SPI Japan 2010

Empowered by Innovation NEC

# 現場主義が道を拓く
## ～ソフトウェア品質会計の経験から～

2010年　11月　11日

日本電気株式会社

誉田　直美

# 人と地球にやさしい情報社会を
# イノベーションで実現する
# グローバルリーディングカンパニー

**NECグループビジョン2017**

# 書籍のご紹介 「ソフトウェア品質会計」

## ソフトウェア品質会計
### ～ＮＥＣの高品質ソフトウェア開発を支える品質保証技術～

誉田　直美著

2010年6月30日発行

日科技連出版社

定価 3150円 (税込)

ISBN978-4-8171-9348-3


- 品質会計は、ＮＥＣ独自のソフトウェア品質管理手法であり、ＮＥＣグループで広く適用されています
- ケーススタディを含め、その使い方を具体的に解説しました
- 現場･現物･現実を重視している点が、大きな特徴です

**日経品質管理文献賞受賞！**

# 事業領域と主な商品･サービス

ITサービス事業

ITプロダクト事業

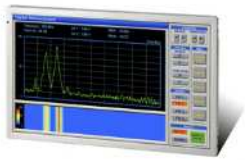

パーソナルソリューション事業




ネットワークシステム事業


社会インフラ事業

エレクトロンデバイス事業



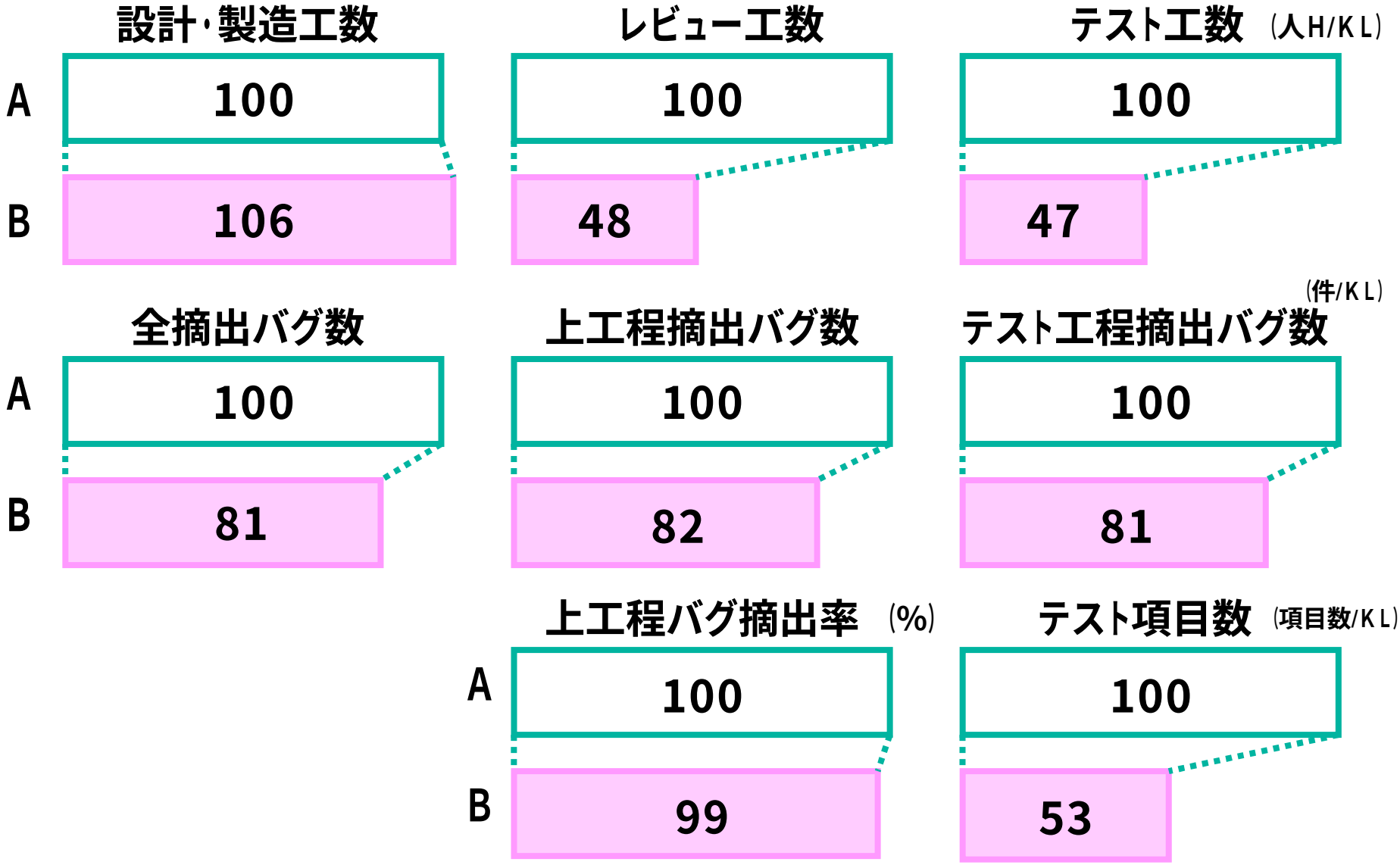
# 本日のテーマ

# 2つの開発組織の比較から

## <組織Aと　組織B >

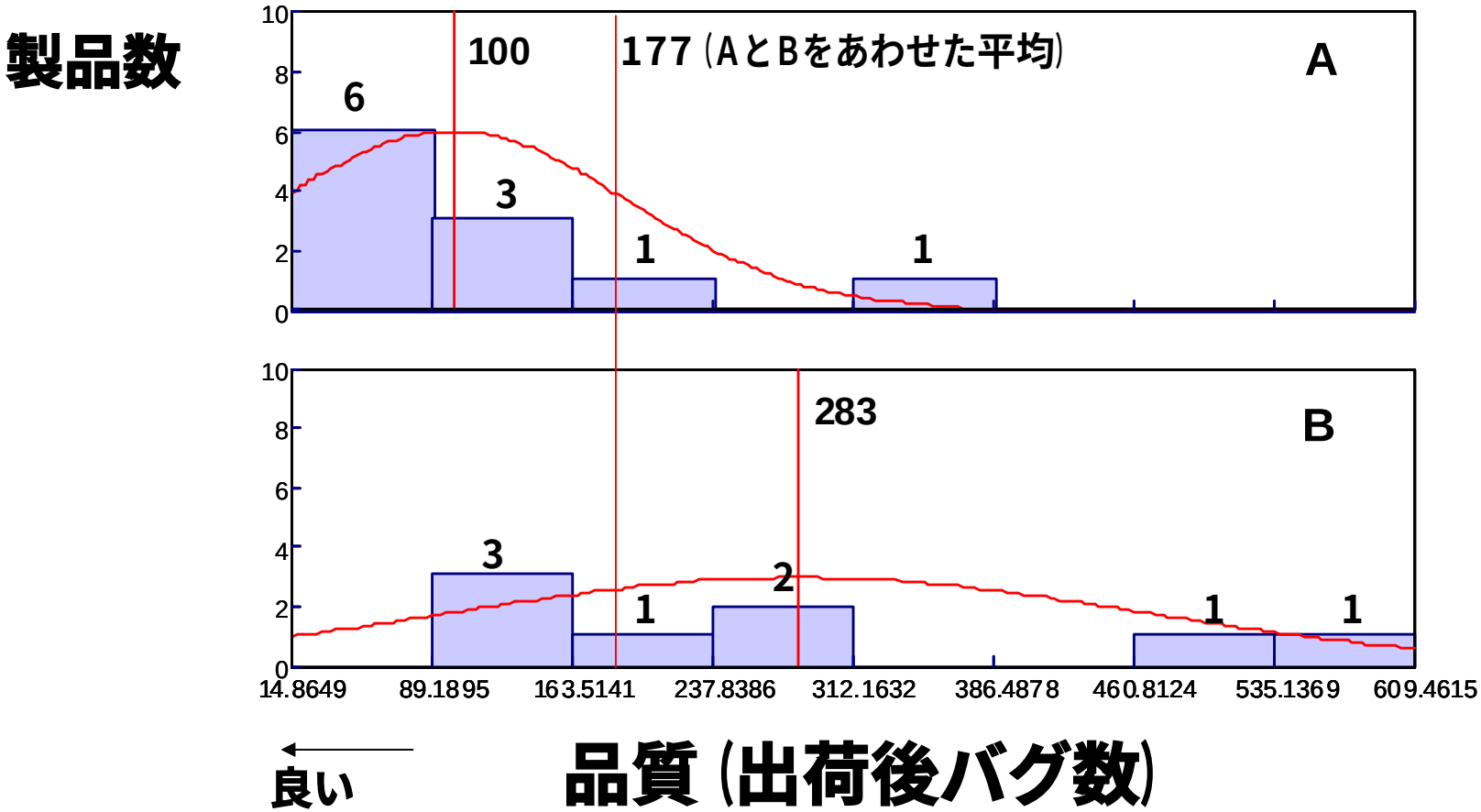
- 開発条件がよく似ている
  - ビジネス領域, ビジネス規模
  - 開発規模, 開発者数
  - ソフトウェアプロセス, 開発技術・管理技術
  - 「品質会計」を適用
  - CMMI　レベル5達成済み
- しかし、成果に大きな差異

# プロセスデータの比較



# 出荷後品質の比較

**組織Bの品質は、組織Aより 2倍以上悪い**



同じ技法･プロセスを適用しても、
同じ成果はでない

なぜか？

# 目次

1. 品質会計とは

2. 品質会計を特徴付ける技法と
   その成果事例

3. 現場･現物･現実を重視するとは

# 1. 品質会計とは

# ソフトウェアを取り巻く状況

## ソフトウェアの重要性が増大

- 現代の経済･社会インフラ
- 製品･サービスの機能，性能，特徴、魅力を決定づける

## ソフトウェアのGDP貢献度

- 日本のGDP：約500兆円，ソフトウェア業の年間売上高：約15兆円
- 実質的GDP貢献度：　極めて大きい
  - 産業，社会，生活のあらゆる場面でソフトウェアが使われている

⇒ソフトウェアを制するものは世界を制する

## 出荷後バグの少ないソフトウェア開発の実現は、ベンダの使命

# なぜ、品質会計は生まれたか

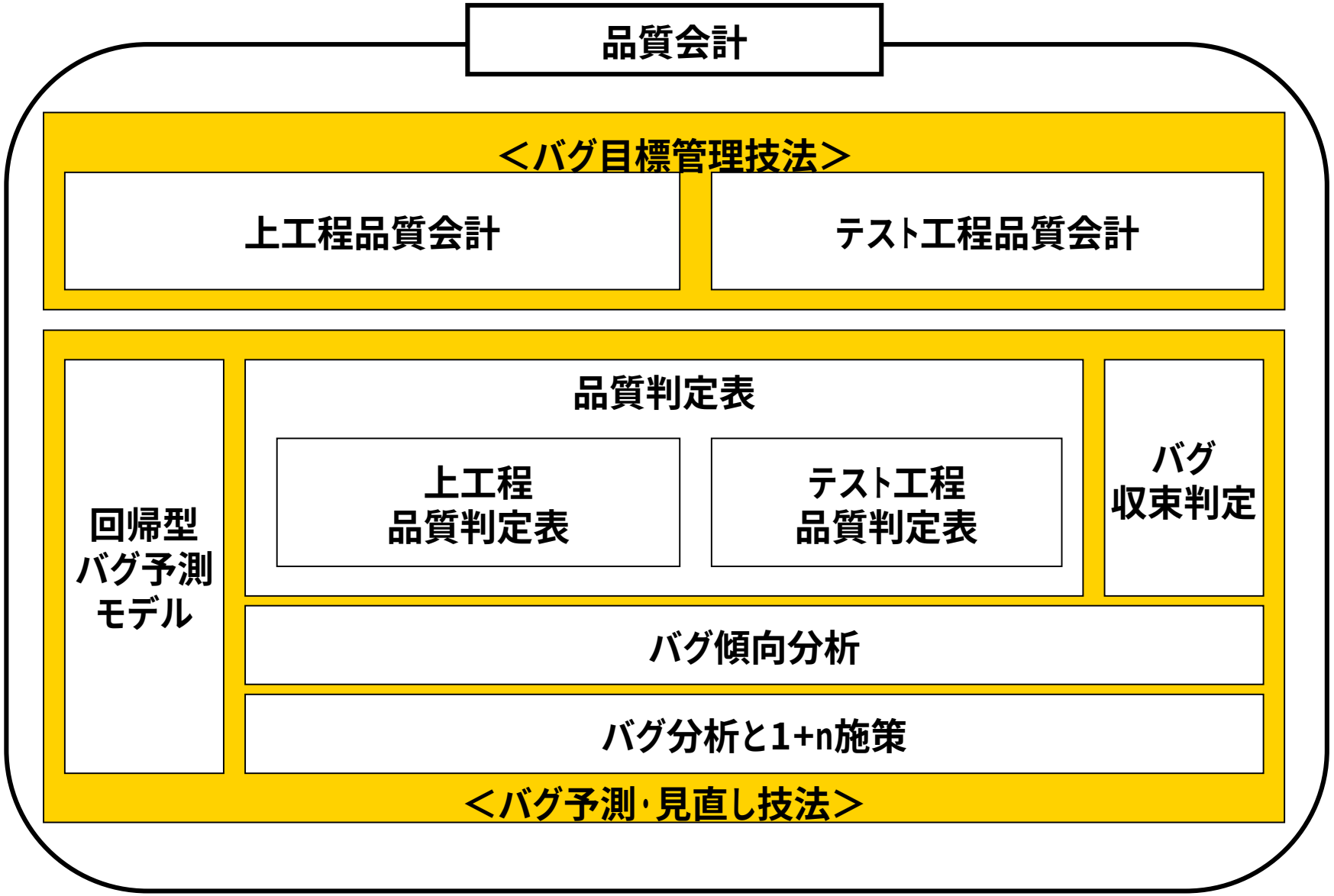
- 果て知れぬフィールドバグとの戦い…
- 負のスパイラルのままでは事業継続できない（徹夜続きで体が持たない）との思いから、現場が考案し発展

ダンプとトラブルの山に埋もれ……

# 品質会計: Quality Accounting System とは

- 「品質」が作り込まれたことを、確かな根拠をもって　説明するソフトウェア品質管理手法
  - “account”とは、もともと　「理由・根拠を説明する」という意味を持つ
  - 品質会計を特徴づける技法
    - 上工程品質会計
    - テスト終盤の残存課題の分析
      - バグ傾向分析、バグ分析と1+n施策、バグ収束判定の3つの組み合わせ
- 1982年頃、ＮＥＣの開発現場で考案された、ＮＥＣ独自の手法
- 「作り込んだバグを負債と見なし、バグ摘出によりこの負債を返済し、負債がゼロになった時点で出荷する」ことを基本とする

