IBM

ソフトウェアテストシンポジウム2009 北海道

# テストやレビューのプロになるために
# -- 計測力と洞察力強化のコツ・ノウハウ --

Nobuhiro Hosokawa (carvin@jp.ibm.com)
Quality Assurance, Services Quality.
IBM Japan Ltd.

# Takeaways

- 計測は力なり – メトリクス計測と産業
  - メトリクス・データ
  - 定量可視化技術
  - …で、測ってどうするのですか？
    - データ測定することの意味
- プロの洞察力・集中力
  - 事例 - 問題意識
  - 集中と没頭
- プロの発想力・創造力

IBM

# メトリクス計測と産業

# テスト技術者に必要な資質とは？

× Inspiration

× Intuition

○ Insight＝洞察力

# 品質を「ハカる」について

- 「はかる」という漢字は多数存在します：
  - 計る 数・時間を「計算・計画」する。例：タイミングを計る
  - 測る 長さ・面積を「測定・測量」する。例：角度を測る・熱を測る
  - 謀る だます・計略にかける。例：暗殺を謀る・悪事を謀る

品質は「図る」でありたい

- 品質＝「図る」意図・工夫をもって「合理化する・解決する」

# 予測と予防：成功の鍵は“測定”

- 一般的にはイロイロ測定されている

何のメトリクスデータ測ってますか？

データ測って何につかってますか？

- 例）「見積り用規模・生産性データ」

  → 測って使って見積もり精度が本当に向上する？

- **品質のメトリクス測定**

# 計測は力なり – メトリクス計測と産業

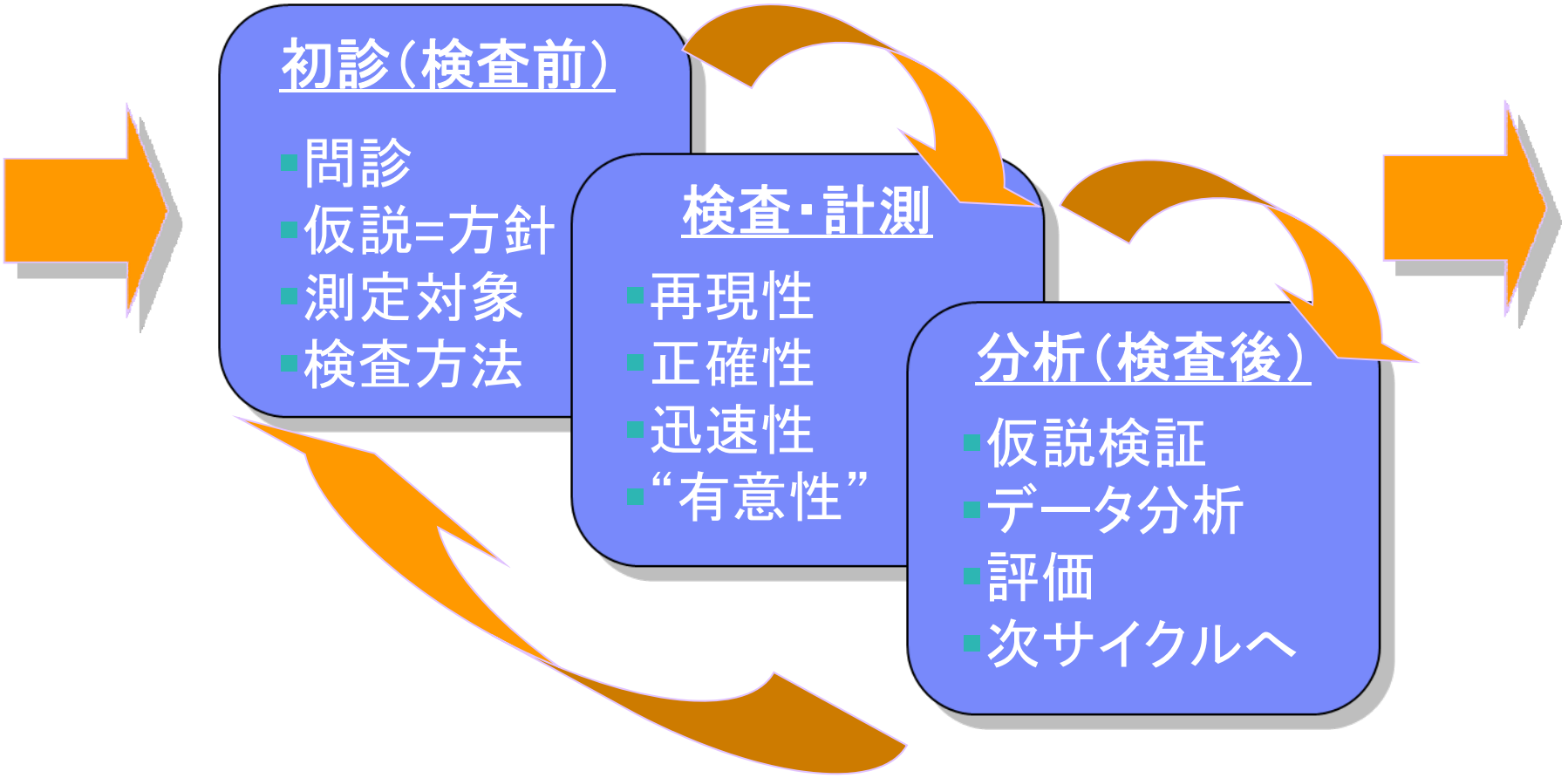
- 産業の中で「測定」を行わない商売を挙げてください
- 日常行っている「計測」作業があれば教えてください

# メトリクス測定の動機

- 品質に関する証拠・確証がほしい
- 製品リリースに拠り所や基準がほしい
- より早い段階で欠陥（とその兆候）を捕らえたい
- 計測データなしではプログラムを納品できないから
- 理由はないが決まりだから・仕方なく
- 数字を測るとなんとなく安心する
- 定量データ好きの管理者がいる・測定マニアがいる
- 経営指標として測定が義務付けられている

# 医学の場合

- 医療行為のサイクル



# 医学の場合：１）計測前

- 医師による診療行為ははじめに「視診」や「聴診」「問診（医療面接）」
- いわゆる「の見立て」。患者の顔色や訴えをよく聞きながら病気のさまざまな可能性を疑う。
- あの病気の可能性がある、こういう病気もありえるといった思考（テスト技術者や品質エンジニアも同じ）
- 初診の結果、病気が特定できてもできなくても、医学の測定活動である「検査」実施。
- 見立て＝医師の主観。検査＝客観的かつ科学的な根拠に基づく定量データの裏づけ・確証
- 検査には次のような種類が存在する

  - **病気の有無・病気かどうかを判断する**
  - **病名を特定する**
  - **病気の程度を調べる**
  - **治療方法を決定する**
  - **治療効果や再発を調べる**

- 医師は、さまざまな検査方法の中から、現在の患者の状態（容態）の中から検査手法を選択。この検査手法は

  - **何のメトリクス（＝指標）を計測するか**
  - **どの検査手法でそのメトリクスを得るか**

  の観点から選択・決定される。
- 「あの病気であればこの指標が異常値を示すはずである」「この値が正常で、この値が異常値であればこの病気であると判断できる」といった科学的根拠と、実績のあるメトリクスを選択する
- また同時に、「この容態の患者にはこの検査は困難である」、「この検査手法は費用が高い」といった事前条件を踏まえたトレードオフも考慮

  （これもテスト技法が目的やテスト戦略に基づいて決定されるのと同じ）

# 医学の場合： 2）計測・検査実施

- 医師の検査オーダーを元に、検査技師が検査を実施。血液や細胞といった患者の一部を採取して行う検査（検体検査）や、運動能力検査( 生体検査）といった各種専門検査を実施。
- 検査結果は「物質量（定量数値）」や「プラス・マイナス」といった形で報告される

1）再現性：何度測ってもばらつきが少ないこと
2）正確性：正しく現象を把握していること
3）迅速性：結果がすばやく得られること
4）「測定結果データ」が医師による診断と治療に役立たなくてはならない

- これもシステム開発において、テスト技術者や品質エンジニアが過剰品質を避け、コスト・期間負担を最小にする方法と同じ。
- なお、得られた貴重な定量データは、データそのものを患者に報告することはあまりない

# 医学の場合：3）計測後

- 初診で得た仮説について、定量データを読み解くことで評価し、裏づけ、修正し、病気を特定する。
- 「異常なし」「様子見」「さらなる精密検査」が必要かといった判断を通じて次の処置を決定
- 最終的に、処置（投薬や開腹手術といった具体的な治療行為）を行い、次回検査で最初の「見立て」と、今回施した処置が正しかったかを判断しながら、もう一度次のサイクルを回す

# 医学から得られる教訓

- 検査は、医師という判断をする主体に対する判断
- 意思決定材料を提供するものである
- 検査手法は、病気の有無・程度を調べる・治療法方決定・治療効果調査といった各種目的に応じて選択される
- 検査手法選択時に事前条件・トレードオフを考慮する
- 検査は精度が問われる。再現性や正確性といった要素が重要である
- 検査結果は、次の活動に対して役立つものでなくてはならない
- 検査結果データそのものは意味を持たない。データを読み解く技術が重要である。

