東海　11月6日

DEI

テストから観た実体のモデリングと実装構造の評価

～検証指向設計の実現に向けて～

デバッグ工学研究所
代表　松尾谷　徹

R-15. 11

© DebugEng Debug Engineering Institute

はじめに

DEI

- 昨年6月は，
  Concolic Testing， Symbolic Execution　のお話し
- 今年は，
  岸本さんが，その実践の紹介
  植月さんが，もっと深いお話し

- 今回のテーマは，テスト自身のモデリングです．
- 抽象的なテーマですが，実体との関係も考えます

© DebugEng Debug Engineering Institute

目次

DEI

■ 概要

- *初めて聞くようなテーマかもしれません.*
- *規範的な教え，実態，疑問点などを整理しましょう*

■ 目次

1. 背景　検証指向設計とは
2. 例題で考える：派生開発
3. スキルの成長
4. テストのモデル：探索モデル
5. プログラムの探索構造
6. まとめ

© DebugEng Debug Engineering Institute

DEI

<テストから観た実体のモデリング>

1. 背景　検証指向設計とは

■ テストから観たモデリング・・・その背景

■ 検証指向の考え方

© DebugEng Debug Engineering Institute

検証指向設計

DEI

実は25年ほど昔から進めている・・・・まだ道半ば

- 検証指向プログラミング技法VOP、第11回日科技連ソフトウェアシンポジウム、PP.61-68,1991
- 検証指向のプログラム設計技法 VOP, 情報処理学会, ソフトウェア工学研究会77-8, PP.45-50,1991
- VOP: Verification Oriented Programming, EOQC, Fourth European Conference on Software Quality,PP.529-538,1994 Oct
- ソフトウェア開発・検証技法,電子情報通信学会,1997

© DebugEng *Debug Engineering Institute*

背景

DEI

- 検証指向設計の背景
  - *1980年後半　設計技法の進歩　データ指向，オブジェクト指向の展開*
  - *流行：　制御指向の設計者が改信するが，中途半端なオブジェクト指向*
  - *その結果*
    - コードレベルのオブジェクト指向（言語はjava，C++，・・）
    - しかし，オブジェクト指向型のモデルで考えてはいない．
      - ✓ その目的は？　流用技術としてご利益追求！！
      - ✓ 対処療法的なプログラムの増加（良く解らないが，目的の機能は動作する）
  - *これらのツケや問題がテストに波及！！*
  - *流用技術により規模増大　－＞　テスト範囲の増大　－＞　大変！！*
  - *テストはブラックなので，（中身を見ないという意味）　規模増大は大変*

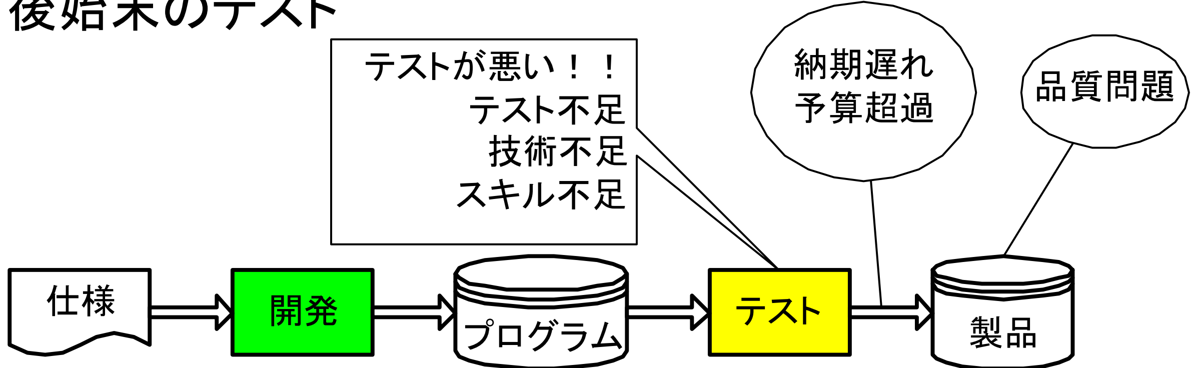
© DebugEng *Debug Engineering Institute*

