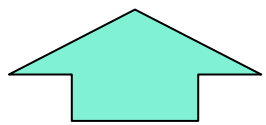
「品質リスクの予測」への展望

～バグの作り込みパターン，除去パターン～

S-open (ソフトウェア技術者ネットワーク)
ソフトウェアメトリクスSIG
堀　明広

JaSST06 東京での発表内容

- 「定量的ソフトウェアテスト完了判断基準の一考察」
  - ソフトウェアテストの目的とは何か
  - GQMパラダイムを利用してテストの完了を定量的に判断するための基準を導出



○	テストの完了判断では様々な要素を考慮する必要があるが，これらをある一例として整理することはできた。
△	まだまだ，曖昧な点が多い！

問題を考察する前に，前回はどんな発表であったか，次ページ以降で簡単に…

ソフトウェアのテストに明快な終わりはない

- 良く言われること
  - テストでバグがあることは，証明できる
  - テストでバグがないことは，証明できない
- テストを続けていれば，バグは出てくるもの
- 経済的な理由から，どこかで何かをきっかけに，テストの継続をあきらめなければならない

テストの「完了」を適切に判断するには，どう考えるべきか？
定量的尺度を使ってテスト完了判断を行うには何が必要か？

検討するテストフェーズ

何れの開発モデルも，
テストは最下流に位置する

完了判断
(顧客にリリースする直前)