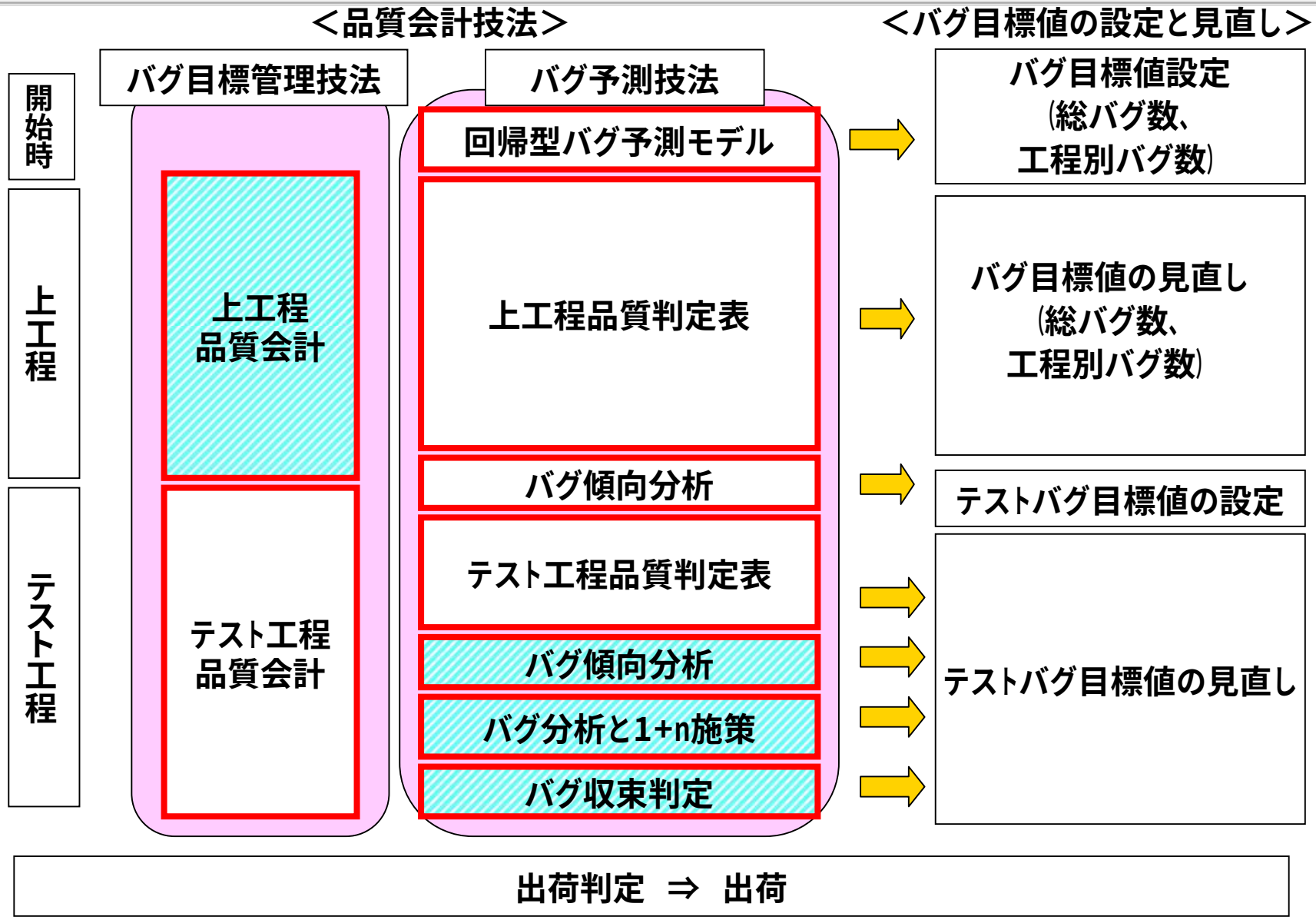
# 品質会計の技術体系



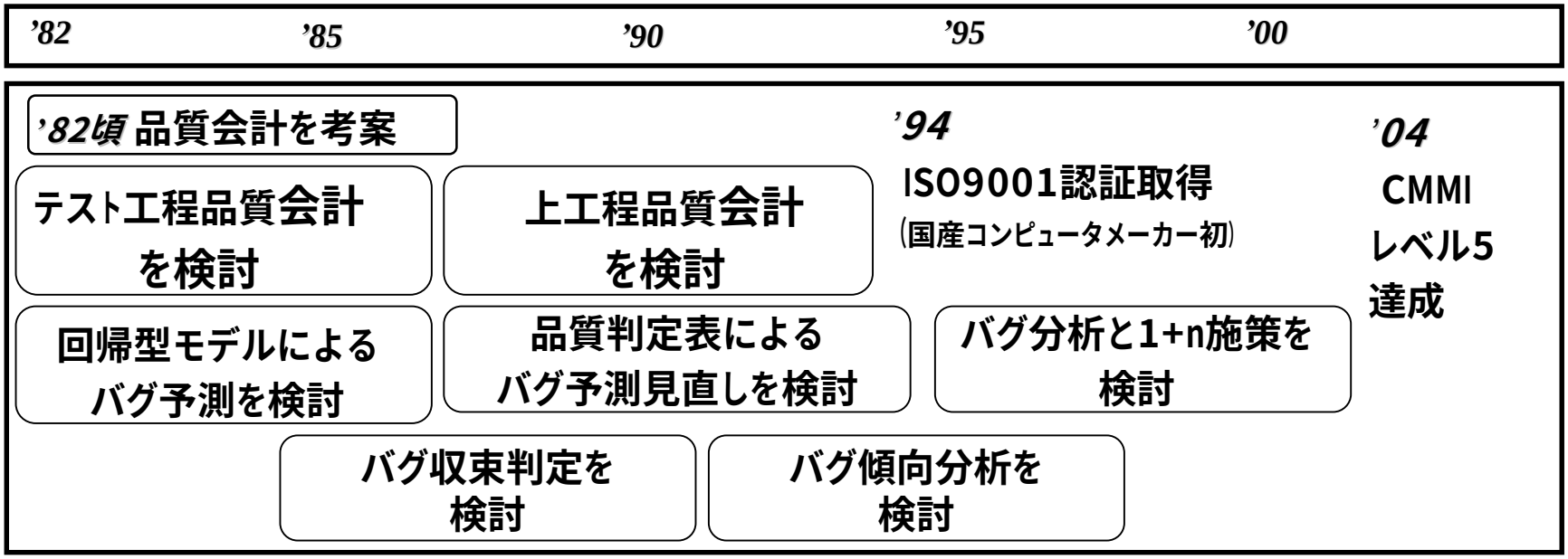
# 品質会計を構成する技法とその特徴

| カテゴリ | 技法 | 使用法と特徴 |
|---|---|---|
| バグ目標管理技法 | 上工程品質会計 | 上工程（設計～製造）用のバグ目標管理技法。バグの摘出工程と作りこみ工程の両面からバグを目標管理する。 |
| | テスト工程品質会計 | テスト工程用のバグ目標管理技法。テスト開始時に残存する、プログラム全体の総バグ数を目標管理する。 |
| バグ予測・見直し技法 | 回帰型バグ予測モデル | 開発開始時に、今回の開発で作り込むであろう総バグ数を予測するためのバグ予測技法。 |
| | 品質判定表 | 開発途中に発生する変化を考慮して、バグ目標値を見直すバグ目標値見直し技法。上工程品質判定表とテスト工程品質判定表がある。 |
| | バグ傾向分析 | 摘出したバグをさまざまな観点から整理することにより、バグの摘出傾向に偏りがないかを分析する技法。 |
| | バグ分析と1+n施策 | バグ1件ごとに真因を分析することにより、開発上の細かい抜け・漏れを発見し、その抜け・漏れに対して、集中的なレビューやテストにより残存するバグを摘出する技法。<br>バグ分析と1+n施策はセットで用いる。 |
| | バグ収束判定 | テスト度合いに対する累積摘出バグ数の推移により、バグ収束を判定する技法。 |

# 開発プロセスと品質会計技法の適用の関係



# 品質会計発展の歴史




[品質改善の視点]

＜第1期＞
テスト中心のバグ叩き出しによる品質向上

＜第2期＞
上工程でのすばやいバグ摘出による品質向上

＜第3期＞
バグ分析と1+n施策にもとづく同種バグ摘出と開発プロセスへのフィードバック

[開発プロセス構築の視点]

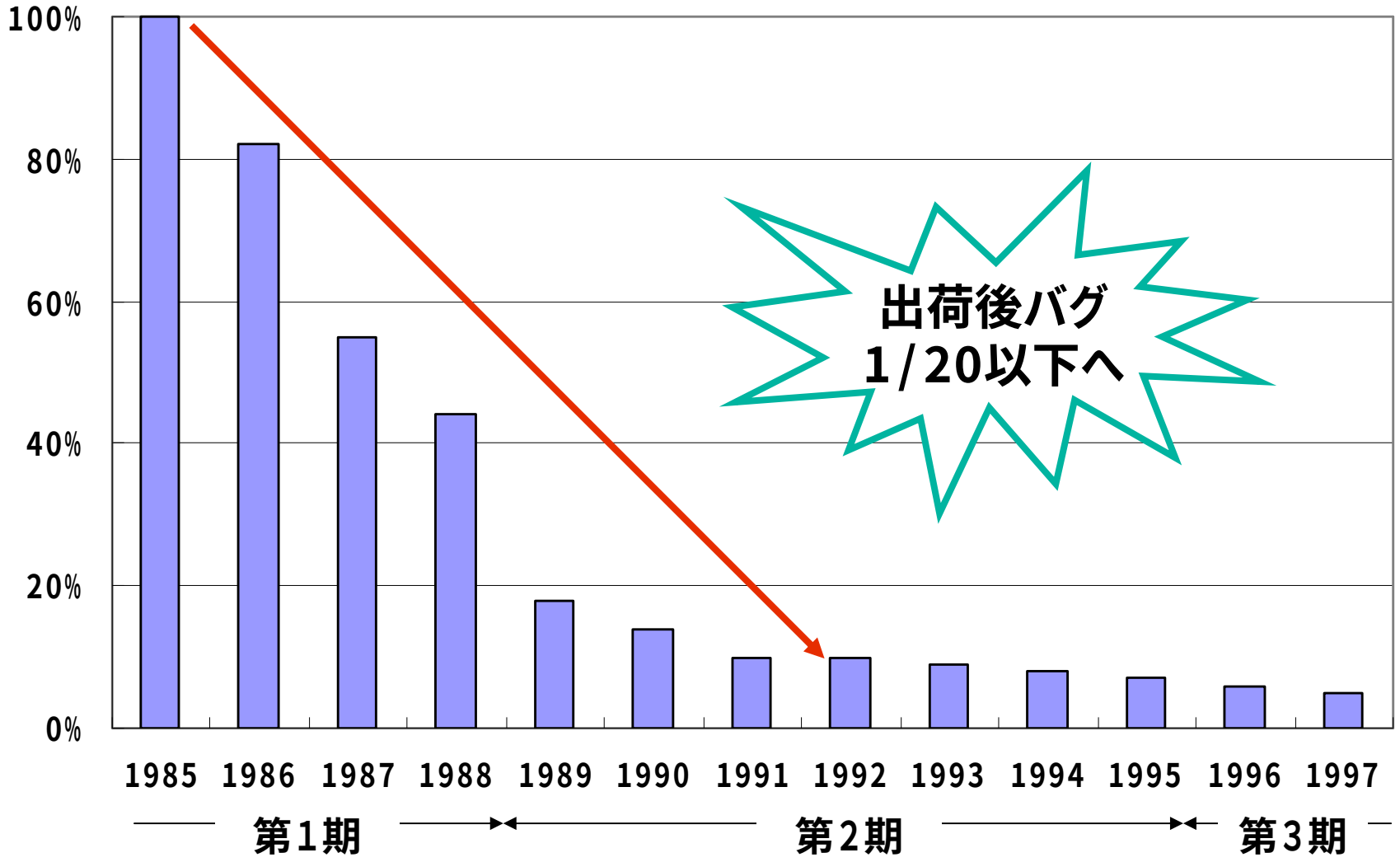
各グループでの手順改善

組織としての仕組み構築･改善
(管理技術 / 開発技術 / 開発環境の標準化)

# 品質会計発案組織のソフトウェア開発の特徴

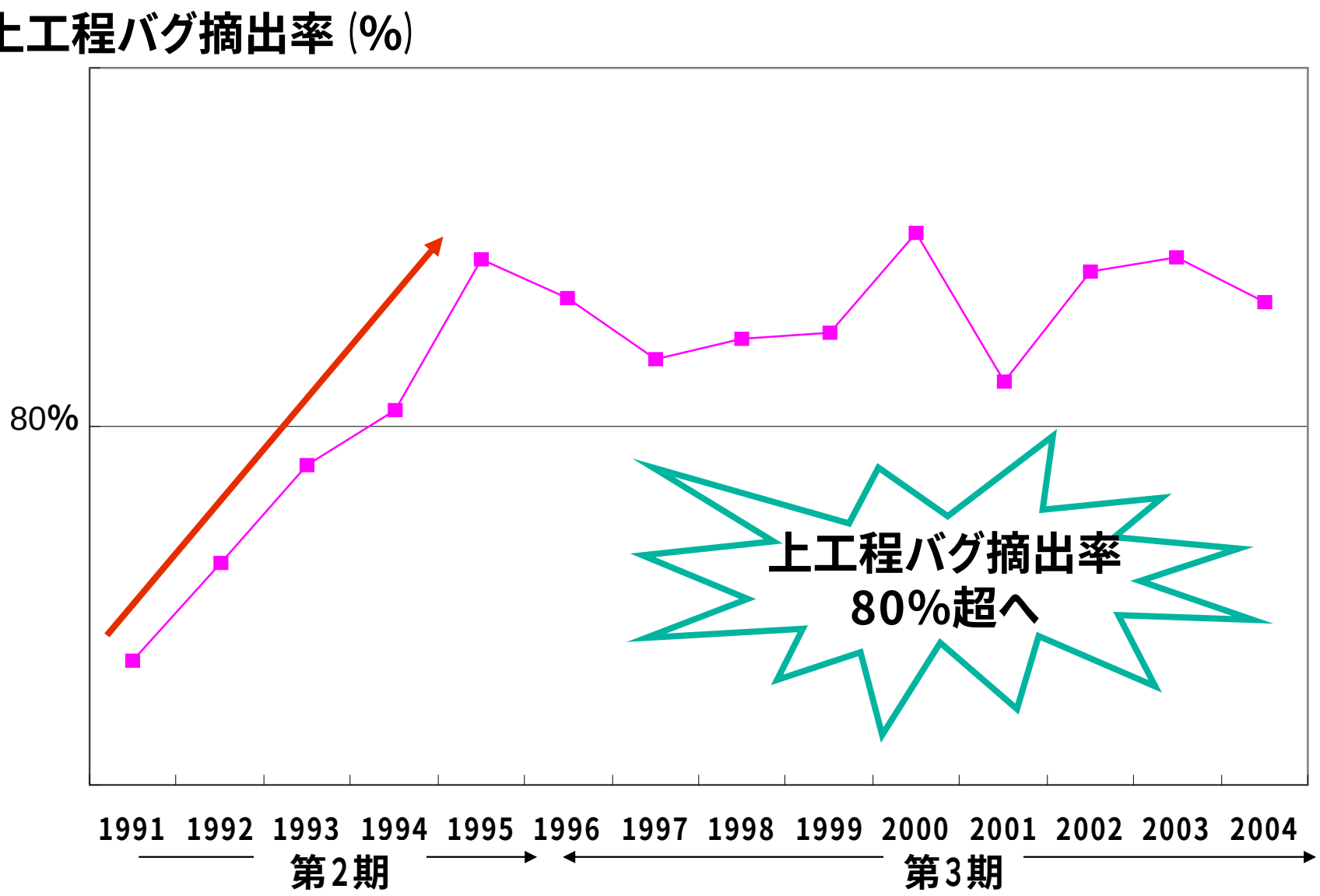
- IT系の汎用ソフトウェア製品 (基本ソフト, ミドルソフトウェア)
  - 特定顧客向けではない
- 要求定義は、その組織自身が市場分析にもとづいて作成
- 汎用品のため、様々な使用方法の考慮が必須
  - 特に高信頼性を実現する製品では、広範囲の動作条件の検証が必要
- 保守段階の未然防止活動が重要
  - 同じトラブルを他のお客様で発生させない