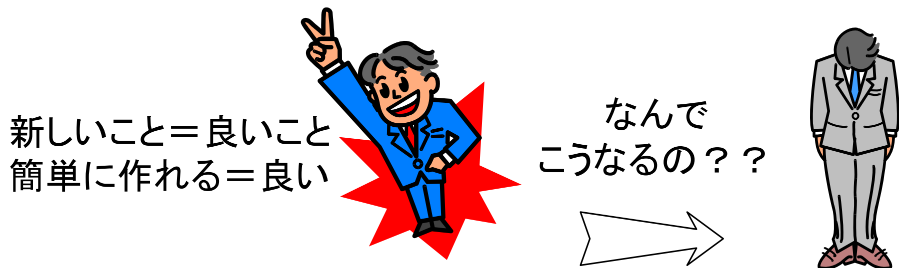
概念的には

DEI

■ 自由な開発

■ 後始末のテスト



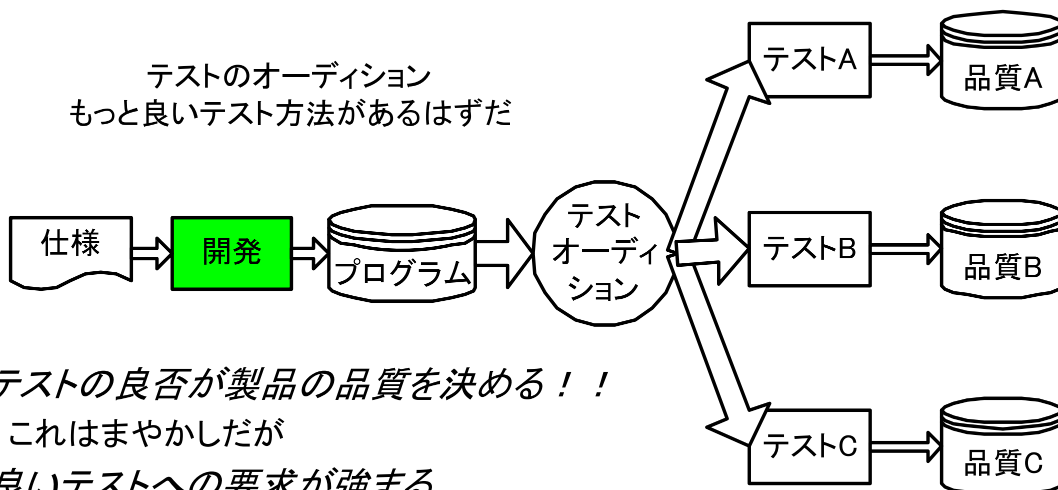


© DebugEng Debug Engineering Institute

考え方：テストのオーディション

DEI

■ テストを調達するトレンド



- *テストの良否が製品の品質を決める！！*
  - これはまやかしだが
- *良いテストへの要求が強まる*

■ 実態

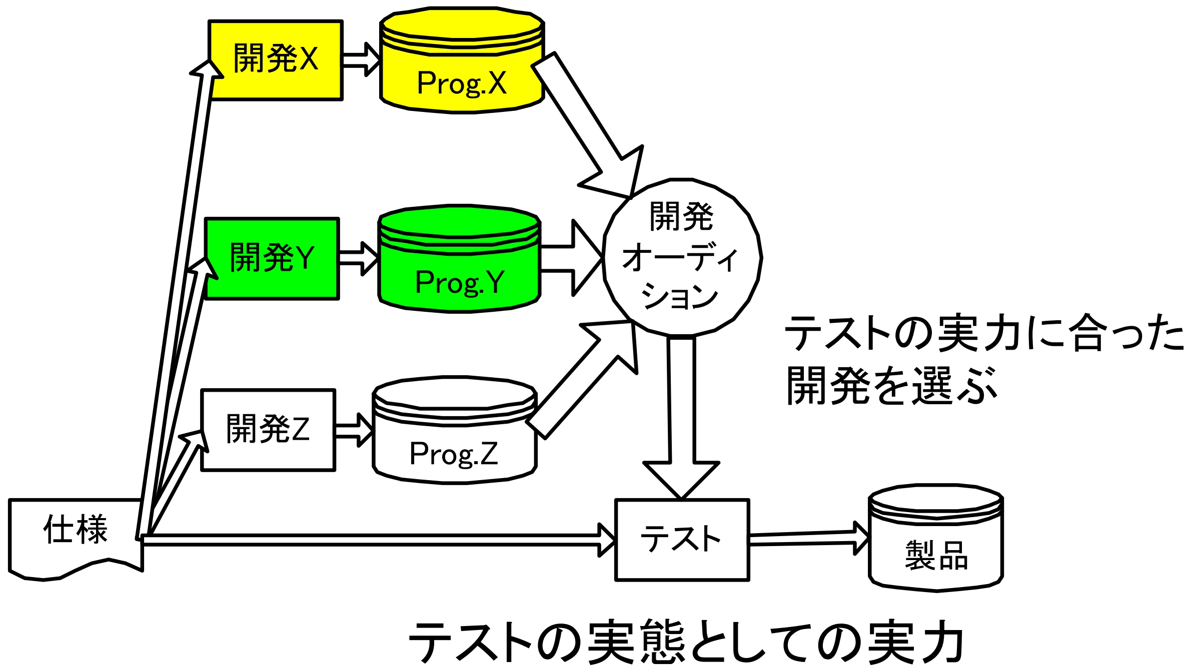
- *20年を経て，実績のある「良いテスト」はいまだに謎*
- *理屈は沢山あるが，客観的な実践成果を見たことがない*

© DebugEng Debug Engineering Institute

逆転の発想　開発のオーディション

DEI

■ 選ぶ基準はテストとの相性（バランス）



テストの実態としての実力

© DebugEng Debug Engineering Institute

まとめると

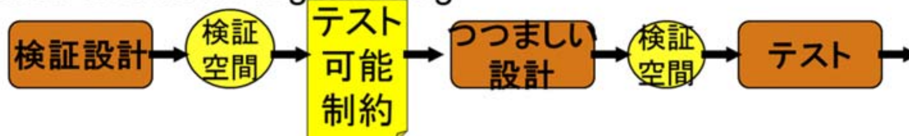
DEI

■ 考え方
- *理解できないものは検証できない*
- *理解できないものでも、部分を集めれば設計できる*
  ⇒ 現在のシステム構築　　テストで破綻
- *しかも、様々な難しい設計方法を好む傾向がある*

■ 対策：テスト能力の範囲内で設計する

Cowboys Programming

Verification Oriented Programming



© DebugEng Debug Engineering Institute

DEI

<テストから観た実体のモデリング>

2. 例題で考える：派生開発

- 開発活動の大半は派生開発
- 回帰テストの謎

© DebugEng Debug Engineering Institute

派生開発におけるテスト

DEI

- 派生開発のテストモデルを考える

- 確認すべき2つの仕様

1. 派生により，追加，変更，削除された機能や振舞い
   - *「派生部分のテスト」と呼びます*
2. 追加，変更，削除されていない機能や振舞い
   - *一般的な「回帰テスト」と呼びます*