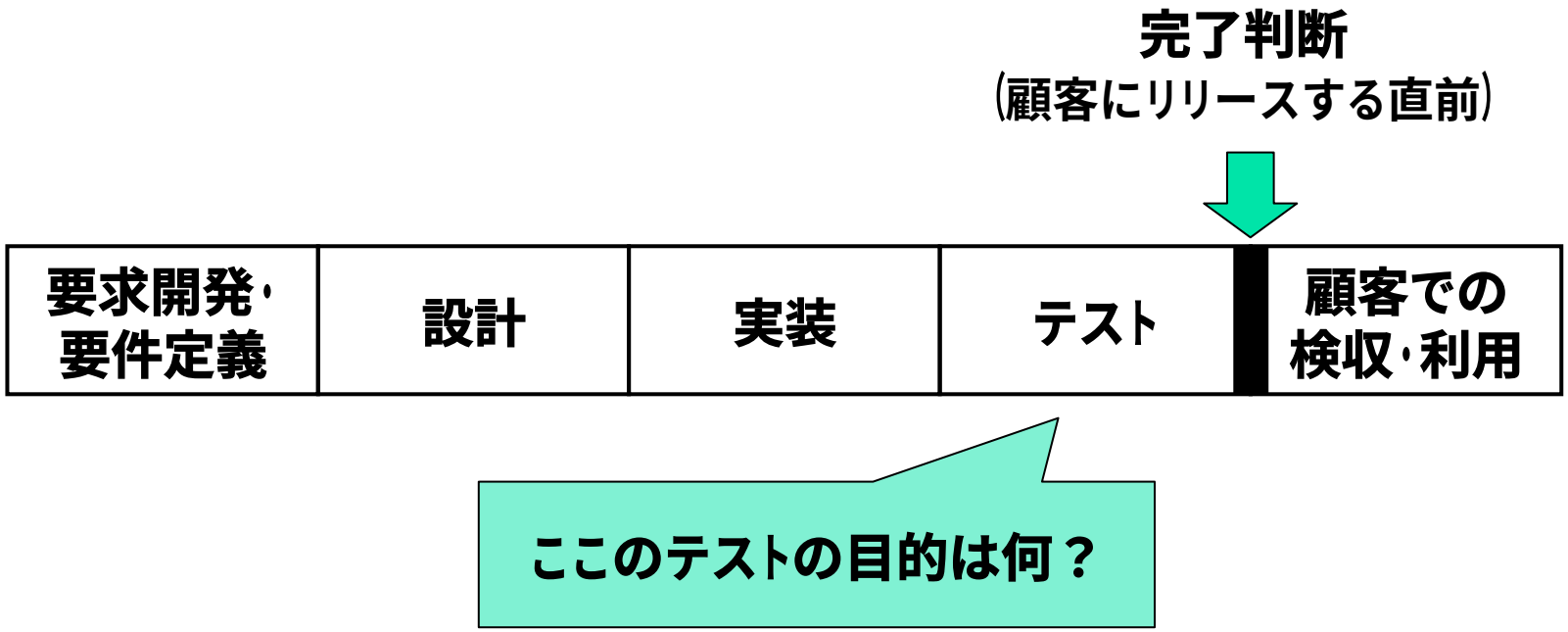
＜検討対象＞

「Verification & Validation」の”Verification”をメインに据えて，
テストの完了判断を定量的に行うための考え方，前提条件を探る

「テストを完了した」とはどういうことか？

＜本稿の考え方＞

そのテストで目的としていたものを達成していると言える状態



テストの4つの目的

従来から言われている目的

- ソフトウェアに潜むバグを検出し，修正
- 求められる要件を満足していることを確認

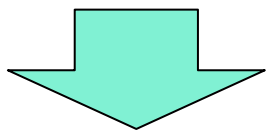
- バグの発生状況から，未だ潜むバグの存在を推測し品質の成熟度を判断
- バグの作込み要因と，より上流の検証フェーズでの流出要因を分析して，開発プロセスの問題を特定・対策

情報を収集し，アクションにつなげる活動とした捉え方

ソフトウェアテストの完了判断基準とは

そのテストの目的を達成しているか，判定する材料

- **本稿のアプローチ**
  - **テストの目的をゴールに据えて，メトリクスを設計・検討するためのＧＱＭパラダイムに当てはめる**

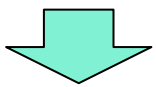


  - **定量的ソフトウェアテスト完了判断基準を構築**

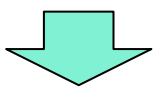
ＧＱＭパラダイムとは

■Basili教授らによって提案されたソフトウェア測定 及び 評価のための総合的な枠組み
■ＧＱＭを階層構造で表すことにより、測定 及び 評価目的を明確化

- 測定の目標(Goal)を明確化



- その目標の達成方法 又は 評価方法を質問(Question)の形式で記述



- その質問に定量的に答えるための測定量(Metric)を定義

ソフトウェアテストのGoalとQuestion

- <G1>ソフトウェアに潜むバグを検出し，修正
  - G1Q1:妥当な質と量のテスト項目を設定
  - G1Q2:設定したテスト項目を実行
  - G1Q3:検出したバグを修正
  - G1Q4:検出したバグを正しく修正していることを確認
- <G2>要件を満足していることを確認
  - G2Q1:要件を網羅するテスト項目を設定
  - G2Q2:プログラムの全機能を動作させるテスト項目を実行
- <G3>バグ発生状況から品質成熟度を判断
  - G3Q1:予め想定したバグの数だけ，バグを検出できている
  - G3Q2:信頼度成長曲線が収束している
- <G4>開発プロセスの問題を特定･対策
  - G4Q1:バグの発生状況を，開発プロセスにフィードバックしている

テスト目的 ブレイクダウン ⇨ 必要なメトリクスを設計 ⇨ 完了判断基準 構築

テスト完了判断基準の大項目

- <0>体系的なテスト設計（前提条件）
- <1>テストの実施状況
- <2>バグ発生状況
- <3>バグの修正状況
- <4>バグの修正品質
- <5>信頼度成長曲線の収束度合い
- <6>検出バグの発生傾向分析
- <7>開発プロセスへのフィードバック状況

※検討経過は省略（結果のみ記載）

<0>体系的なテスト設計（前提条件）

- テスト項目は，的確・効率よくバグ検出できるように，論理立てて設計していなければならない。
- これを測るため，着目したカテゴリ別にテスト項目数が分かるようにすることが必要。
- 要件を満たしているか確認するテストの実施状況を表すため，要件とテスト項目数を対比できるようにしなければならない。
- これらはテスト対象の規模に見合った量でなければならない。
- 上記の条件を満たしつつ，有限なリソース内で収まるように，計画する必要がある。