# 品質会計の効果：出荷後バグの削減推移

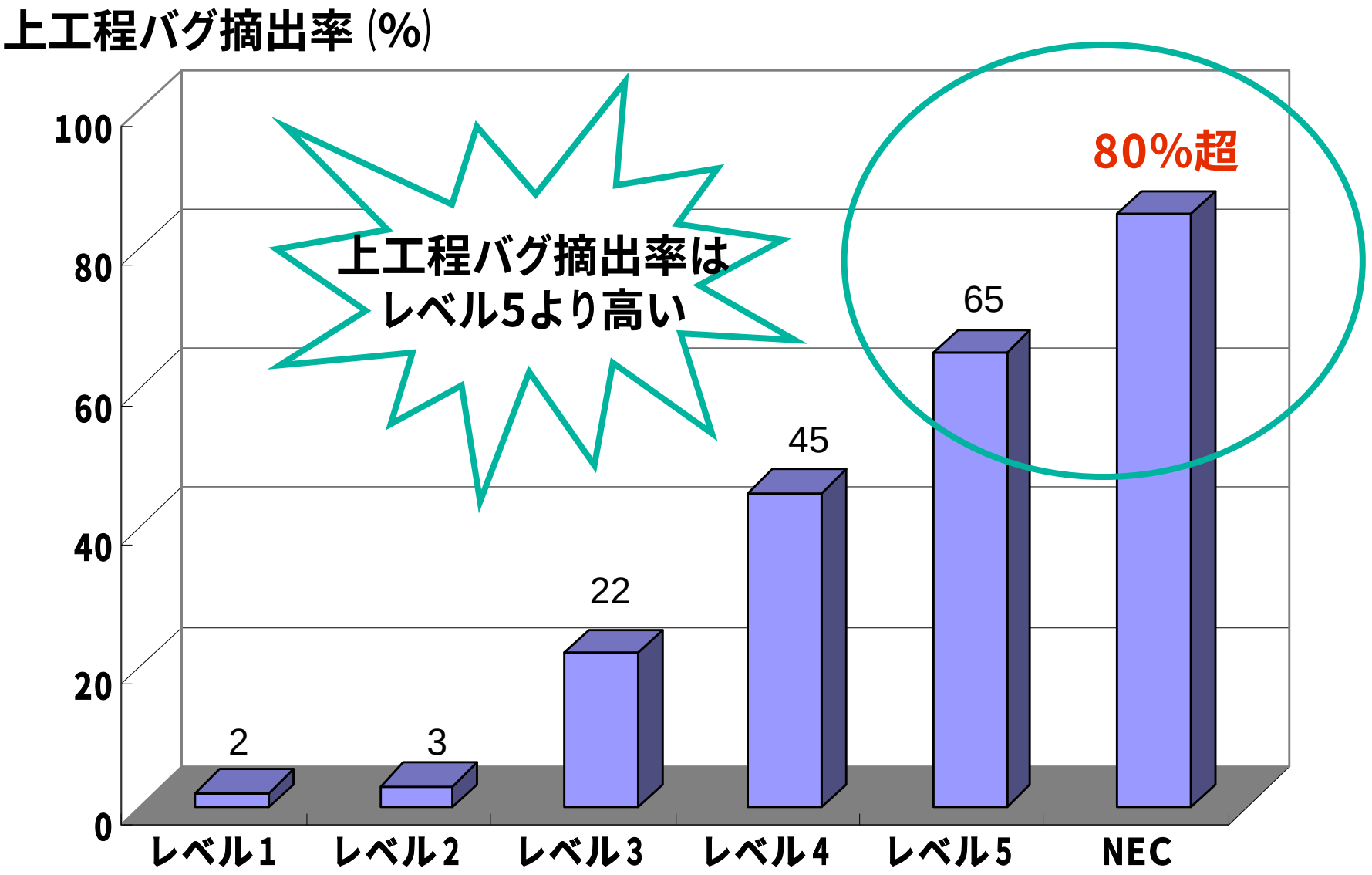



※1985年の出荷後バグ件数を100%とした相対比を表示

# 品質会計の効果：上工程バグ摘出率の推移



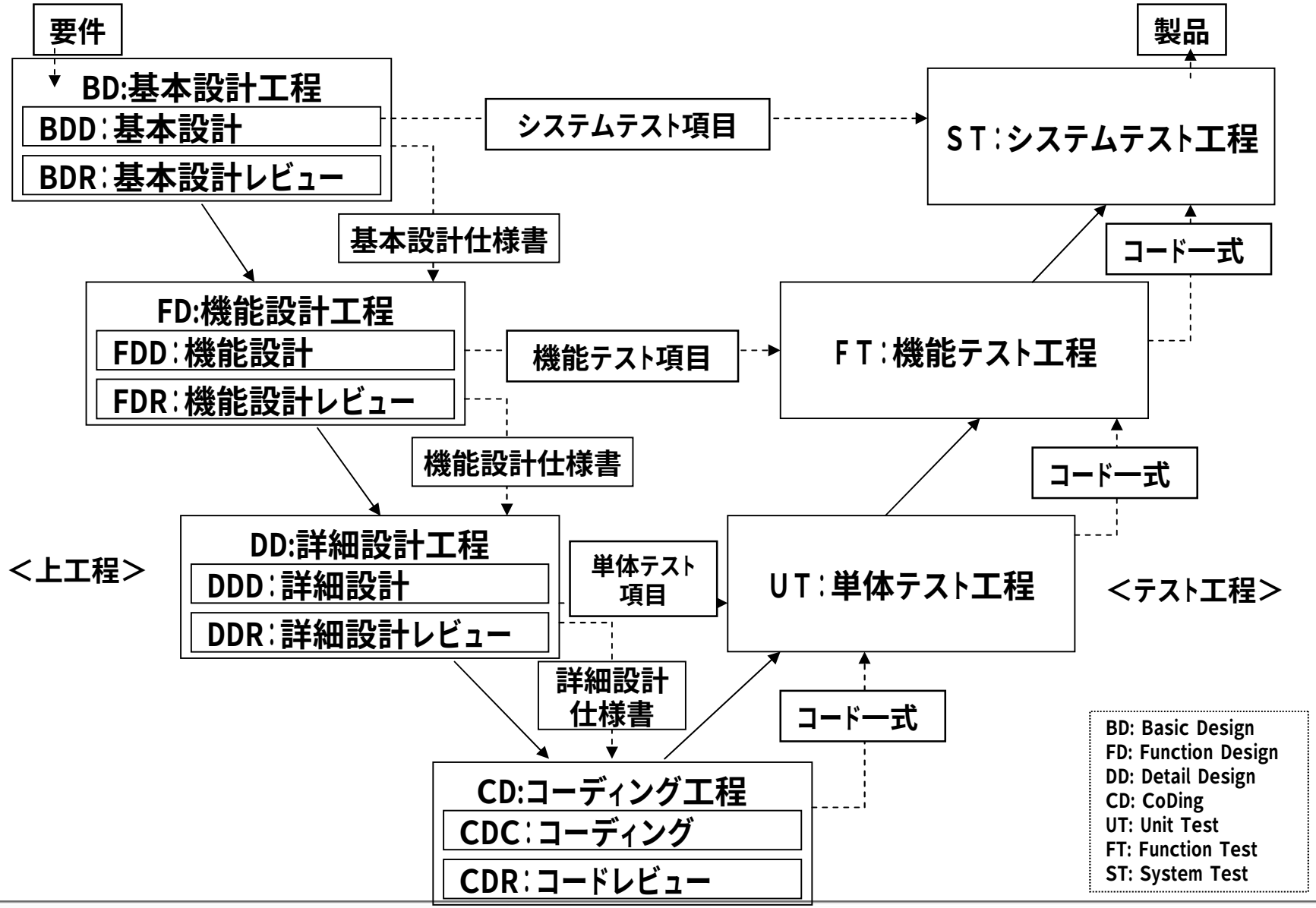
# 品質会計の効果：上工程バグ摘出率とCMMIレベル




*レベル1～5のデータ出典：日経コンピュータ(2001.7.30.号)

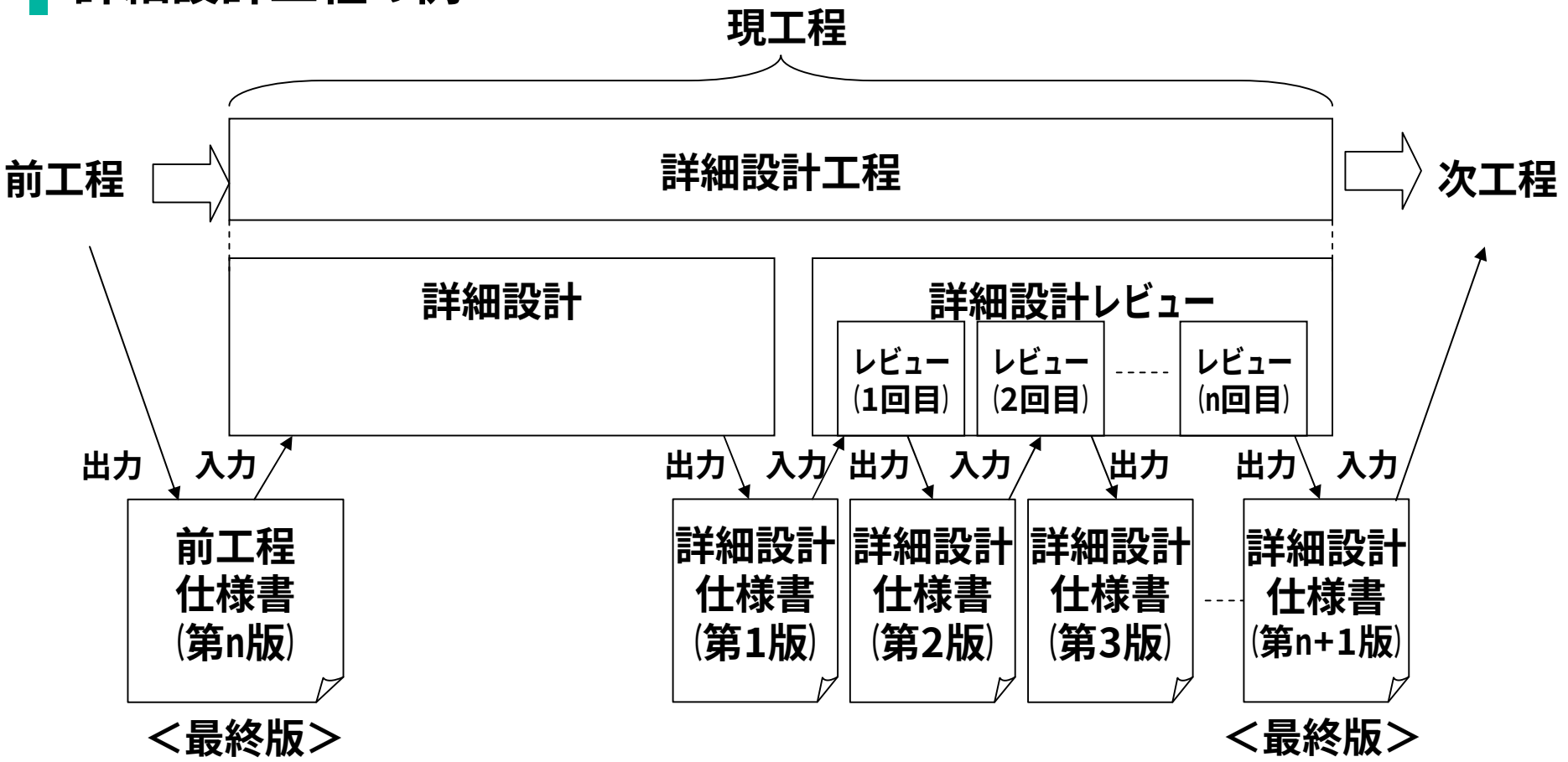
# 2. 品質会計を特徴付ける技法とその成果事例

# 開発プロセス



# 「設計」と「レビュー」の関係

## 詳細設計工程の例



# 品質会計の原則

＜品質会計の原則＞

バグは作り込まない。作りこんだバグはすばやく摘出する。

＜上工程品質会計の原則＞

作り込んだバグは次工程までに摘出する

- 作り込み工程で80%摘出
- 次工程で残り20%摘出

＜テスト工程品質会計の原則＞

作り込んだバグは、すべて摘出してから出荷する

＜目標＞

上工程バグ摘出率　80%

# 上工程品質会計

# 上工程品質会計　とは

## 特徴

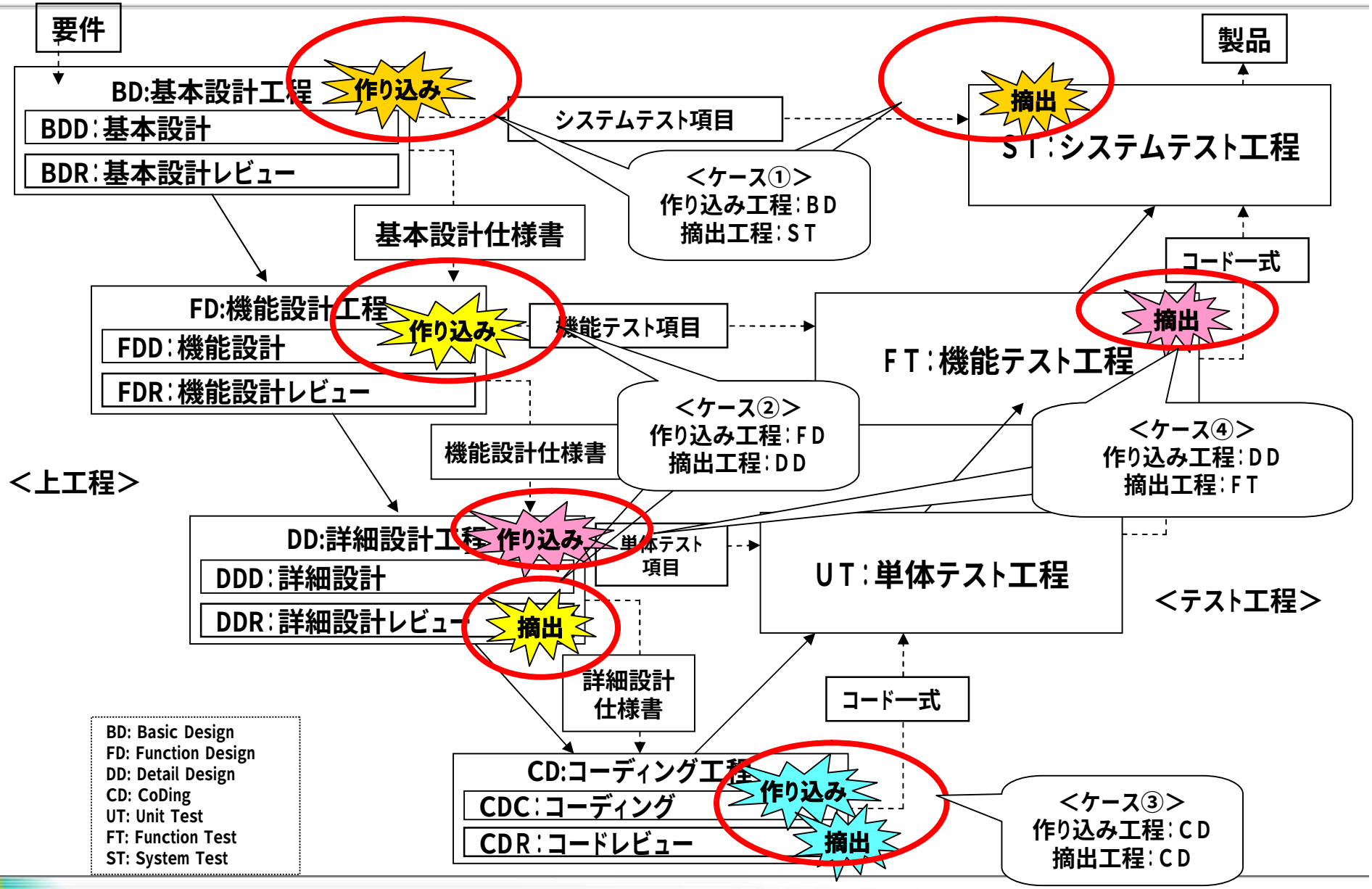
- 上工程（設計～コーディング）用のバグ目標管理技法
- バグの摘出工程と作り込み工程の両面からバグを目標管理
- レビューでのバグ摘出を推進する技法

## 目標管理の視点

- バグ摘出工程別の管理
  - 開発の経緯につれて、どのようにバグを摘出しているか
- バグ作り込み工程別の管理
  - どの開発工程の作業内容に問題があるか

## 上工程品質会計は、品質向上の大きな原動力

# バグの作り込み工程と摘出工程



# バグ目標値の初期設定：回帰型バグ予測モデル

## 特徴

- 開発開始時に今回の開発で作りこむと思われる総バグ数を予測するためのバグ予測技法

## 注意事項

- 予測のために事前に準備すべきもの
  - 過去の開発データを統計解析して作成した回帰型バグ予測モデル
  - 過去の開発データにより作成した摘出工程別バグ目標比率

⇒過去の開発データがない場合は、他組織や書籍の事例などから、暫定的に算出すればよい

- 回帰型バグ予測モデルの特徴と限界
  - 過去に経験した開発通りに開発するものと仮定した予測である
  - 過去に経験したことがない特性を持った開発の場合は、予測できない
  - モデルは常に最新データを使って見直す
    - 最新の開発状況をモデルに反映し、予測精度を向上

# 回帰型バグ予測モデル

$$B = C \cdot \alpha_1 \cdot \alpha_2 \cdot \cdots \cdot \alpha_n \cdot S$$

- B：バグ予測値
- C：標準バグ件数（定数）
- α：バグへの影響要因
- n：バグへの影響要因の数
- S：開発規模(KLOC)

**モデルの意味**

・開発規模に比例してバグ数は増え、影響要因により変動する

※モデルの意味が成立しない場合は、モデルの形を見直す

**影響要因の考え方**

・バグへ影響する要因を選択

・環境変化に応じて影響要因は変化する（常に同じではない）

## バグへの影響要因（最近、主に使用しているもの）

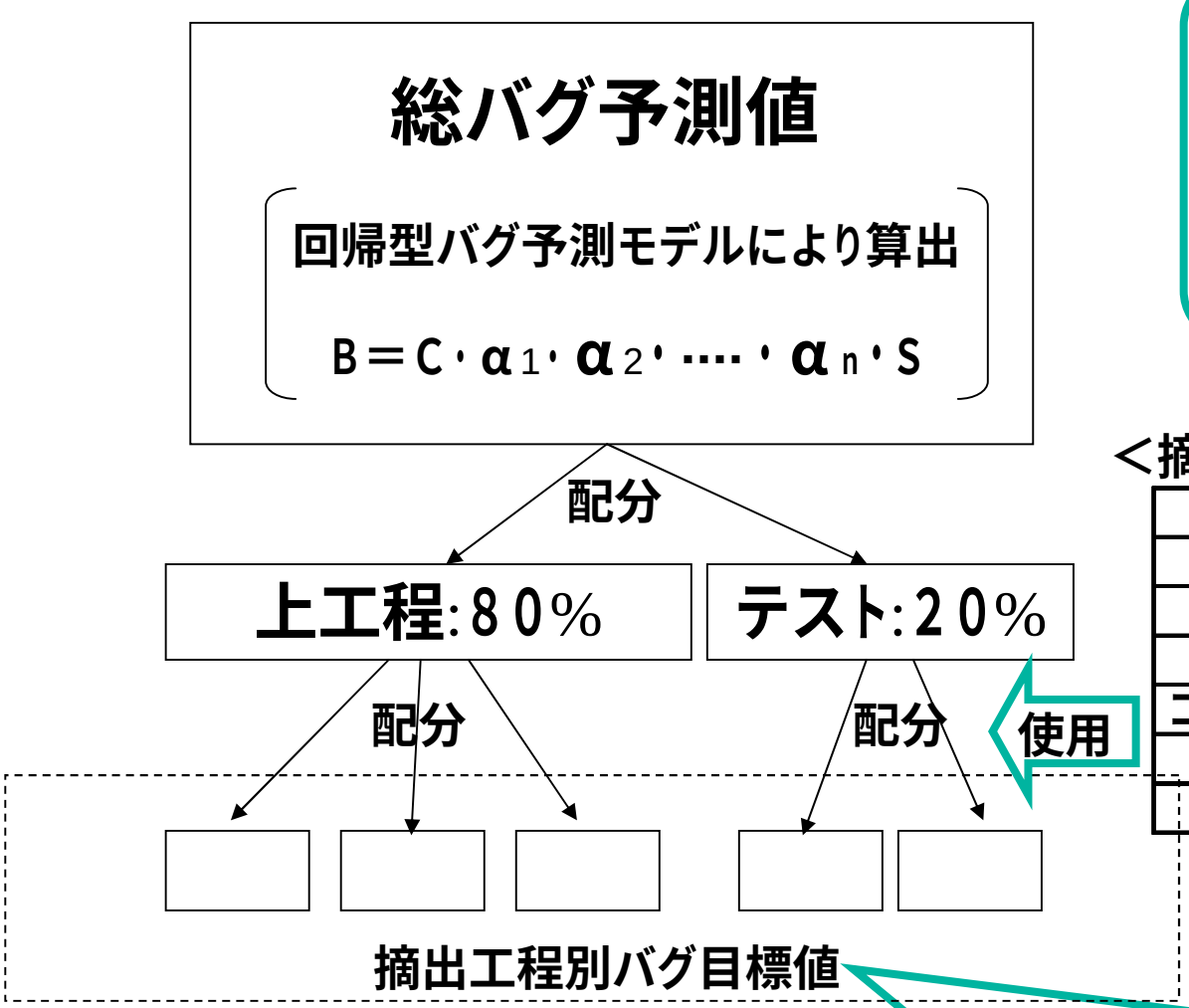
### 開発グループの「技術力」

| 段階 | 技術力（α1） | 数値 |
|---|---|---|
| 5 | 高い | |
| 4 | やや高い | |
| 3 | 平均的 | |
| 2 | やや低い | |
| 1 | 低い | |

### 開発対象ソフトウェアの「難易度」

| 段階 | 難易度（α2） | 数値 |
|---|---|---|
| 5 | 難しい | |
| 4 | やや難しい | |
| 3 | 平均的 | |
| 2 | やや易しい | |
| 1 | 易しい | |