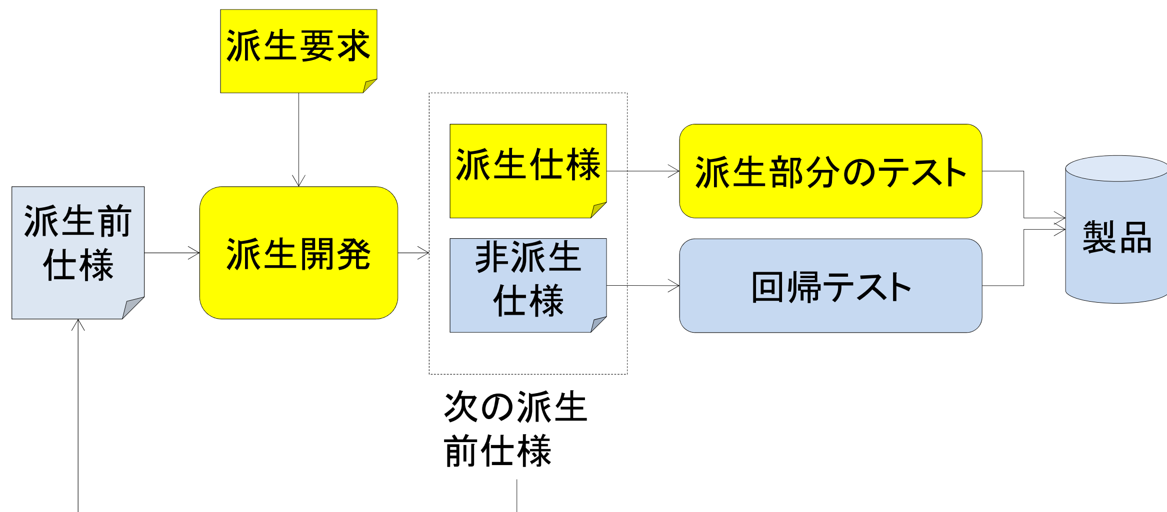
- 開発側は
  - *派生要求の真の目的は何か？*
  - *どこを改造すれば良いのか？*

© DebugEng Debug Engineering Institute

頭の中(だけ)のモデル

DEI

■ 仕様ベースのテストなら

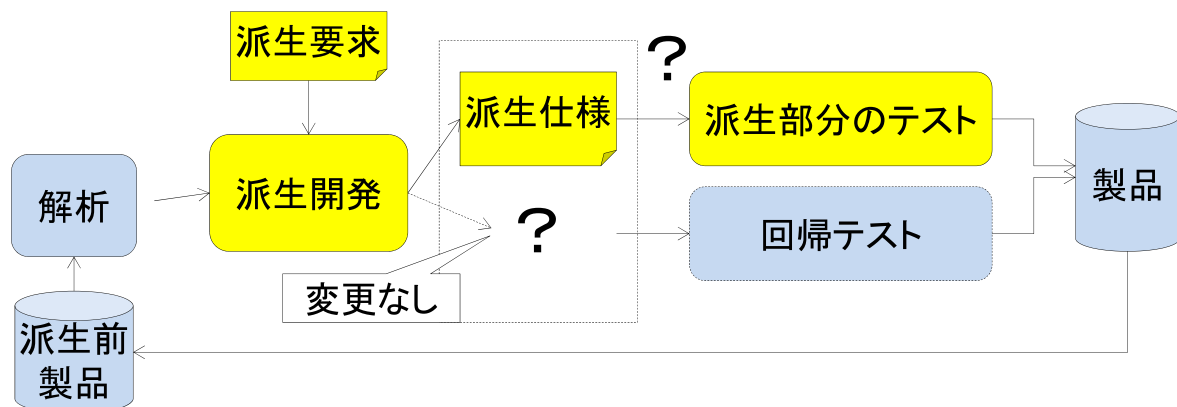


© DebugEng Debug Engineering Institute

実態モデル

DEI

■ 回帰テスト設計の入力は何か・・・・・無い！ ！



■ 繰り返すと，どうなるのか？　品質は？生産性は？

- *直ぐ判るが，悪いことは考えない(先送り:国民性)*

© DebugEng Debug Engineering Institute

後工程は？

DEI

- 製造業の美徳
  - *後工程は，お客様*

- 製造業の内なるソフト業は
  - *後工程は，“あかの他人”*
  - *つまりCowboys Programming*
  - *結末は，共倒れ*

- 解決策は？
  1. *美徳を守るプロセスへ改善！　・・・・精神論でごまかすな！！*
  2. *現実として認める（人力では困難）　・・・・新たな解決方法へ*

© DebugEng Debug Engineering Institute

まとめ

DEI

- テストの観点から派生開発を眺める

1. 派生部部分のテスト・・・新規開発と同様
2. 回帰テスト・・・大きな問題

問題は

- 責任の所在は別として，現テスト力では対応不可
  - *対応不可とは，現実的な方法で確実に品質を確保できない*
  - *現実的な方法　仕様ベースのテスト，テストの蓄積，要員のスキル，工数*
  - *仕様がない，テストの蓄積がない，工数がない，効果がない・・・・*

非効率だが，他に手段が無いので放置されている

© DebugEng Debug Engineering Institute

いろいろ課題はあるが，薄明り

DEI

- 派生開発の回帰テスト問題
  - *破綻するのは，仕様ベーステストだから*

- 発想の転換/モデルの転換
- 検証指向でプロセスを変える
- 実装ベースのテストで乗り越える
  - *『Concolic Testing を活用した実装ベースの回帰テスト －人手によるテストケース設計の全廃－』*
  - *ソフトシンポ2015*
  - *http://sea.jp/ss2015/award.html*