**上記文章の「検査」を「テスト」や「測定」に、「病気」を「欠陥」に読み替えるとテスト技術者にとって、まったくそのまま当てはまる**

IBM

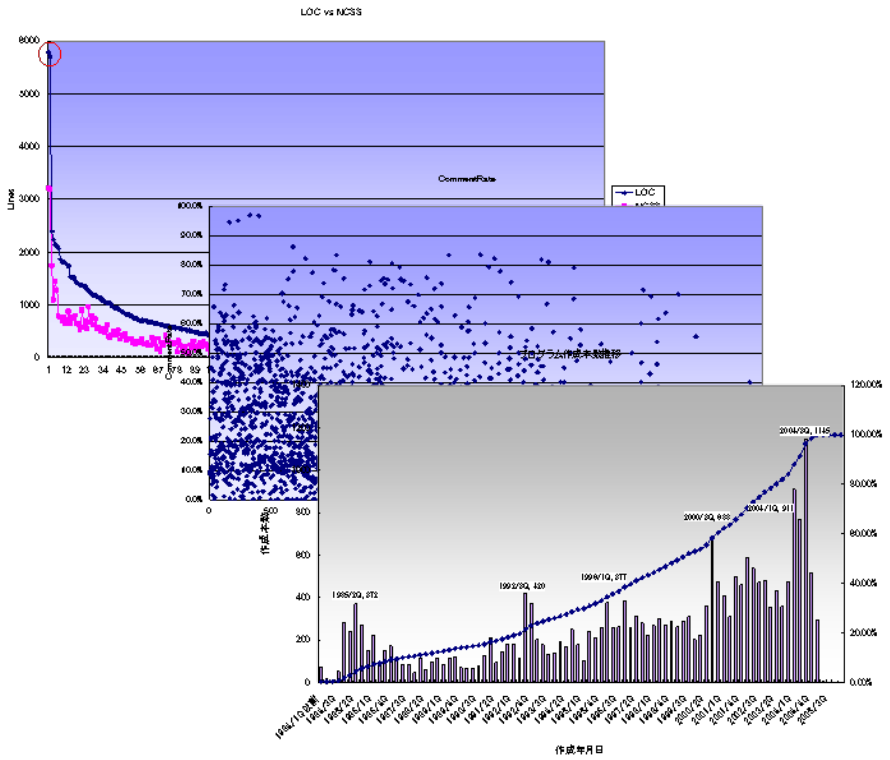
# 測定対象の決定と
# メトリクス測定の基礎技術

# では、何を測定すればいいのか？

- 様々なクラシカル・メトリクス
  - Lorenzメトリクス
  - CKメトリクス
  - Halsteadボリュームメトリクス
  - 各種サイズメトリクス
  - コメント率 （Comment To Code Ratio）
  - McCabeの循環的複雑度

:

:



# メトリクス測定の重要性とその活用

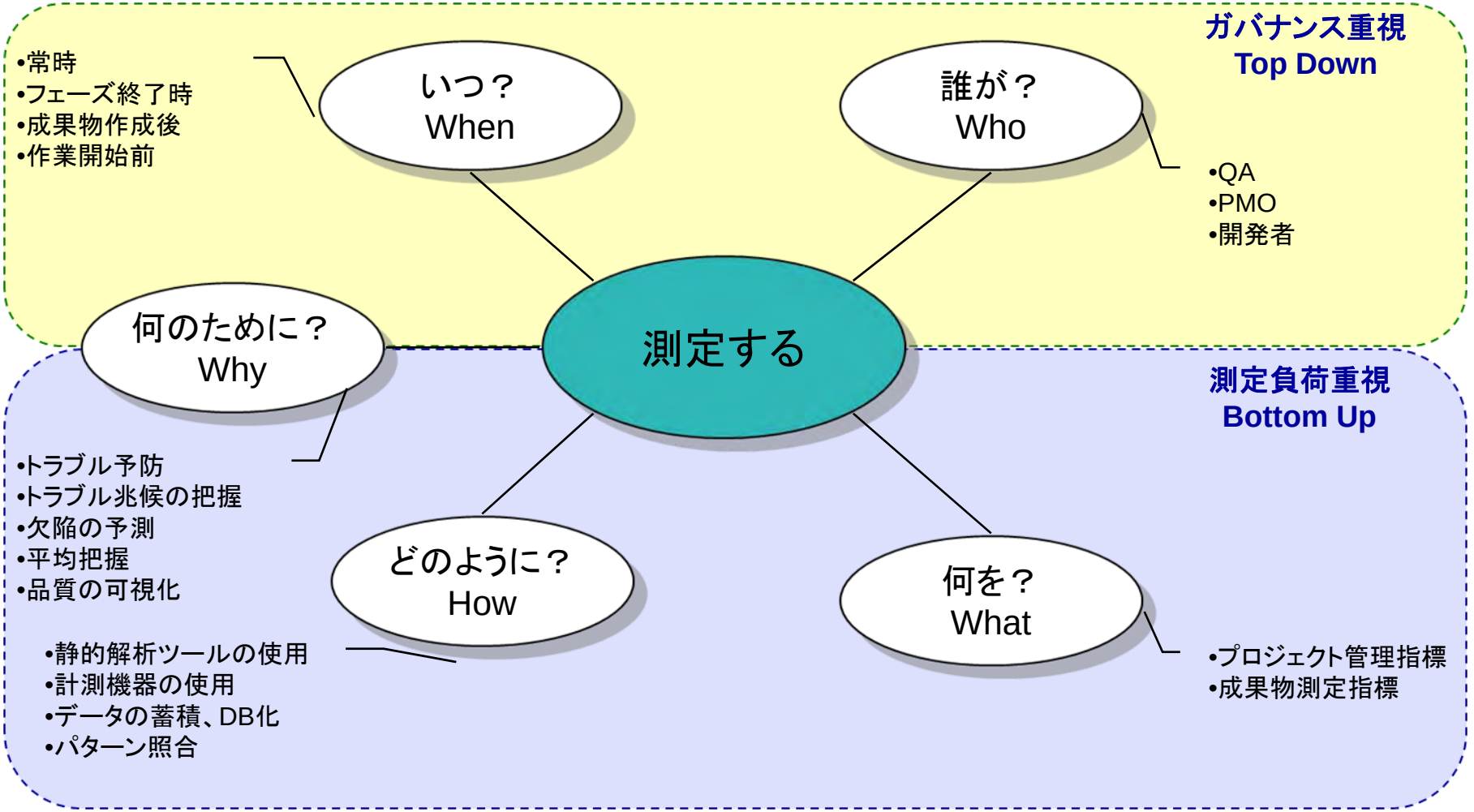
## メトリクス（＝指標）計測には2側面ある

| | 1プロジェクトのためのメトリクス | | 全社レベルのメトリクス | |
|---|---|---|---|---|
| 測定対象 | 正常案件 | トラブル案件 | 正常案件 | トラブル案件 |
| | •モニタリングのための品質メトリクス測定 | •トラブル回復のためのメトリクス測定 | •規模メトリクス<br>•生産性メトリクス<br>•リスク候補 | • ビジネスインパクト<br>• 検出時期<br>• 検出時の出来高 |
| 提供データ | • プロジェクト健全性メトリクス（実データ＋平均値）<br>• プロジェクトマネジメント技術の一部 | | •失敗・成功パターン<br>•ナレッジ （＝べし・べからず集） | |
| 特徴 | • ヒューリスティック・ベース<br>• 場当たり的、しかし最もタイムリー | | • ゴール・ベース<br>• 測定フレームワークに依存<br>• クラシカル・ソフトウェア・エンジニアリング | |
| 目的 | トラブル予防のための品質メトリクス | | トラブル予測のための品質メトリクス | |

# 効果的なソフトウェアメトリクスの持つべき属性

- 単純で計算可能なこと
  - 比較的習得しやすく、計算に時間がかかりすぎてはならない
- 経験や直感に照らし合わせても説得力があること
  - 技術者が直感的に持つ感覚と一致しなければならない
- 一貫性があり客観的であること
  - メトリクスは常に明確な結果を出さなければならない
- 単位や次元が一貫した使い方をされていること
  - メトリクスを計算する場合には、妙な単位の組み合わせにならないようにする
- プログラミング言語から独立していること
  - メトリクスは分析モデルや設計モデル、あるいはプログラムの構造そのものから導出されなければならない
- 品質を改善するための効果的なメカニズムがあること
  - メトリクスはより高い品質の製品を作り出せるようにしなければならない