<0>体系的なテスト設計（前提条件）（続き）

- バグを終盤で発生させると，その対処にはコストと危険が伴うため，バグを可能な限り早期に検出する必要がある。
  ムダにテストを行うと時間のロスにつながるため，テスト項目数はむやみに増やすことはできない。
- リソースと効率的なテストとのトレードオフで，テストを実施しないと意志決定した領域がもしあれば，それはリスクとして管理下に置き，モニターする必要がある。

テスト完了判断基準

- <1>テストの実施状況
  - 設定したテストは，全て実施済みでいること。
  - 必要なテストが，環境面や他の事情で実施できない項目があれば，その理由とリスクの度合い･対応策を明確にする
  - カバレッジとテスト項目数，過去の実績と照らし合わせ，一定の基準を満たしていること。
- <2>バグ発生状況
  - 過去の実績と照らし合わせて目標に定めた件数に見合った数のバグを検出できていること。
  - バグとテスト項目数の比を検討し，妥当な基準にあること。
  - 少数のテストであまりに多くのバグが検出されていたり，バグの検出数が目立って少ないなどが見受けられた場合には，その理由を分析し，対策すること。

テスト完了判断基準 (続き)

- <3>バグの修正状況
  - 検出したバグは原則として全て修正していなければならない。ただし，重要度と他への影響に応じ，残存するバグ件数を許容する基準を設けても良い。
  - その場合には予め下記を実施していること。
    - 重要度の定義，個別のバグのランクを意志決定する仕組みを予め計画に盛り込み，関係者間で合意している
    - 顧客に残存するバグの存在を通知し，了承されている

テスト完了判断基準 (続き)

- <4>バグの修正品質
  - デグレードが発生した件数と，発生状況の分析結果により，未だ存在するデグレードバグを類推し，リスクが許容できる範囲であること。
- <5>信頼度成長曲線の収束度合い
  - 信頼度成長曲線が収束傾向にあること。
- <6>検出バグの発生傾向分析
  - 欠陥分析アクティビティの分析結果によりリスクを評価し，リスクが許容範囲であること。
- <7>開発プロセスへのフィードバック状況
  - 欠陥分析アクティビティの分析結果を開発プロセスにフィードバックしたものに，積み残しがないこと。
  - 実施すべき見直しが全て完了していること。

テスト完了の意志決定

- 導出した基準を全て満足するものが，完了判断基準ではない
  - メトリクスは，ある観点での側面を切り出して表したもの
  - それぞれのメトリクス単独では全体像を正しく見渡せるものではない
  - 様々な角度からの情報を集約し，総合して全体像を捉えることが，合理的なテスト完了判断につながる

単に数値と基準の差異を見るのではなく，その意味するところを，
開発チーム，テストチーム，品質保証チーム等の関係者で議論を尽くす

出来ていることと出来ていないことを明確化し，
「テストを継続しない」ことに対するリスクを評価

ソフトウェアのテストに明快な終わりはない

- テスト完了判断基準は，それ単独で機能するものではない。
- ソフトウェア開発のライフサイクル全体を通して品質保証計画を立て，その中の一要素として基準を設計することが必要。
- 基準を満たしていてもなお未知のバグが存在する可能性がある。万能な完了判断基準は存在しない。
- 導出した7つの基準は，テストフェーズのタイムリミット間近になってからデータをかき集め，後追いで基準に照らし合わせて判断に用いるものではない。