- 言いたいことは，「大局的なモデル」を考える必要性

© DebugEng *Debug Engineering Institute*

DEI

<テストから観た実体のモデリング>

3. スキルの成長

- プログラミング演習　と　テスト演習
- デバッグなしのプログラミングは有り得ない
- デバッグなしのテスト設計は当たり前・・何で？

© DebugEng *Debug Engineering Institute*

プログラミング演習

DEI

- 2つのプログラミング演習法
1. 机上でプログラミングを学ぶ
2. プログラムを組んで実習から学ぶ

- 演習後，同じ仕様を与え正解を得るまでの時間測定
  - *マラソン同様，打ち切り時間あり*

- 結果の予想？
- その原因は？

© DebugEng *Debug Engineering Institute*

プログラミング･スキルの中身

DEI

- 新たな言語を速く習得する人，そうでない人
- プログラムを速く組める人，そうでない人

････

- 文章/論文を書く場合，　高品質で速い人，そうでない
- 身体運動と連携する脳の働きは，もっと明快
  - *子供が自転車に乗れるようになる*
  - *原始人が吹き矢のスキルを習得する*

- これらのスキル習得に共通するものは何か？

© DebugEng *Debug Engineering Institute*

「野生の思考」: ブリコラージュ

DEI

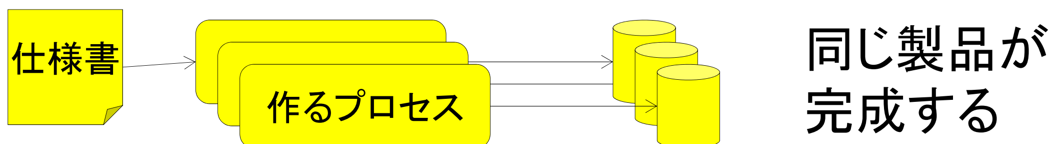
- ブリコラージュ・・・・・デバッグで成長する
  - *手に入るモノを寄せ集めて，試行錯誤しながらモノを作る*
    - その本質は，創造性と機智
- エンジニアリング・・・・緻密な論理の積上げ
  - *理論や設計図に基づいてモノを作る*
    - その本質は，再現性，属人性の排除
- クロード・レヴィ・ストロース
  - *ブリコラージュの発見者*
  - *サルトルの実在主義　を論破*
  - *新たな哲学概念　「構造主義」の一人*
  - *民俗学者，神話学者，哲学者*

© DebugEng Debug Engineering Institute

プログラミングは，エンジニアリングか？

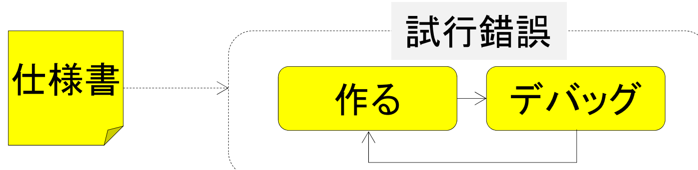
DEI

- エンジニアリングであるならば
  - *仕様書から，プログラムが作成される*
  - *同じ仕様書なら，だれが作っても，同じコードが出来る*



同じ製品が
完成する

- ブリコラージュであるならば
  - *仕様書だけでは，均一のプログラムは作成できない*
  - *作成する人の創造性と機智と**試行錯誤**に左右される*



© DebugEng Debug Engineering Institute

ソフトウェア工学の夢

DEI

- プログラミングに再現性は無い
- これは困ったこと（エンジニアリングとするならば）
- 何とか，再現性を高めよう！！
- そのために，工学的アプローチが必要

- **ソフトウェア工学の提唱(1968)**
- エンジニアリングにしたい（要望，願望）
- 半世紀を経てもエンジニアリングにはなっていない
  - *そもそも，エンジニリングでは無いのでは？*
- エンジニアリングを進める前に
  エンジニアリングに近づける工夫！！

© DebugEng Debug Engineering Institute

プログラミングのスキル実態

DEI

- エンジニアリングでは無い （知識だけではできない）
- 試行錯誤が必要
- プログラミングの試行錯誤＝デバッグ

- 高スキル者の行動
  - *速くそこそこのコードを書く*
  - *デバッグ環境を速く作る*
  - *デバッグにより早く誤りや抜けを見つけ，修正する*
- つまり，ブリコラージュ

© DebugEng Debug Engineering Institute

テストスキルの実態

DEI

- ソフトウェア分野におけるヒューマンファクタ研究の流れ，電子情報通信学会FTS研究会，第15回ソフトウェア信頼性シンポジウム，PP.16-21,1994 Dec
- 題名不明，JaSST2016 Tokyo，詳細不明 （未確定）

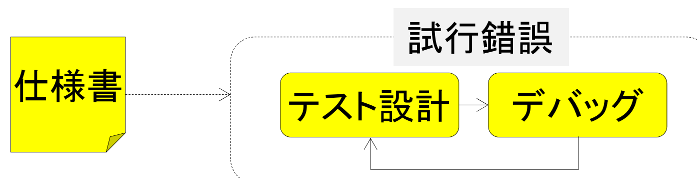
- 幾つかのテスト演習実験
- 現実のスキルだと
- 時間内に8割程度の網羅達成者は5％以下
- 技法を習得により，多少向上するが網羅には至らず
  - *技法習得で部分的な気付きは生ずるが*
  - *網羅を達成できるテスト技法が不足*