# 工程別バグ目標値の設定方法




ポイント！

・テスト工程のバグ数は増加することが多いので、それを加味した意志の値とする

・上工程バグ摘出率80%が達成できるようにする

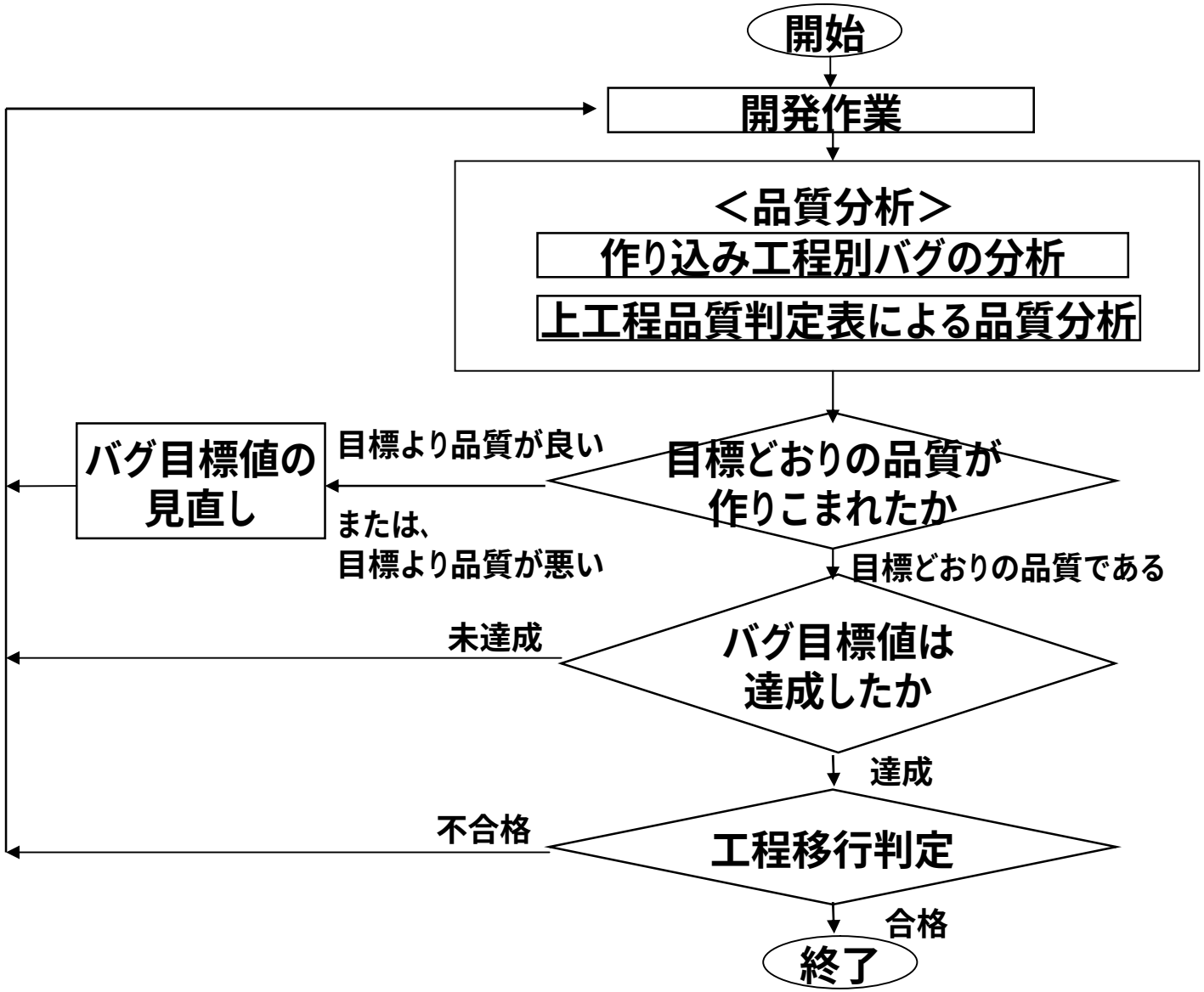
＜摘出工程別バグ目標比率（例）＞

| 摘出工程 | 目標比率（%） |
|---|---|
| 基本設計工程 | 4 |
| 機能設計工程 | 18 |
| 詳細設計工程 | 20 |
| コーディング工程 | 40 |
| テスト工程 | 18 |
| 計 | 100 |

算出結果に基づき、今回の開発特性を考慮してバグ目標値を決定

※必ずしも、算出結果をそのまま受け入れる必要はない

# 上工程品質会計の流れ



# 作り込み工程別バグ分析の分析観点

1. レビュー回数を重ねるにつれて、摘出されるバグ数は減少しているか？
2. 前工程の作り込みバグが多く摘出されていないか？
3. 前工程より上流で作りこまれたバグが摘出されていないか？

～　品質会計の原則　～
作り込んだバグは次工程までに摘出する
（作り込み工程で80%、次工程で残り20%を摘出）

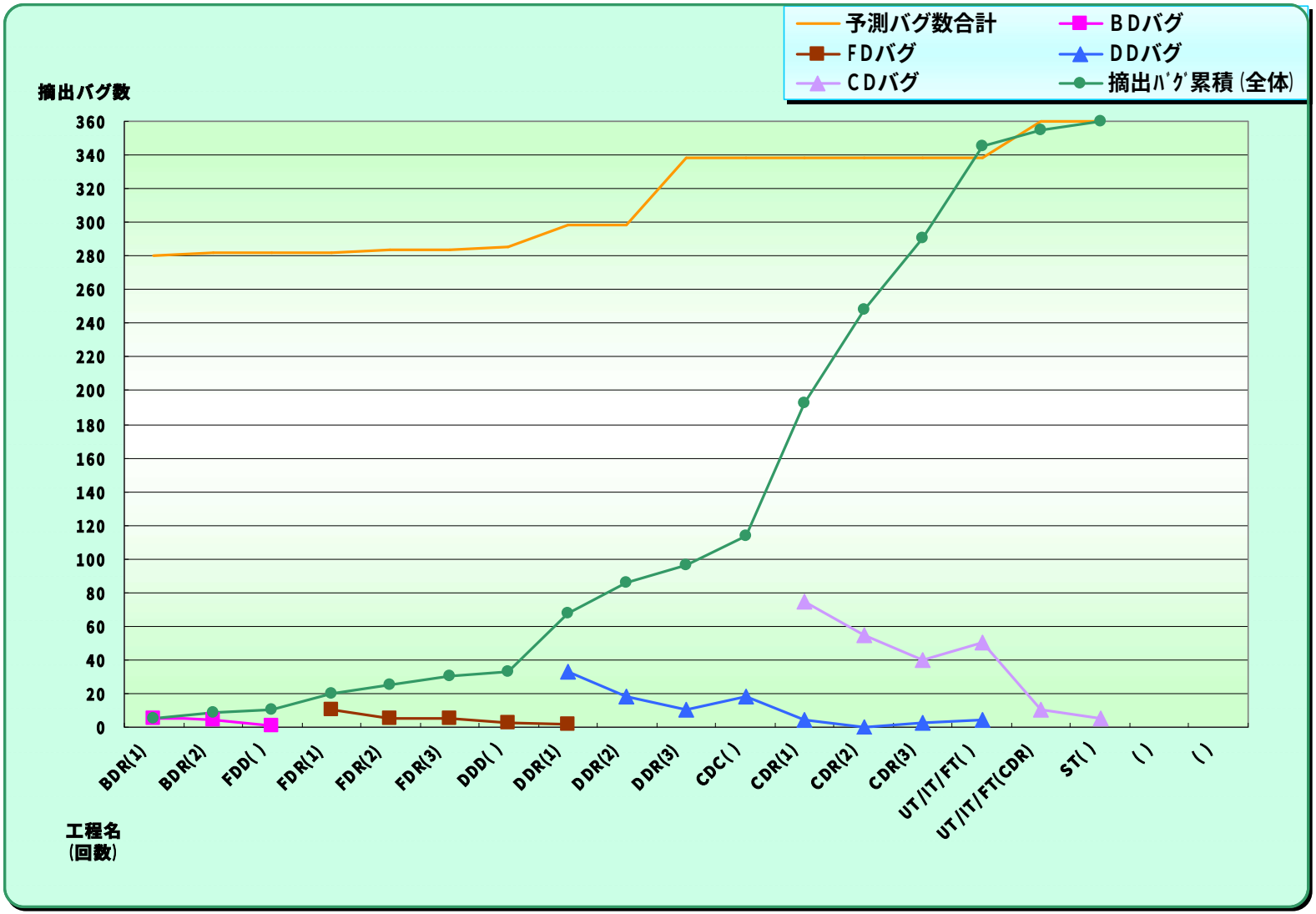
# 作り込み工程別バグ分析のためのデータ表 (例)

## 現工程：詳細設計工程

| | 詳細設計 | レビュー(1回目) | レビュー(2回目) | …. | レビュー(n回目) | 累計 |
|---|---|---|---|---|---|---|
| 基本設計バグ | 0 | 0 | 2 | | 0 | 2 |
| 機能設計バグ | 8 | 2 | 5 | | 0 | 18 |
| 詳細設計バグ | - | 13 | 16 | | 5 | 38 |
| 合計 | 8 | 15 | 23 | | 5 | 58 |
| 累計 | 8 | 23 | 46 | | 58 | - |

1. レビュー回数を重ねるにつれて、摘出されるバグ数は減少しているか？
2. 前工程の作り込みバグが多く摘出されていないか？
3. 前工程より上流で作りこまれたバグが摘出されていないか？

# 参考：上工程品質会計グラフ（例）



# 上工程のバグの定義例

## ■上工程のバグとは

- 設計工程では設計書ドラフト版完了時から、ＣＤ工程ではコンパイル完了時から修正を要した点はすべて当該工程のバグ

## ■上工程のバグの定義

・設計書のバグ

- 標準に沿っていないものはバグとする。
- 前工程の仕様書に沿っていないものはバグとする。
- 記述がわかりにくいため、ほかの担当者が誤解する可能性の高いものはバグとする。誤解する可能性の低いものはバグとしない。
- 他グループへの確認不足によるものはバグとする。
- 記述されていない部分はバグとする。ただし、誤字脱字（てにをは）などの記述ミスはバグとしない。
- その仕様書の入力となった前工程の仕様書に問題があったためによるバグは、前工程のバグとする。
- 他グループの仕様書のバグによるバグは、他グループのバグとする。

・コーディングのバグ

- コーディング規則に沿っていないものはバグとする。
- コーディングミスはバグとする。

・重複バグは、全体で１件とする。

# 品質判定表　＜上工程品質判定表＞

| 上工程品質判定表 | | レビュー工数/KLOC | | |
|---|---|---|---|---|
| | | 実績＜計画 - n％ | 計画-n％≦実績 ≦計画+n％ | 計画+n％＜実績 |
| レビューによる摘出バグ数/KLOC | 実績＜計画-n％ | 品質を判断する時期ではない<br>⇒レビュー継続 | 品質は計画よりも良い<br>⇒①式で見直し | 品質は計画よりも良い<br>⇒②式で見直し |
| | 計画-n％≦実績≦計画+n％ | 計画より品質が悪い<br>⇒①式で見直し | 品質は計画通り | 品質は計画通り |
| | 計画+n％＜実績 | 計画より品質が悪い<br>⇒①式で見直し | 計画より品質が悪い<br>⇒①式で見直し | 計画より品質が悪い<br>⇒②式で見直し |

＜バグ目標値見直し式＞

①式：新総バグ目標値＝旧総バグ目標値×
$$\frac{\dfrac{\text{レビューによる摘出バグ数実績値}}{\text{レビュー工数実績値}}}{\dfrac{\text{レビューによる摘出バグ数目標値}}{\text{レビュー工数目標値}}}$$

②式：新総バグ目標値＝旧総バグ目標値×
$$\frac{\text{レビューによる摘出バグ数実績値/KLOC}}{\text{レビューによる摘出バグ数目標値/KLOC}}$$

★品質判定表の考え方が重要。メトリクス等は組織にあわせて変えてよい★

# 品質判定表の考え方

## 開発途中に発生する変化を考慮して、バグ目標値を見直すバグ目標値見直し技法

## 上工程における（計画時には考慮していなかった）品質上の変化を、いかに把握するか

- レビュー状況が最も分析しやすい＜ITソフトウェア事本のケース＞

⇒レビュー度合い（レビュー工数）に対する摘出バグ数というメトリクスを使用

★レビュー技術に変化がないと仮定したとき、レビューにかけた工数に対する摘出バグ数によって、品質の程度を判断

- レビュー工数に対して摘出バグ数が多い

⇒品質が悪い（作りこまれた品質は、計画より悪かった）

- レビュー工数に対して摘出バグ数が少ない

⇒品質が良い（作りこまれた品質は、計画より良かった）

- レビュー工数に対して摘出バグ数は計画通り

⇒品質は計画通り（作りこまれた品質は、計画時と同等）

※テスト工程も同様の考え方に基づく

# ケーススタディ：機能設計完了時の品質分析

## 機能設計工程のバグ目標値を大幅過達⇒見直し

＜Zグループの機能設計工程におけるレビューの推移＞

| 実績値の推移 | | FDD | FDR(1) | FDR(2) | FDR(3) | 累計 |
|---|---|---|---|---|---|---|
| 摘出バグ(件) | 基本設計バグ | 0 | 1 | 0 | 0 | 1 |
| | 機能設計バグ | - | 21 | 25 | 5 | 51 |
| | 合計 | 0 | 22 | 25 | 5 | 52 |
| | 累計 | 0 | 22 | 47 | 52 | - |

＜Zグループの機能設計工程における開発データ＞

| | 目標 | 実績 | KLOC当たり | | KLOC当たり目標値の±10% | |
|---|---|---|---|---|---|---|
| | | | 目標 | 実績 | -10% | 10% |
| 摘出バグ数 | 46 | 52 | 1.70 | 1.93 | 1.53 | 1.87 |
| レビュー工数 | 103 | 104 | 3.81 | 3.85 | 3.43 | 4.20 |

使用

＜Zグループの機能設計工程における上工程品質判定表＞

| 上工程品質判定表 | | レビュー工数/KLOC | | |
|---|---|---|---|---|
| | | 実績＜目標-10%<br>(実績＜3.43) | 目標-10%≦実績≦目標+10%<br>(3.43≦実績≦4.20) | 目標+10%＜実績<br>(4.20<実績) |
| 摘出バグ数/KLOC | 実績＜目標-10%<br>(実績＜1.53) | 品質を判断する時期ではない<br>⇒レビュー継続 | 品質は目標よりも良い<br>⇒①式で見直し | 品質は目標よりも良い<br>⇒②式で見直し |
| | 目標-10%≦実績≦目標+10%<br>(1.53≦実績≦1.87) | 目標より品質が悪い<br>⇒①式で見直し | 品質は目標どおり | 品質は目標どおり |
| | 目標+10%＜実績<br>(1.87＜実績) | 目標より品質が悪い<br>⇒①式で見直し | 目標より品質が悪い<br>⇒①式で見直し | 目標より品質が悪い<br>⇒②式で見直し |

# ケーススタディ：機能設計完了時のバグ目標値見直し結果

## 上工程品質判定表による新バグ目標値の算出

$$\text{①式：新総バグ目標値} = \text{旧総バグ目標値} \times \frac{\dfrac{\text{レビューによる摘出バグ数実績値}}{\text{レビュー工数実績値}}}{\dfrac{\text{レビューによる摘出バグ数目標値}}{\text{レビュー工数目標値}}}$$

$$= 259 \times \frac{\dfrac{52}{104}}{\dfrac{46}{103}}$$

$$290$$

## 見直し後バグ目標値

| 摘出工程 | 見直し前 | 計算値 | 見直し後 |
|---|---|---|---|
| 基本設計工程 | 10 | 12 | 10 |
| 機能設計工程 | 46 | 52 | 53 |
| 詳細設計工程 | 52 | 58 | 58 |
| コーディング工程 | 104 | 116 | 117 |
| テスト工程 | 47 | 52 | 52 |
| 合計 | 259 | 290 | 290 |