自らが置かれた状況で，日々きめ細かい軌道修正をし，マネージメントすることこそが，何よりも重要

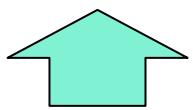
※前回発表分 ここまで

目指したいこと：「品質リスクの予測」

- ここで言う品質リスクとは：
  - 「そのソフトウェアに残存する欠陥の可能性」と定義

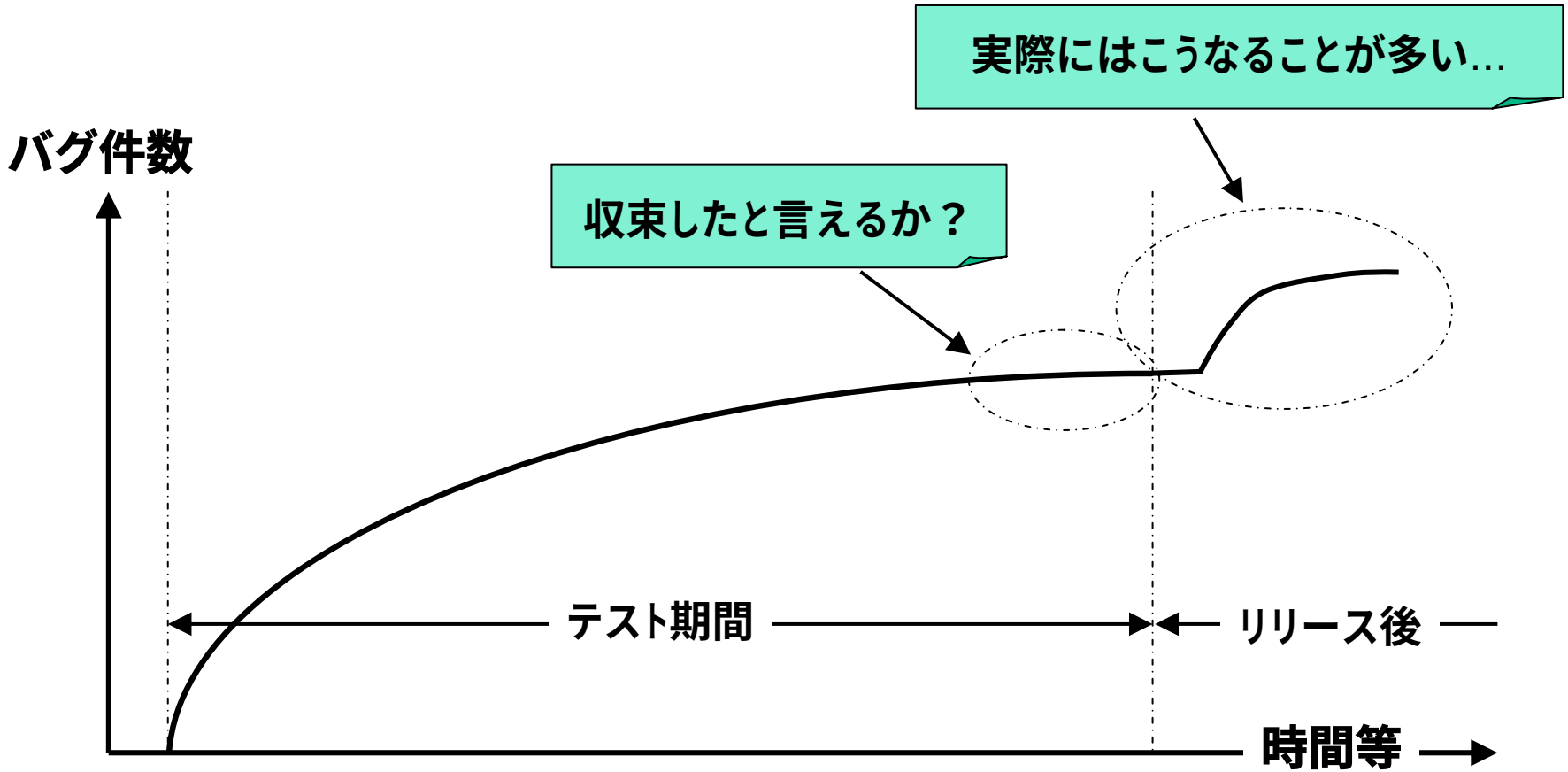