© DebugEng *Debug Engineering Institute*

改善策とその実験

DEI

- 今年10月5日（大府），10月8日（高松）で実験
- 「テストのデバッグ環境を与えたテスト演習」
- 両実験とも，時間内に全員達成（100％網羅）



- テストデバッグとは
  - *実体はテスト対象プログラムとgcov*
  - *被験者は，分岐網羅で確認（デバッグ），テストケースを追加*

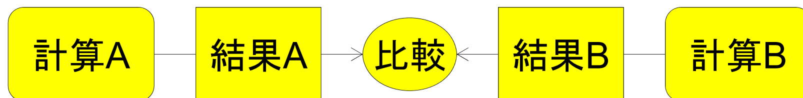
© DebugEng *Debug Engineering Institute*

テスト対象の出来栄え

DEI

- テストデバッグにより，テストスキルは飛躍的に向上
  - *テストデバッグには，テスト対象が必要*
  - *テスト対象の出来栄はどの程度影響するのか？*

- 検算の原理
  - *検算は，2つの計算結果を比較する*
  - *比較によって見つかるのは，不一致 ／ 一致*
  - *不一致は，（計算A or 計算B） が間違っている*
  - *一致は，両方とも（正しい or 誤っている）のどちらか*
  - *両者の誤り率を10%とすると，その9%を検出できる*
    - 両者が90%の信頼性なら，99%に向上する （ABが無相関なら）

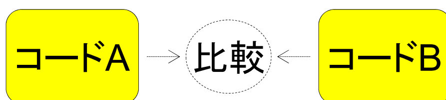


© DebugEng Debug Engineering Institute

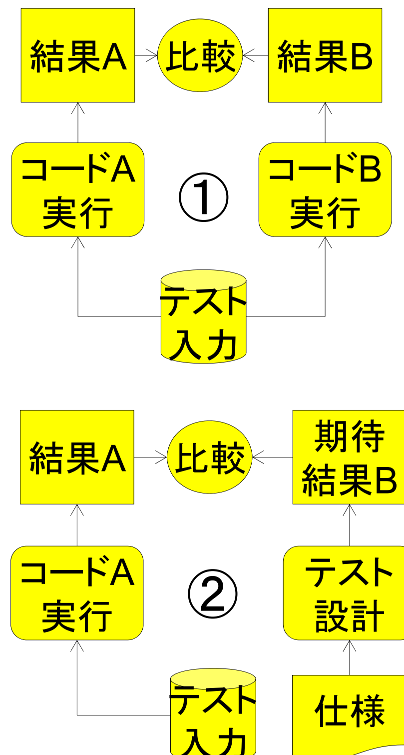
テストは検算の変形

DEI

- プログラム間の比較



コード間の比較は意味が無い



① 動作の比較が必要
- *動作結果の比較はできる*
- *そのためにテスト入力が必要*

② 期待結果があれば
- *コードBを作成する必要は無い*
- *これがテスト*

© DebugEng Debug Engineering Institute

まとめ

DEI

- テストのモデルは，検算の変形
  - *検算 → 不一致 → 片側の再計算 → 繰り返す*
  - *不一致（バグの場合）プログラムの修正だけでは不十分*
  - *テスト側の誤りもある→ テストの修正*
- デバッグは，プログラム/テストの両方に必要

- 現状：
- テスト側のデバッグが欠如している

- 解ったこと：
- テストのデバッグ環境を与えると，飛躍的に漏れが少なくなる

© DebugEng Debug Engineering Institute

DEI

<テストから観た実体のモデリング>

4. テストのモデル：探索モデル

- 探索目標，探索空間/探索構造，探知力

© DebugEng Debug Engineering Institute

デバッグを支える探索理論

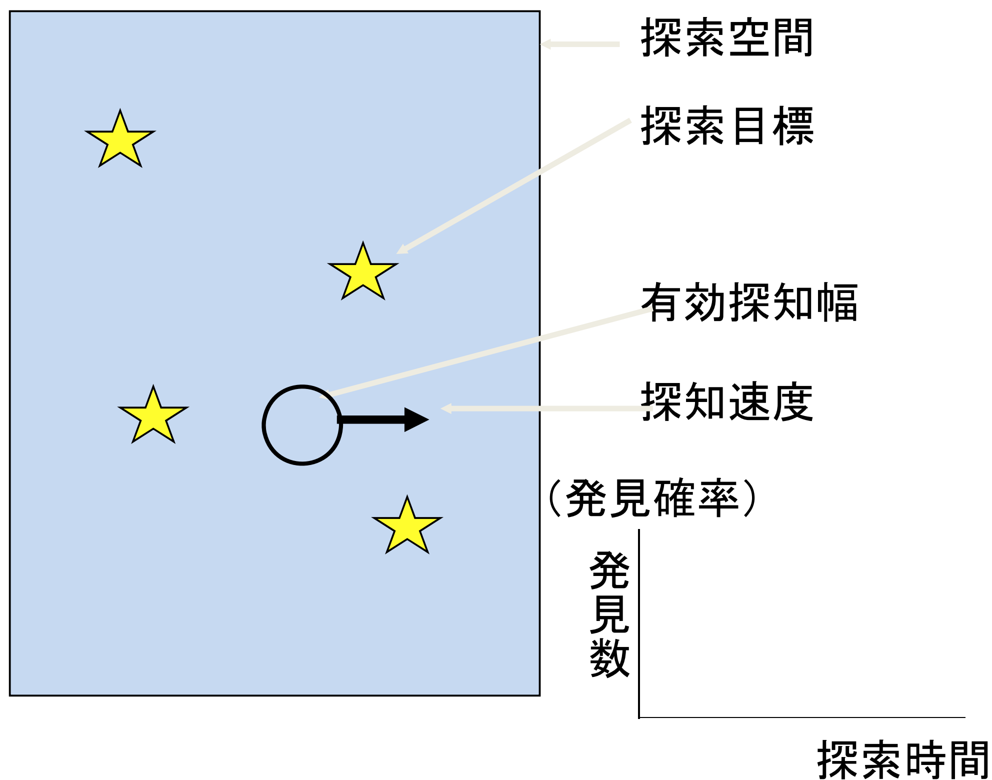
DEI

- デバッグ：バグ（目標物）を探索する行為
- 探索に関する理論 「探索理論」
  - OR（オペレーションズ・リサーチ）における研究
  - 第二次世界大戦における3大OR上の発見
  - 米海軍の独国Uボート探索から生まれた
- 探索要素（探索性能を決める要素）
  - 探索空間： 駆逐艦が担当する海域 広さ、海流・・etc
  - 探索目標： Uボート（潜水艦） 速度、数（存在密度）
  - 有効探知幅： 駆逐艦のソナー探知域
  - 探知速度： 駆逐艦の運動能力（速度、操舵性）