品質判定表の結果が現実を正しく表しているか、現場へ行って確認する。

☆必ずしも品質判定表の結果を採用しなくてもよい☆

# テスト終盤の
# 残存課題の分析

# テスト工程品質会計　とは

## 特徴

- テスト工程用のバグ目標管理技法
- テスト開始時に残存する、プログラム全体のバグ数を目標管理する

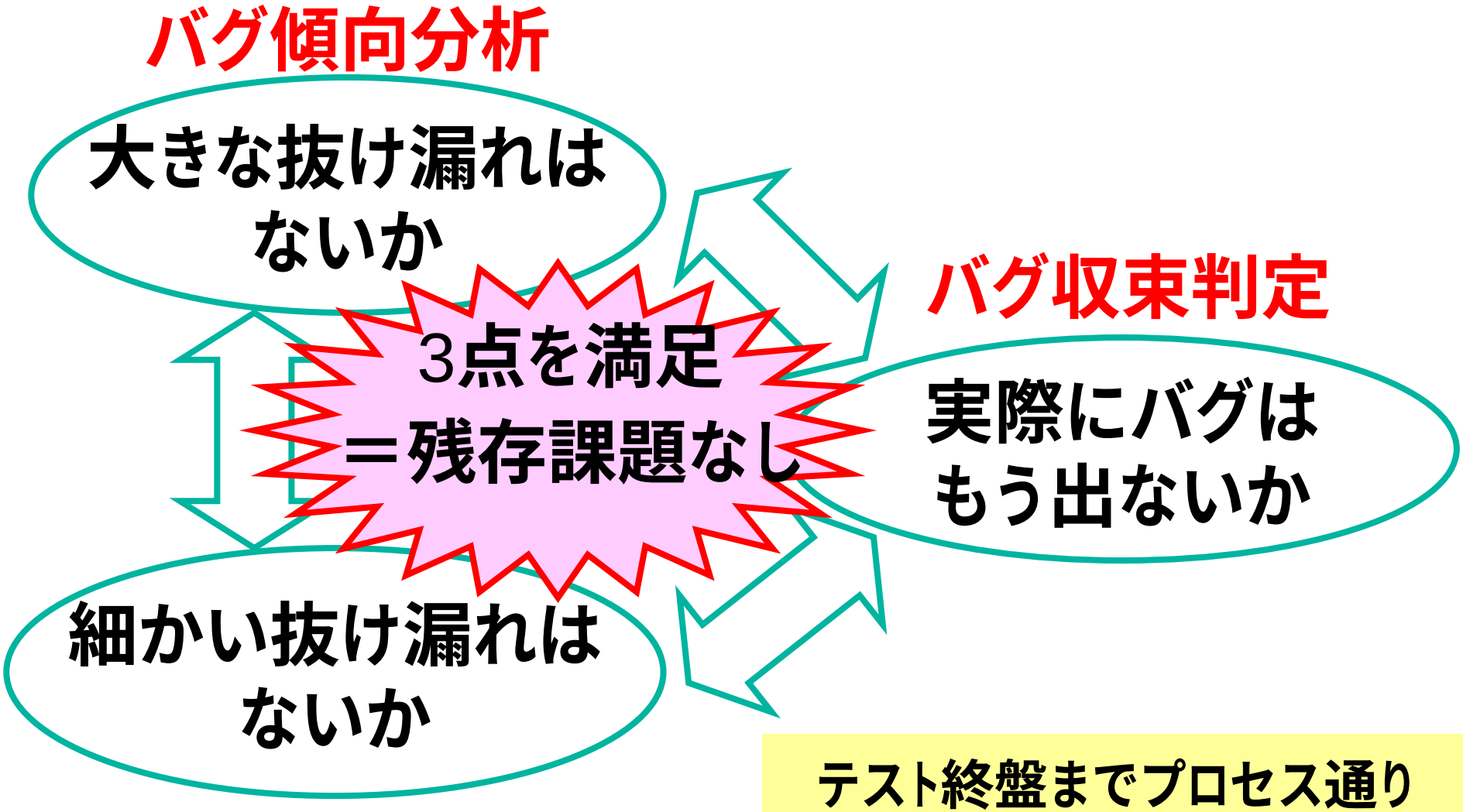
## バグはもう本当に残っていないか？を問い続ける＜残存課題の分析＞

- テスト終了条件は以下の2つ

1. 残存バグがゼロ
   - テストすべきことをすべて実施し、確認すべきことが残っていない
   - バグ傾向分析、バグ分析と1+n施策により、残存課題がないことを確認する
2. テスト工程のバグ摘出推移が収束
   - バグ収束判定技法により判断する

**★テスト終盤の残存課題の分析が、品質会計のもう一つの特徴**

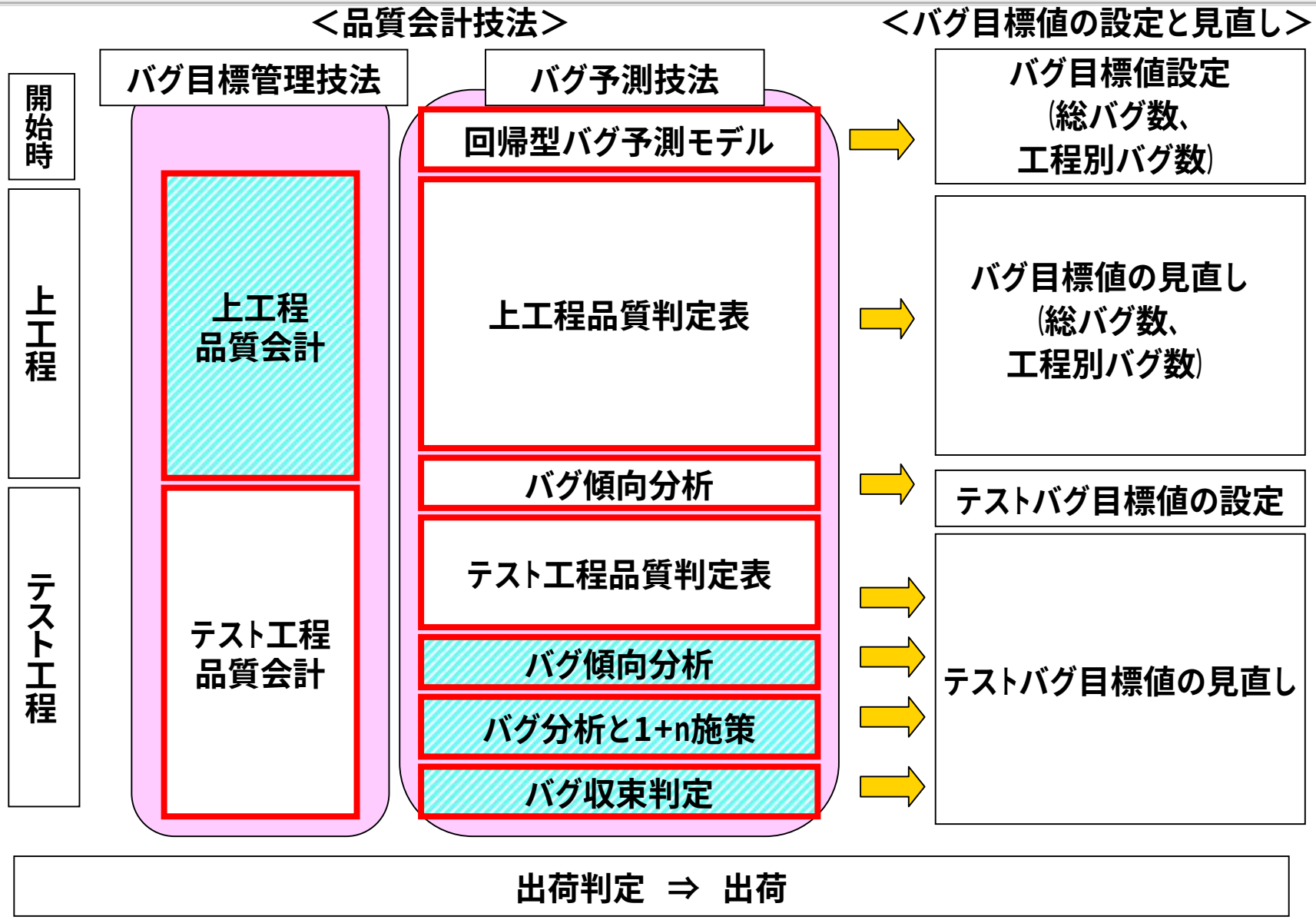
# テスト終盤の残存課題の分析の考え方




テスト終盤までプロセス通り
実施されていることが前提

# 開発プロセスと品質会計技法の適用の関係

再掲



# バグ傾向分析の狙い

## 大きな抜け･漏れはないかの確認

- 開発したソフトウェアの特性に沿ったバグが出ているか
- バグの出方が偏っていないか
- レビューやテストの目的に沿ったバグが摘出されているか
- レビューやテストに系統的な抜け漏れはないか

## バグ傾向分析には、決まった分析手順はない

- 対象ソフトウェアの特性を考えて分析する

# バグ傾向分析の観点

| 分析観点 | 内容 |
|---|---|
| 正規化 | 規模などの単位当たりでバグ数を正規化する。バグ数は規模の大小の影響を受けるが、単位規模当たりのバグ数であれば、数値の比較が可能となる。 |
| 構成する機能 | バグを、構成する機能ごとに分類する。どの機能のバグ数が多いかを把握することができる。 |
| バグ作り込み工程 | 摘出したバグを作り込み工程で分類する。設計・コーディングのどの工程に問題があったかを知ることができる。 |
| バグ重要度 | バグを利用者に与える重要度（致命的、重要、軽微など）で分類する。重要なバグが出ているかを把握することができる。 |
| バグ作り込み原因 | バグを作り込んだ原因で分類する。一般的な作り込み原因には、設計ミス・考慮漏れ・単純ミス・手順ミス・仕様理解不足・デグレードなどがある。作り込み原因は、分析対象ソフトウェアに応じて設定したほうがよい。 |
| 発生条件 | バグを発生条件（正常系、異常系、タイミング、組合せ、限界値など）で分類する。 |
| 発生現象 | バグを発生現象（結果異常、システム停止、データ破壊など）で分類する。 |
| 摘出者 | バグを摘出者で分類する。意図せずバグが摘出された件数を把握できる。 |

# ケーススタディ：テスト工程摘出バグのバグ傾向分析①

## 詳細機能別にＦＴ～ＳＴのバグ傾向分析を実施

⇒Ｂ-２機能は、規模も大きくバグ数が多い。ＦＤバグも摘出されている。

＜詳細機能別作り込みバグ（ＦＴ～ＳＴ）＞

| | Ｂ-1機能 | Ｂ-2機能 | Ｂ-3機能 | Ｂ機能全体 |
|---|---|---|---|---|
| ＢＤバグ | 0 | 0 | 0 | 0 |
| ＦＤバグ | 0 | 2 | 0 | 2 |
| ＤＤバグ | 0 | 3 | 1 | 4 |
| ＣＤバグ | 1 | 15 | 4 | 20 |
| 合計 | 1 | 20 | 5 | 26 |

＜詳細機能別の規模当たりのバグ（ＦＴ～ＳＴ）＞

| | Ｂ-1機能 | Ｂ-2機能 | Ｂ-3機能 | Ｂ機能全体 |
|---|---|---|---|---|
| 規模（ＫＬＯＣ） | 2.9 | 15.3 | 7.2 | 25.4 |
| 件/ＫＬＯＣ | 0.34 | 1.31 | 0.69 | 1.02 |

Ｂ-2機能はＢ機能の中核部分を実現している

# ケーススタディ：テスト工程摘出バグの傾向分析②

## 重要度別バグでは、B-2機能から、致命的2件・重要8件のバグが摘出
## ⇒B-2機能へ絞った品質分析を実施

致命的なバグが後工程に摘出されるのは問題
(実件数で考える)

＜詳細機能別の重要度別バグ（ＦＴ～ＳＴ）＞

| 重要度 | B-1機能 | B-2機能 | B-3機能 | B機能全体 |
|---|---|---|---|---|
| 致命的 | 0 | 2 | 0 | 2 |
| 重要 | 0 | 8 | 1 | 9 |
| 軽微 | 1 | 10 | 4 | 15 |
| 合計 | 1 | 20 | 5 | 26 |

結論：B-2機能のこれらの観点の追加確認が必要

画面表示系のテスト強化の結果

＜Ｂ-２機能の作り込み原因別バグ（ＦＴ～ＳＴ）＞

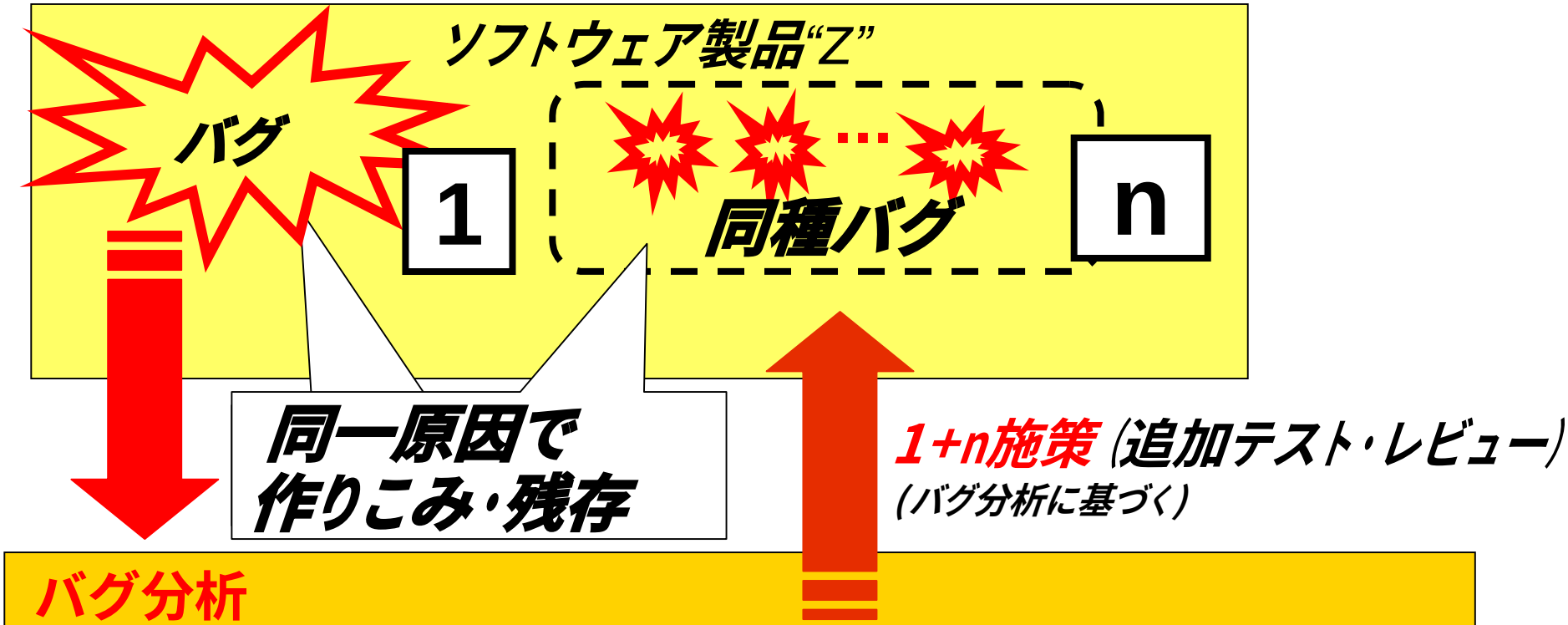
| バグの作り込み原因 | ＢＤバグ | ＦＤバグ | ＤＤバグ | ＣＤバグ | 合計 |
|---|---|---|---|---|---|
| GUI表示形式の不正 | | | 1 | 8 | 9 |
| 他機能とのインタフェース考慮不足 | | 2 | 1 | 3 | 6 |
| 異常系の考慮不足 | | | 1 | 4 | 5 |
| 既存仕様の誤解 | | | | | 0 |
| 設計仕様書の記載不足 | | | | | 0 |
| 合計 | 0 | 2 | 3 | 15 | 20 |

ＦＤレビュー時にＢ機能全体をよく知る技術者がたまたま欠席。他レビューでカバーしたつもりだったが…。

うち1件はＱＡバグ

# 「バグ分析と1+n 施策」とは

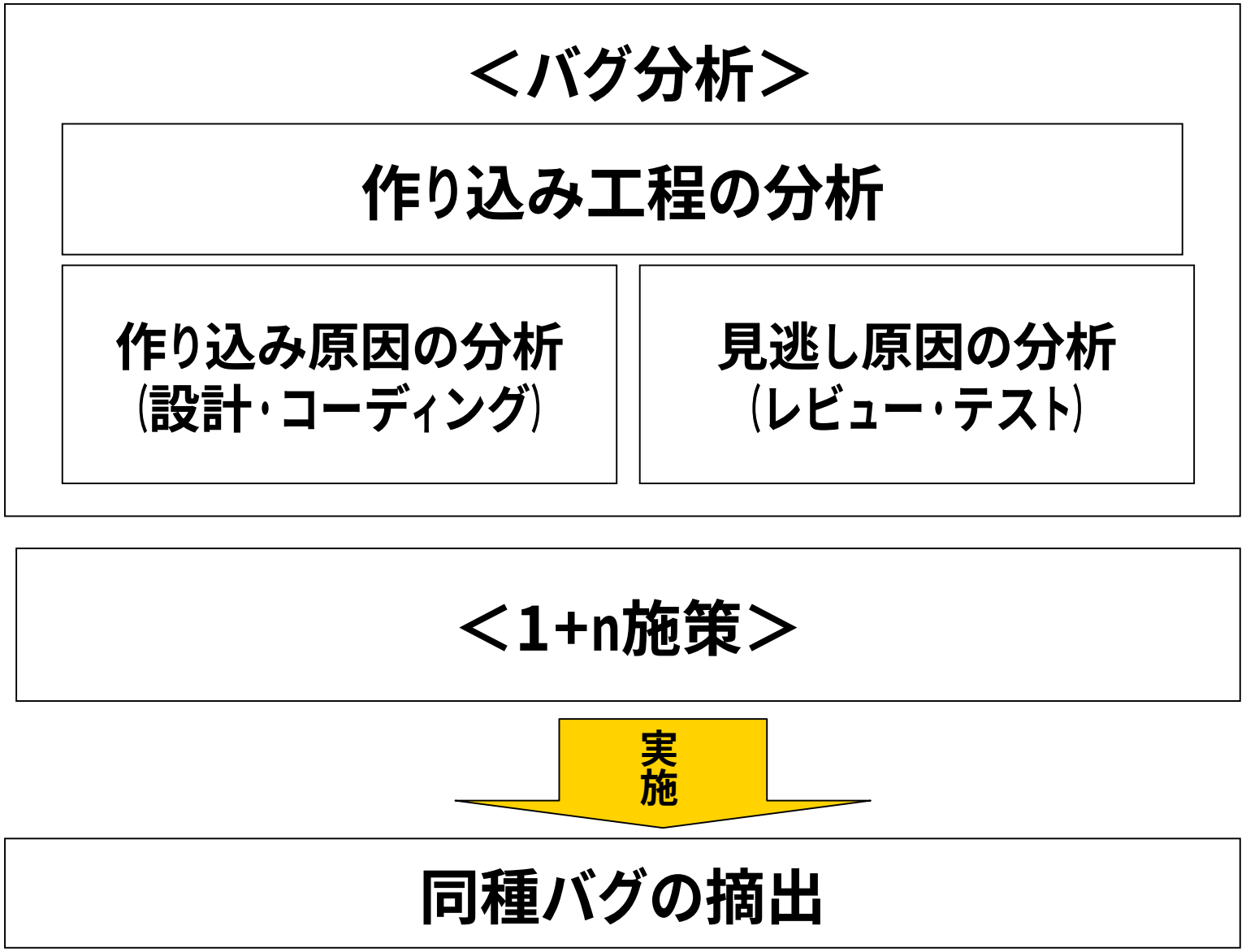
開発上の細かい抜け漏れを発見し、その抜け漏れに対して集中的にレビューやテストを実施することにより、残存バグ（同種バグ）を摘出する方法