- 理想の姿

「作り込んだ欠陥の総数」 － 「検出・修正した欠陥の総数」 ＝ 0



これは果たして実現できるのだろうか？

信頼度成長曲線の悩み

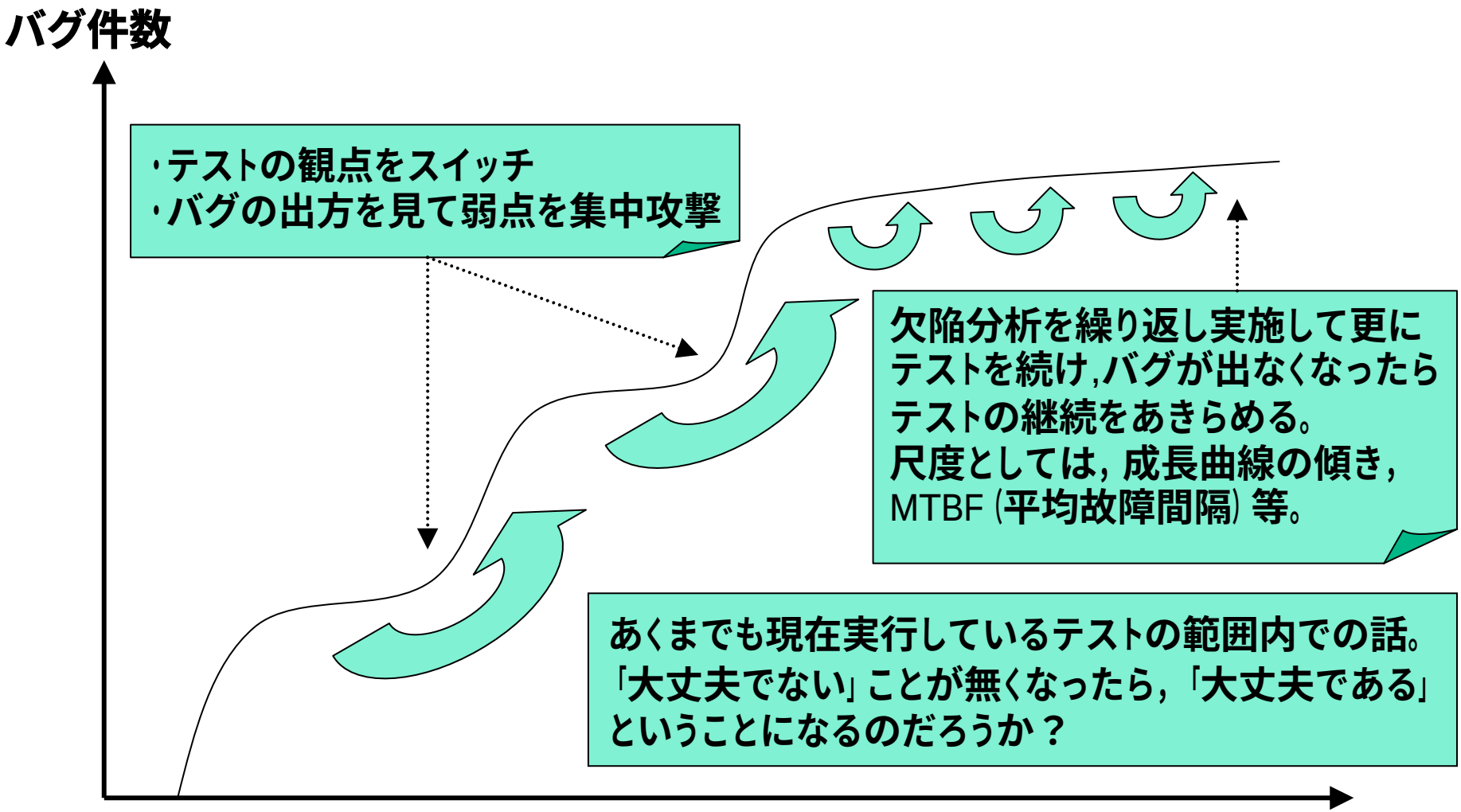


信頼度成長曲線は，そのソフトウェアをテストした**観点・範囲内**でのものに過ぎない

