers「ソフトウェア・テストの技法」

© Hitachi, Ltd. 2009. All rights reserved.

Episode 2　事故対策の現場にて

- ソフトウェアに起因する事故の説明資料を作成
- どのような条件の組み合わせで事故が発生したのかを詳細に説明した資料作成
  - 言外に、「これだけの条件が重なっていたので、発生してしまった」という弁解の意図があった。
- ソフトウェア側からの技術的な説明を歓迎するお客様もいるが、不快感を持つお客様も多い。
  - ソフトウェア側から見ると希少な確率でも、そのお客様の環境では必ず起きるような場合。
  - なんで、そのお客様の環境をテストしていないのかという理由にはならない。

© Hitachi, Ltd. 2009. All rights reserved.

Idealist's Cry!

Design methods determine reliability
Testing shows presence of bugs
ONLY software products can be verified without Testing
→Testing is unnecessary

Pessimist's Sigh...

Faults = f (manpower)
(independent of design methods)
Infinite program paths
Impossible to remove all faults
→Testing is fruitless

Industrial Eye !?

Best design, some faults
Finite failures (or faults)
Any fault can be found by testing
Possible to remove all faults by software testing
→Testing is important

出典：IKOMA, Mikio, Software Testing and Reliability of the Japanese Software Industry, COMPSAC '91

© Hitachi, Ltd. 2009. All rights reserved.

改善指向：ソフトウェアテストの知識の形式知化、ツール化

推

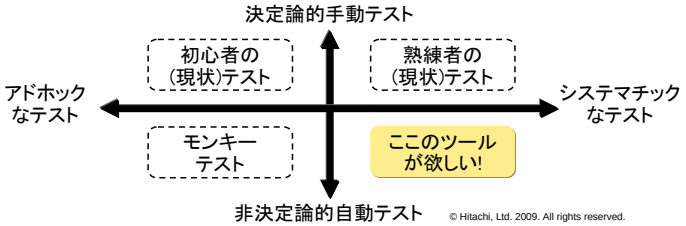
- 実戦的なソフトウェアテストの暗黙知はある。
- 例えば
  - 広い定義域を持つ関数をテストするときのテストする値の決め方
  - 性能向上でソースを変更した場合のレビュー又はテスト方法
  - OS系ライブラリをテストするときの条件の設定方法（マルチスレッド、シグナル、forkなどで安全に動作等々）
  - 永続的な資源の有無とリセット条件、…
- このような実戦的なソフトウェアテストの知識を形式化するだけでなく、ツール化
  - (直交法も、こういうソフトウェアテスト改善の一環かも)

© Hitachi, Ltd. 2009. All rights reserved.

このような自動テストツールが欲しい！

研

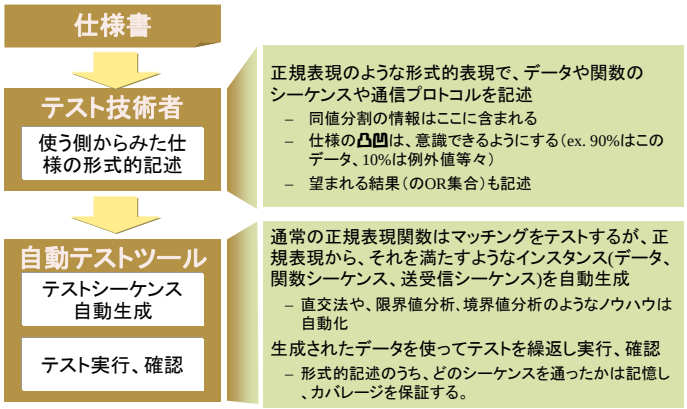
- 現状は、GUI系のテストを省力化するテストツール
- 今後望まれる自動テストツール
  - 仕様を入力にして、テストケース生成、実行を自動化するツール
  - 何百万回でもテスト可能
  - ソフトウェアテストケース作成の各種ノウハウをツールに実装
  - 自動的に、ソフトウェアの凸凹をかき分けてテスト可能になれば最高！



© Hitachi, Ltd. 2009. All rights reserved.

自動テストツールのイメージ(1)

研

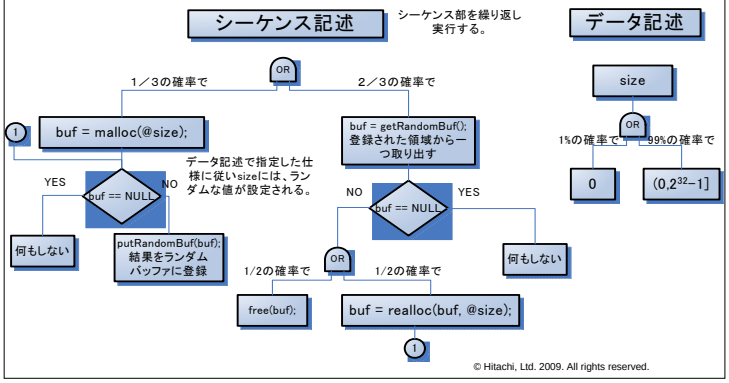


© Hitachi, Ltd. 2009. All rights reserved.

自動テストツールのイメージ(2)

研

- 形式的なテストの記述のイメージ(mallocのテスト例)



© Hitachi, Ltd. 2009. All rights reserved.