**バグ分析**

なぜ、そのフィールドバグ を作りこんだか？
なぜ、そのフィールドバグ はレビューで摘出できなかったか？
なぜ、そのフィールドバグ はテストで摘出できなかったか？

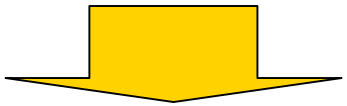
# バグ分析と1+n施策



# Case study

フィールドバグ

- ある入力フィールドへ範囲外の値を入力したところ、プログラムが異常終了



追加テスト・レビュー

- 他の入力フィールドに対して、範囲外の値のテストをする

*充分か?*

# Case study バグの真の原因分析 (1)

フィールドバグ
- ある入力フィールドへ範囲外の値を入力したところ、プログラムがアボート

なぜ このフィールドバグは発生したか？

1. 割る数が0のときのエラー処理実装漏れ

なぜ その実装漏れがおこったか？

2. その担当開発者は、割る数が0のケースがあることを知らなかったため

# Case study バグの真の原因分析 (2)

## ③ なぜ その開発者は割る数が0のケースが発生することを知らなかったのか?

その開発者は、開発途中に急に交代した交代要員だったため

- 旧開発者は、詳細設計書へ割る数が0のケースについての仕様を記載していなかった。(通常、設計と実装は同一開発者が担当するので、細かい仕様は記載していなかった)
- 現開発者は、設計仕様書に基づいて実装した。

➡ 真の原因は、**“書かれざる仕様”**

- このプログラムは、旧開発者によりレビューされなかったため、“書かれざる仕様”の未実装は発見できなかった。
- 誰も“書かれざる仕様”を知らないので、テストもしなかった

# Case study 1+n 施策

フィールドバグ
- ある入力フィールドへ範囲外の値を入力したところ、プログラムがアボート

追加テスト・レビュー
- 現開発者が実装したすべてのプログラムを、旧開発者がレビューする
  - 他に“書かれざる仕様”の未実装がないかを確認する
- 以下の観点からのテスト
  - レビューにより摘出された 全“書かれざる仕様 ”
  - すべての入力フィールドへの範囲外の値指定

# バグ分析のコツ

## 真の原因は1つではない

- バグ分析の目的によって真の原因が異なる
- 改善したい目的に応じて、分析の方向性を意識して変えるべき

## 目的を明確にする

- 1+n施策（同種バグの摘出）
  - 作り込み原因、見逃し原因（レビュー、テスト）を分析する
  - その原因を裏返したとき、それをやっていたらそのバグを作りこまなかった、見逃さなかったと判断できるところがゴール
- 次プロジェクトでの改善　⇒プロジェクトに特化して改善策を考える
  - 当該プロジェクトの範囲で分析する
  - 真の原因を裏返したとき、次プロジェクトではその問題は発生しないと判断できるところがゴール
- 組織的なしくみの改善　⇒組織の特性をふまえてしくみ化の方向で考える
  - 組織の範囲（その組織で実施されるさまざまなプロジェクトを包含）で分析する
  - 真の原因を裏返したとき、その組織では2度とその問題が発生しないと判断できるところがゴール

# バグ作り込み原因と見逃し原因一覧

- 分析精度を向上させるため、過去の分析事例から真の原因を整理してテンプレート化
  ⇒整理された選択肢を見ながら、真の原因を考える

| | | 技術面の問題（例） | 進め方の問題（例） |
|---|---|---|---|
| 作り込み原因 | 設計・コーディング | 技術調査不足<br>設計・プログラミング技術不十分<br>設計・コーディングノウハウ不足<br>入力資料のミス<br>設計仕様書の記載不十分・誤り | 必要な開発工程を省略<br>標準・ルールに則っていない<br>担当者間の連携・引継ぎミス<br>関係部門との調整不十分<br>開発計画が不十分 |
| 見逃し原因 | レビュー | レビューチェック観点漏れ<br>レビューアの問題<br>レビュー技術の問題<br>レビュー不足 | レビューをしていない<br>レビュー実施方法に問題<br>レビュー未完了<br>指摘事項の修正漏れ・誤り<br>レビュー計画が不十分 |
| | テスト | テスト項目にあがっていない<br>テスト項目の誤り<br>期待するテスト結果の誤り<br>テスト結果確認方法の誤り | テスト項目の実施漏れ<br>テスト工程項目実施方法に問題<br>担当者間の連携・引継ぎミス<br>テスト計画が不十分 |

# 1+n施策のコツ

## 真の原因の裏返しが1+n施策となる

- 施策内容や範囲を、裏付けなしに絞り込まない
- 1+n施策の結果、同種バグを摘出できなかった場合は、バグ分析の的確性を確認する
- 1+n施策の結果、摘出したバグはその内容を確認する

  ⇒必要なら追加1+n施策を実施

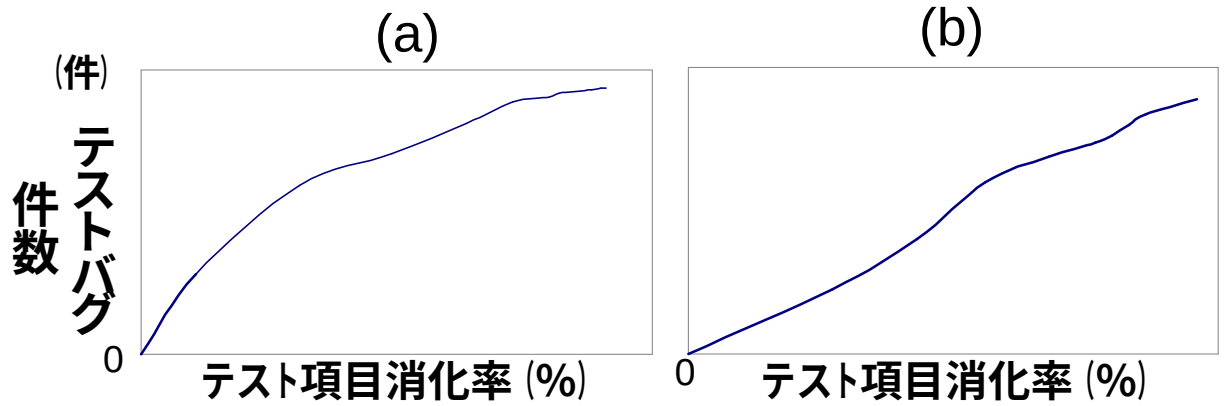
# バグ収束判定

## 新たな追加テストを実施しても、バグが摘出されないことを実証することが目的

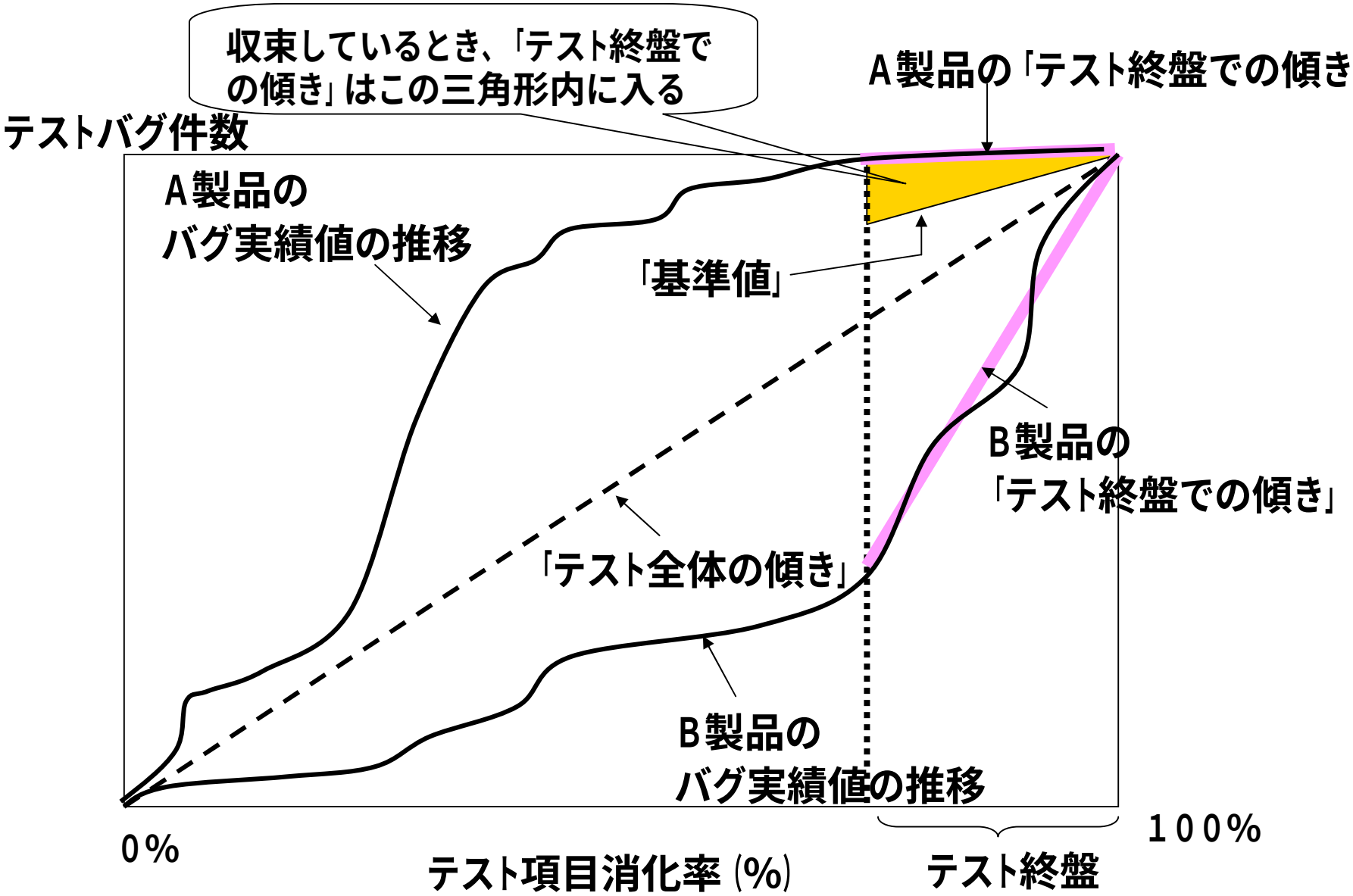
- ×：もうバグは摘出されない「はず」
- ○：追加テストしても、「実際にバグは摘出されない」