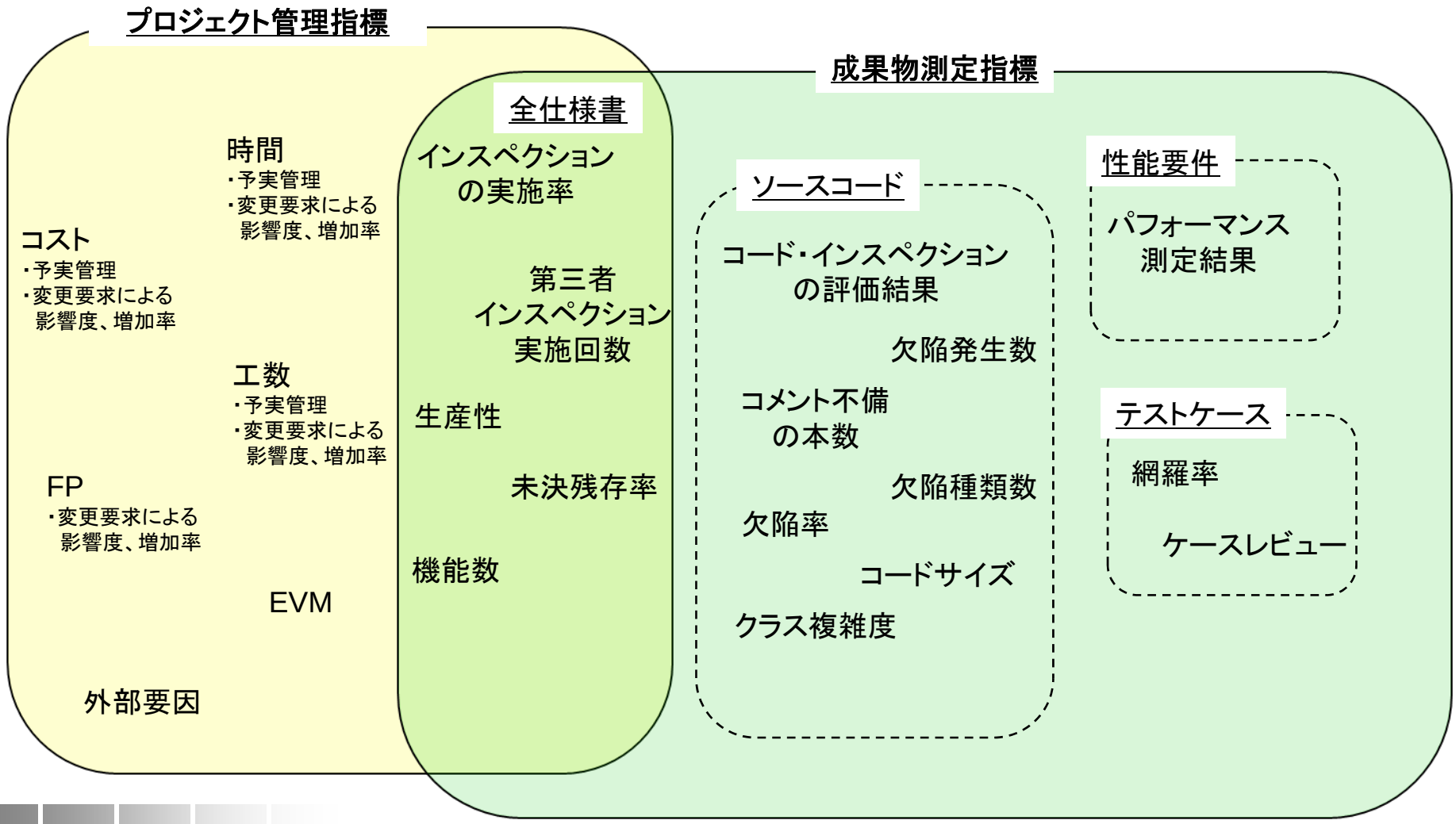
# 測定の5W1H


ガバナンス重視
Top Down
•常時
•フェーズ終了時
•成果物作成後
•作業開始前
いつ？
When
誰が？
Who
•QA
•PMO
•開発者
何のために？
Why
測定する
測定負荷重視
Bottom Up
•トラブル予防
•トラブル兆候の把握
•欠陥の予測
•平均把握
•品質の可視化
どのように？
How
何を？
What
•静的解析ツールの使用
•計測機器の使用
•データの蓄積、DB化
•パターン照合
•プロジェクト管理指標
•成果物測定指標

# 測定はいつ行うべきか


要件定義
基本設計
詳細設計
①常時（日次～定期的）
②フェーズ終了時
③成果物作成後（Adhocに）
④作業の開始前
（ベースライン法）
コーディング・UT
統合テスト
システムテスト
ユーザー受入テスト
①
②
③
④

# 何を測るか？


プロジェクト管理指標
時間
・予実管理
・変更要求による
影響度、増加率
コスト
・予実管理
・変更要求による
影響度、増加率
工数
・予実管理
・変更要求による
影響度、増加率
FP
・変更要求による
影響度、増加率
EVM
外部要因
成果物測定指標
全仕様書
インスペクション
の実施率
第三者
インスペクション
実施回数
生産性
未決残存率
機能数
ソースコード
コード・インスペクション
の評価結果
欠陥発生数
コメント不備
の本数
欠陥種類数
欠陥率
コードサイズ
クラス複雑度
性能要件
パフォーマンス
測定結果
テストケース
網羅率
ケースレビュー

IBM

# 何を測るか？ メトリクスの“驚くべき”歴史

- 以下にプログラム行数（ステートメント数）がいかに曖昧なメトリクスかを示す歴史的研究を紹介します

- 行数は「一行の長さ」が一定の場合に初めて有効になる。(50,72/80/128等)

- 一行に複数文を許すプログラム言語では、例えばループ全体を1文にして見掛けの複雑性を減らすこともできる。

プログラムが小さいうちは、他のメトリクスと同程度にプログラム行数は有効であるが、大規模プログラムに適用すると実際の値をかなり下回る (Curtis, Sheppard, Millimam('79))

そもそもプログラム言語毎に「ステートメント数」の定義が一定ではない

**プログラム行数と欠陥発生の間に 顕著な相関関係はない（Rubey('75)）**

小さいプログラムの不良発生率は1．3%から1．8%であるが、大きなプログラムでは、2．7%から3．2%の範囲に上昇する。バグ発生率とプログラム行数がほぼ線形に増加するのは100行以下の小さいプログラム。

1ステートメントあたりのバグ発生率と、ルーチンのステートメント数の間には、**非線形の相関関係**があるが、プログラム言語特性や、モジュール化（＝設計）による影響が大きい

# メトリクス測定の重要性 :メトリクスの歴史

1950年代中旬 :「まずプログラム行数を数えること。
知りたいことは何でもそこから導出できる」

Zogorski('65), Weinwurm('67)の研究: コスト予測の「最初の研究」で行数を否定

1. 「ソフトウェア開発コストはプログラムの行数という単純なものとは直接何の関係もない要因などに大きく依存する」
2. 迷信の否定:「リアルタイムソフトはバッチより難しい」 → ×
3. コスト予測は、開発サイズに依存する

1980年代以降: 「大きさ」に関する研究

- ソフトウェア定量化の基本命題=「どれくらい大きいか」
- プログラム行数=主観的尺度として否定
- 近代的なソフトウェア開発モデル(コスト、期間、工数...)

- メトリクスの歴史は「プログラム行数の否定」によって作られてきました。

# 参考） メトリクス測定・定量化の基本命題＝「どれくらい大きいか」

- プログラムの「大きさ」をどのような尺度で捕らえるかが歴史上未だ解明されていない大命題です。ただしもしも、本当に「プログラムの大きさ」をプログラム行数で表すことができるなら、不思議な現象が発生します

例１）欠陥密度による残存欠陥の予測

**欠陥密度 =単位行数（一般に1000行＝キロ行）あたりの欠陥数**

もし上記が本当ならば、
１） よく設計されて構造的複雑度も全く整った短い（100行程度）ソリッドなコードに、汚いバグ入りメソッドを1つ追加すると値が突然跳ね上がる
２） １本10000行のコードに100個欠陥があるのと、100行のコードに1個あるのは同じ品質

- 例２）総プログラム行数によるシステム工数見積り

**プログラム行数だけで全システムの開発工数が見積もることができる**

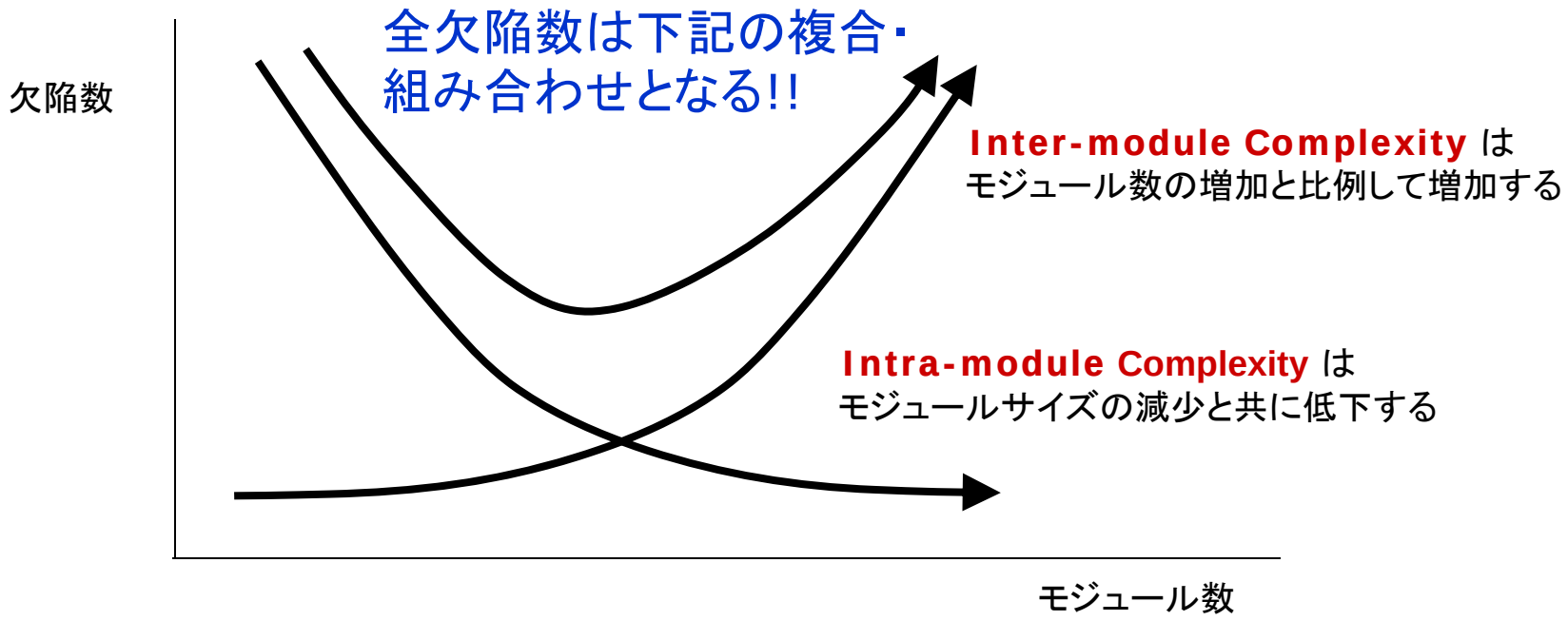
もし上記が本当ならば、
１）ソースコードを全部プリントアウトして、そのダンプの紙束の厚みで見積りが可能
２）もっと簡略化する方法として、ソースダンプの「目方・総重量」で見積りが可能