© DebugEng Debug Engineering Institute

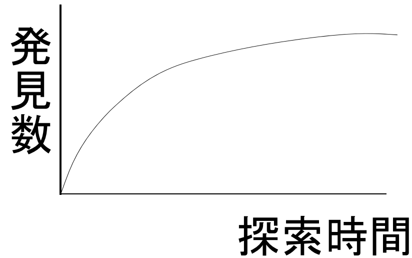
探索モデル　探される側／探す側

DEI



© DebugEng Debug Engineering Institute

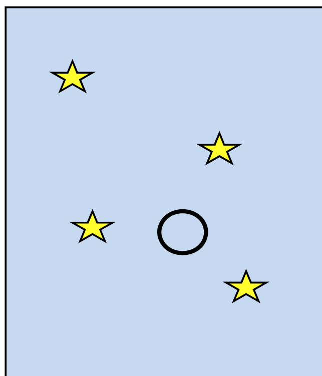
探索戦略・ランダム探索



手当たりしだいに探す
航路を見ると酔歩型の探索
<通常の探索＝最悪の探索>

ランダム探索

発見確率　$1 - e^{-rt}$



この方法は、最も効率の悪い
探索方法である.

むだ：探索の重複がある

むら：未探索個所が不明

© DebugEng Debug Engineering Institute

探索理論をテストに当てはめる

- 状況として：　バグ成長曲線＝（逆）指数型
  つまり，最悪の探索
- 何故，指数型になるのか？
  - *バグが移動する？　（バグは逃げるのか？）*
  - *地図を持っているのか？*
  - *探索者間で探索範囲の調整はあるのか？*
    - レビュー，単体，結合，システム
    - 何でこんなに多重探索をするの？
  - *テスト（ケース）の有効探知幅は？*
  - *テストの探索空間は？*

- 疑問に答えられますか？
- 疑問を持ちませんか？

© DebugEng Debug Engineering Institute

まとめ

DEI

- 探索モデル
- テストを評価する上で役立つモデル

- 現状のテストは，探索モデルから観れば

非効率

- 改善点は沢山ある

© DebugEng Debug Engineering Institute

DEI

<テストから観た実体のモデリング>

5. プログラムの探索構造

- 検証指向のプログラムとは何か

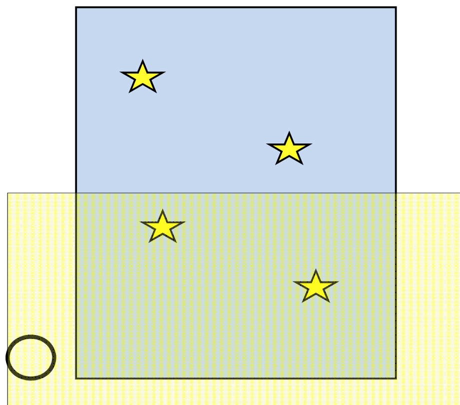
© DebugEng Debug Engineering Institute

テスト効率を左右する探索問題

DEI

■ いろいろあるが，主な問題としては

1. そもそも探索していない：　探索漏れ問題
2. 時間や工数がかかる：　　探知効率の問題



探索空間と探知空間
（対象の空間とテスト空間）

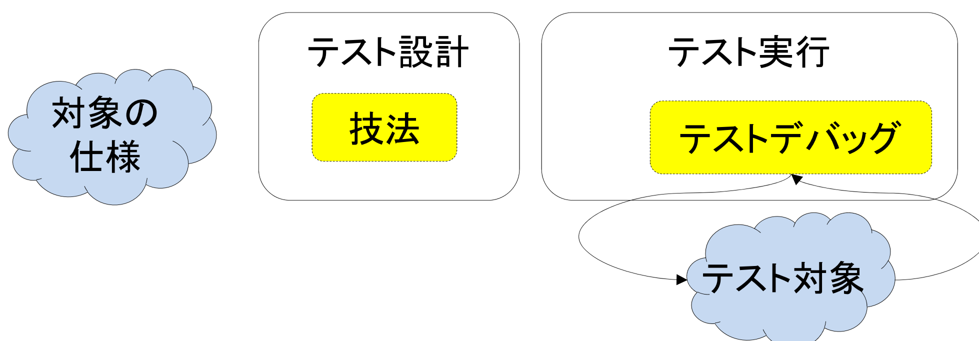
テスト空間内の合理的な探知

© DebugEng Debug Engineering Institute

探索漏れ問題

DEI

1. テスト技法の課題
   - *個々のテストケースを詳細化する技法が偏重され，*
   - *系統的な技法が敬遠されている（理由：　技法が難しいから）*
2. テストのデバッグ課題
   - *探索済/漏れをフィードバックする＝デバッグ環境の欠落*
   - *網羅計測などtoolで可能*