改善指向:
テスト技術の改善モデル案

推

- 数学のように、未解決問題を管理するような方法

© Hitachi, Ltd. 2009. All rights reserved.

品質の作りこみ:
テスタビリティ設計の二つの意味

研 活

- テストできる手段を備えているか？
  - テスタビリティアセスメント
  - テストを制御可能か(Controllability)。テスト結果を観測可能か(Observability)
- ソフトウェアの凸凹が少しは平らになっていないか？
  - 必要とするテスト項目を減らす
  - 正確には、合理的に絞ったテスト項目で実質的に十分なテストができるようにする。

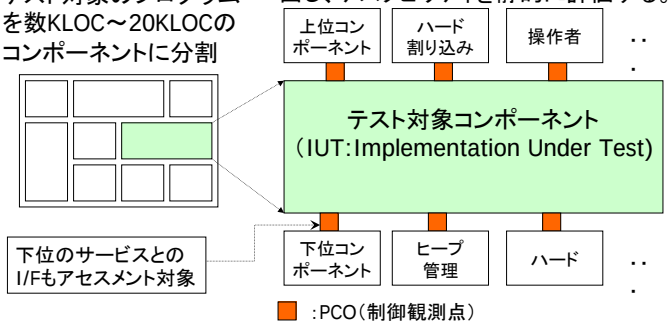
© Hitachi, Ltd. 2009. All rights reserved.

テスタビリティアセスメント

研

テスト対象のプログラムを数KLOC～20KLOCのコンポーネントに分割

テスト対象のコンポーネント毎にPCOを抽出し、テスタビリティを静的に評価する。



用語及び構成の部分:出典 ISO/IEC 9646 “ Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework”

© Hitachi, Ltd. 2009. All rights reserved.

現場・現物主義:
現物主義のテスト方法論

研

- 現物指向を形式知、または確立された技術にできないか？
  - 例:テストがどの程度、本番に近いのか？を示すメトリクス
    - 同じハード構成(コンピュータ、周辺装置)
    - 同じネットワーク構成(機器、プロトコル、性能)
    - 同じ基盤ソフト構成(OS、ミドル)
    - 同じデータ、データの分布(オペレーショナルプロファイルの妥当性)
  - 例:現物指向に基づくテストプロセスの確立
    - ドライバ、スタブ、シミュレーション、エミュレーション、ICEといった、テスト用の技術から最終的なValidationまでのプロセスの明確化

© Hitachi, Ltd. 2009. All rights reserved.

Episode 3　現物主義 vs. セキュリティ

- 世界的に認められた日立製の暗号アルゴリズムをプログラミング
  - 「ムダだよ」という研究者のアドバイスを無視して、品質部署の人間が早速検査
  - 研究者。「これが有名な日立の品質保証か…」と感心
- アルゴリズムの強度とそれを実装したソフトウェア(又はシステム)の強度は全く別物。
- 現物主義の観点で品質保証の技術者の行動は正しい。

© Hitachi, Ltd. 2009. All rights reserved.

全員参加

推 活

- （幸か不幸か）日本ではソフトウェア開発におけるテスタという職種はあまり専門化されていない。
- テスタが専門化されていても、工程間、職種間で協調できるのが日本の強み
- 他国より容易に前工程からのソフトウェアテスト技術確立の活動ができるのではないか。

© Hitachi, Ltd. 2009. All rights reserved.

本日の講演のまとめ

- ソフトウェア、ソフトウェア信頼性、ソフトウェア開発プロセス、どれをとっても**凸凹**である。
- **凸凹**を意識しないで、ソフトウェアテスト技術を過信するのは極めて危険。
- 今後のソフトウェアテスト技術を発展させるため
  - できるだけ**凸凹**のないソフトウェアをつくり、従来のソフトウェアテスト技術を最大限に活用しよう。
  - 多少、**凸凹**があっても適用可能なソフトウェアテスト技術を創生しよう。
  - 日本の品質管理の強みを活かし、ソフトウェアテストの壁をぶち破ろう！

© Hitachi, Ltd. 2009. All rights reserved.

参考文献

- ボーリス・バイザー「ソフトウェアテスト技法—自動化、品質保証、そしてバグの未然防止のために」 日経BP出版センター ; ISBN: 4822710017
- Glenford J.Myers「ソフトウェア・テストの技法」近代科学社 ; ISBN: 4764900599
- ISO/IEC 9646 “ Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework”
- 西康晴「ソフトウェア品質保証技術の動向」より http://www.jasst.jp/archives/jasst08t/pdf/2.pdf
- IPA SEC「ソフトウェア開発データ白書2008」日経BP出版センター ; ISBN978-4-8222-6227-3
- 秋山浩一「直交表を活用したソフトウェアテストの効率化 - HAYST法の活用-」 http://www.jasst.jp/archives/jasst05w/pdf/S4-1.pdf
- 久保宏志 監修「富士通におけるソフトウェア品質保証の実際」; 1989ISBN4-8171-6022-5
- ベノワ・B・マンデルブロー「禁断の市場 フラクタルでみるリスクとリターン 」ISBN-10: 4492654178
- IKOMA, Mikio, Software Testing and Reliability of the Japanese Software Industry, COMPSAC ‘91

© Hitachi, Ltd. 2009. All rights reserved.