# コードメトリクス： 複雑度の考え方

- 複雑度には2つの考え方があります

①Intra-module complexity （例： Thomas McCabe's Cyclomatic Complexity）
  - 1モジュール内の複雑度

②Inter-module complexity （例： Henry-Kafura's Inter-Module Complexity ）
  - 他モジュールとのインタラクションの複雑度

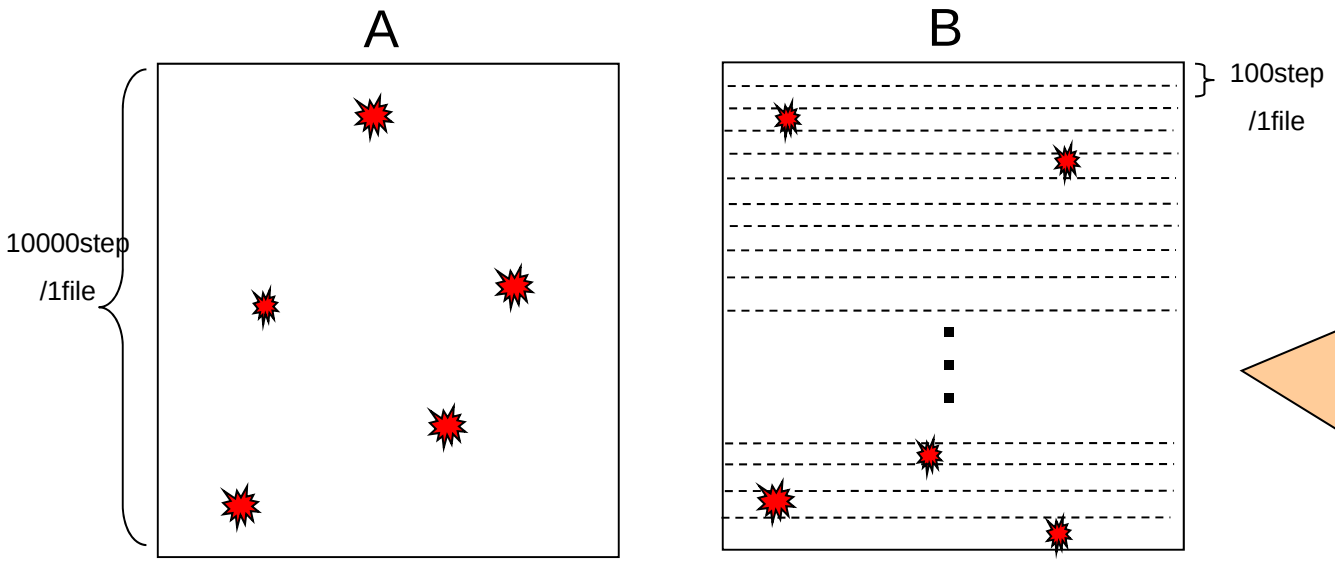


# 参考）欠陥密度：算出結果の比較（よくある誤り）

- 例1）同じ欠陥密度≠同じソフトウェア品質
- ⇒部品粒度（＝設計仕様）を考慮していますか？
  - 欠陥密度: 0.5/KLOC（10KLOC:10000ソースコード行あたり5つの欠陥）

project A ：total 10000step、ファイル数1、欠陥数5

project B ：total 10000step 、ファイル数100、欠陥数5




同じ0.5/KLOCの結果でも、元のクラス設計が異なる例。単純に算出結果のみを比較するのは危険。

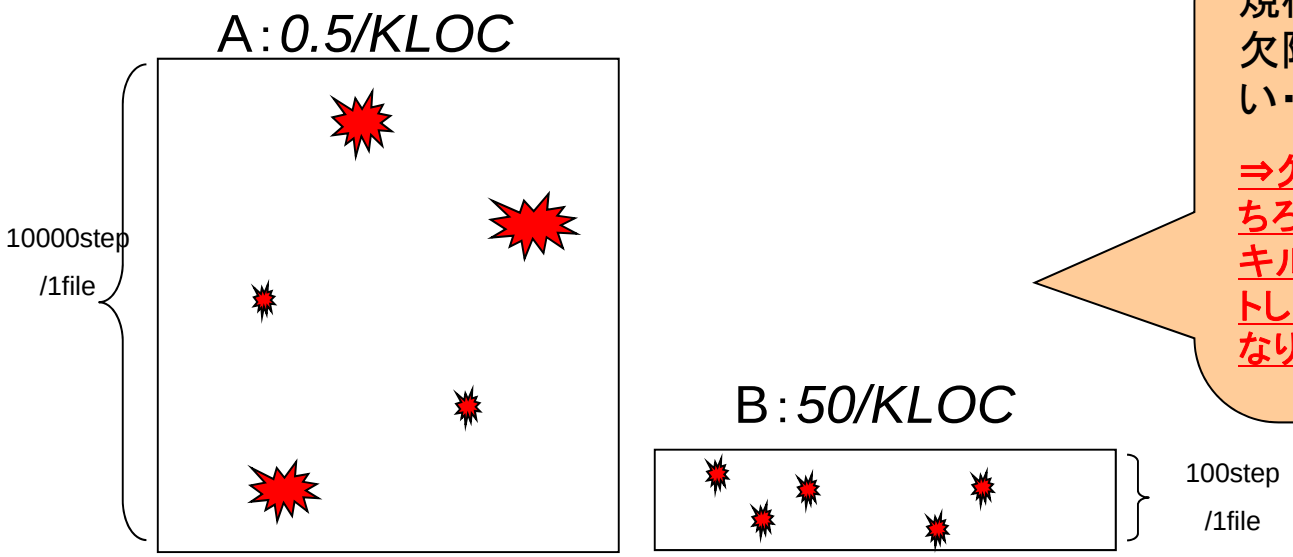
⇒「クラスサイズ」「クラス数」で割ってクラス当たりの欠陥密度」にして比較。

# 参考）欠陥密度：算出結果の比較（よくある誤り）

- 例2）欠陥密度の高さ・低さ≠ソフトウェア品質の高さ・低さ
- ⇒他プロジェクトの欠陥密度の絶対値と比較して、品質を評価していませんか？
  - 欠陥密度: A:0.5/KLOC　VS.　B:50/KLOC

project A ：total 10000step、ファイル数1、欠陥数5

project B ：total 100step 、ファイル数1、欠陥数5



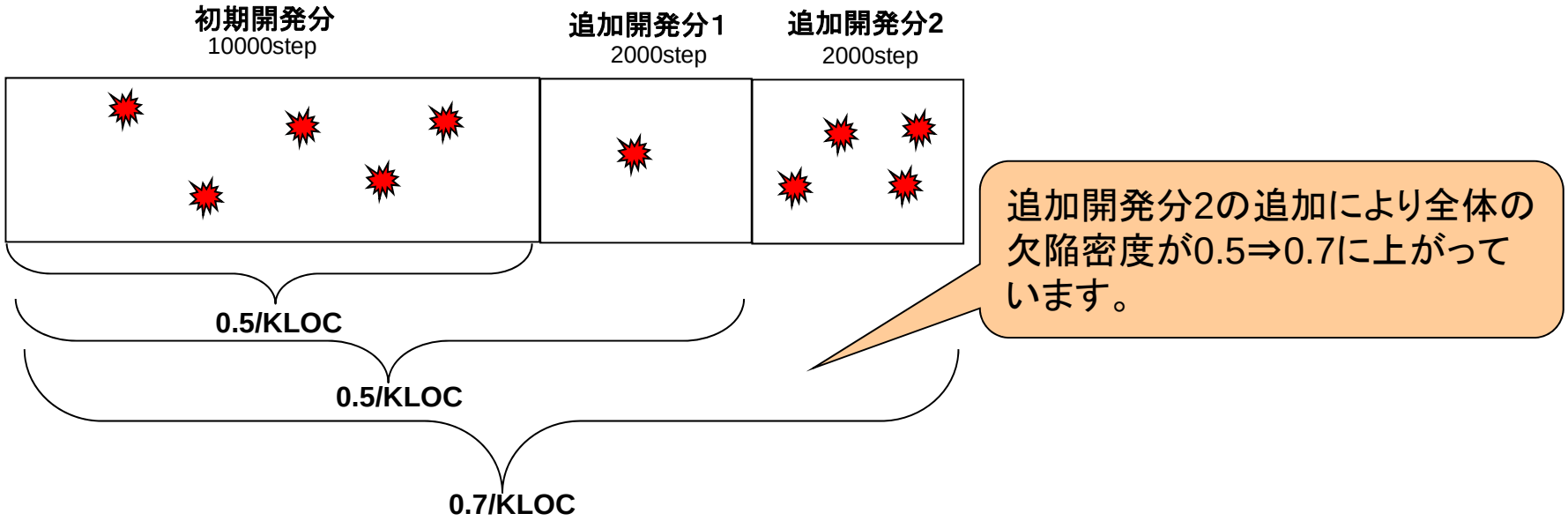

規模の異なる他プロジェクトの欠陥密度と比較して品質が高い・低いと判断するのは危険

⇒欠陥密度はプロジェクト規模はもちろん、アプリの複雑度、開発者スキル等にも依存します。また、カウントしている欠陥の種類、重要度も異なります。

# 参考）欠陥密度：算出結果の比較

- 欠陥密度：同じ算出方法、同条件下の値と比較をして初めて品質の比較が可能
  - ※ただしこの場合も欠陥の種類、重要度は考慮が必要

例）同じプロジェクトで、同じプログラム（同じ開発者、同じ複雑度）の保守を行う場合



# 参考）GQM（Goal/Question/Metrics）手法

Goal　　プロセスアクティビティや製品特性を評価するために測定目標を確立

Question 目標に到達するために答えなければならない質問を定義

Metrics　質問に答えるのに役立つメトリクスを特定

参考：Goal定義）　目的定義テンプレート

- 測定の対象　　：測定したいアクティビティや属性の名前
- 分析の目的　　：分析全体の目的
- 対象の側面　　：測定したいアクティビティや属性の側面や特性
- 利害関係者　　：この測定を必要とする人々の持つ観点
- 測定の背景　　：測定が必要となった状況や環境