信頼度成長曲線は，作り込んだバグを取り除けたペースを表したものに過ぎず，ソフトウェアの信頼性を表したものではない

テストの"質"と一緒に考え合わさなければ間違った判断を下してしまう

信頼度成長曲線の実際の姿



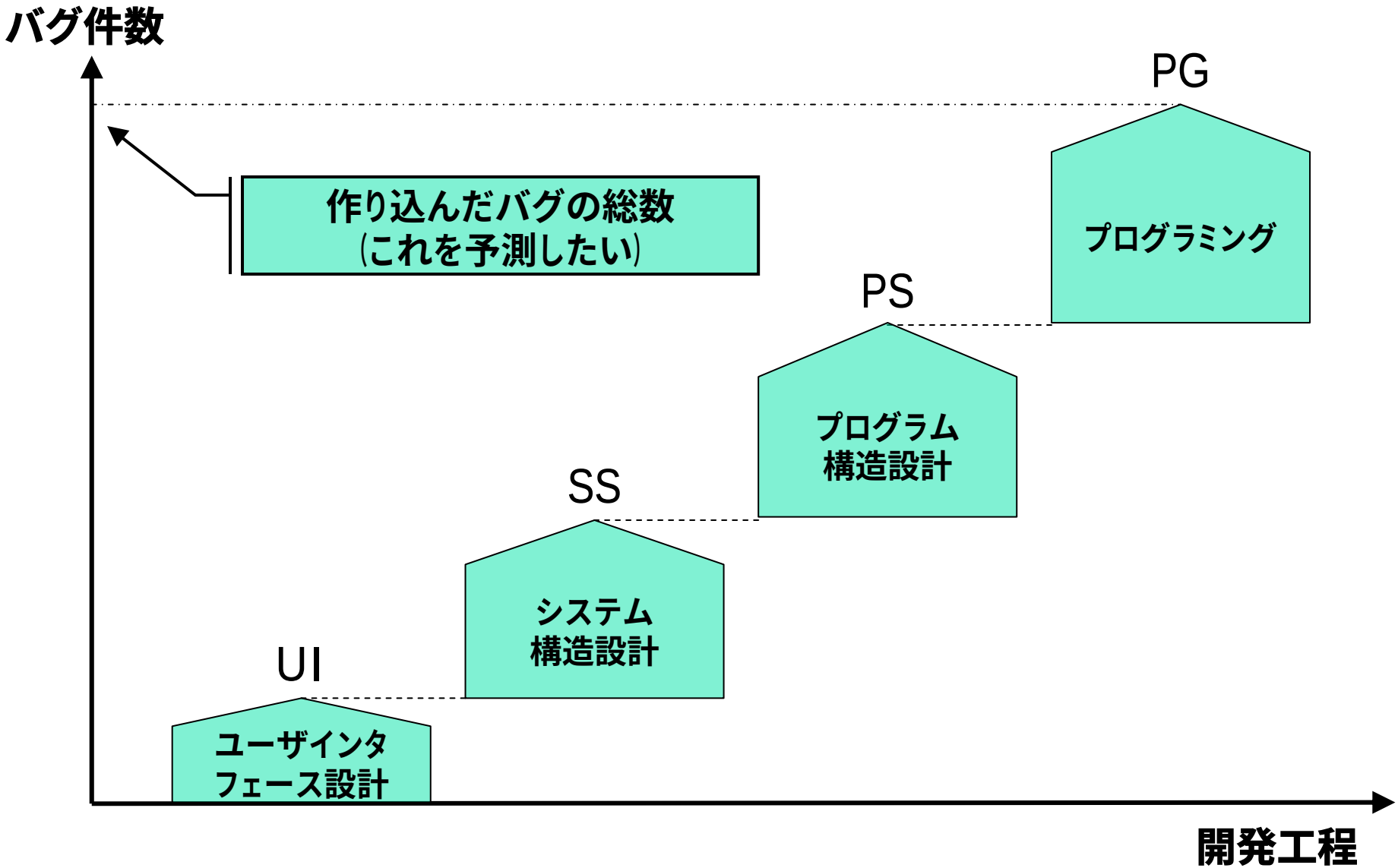
現在実行しているテストの"質"を把握していなければMTBF等を信用するのは難しい？
バグが見つからないのはテストが甘いからか？　品質が高いからか？