© DebugEng Debug Engineering Institute

テスト技法の課題

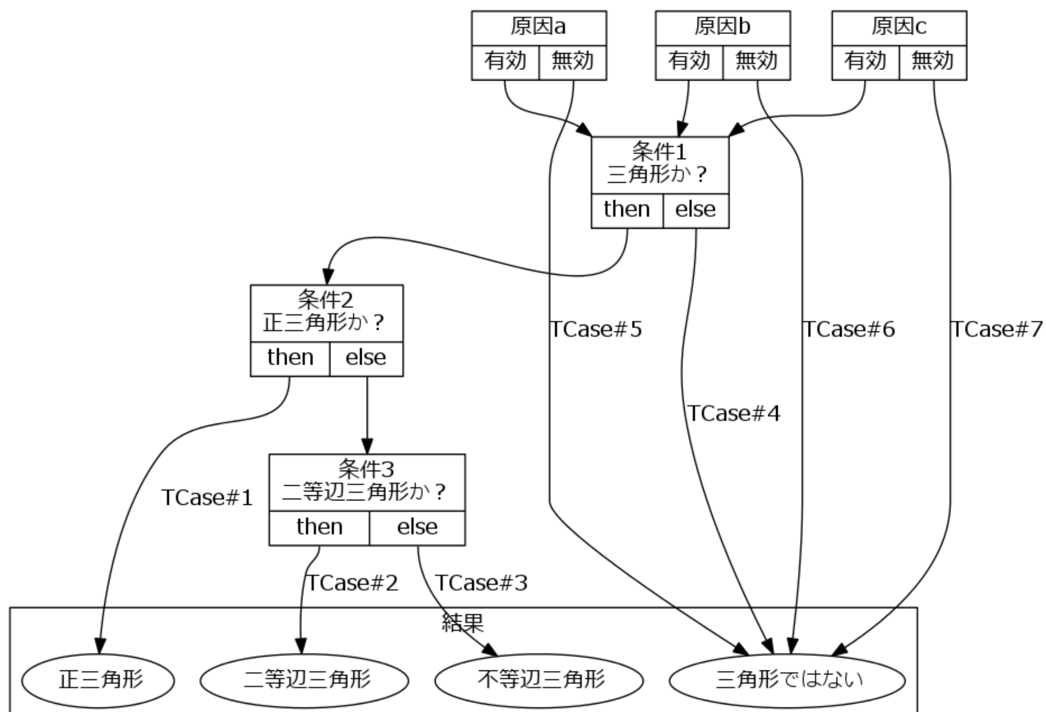
DEI

- 例えば，マイヤーズの三角形問題
  - 三角形の3個の辺の長さa，b，c　が入力された時、その三角形の形状は正三角形か、2等辺3角形か、不等辺三角形かを識別する．
- 経験的な方法ではなく，
- 漏れなく，テストケースを抽出する技法を使えますか？
  - *マイヤーズは，原因結果グラフを使うことを推奨しているが*
  - *回答を示していない*

© DebugEng Debug Engineering Institute

CFD技法なら

DEI



© DebugEng Debug Engineering Institute

テストのデバッグ課題

- テスト対象　triangle.c　ドライバー testdriv.c
- テストデータ sample.in
- コンパイル
  - *gcc -fprofile-arcs -ftest-coverage -o test testdriv.c triangle.c*
- 実行
  - *bash-4.1$ ./test sample.in*
- 網羅確認
  - *gcov -b triangle.c*
  - *File 'triangle.c' Lines executed:100.00% of 12*
  - *Branches executed:100.00% of 20*
  - *Taken at least once:80.00% of 20*　もれ発見
  - *No calls*
  - *triangle.c:creating 'triangle.c.gcov*　詳細はここを見る

© DebugEng Debug Engineering Institute

探索漏れの課題

- 関数や機能単体については，
  - *系統的なテスト技法（設計技法でもある）*
    - マイヤーズの三角形を実装する“良いプログラム構造”を示せ！！
  - *カバレッジツールによるテストのデバッグ*
- 統合テストの場合は
  - *探索空間のズームアウト：　何を網羅するのか*
  1. *機能の網羅　　機能とは，結果の同値に相当する*
  2. *関数やメソッド網羅　　Toolが使える*
  3. *対象範囲を絞ったgcovなどのTool*
- 回帰テストの場合は
  - *前の版との差をToolにより検出*

© DebugEng Debug Engineering Institute

それでも漏れる部分　→ 設計側で検証機能を付加

DEI

- 内部に隠ぺいされた状態変数
  - *原理的にテスト不能です*
  - *検証を可能にするため，状態のRead/Write 機能を実装する*
  - *ICの設計，論理回路設計では常識*
    - ソフト技術者の中には，原理的にテスト不能であることを知らに者が居る？
- 誤って（無意識に）隠ぺいされた状態変数を作る
  - *初期化していない変数の使用*
  - *external で定義した変数で，繰り返し参照，更新されるもの*
  - *Toolで検出困難　（原理的にはデータフローと制御フローの解析）*
    - 常数（定数）は，本当に定数なら問題ない
    - 特別に設計された変数以外は，automatic とする

© DebugEng *Debug Engineering Institute*

網羅チェックで問題になるループ

DEI

- ループ処理は，あるデータ構造に対する処理
- よって，データ構造と共にテストを設計すること
  - *データ構造を処理するためにループが有る*
  - *データ構造を処理する時の条件を洗い出し，網羅的なテスト設計*
  - *コードのループ処理から，逆に推測するのは困難*
- 例　マスター更新（マスターファイルとトランファイル）
- つまり，設計者は
  - *データ構造を正確に示す*
  - *起こり得る原因と対応する処理（その結果）を明示する*