# GOAL/QUESTION/METRIC 手法： アプローチ概要

- 企業全体、部門、プロジェクト各単位のビジネスゴールと生産性と品質関連する測定目標を設定する
- 定量化可能な方法で、それらのゴールをできるだけ完全に定義する質問（モデルに基いた）を設定する
- 質問に対する答えやプロセス、製品とゴールの一致を追跡するために収集すべき測定を指定する
- データを収集する仕組みを開発する
- データを収集、その場で分析しアクションを正すべくプロジェクトにフィードバックする
- 収集したデータをゴールと適合しているか、そして先々の改善に向けた推奨アクションを提示するために過去データと合わせて分析する

IBM

# 定量可視化技術

# 測定 → 視覚化のプロセス ： 測定結果の「分析・評価」とは？

1. **データ収集（Acquire）**
   - データを収集します。企業活動のさまざまなメトリクスである場合や、プロジェクト活動，プログラムから得られるソフトウエアメトリクスデータの場合もあります
2. **解析（Parse）**
   - データの意味をもとに構造を付加しカテゴリ分けします
3. **フィルタリング（Filter）**
   - 関係のないデータをすべて除去します
4. **マイニング（Mine)**
   - 統計学的手法やデータマイニング手法を適用し、パターンを見つけたり数学的処理を行えるようにします
5. **表現（Represent）**
   - 棒グラフ、リスト、Tree構造など、ベースとなる視覚化モデルを選択します
6. **精緻化（Refine）**
   - より明快で視覚的に魅力に富んだものになるよう基本表現を改善します
7. **インタラクション（Interact）**
   - データを操作したり、何を表示するかを制御したりする手段を追加します