## 傾きを数値化して収束判定

(a)と(b)は各々、収束していると判断しますか？

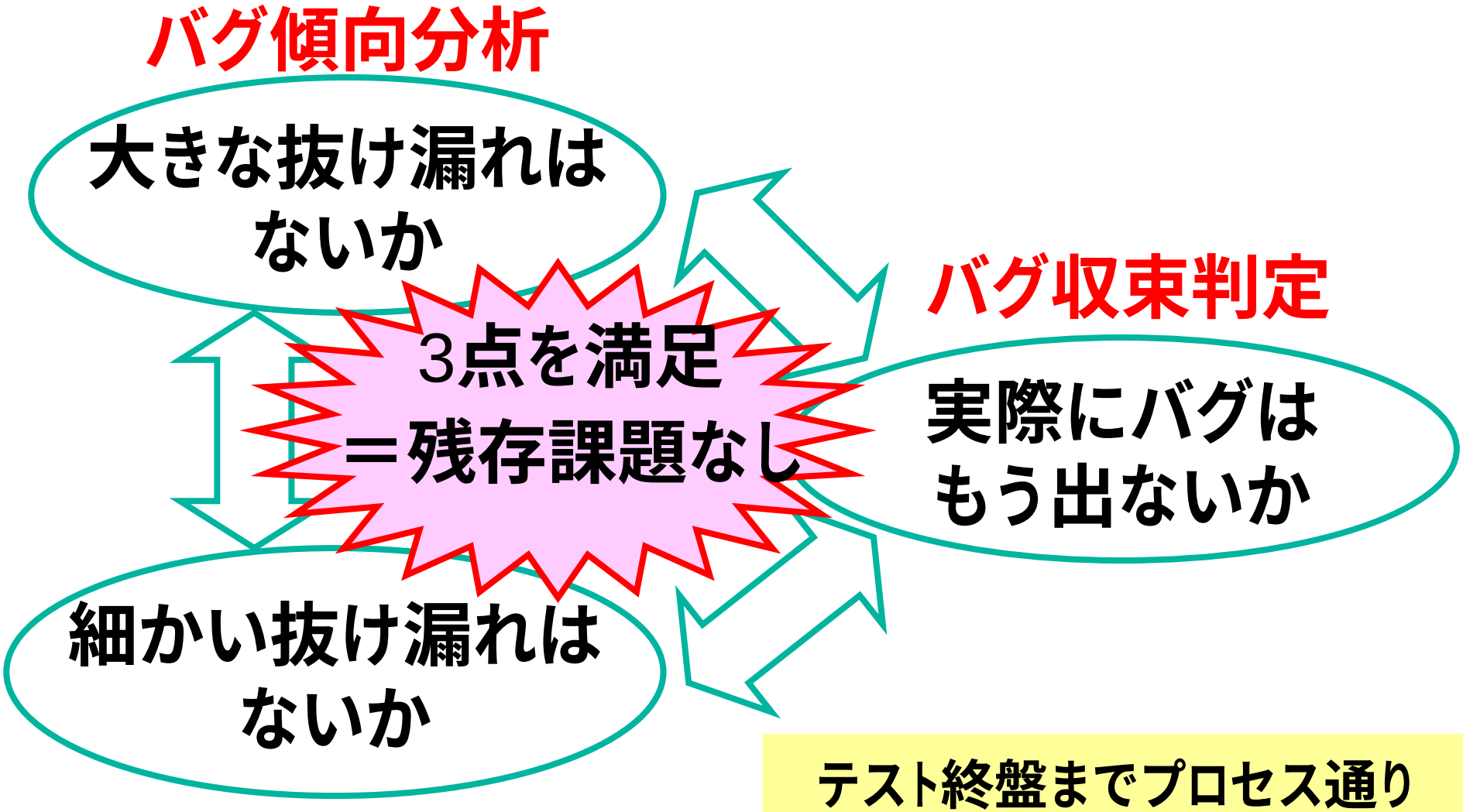


# バグ収束判定の考え方



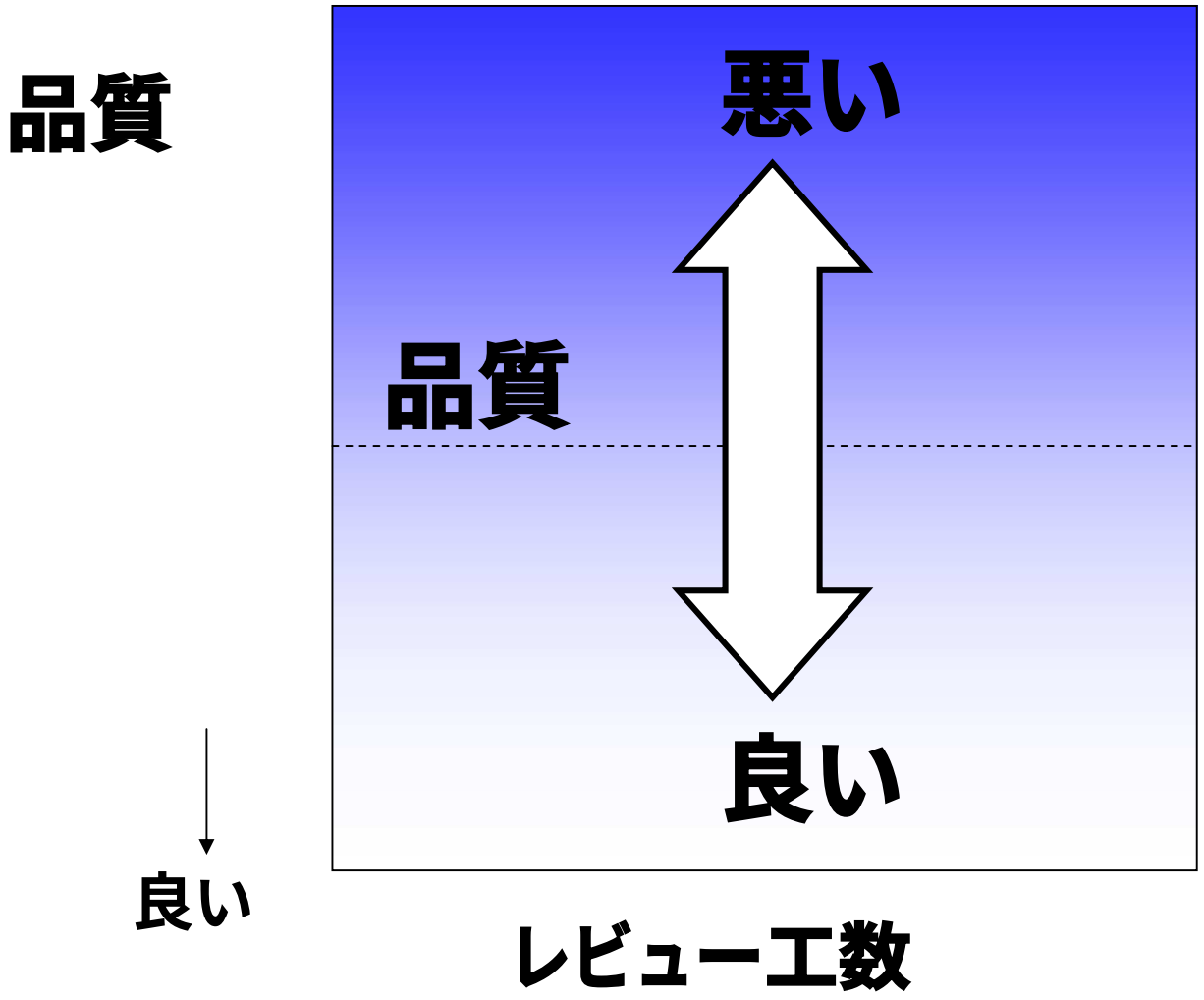
# テスト終盤の残存課題の分析の考え方

再掲

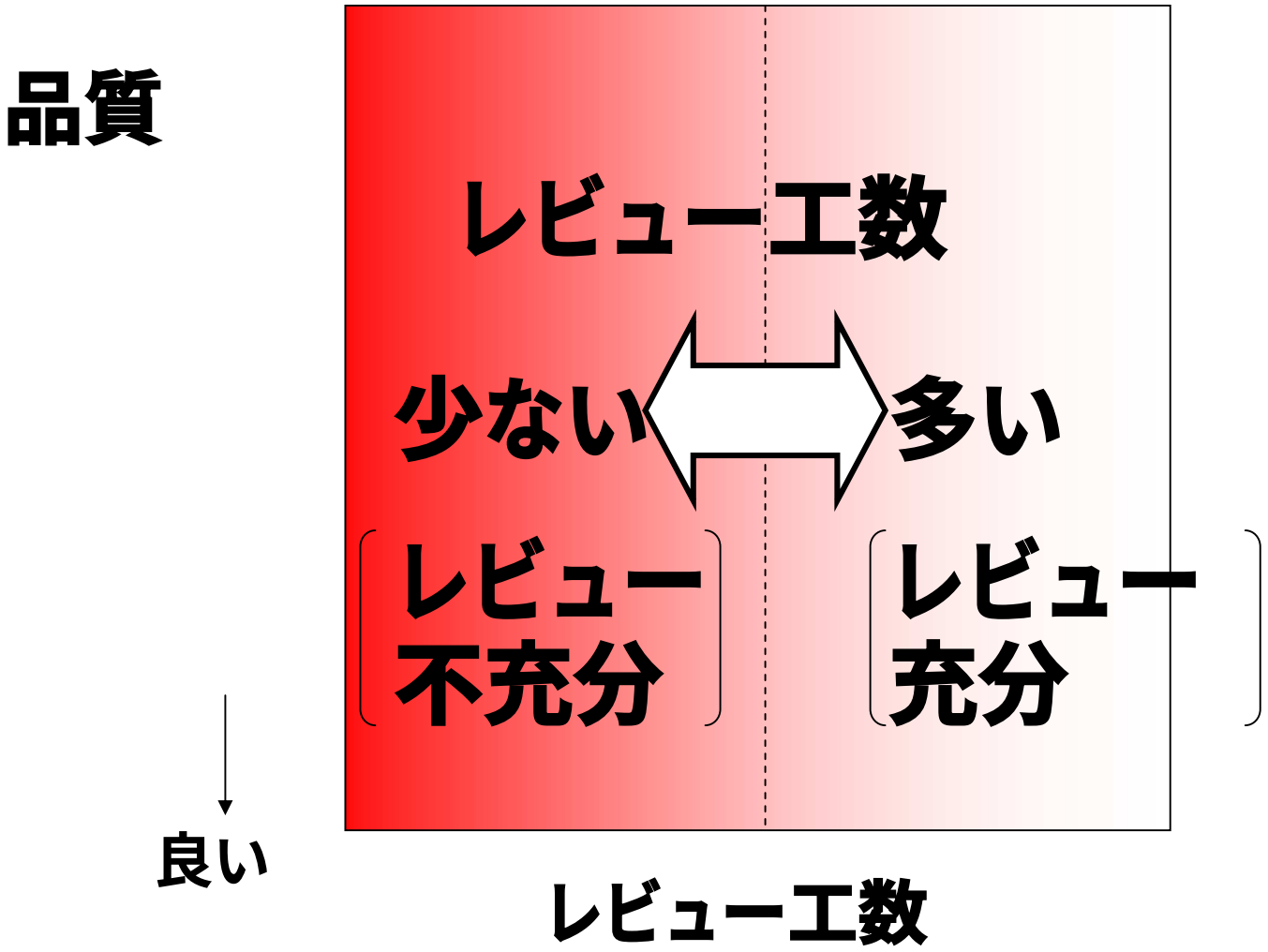



テスト終盤までプロセス通り
実施されていることが前提

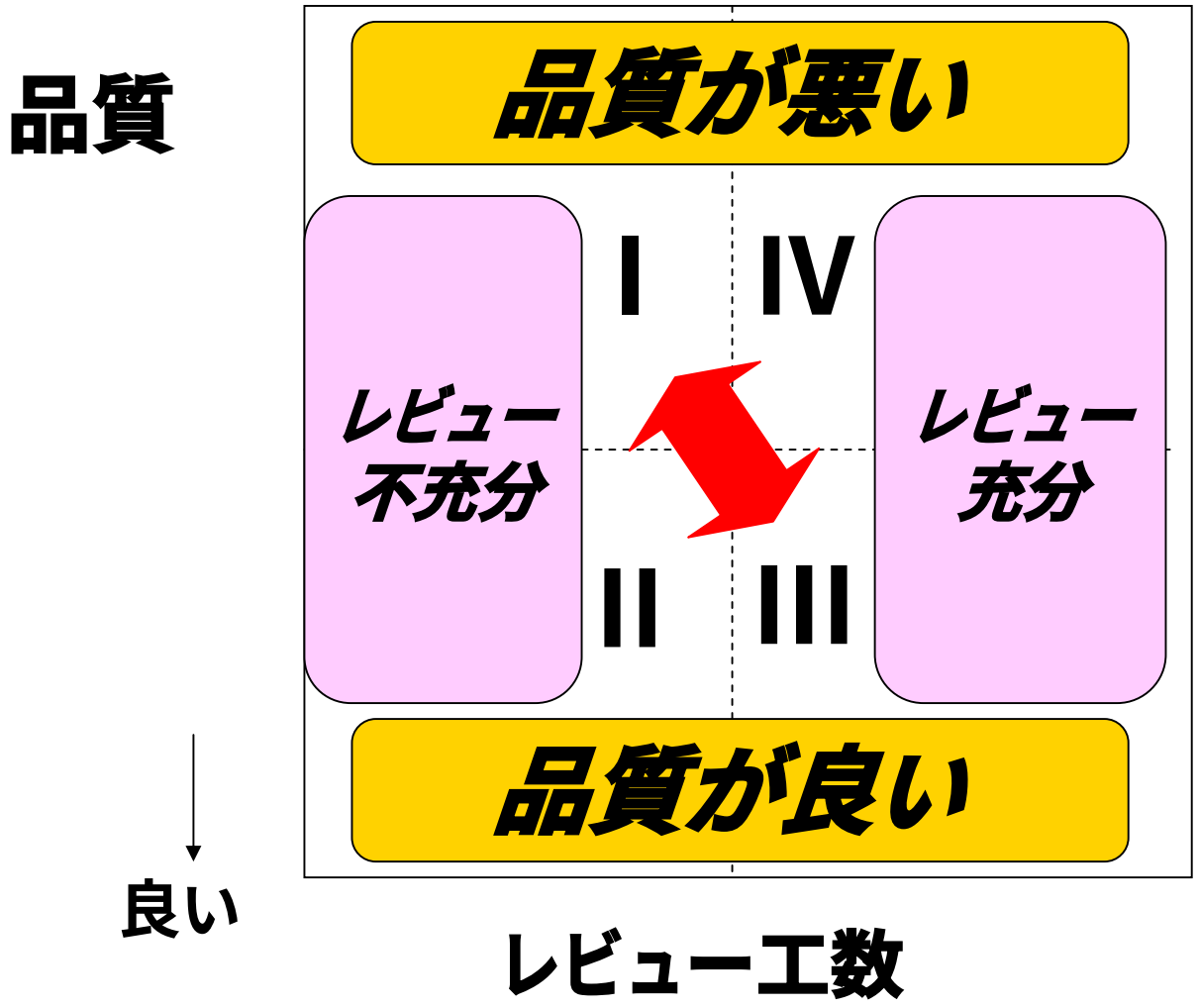
# 分析方法 1



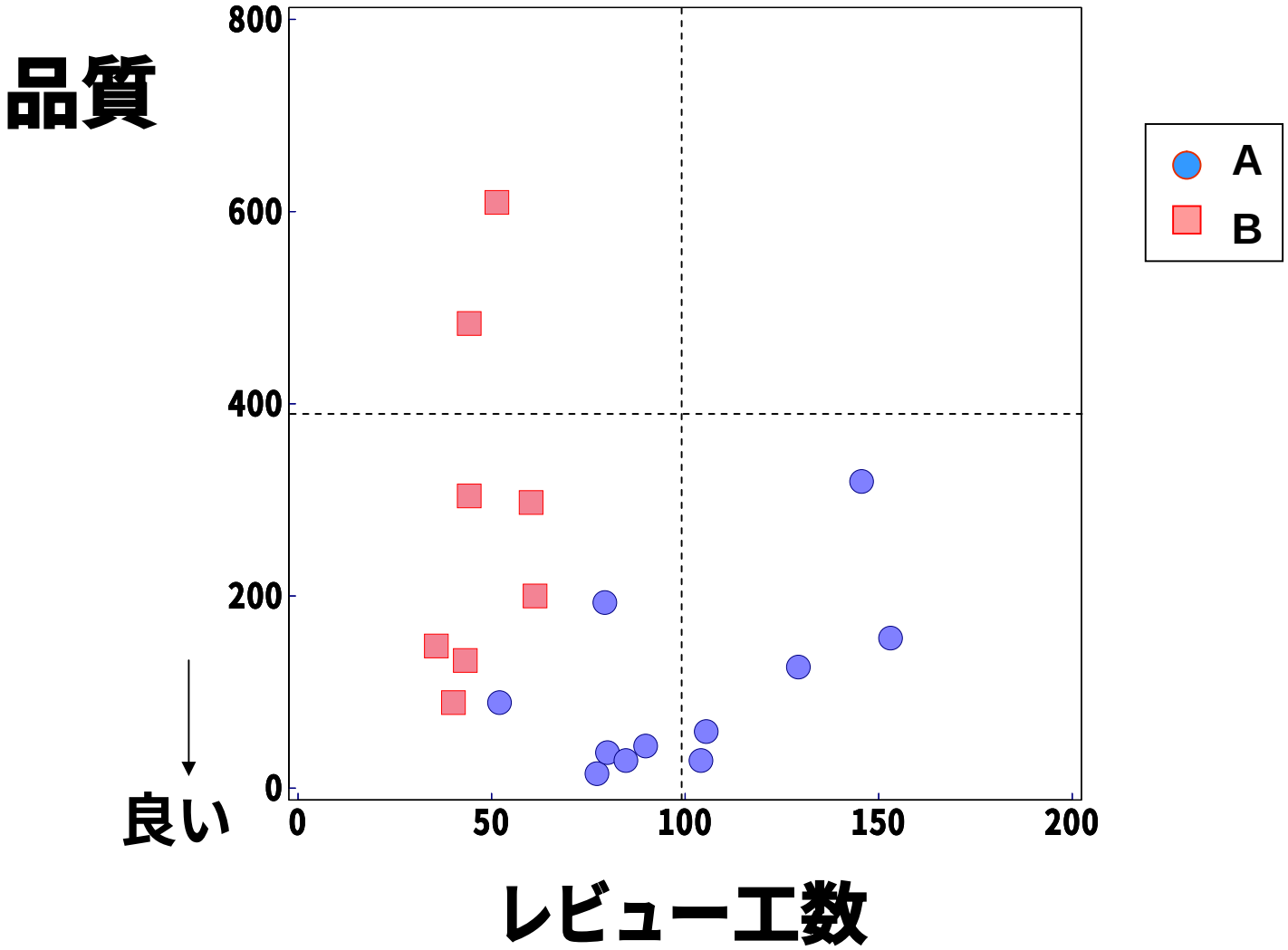
# 分析方法 2



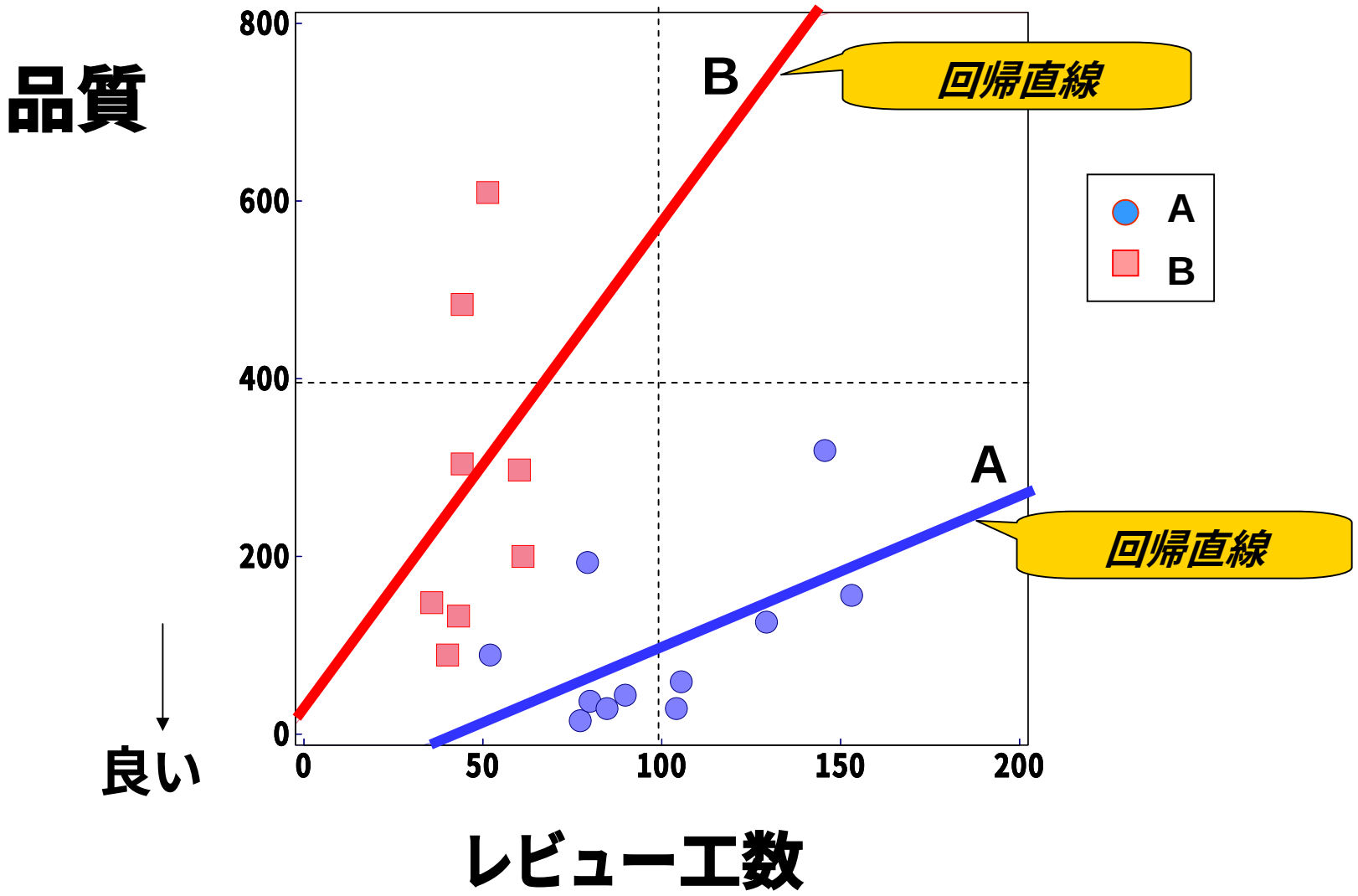
# 分析方法 3



# レビュー工数 vs. 品質

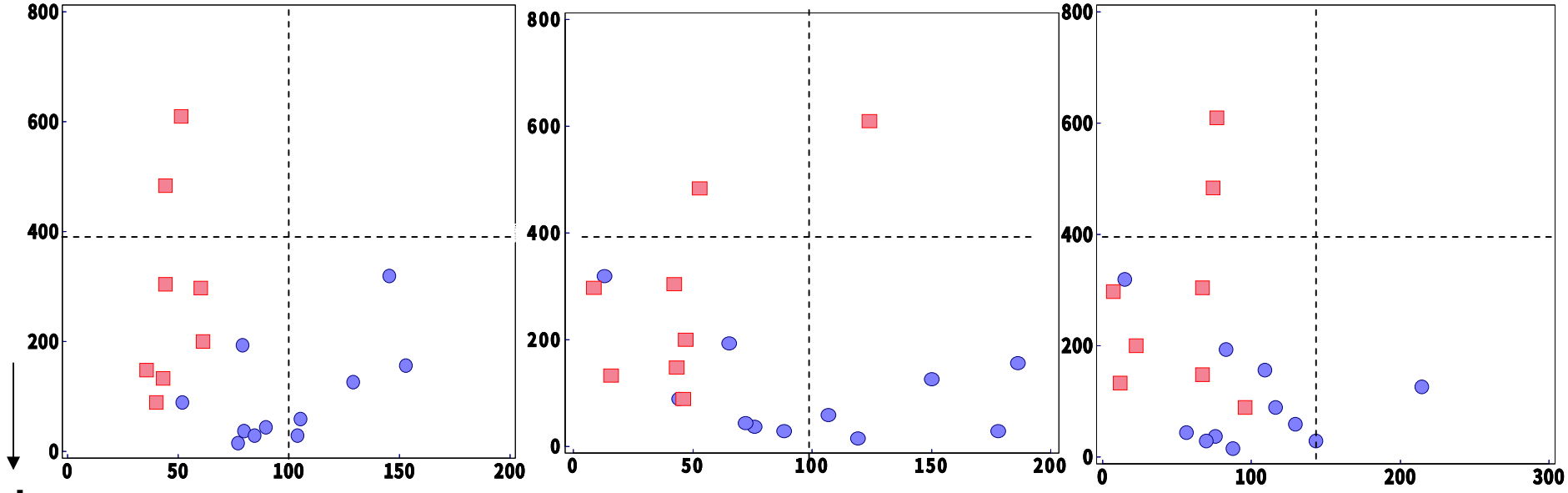


# レビュー工数 vs. 品質



# 比較 1-1

品質



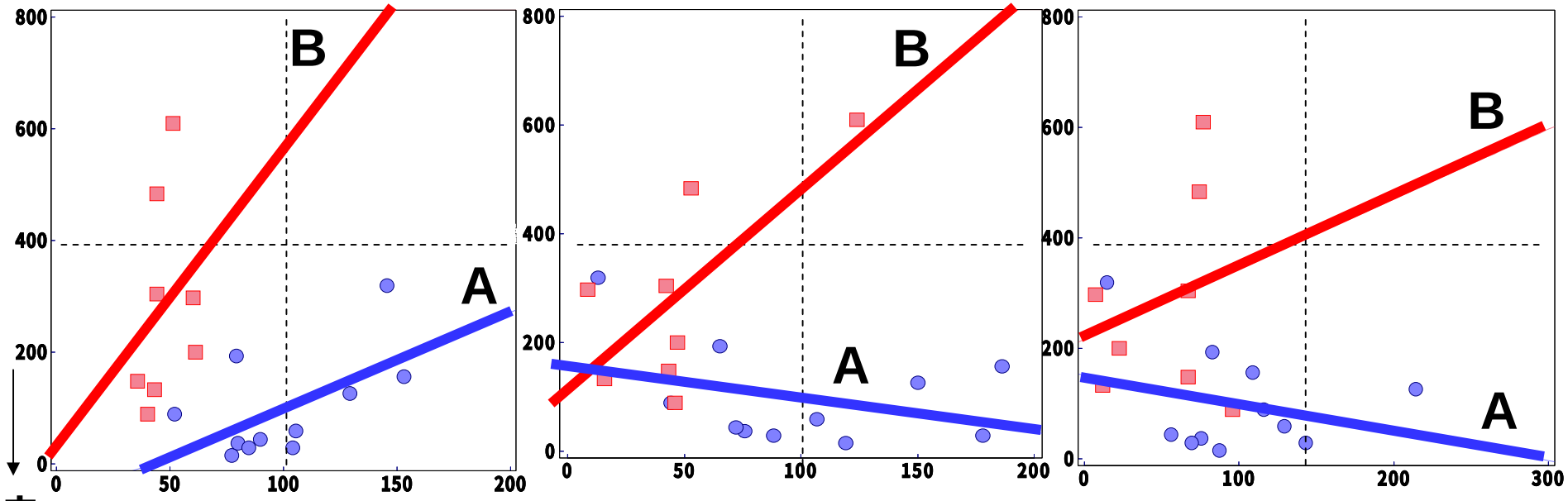

良い

レビュー工数　　テスト工数　　テスト項目

A
B

# 比較 1-2

品質

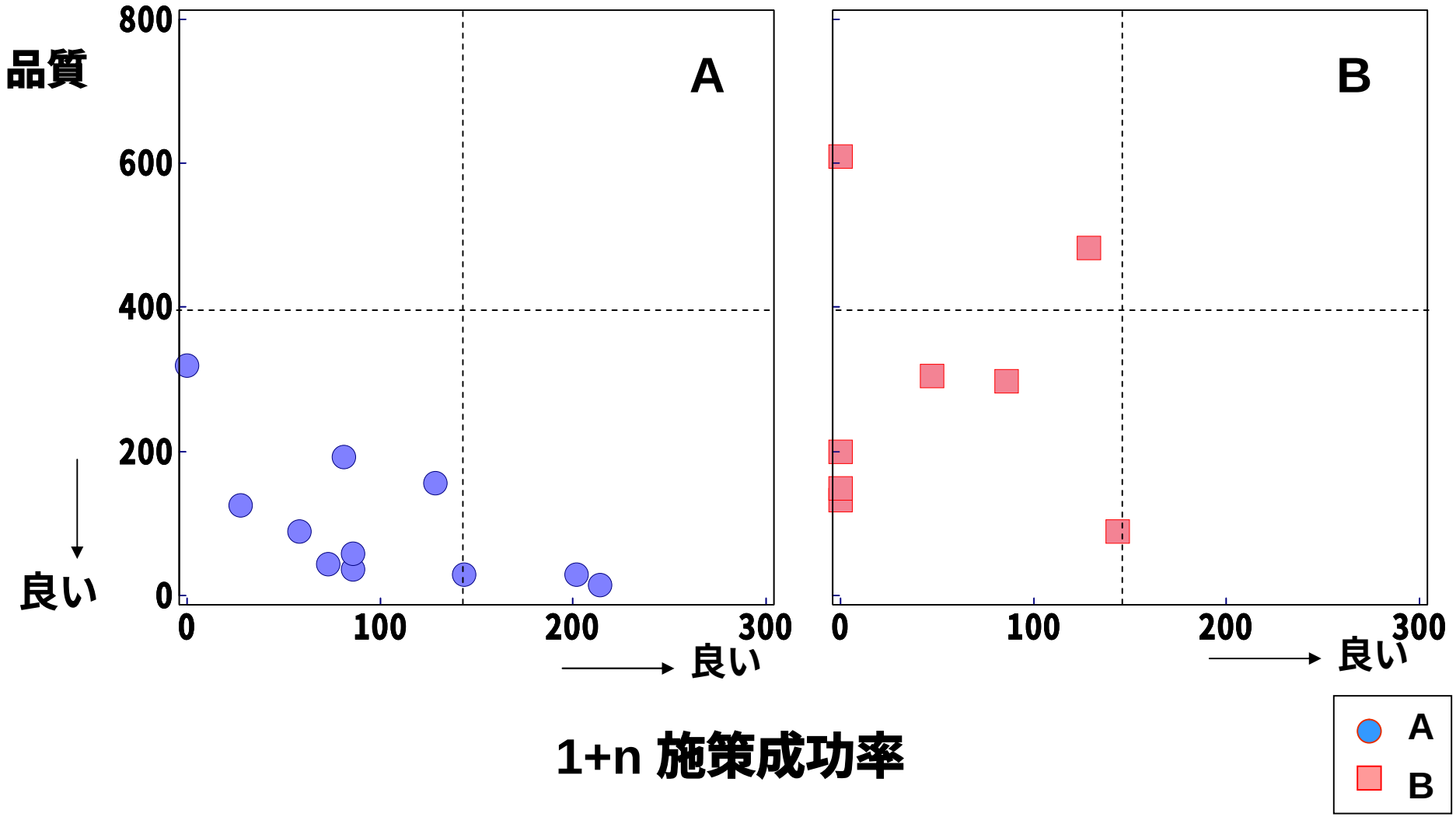



良い

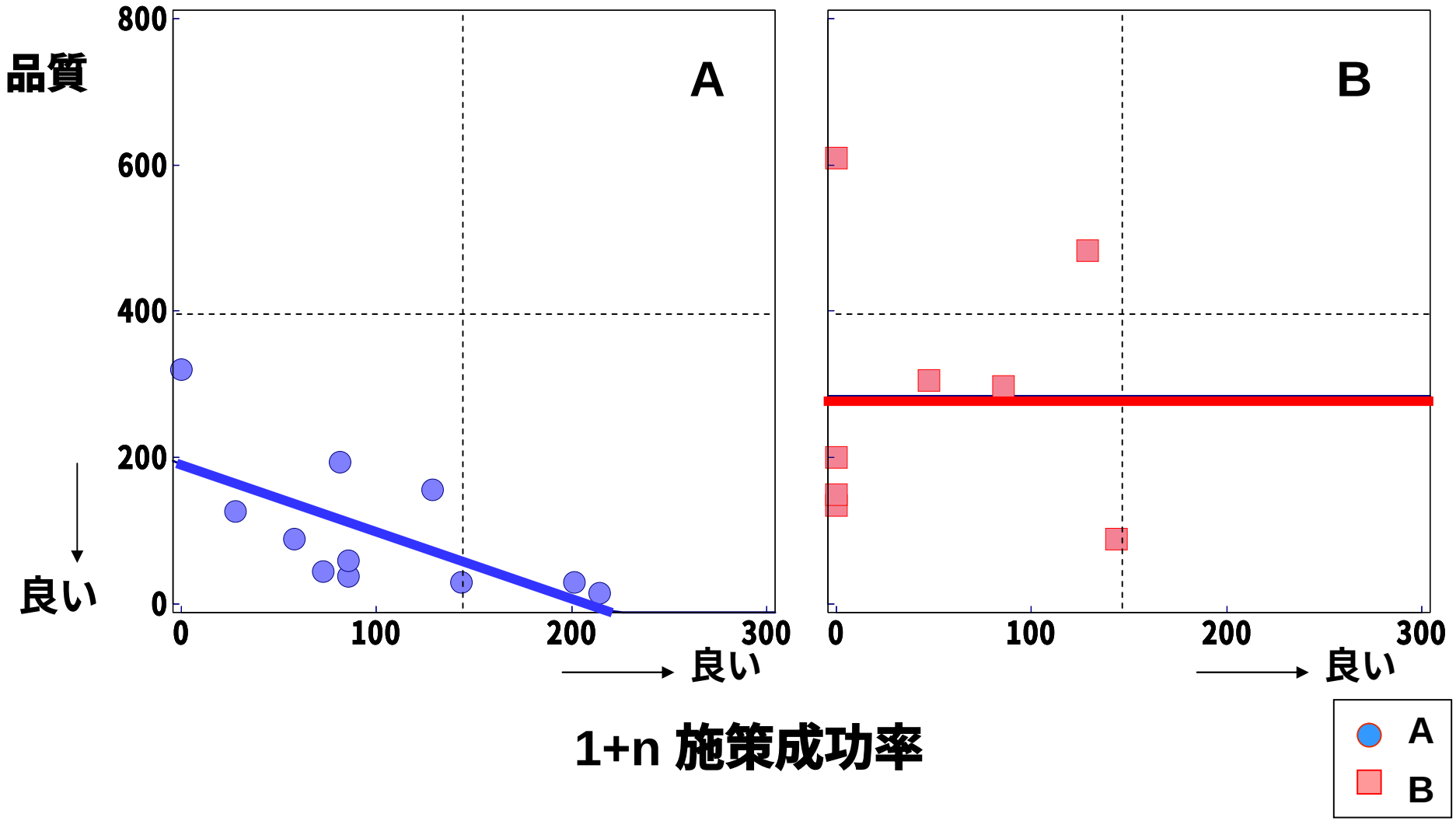
レビュー工数　　テスト工数　　テスト項目

A
B

# 1+n 施策成功率　vs. 品質



# 1+n 施策成功率　vs. 品質



# 3. 現場･現物･現実を重視するとは

- 同じ技法・プロセスを適用しても、同じ成果はでない
- データだけでは判断を誤る
- 現場へいかなければ、本当に良いかどうか判断できない

# 上工程で「現場へ行く」ことの重要性

## 上工程は把握しにくい

- 実際の成果物を見て、ほんとうにできているかを確認する
  - 適切な技法を使用して設計しているか
  - 記載の詳細度合いは適切か、概要すぎていないか
- 記録類を見ながらヒアリングし、開発経緯を確認する
  - レビューは具体的にどのように実施したか
  - レビュー結果の反映および反映結果の確認はどのように実施したか
  - 関係するチームとのレビューを実施したか
- 現場へ行って、プロジェクトの雰囲気を確認する
  - コミュニケーションはとれているか
  - 信頼関係は醸成されているか
  - 関係するチーム間のコミュニケーションはとれているか
  - 不満はないか

**現場・現物・現実を確認したうえで、データを分析する**

# テスト工程で「現場へ行く」ことの重要性

- 「少々進捗遅れが出てきましたが、がんばって挽回します！」 本当か？
  - バグ実績値が急増
    - テスト内容が大きく変わった（例：統合テストを開始した）
    - 今までテストしていなかった部分のテストを開始した.....
  - バグが収束しない
    - テスト項目が大雑把過ぎて、実はそれまできちんとテストできていなかった
    - テスト工程でも仕様変更に対応し続けていた
    - バグを修正するとデグレード発生、の繰り返し....
  - 急に進捗が止まった
    - テスト担当者が病気になった
    - 同じ現象があちこちで発生し、テストは進まず、原因もわからない....

- 現場へ行ってほんとうの理由を把握するからこそ、品質判定表の結果を活かせる