© DebugEng *Debug Engineering Institute*

探知効率の問題

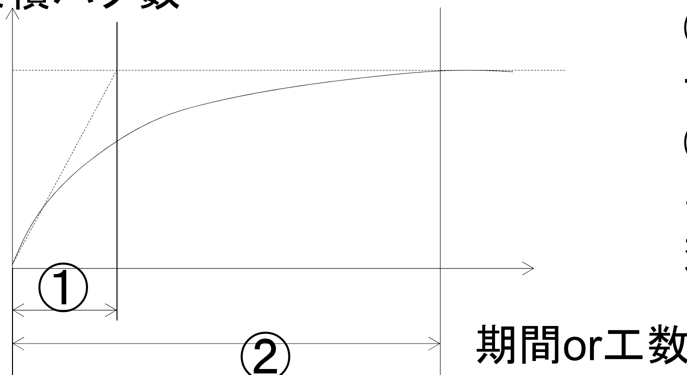
- 漏れ問題の背景に
  - *漏れを防ぐには　⇒ テストケース数の増加，さらに爆発*
  - *テストケースの実行ができない．*
- **テストの杭打ち問題**
- 網羅地盤まで達していないが，納期があるので

- 「探知効率」が良くなれば探索漏れも改善される
- 「探索漏れ」が少なくなれば探知効率も改善される
  - *探索の優先順位づけ技術など*

© DebugEng Debug Engineering Institute

探知効率の推測

- 行ったテストの効率が良かったか悪かったか？
- 推測方法
  - *規模辺りのテスト工数により過去や他と比較*
    - テストの品質で変わる．　比較する適切な他や過去のデータが無い
    - 実態として，評価されていない
  - *簡単な方法　バグ成長曲線を使う*
    - 観測された単位期間中最大の傾きから理想を推測



②実測された期間
ランダム探索
①理想的な期間
平行探索
効率＝①／②

© DebugEng Debug Engineering Institute

無知の探索⇒ランダム探索

DEI

- 効率を落とす一番の原因は，ランダム探索
- 多くは，意図しないランダム探索（モデル化不足）

1. 改修，変更とテストのコンカレント問題
2. 探索結果から探索計画をつくらない問題
   1. *推測の探索地図とバグ確率*
3. 探索地図を書けない問題
   1. *タイミングや波形に関するテスト*
   2. *性能問題，デッドロック問題*
   3. *状態間の組合せ*
4. 最適化の概念欠如
   1. *限られたリソースの配分，輸送問題*

© DebugEng *Debug Engineering Institute*

大局的なモデルで考える

DEI

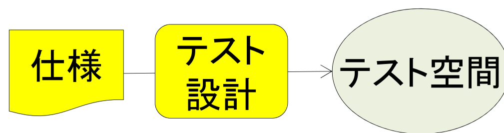
- 様々な工夫が存在する
- 今までの常識は，非常識

- 例えば，
- 「仕様ベースのテスト」に固守する石頭

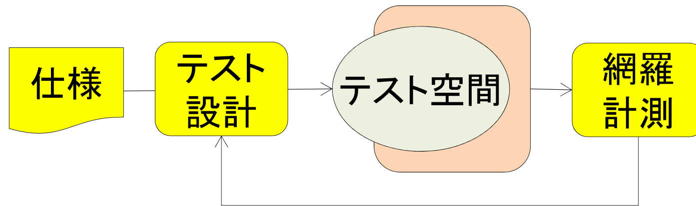
© DebugEng *Debug Engineering Institute*

仕様ベースのテスト

- デバッグなし仕様ベース



- デバッグあり仕様ベース



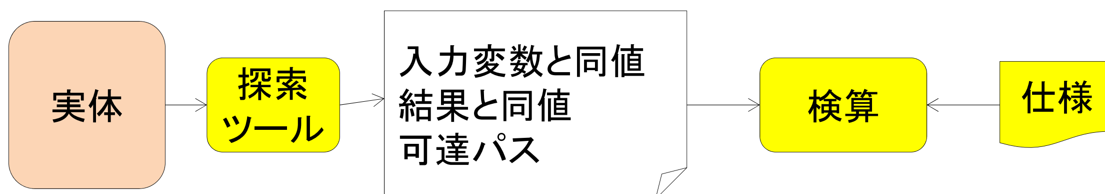
テスト設計だけで
網羅するには
高スキル＋工数

網羅計測により
フィードバックあり
テスト設計は
そこそこ

© DebugEng Debug Engineering Institute

実装ベースのテスト

- 実装ベースのテスト



- 検算の本質：何を比較するのか
  - *テスト入力と結果を使ったテスト　小さな点の集合で網羅を考える*
  - *実装ベースでは，可達パスから網羅を考える*
    - 可達パスの特性を考慮すれば，大幅な改善

© DebugEng Debug Engineering Institute

まとめ

DEI

- 大局的なテストのモデル
- 探索理論
- リソースの配分問題
- などなど，ブリコラージュ＋科学的手法

© DebugEng Debug Engineering Institute

DEI

<テストから観た実体のモデリング>

まとめ

© DebugEng Debug Engineering Institute

まとめ

DEI

- 老人の独り言
- 現在のテストは，行き過ぎた規範的でプロセス指向
- 実務現場は「杭打ち問題」に近い
  - *本物の杭打ち問題との違いは＜正しい Vs. えせ＞の違いが逆転*

- 解決に向けて
- ソフト・エンジニアのスキルは，ブリコラージュ
  テストについては，テストのデバッグによる強化
- 真のエンジニアリングは，
  モデル化ー＞方式ー＞ツール化
- 組織としては，良い武器を与える
  - *オープン系の活用*

© DebugEng *Debug Engineering Institute*

Q&A

DEI

© DebugEng *Debug Engineering Institute*