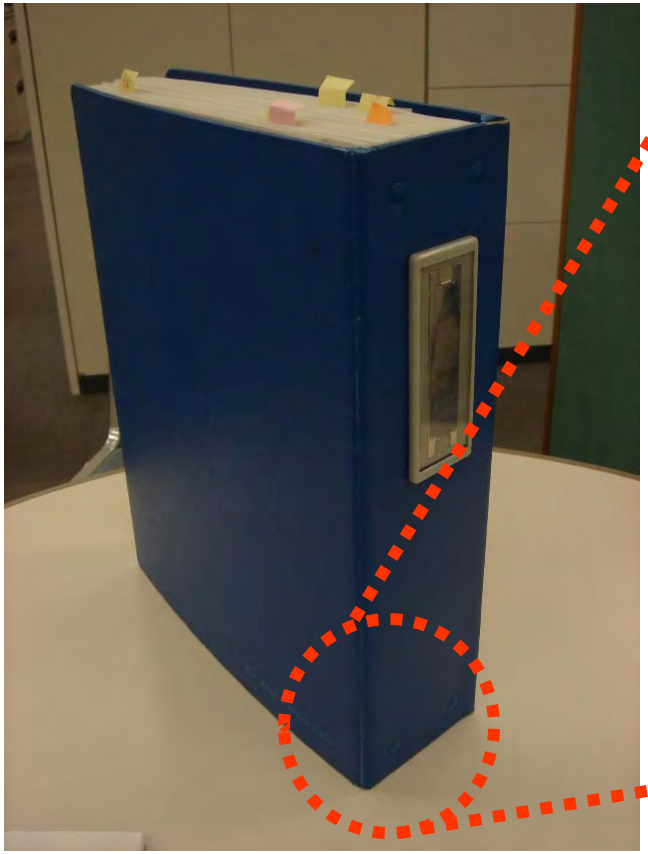
# 可視化表現の例） Tree表現

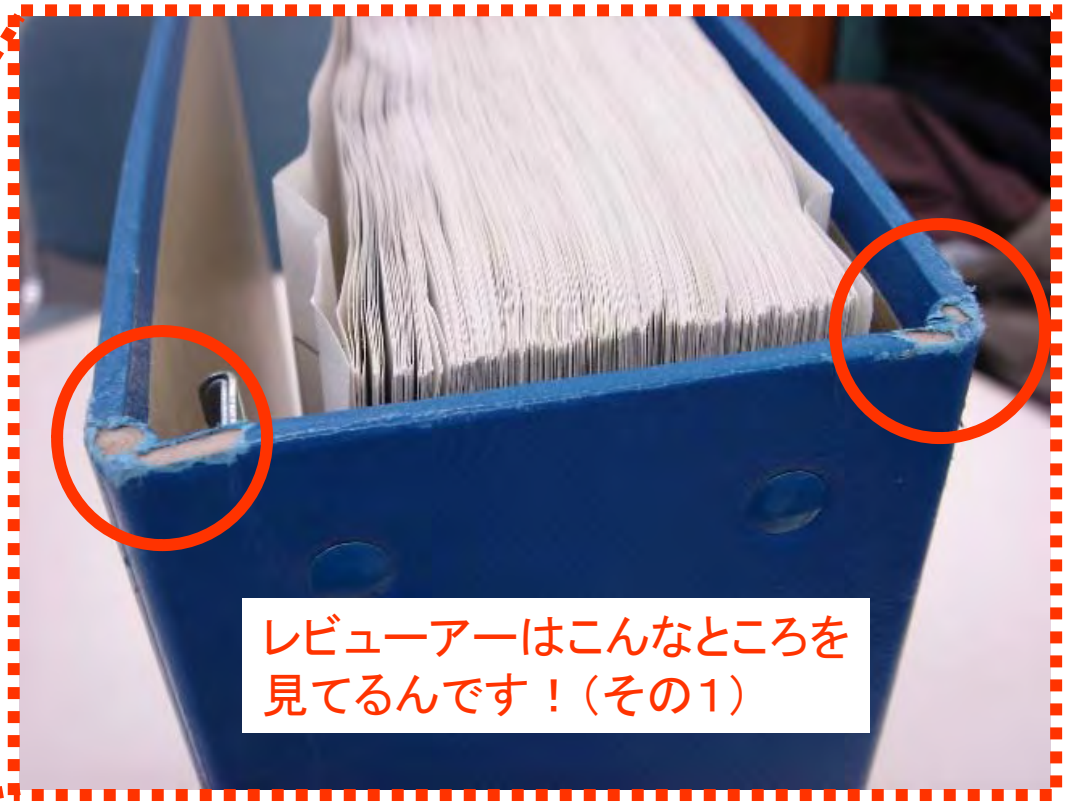
# 可視化表現の例） Graph表現

# 可視化表現の例5） Graph表現

有効だが定量測定ではない
定性的検査の例

# 測定例？） バインダーのハジっこ

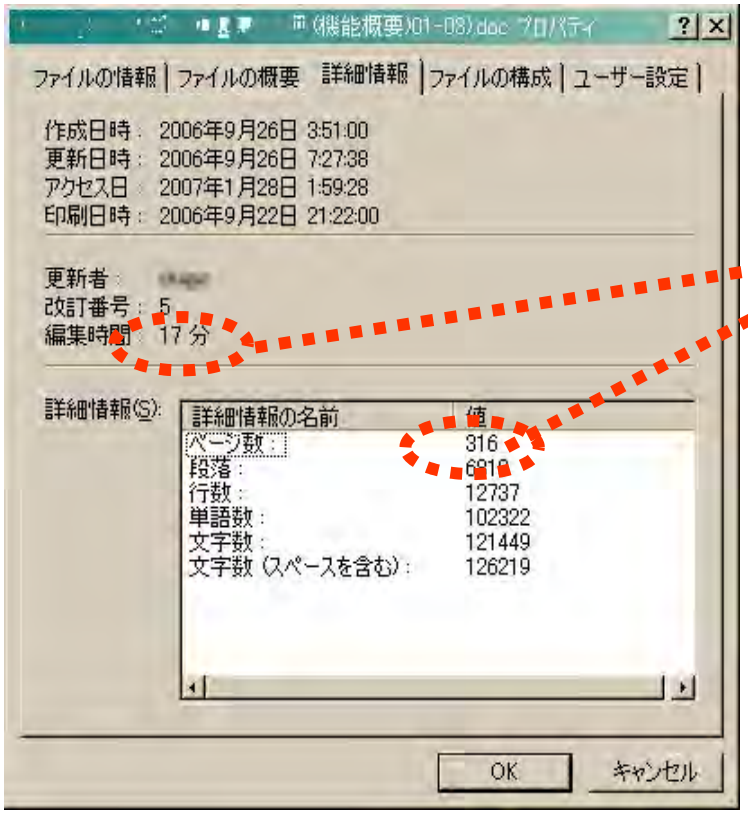





バインダーの底辺カドが擦り切れている仕様書は「使われている仕様書」である

# 測定例2） ファイルのプロパティ情報

レビューアーはこんなところを見てるんです！（その2）




- レビュー対象がMS-Officeの場合：
  - ーファイル内のページ数
  - ー編集時間
  - ー文字数

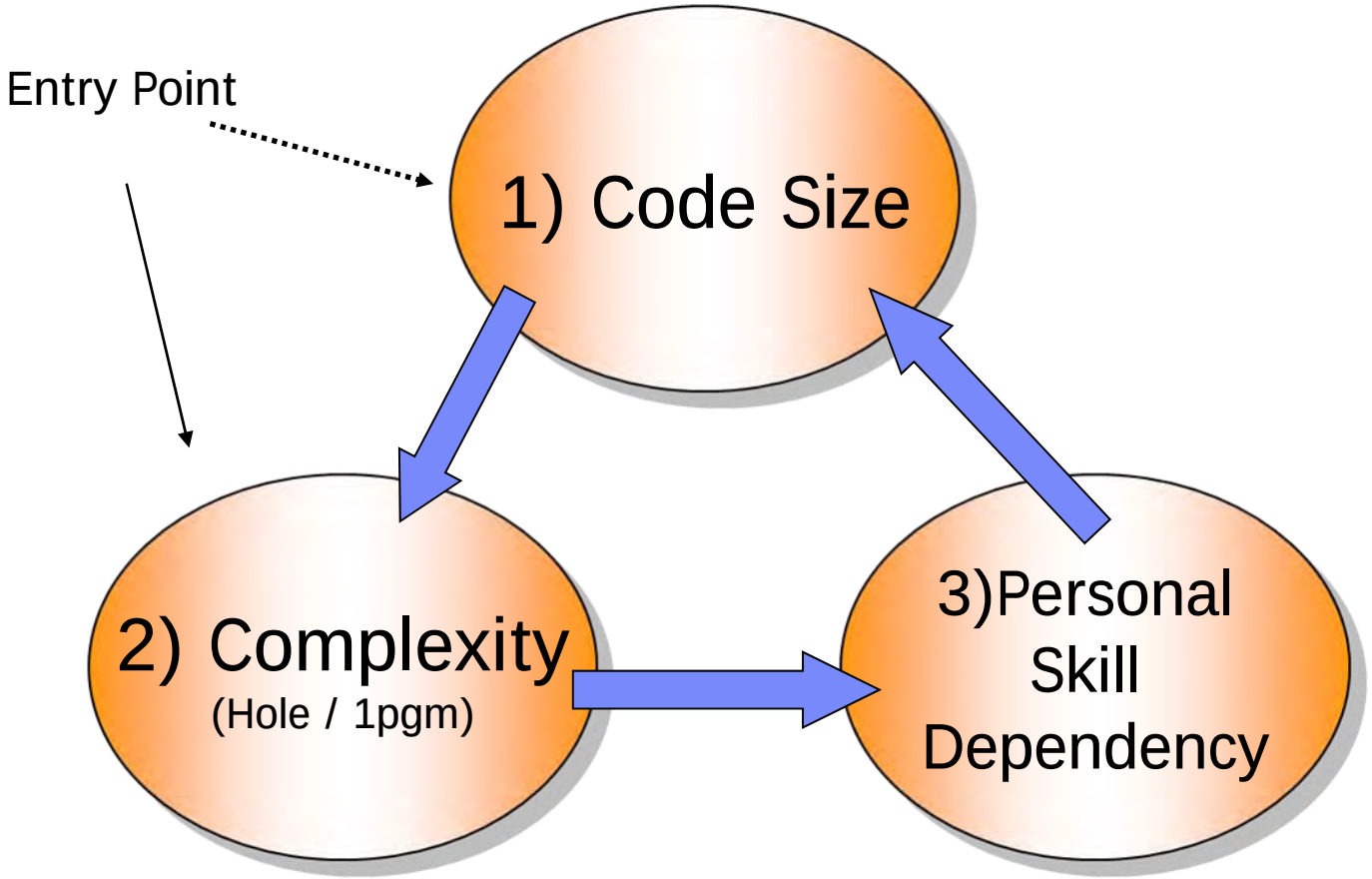
- 左例の場合、
  ファイル内ページ数　：316ページ
  編集時間　：17分
  文字数　：126,219文字

- 316頁÷17分 = 平均3.22秒／頁で作成？
- 126219字÷17分 = 123.73字／秒で入力？

- この仕様書およびプロジェクトの一番の問題は何でしょう？

# Quality Insight from source code : 3 aspects

- Base line : “Check the data from 3 aspects”

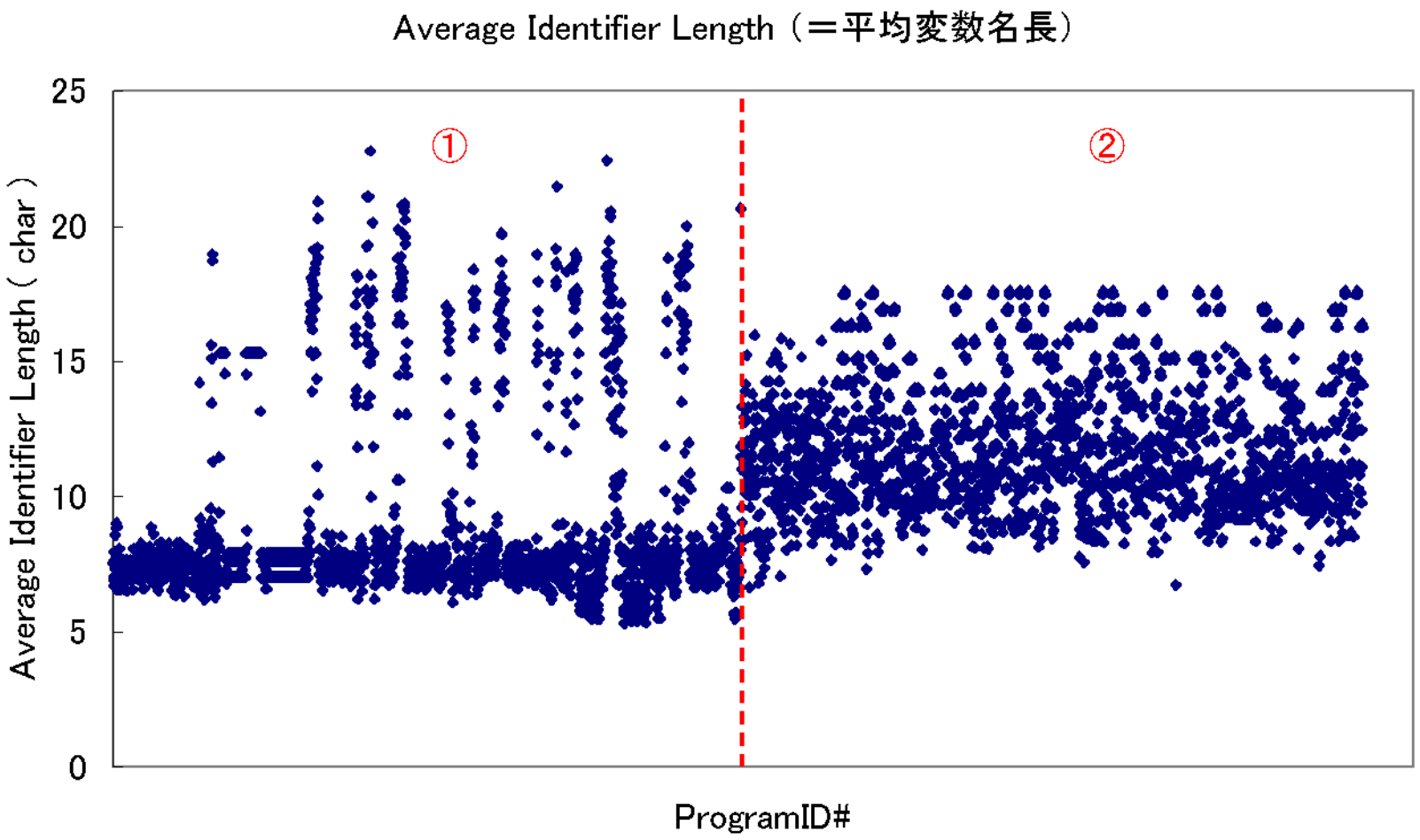


# 予測のための指標例: 平均変数名長

- メトリクス名: 平均変数名長
- 英語名: Average Identifier Length
- 単位: 1ソースコードあたり平均値1つ
- カテゴリ: 属人性指標
- 取得方法: 1ソースコードに記述された全て変数名の長さ(＝文字数)の平均値(単純総和平均)
- 備考:

当該ソースコードの開発者がどのような開発言語経験を持つか推定することができる

# 予測のための指標例：平均変数名長（AIL）の例



# 予測のための指標例：平均変数名長をどう使うか？

1）以下の判定条件式にしたがって、開発者の言語経験を推定する

| | | | |
|---|---|---|---|
| IF | 平均変数名長≦8 | THEN | COBOL言語経験者 |
| IF | 9≦平均変数名長≦12 | THEN | C/C++言語経験者 |
| IF | 平均変数名長≧18 | THEN | JAVA言語のみの経験者 |