# 参考：ＱＣＣ（品質中心の組織文化）とは

| | 丸投げ文化 | 対極 | ＱＣＣ（品質中心の組織文化） |
|---|---|---|---|
| ルールの考え方 | ルールだから実施する | | 目的を理解したうえでルールを使おうとする |
| 判断基準 | 組織基準値に従って判断する | | 自ら最終目的を想定して判断する |
| 仕事への取り組み | 決められた通りに仕事を進める | | 常に仕事のしかたを改善しようとする |
| 価値観 | 場面によってＱＣＤの優先順位が変わる<br>短期志向 | | 常にＱ中心である<br>長期志向 |

# 最悪の品質会計①：数字合わせ

- 開発開始時に、テストでの摘出バグ100件と予測した。
  ちょうどテストでの摘出バグが100件となったので、
  テストを終了した。

# 最悪の品質会計②：なんくせをつける

テスト終了段階に近づいたある日、開発者がバグ目標値を下方修正する見直し申請をしてきた。

それを受けたスタッフは、「ああだ、こうだ」と言って下方修正を認めない。開発者がいくらテストの経緯を説明しても理解してくれない様子である。

# 最悪の品質会計③：通り一遍の報告

「2週間の進捗遅れが設計工程で発生していますが、上級設計者に設計支援させることにより、リカバリを図る予定です。」

「2週間の進捗遅れが設計工程で発生しています。その原因は、同期をとって動作する別機能との間で、今までの考慮していなかったデータのやりとりをする必要があることがわかったからです。この設計には、別機能の設計担当者との検討が必要なため、本機能の設計を熟知している上級技術者に設計支援させて、やりとりすべきデータを設計するとともに、発生しうるすべてのイベントを洗い出して問題ないことを確認する必要があります。この作業に既に着手しており、あと1週間で完了する見込みです。」

# 「品質会計しています」とはどういうことか

×

- 品質会計の帳票にデータを書いて出している
- バグ目標値＝バグ実績値である

○

(データを出すのは、スタートライン)

- 開発途中の問題と対応を説明できる
- なぜそのバグ目標値でよいのかを説明できる
- 特にテスト最終段階での品質と対応状況を説明できる

⇒「品質」を作り込んだことを、確かな根拠をもって
具体的に説明できる

# 品質会計で、なぜバグ目標値を常に見直すか

## ソフトウェア開発は予定通りには進まない

- 予定した技術者を確保できない
- 想定したより技術者のスキルが低い
- 同時に動作するソフトウェアが、調査時とは異なる動きをする.....

## 要求事項は変化する

- お客様からの要件（機能、動作条件、性能...）
- ＱＣＤ....

あらゆる変化を見越したバグ予測はありえない

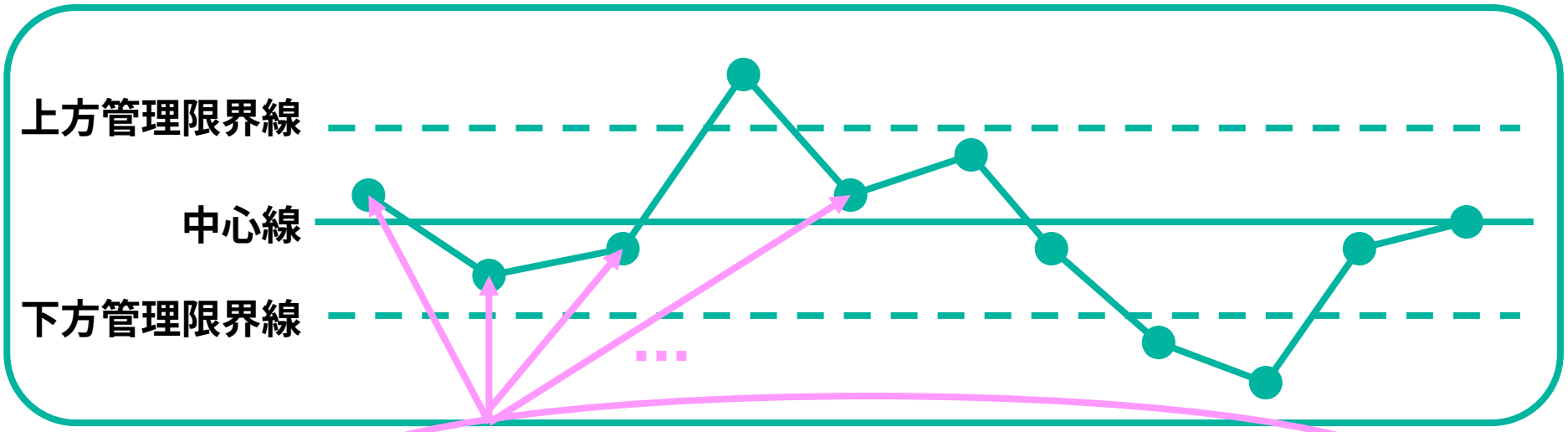
開発途中のさまざまな変化を入力として、バグ目標値を見直す

# 品質会計におけるバグゼロの意味

## 一般的な品質管理手法：バラツキの管理

- 予測と実績の変動を最小限に抑えることにより、出荷製品の不良率のバラツキを最小限に抑える

＜管理図の例＞




## 品質会計

- 管理限界内に入れるだけでなく、開発したひとつひとつのソフトウェアに「残存課題がない」ことを確認する

# ー銀の弾丸はない，しかし道はあるー

- 技術にせよ管理手法にせよ、 単独でソフトウェアの生産性や品質を飛躍的に改善するものはない

- 王道はない．しかし，道はある．

ーF.ブルックス「人月の神話」より

## 現場主義が、成功への道を拓く

Empowered by Innovation