2）当プロジェクトでの使用言語と1）の経験言語を照合し混入欠陥を予測する

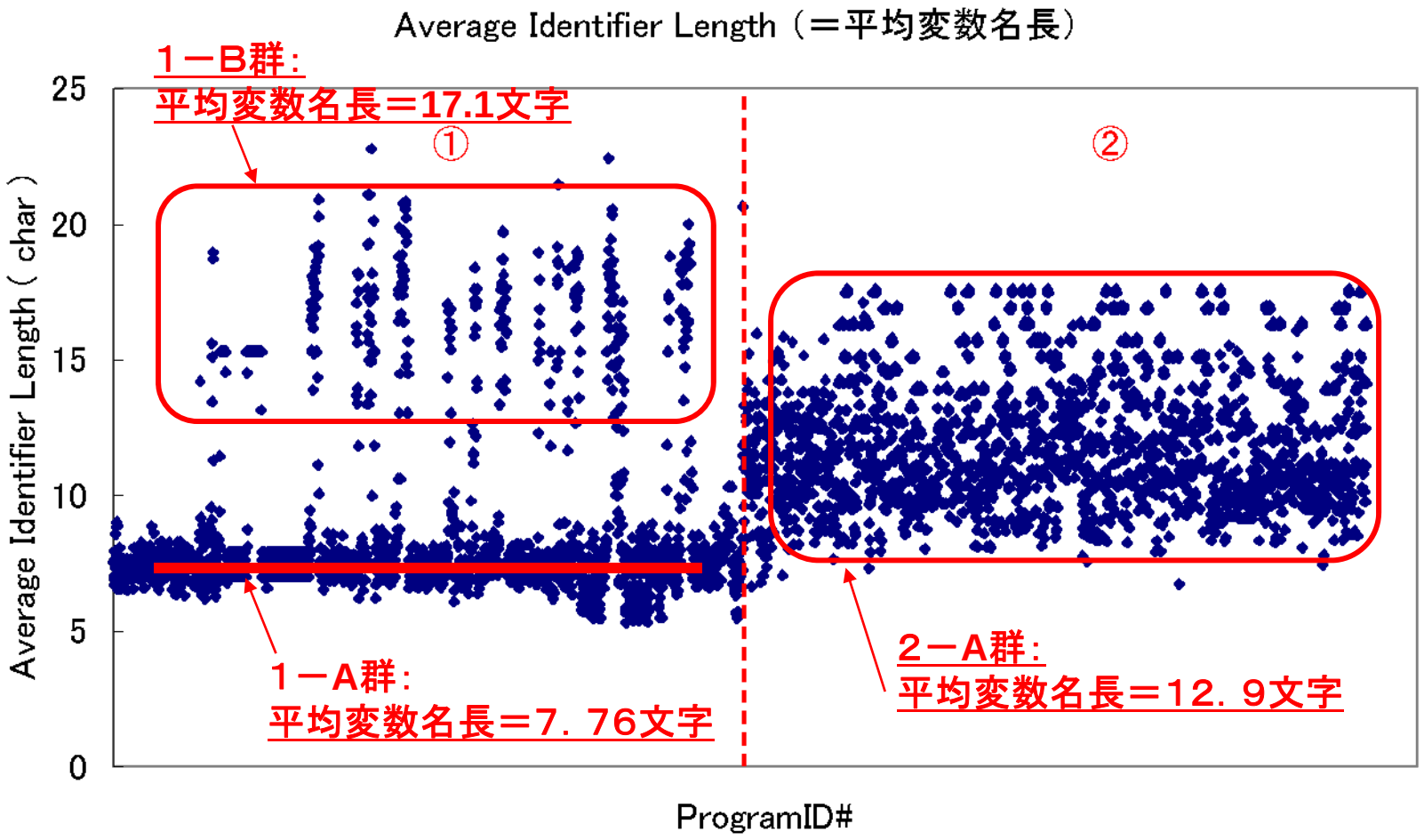
今回利用言語が「Java」で、

開発経験言語がCOBOLなら、「例外処理の漏れ／try-catch」を疑う

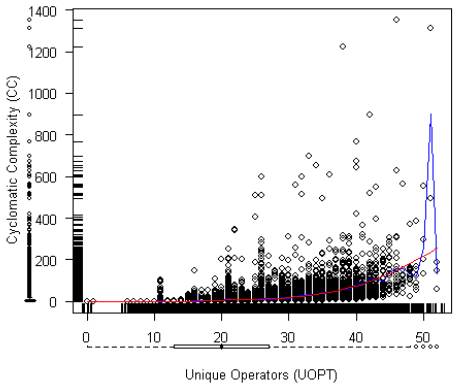
開発経験言語がC・C++なら、「Null Pointer エラー類」を疑う

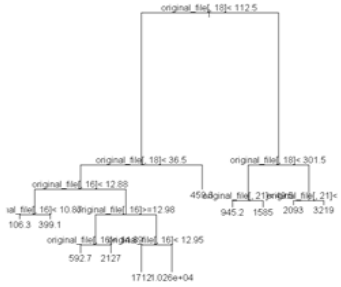
開発経験言語がJavaなら、「リソース解放忘れ／new多用」を疑う

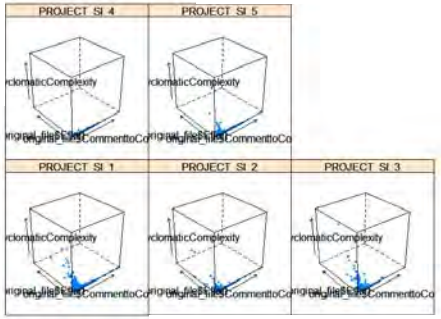
# 再掲： 平均変数名長(AIL)

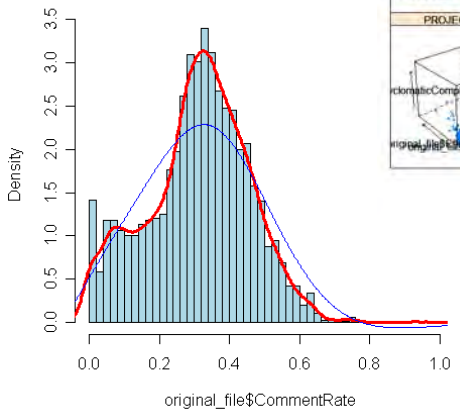


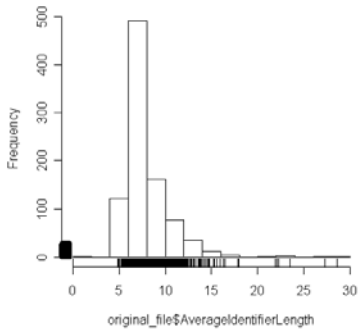
# メトリクス測定データの利用：“組み合わせて使う”






品質には絶対値がないため、品質メトリクスを単独で使っても無意味

↓

組み合わせて利用する

描画して「品質傾向を掴む」

- 技術的属人性はあるか？
- 管理に問題はあるか？
- 予防/抑止できるか？
- 試験性は確保されているか？
- 慢性か急性か？
- スキルか経験か？
- 真の原因は何か？

# …で、測ってどうするのですか？

- テストに使う
- 品質管理に使う
- 報告に使う
- 予防と予測に使う

そのデータ、本当に使えますか？

IBM

# プロの洞察力・集中力

# 誰の何のために測定を行うか？

1） 開発者自身の安心と自信のため

2） リーダー／PMにとっての、プロジェクトの成功のため

3） 企業にとっての利益のため

4） 技術者にとってのプライドのため

5） イイモノを作りたいという人間の「欲求」

# 参考）大量のJavaコードの測定結果：LOC

# 参考）大量のJavaコードの測定結果：コメント率（ヒストグラム）

# 参考）大量のJavaコードの測定結果: コメント率（ヒストグラム）

# COBOLコードの内部構造図

# 欠陥の予防と予測

- 欠陥の数を管理する → ×
  ↓
- 欠陥の種類を特定する → ○
  ↓
- 欠陥原因を特定してから除去
  ↓
- 再発予防する＋次を予測する

**Point）過去経験から欠陥を知ることが重要**

- 当該業務・アプリ固有の予測できる欠陥は？
  - 例）与信審査業務＝長いワークフロー＝承認却下時の例外設計漏れ
- 当該プロジェクト体制固有の予測欠陥は？
  - 例）COBOL経験のあるJava初心者＝try-catch構文のcatch節の抜け欠陥

# 定義: “ヒューマン・エラー”

ヒューマン・エラー :

システム最終成果に対して確実に欠陥を埋め込むことになる、 開発者が当然行うべき必要な行動の欠如または誤り

- ヒューマンエラーの原因を突き止めない限り、今、目の前で除去した欠陥が必ず再発する
- もし誰かが犯したヒューマンエラーを、自動的に検出／指摘することができたら、確実に埋め込まれてしまう　未来の欠陥を事前に予防することができる

# 2つの予防技術:EB & ESR

上流: エラー・ブロッキング (EB)
(EB: Error Blocking)

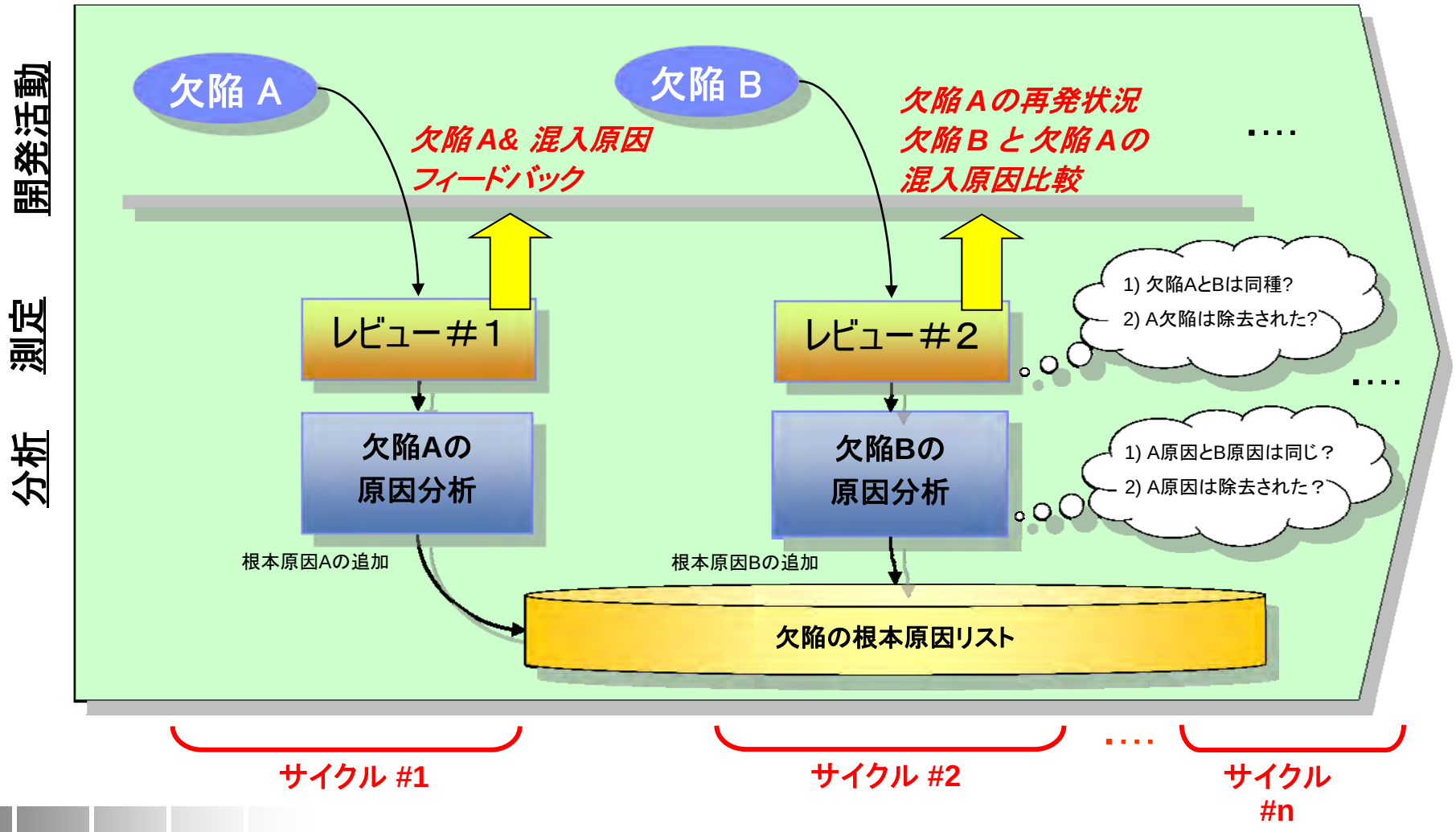
- ヒューマンエラーを原因とする欠陥混入防止
- ソフトウェアプロセス、標準化ガイド、検収プロセスなどを原因とする欠陥の除去を最優先とするく広範囲な修正影響の欠陥から除去／再発予防

下流:エラー原因除去(ESR)
(ESR: Error Source Removal)

- 既検出欠陥の将来の再発予防手法
- 欠陥そのものではなく、人間の行動・ 思考の誤りに焦点を当て原因除去
- ヒューマンエラーに隠された真の 欠陥原因分析や理由分析を行った 後にしか欠陥を除去してはならない

- 欠陥混入機会の減少
- 開発者共通の欠陥を優先して予防
- 今の欠陥検出・除去よりも将来の欠陥混入・再発を予防

# ESR: Error Source Removal



IBM

# プロとしての仕事：
# 発想・創造・集中と没頭

# メトリクス例：どちらのプロジェクトが安定品質か？

・Project A

・Project B

# Pattern example : “CRH” Comment Rate Histogram

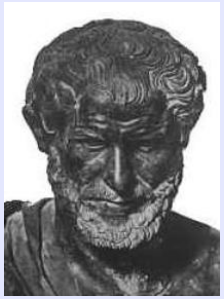
# Comment-Rate Histogram : Example Concludion

# プロの発想力・創造力

- 必殺技主義：ライダー・ウルトラマン現象の否定
- No.1よりOnly One
- ないから作る
- 基礎／ベース／古典／基軸が必要なのは当たり前
- Jazzか、クラシックか？

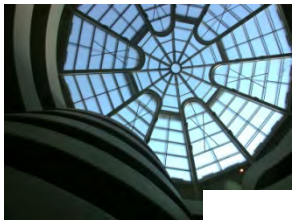
# 参考）アーキテクトの語源

arche
（初，原理，頭）



Αριστοτέλης

teckton
（工匠・職人）



architekton
ギリシャ語

「原理を熟知した工匠」

Architectus
ラテン語

「首位の建築家」

Architect
英語

「建築家」

# 参考）アーキテクトの語源（続き）

**語源**

**「arche」（初，原理，頭）＋「teckton」（工匠・職人）＝ architekton は「原理を熟知した工匠」**

> **「建築家とはギリシア時代の観念として「単なる職人あるいは手技の工人の術ではなく、原理的知識を持ち、職人たちの頭に立ち、諸技術を統べ、製作を企画し指導しうる工匠の技術」をもった人間だとしています。(※2)**

# アーキテクト職 ： 必須の思考・観点

- 国内外の市場・他社・社内といった、あらゆる他者へ影響を与えることをミッションとする
- ソフトウェア・エンジニアリング技術、特に品質を中心とした責務と役割を担う
- 歴史を武器にする・過去経験を武器に、独創性を武器にPM職と共存共栄／相補

## アーキテクト ＝ 技術を武器に影響力を発揮する役割

| | |
|---|---|
| 1.返報性 | 必ず返答を約束し、結果を出すという「コミットメント」 |
| 2.責務と一貫性 | 企業ブランド貢献と一貫した品質最優先文化 |
| 3.社会的承認 | 他者が理解できる・認知できる活動であること |
| 4.好意 | プロフェッショナルとしての高いモラルと低姿勢 |
| 5.権威 | Respectされるに足る知識量と継続的学習 |
| 6.希少性 | 他の追随を許さぬOnly Oneな発想力・創造力 |

# 1）返報性 = 期待に答えること

- 返報性とは、投げかけられた疑問・課題・期待・希望に対して、誠実さとプライドで答えるエンジニア・技術者の「意気込み」のことです。決して損得勘定だけで動いてはいけません

- まず第一に期待されたのなら返答を約束すること。 次に期待されたのなら結果を出すこと、そしてその結果はエンジニアとしてのプライド（誇り）を持って期待に答えなくてはなりません。そして次回もその次もずっと期待される立場になること。高い理想を追い求めなければ高い位置には辿り着けません。

# テスト技術者・品質エンジニア・QA担当者の資質

## 「必須」の資質

1. 配水の陣
2. 笑顔+明るさ
3. 長期的視点とスピード感
4. 絶対的な信頼感（圧倒的情報量＋知識量）

# テスト専門家の育成 ：デバッグに必要な「感性」と「視点」

- 感性：日本人の最も持ち合わせていない感覚
  - 「俯瞰する」能力
  - 構造化して「状況を整理する」能力
  - 背理法，演繹法等の「論理的思考」能力
- 視点：文化背景や教育に依存する不足能力
  - 疑う力
  - 観察力
  - 洞察力
  - 推理力

# 品質専門家の育成

- プロとして＝不変技術としての欠陥除去活動

インスペクションやテスト技法はここ20年大きく変化していない。
開発者・設計者の真反対の，集中力・ハードワークを要する地味な世界。

1）仕様・プログラムを作る技術

2）仕様・プログラムを評価する技術

- 日本の技術者は1）を修得後に2）を修得する
- 欧米，特に軍需系技術者は2）を修得後に1）を修得。

つまり，プログラムを読む技術を先に修得し，「良いプログラム／悪いプログラム」を体得してから開発技術を学ぶ。（例：米国国防総省のADAコード開発者の教育カリキュラム）

10年後にも陳腐化しない息の長い技術として修得すべき技術＝インスペクションやテスト等の評価・検証技術

# 最後に

全ての測定は何のために行われるか？

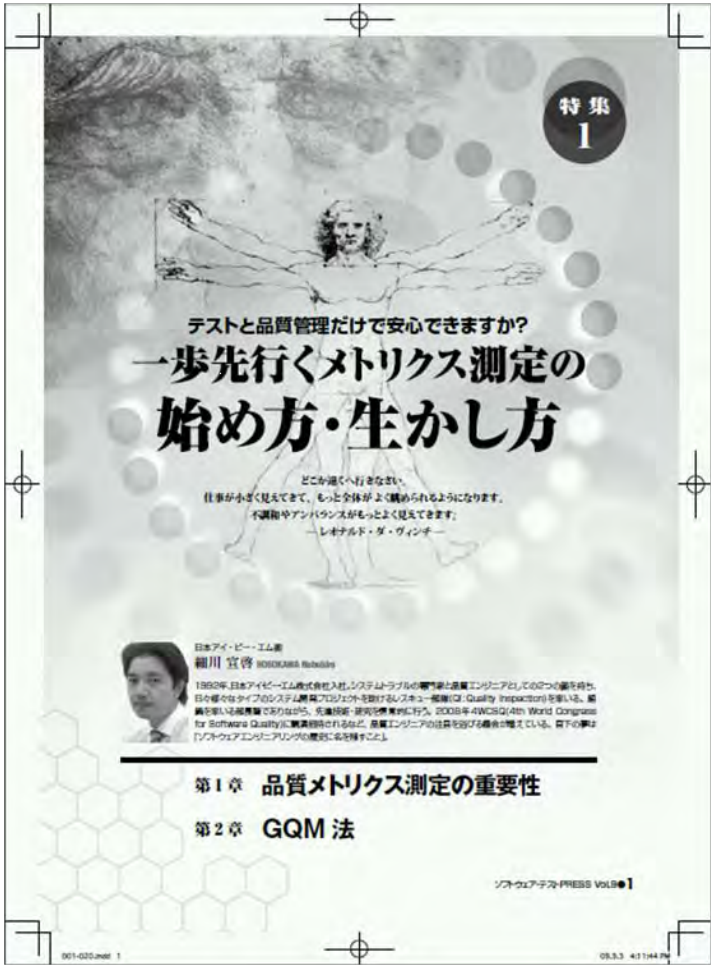
1）安心と自信のため

2）明日、今日より大きな成功をつかむため

3）この国をリードするため

4）プロのエンジニアであり続けるため

# 各種メトリクス測定について: オススメの技法「GQM法」




Software Test Press Vol.9号掲載予定