テストの質とは？

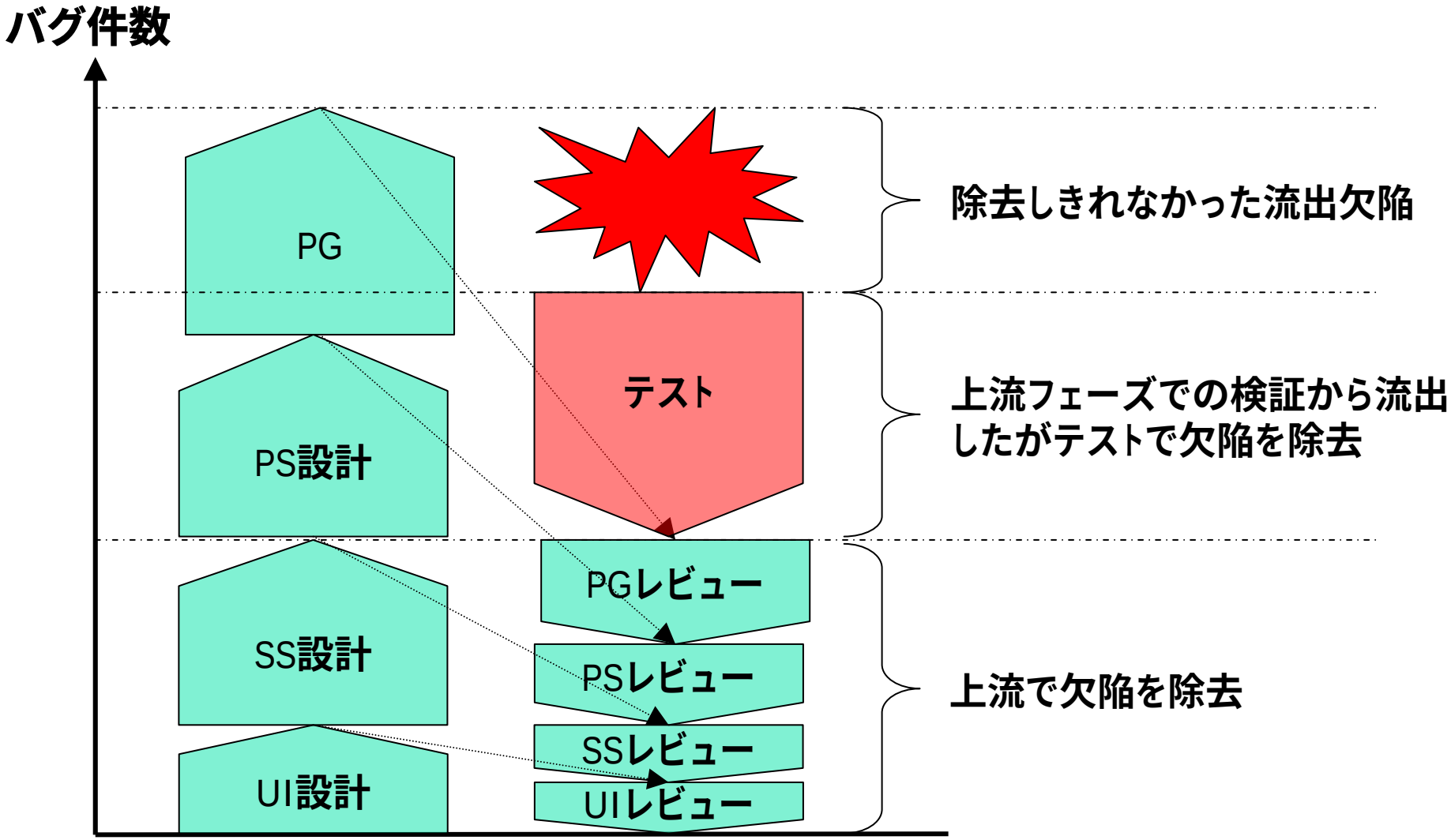
- 網羅度（カバレッジ）が100%？
  - ステートメント実行，分岐，分岐の組み合わせ(C0/C1/C2)，状態，構成
  - カバレッジは目標としては使えるが目的にはならない。
  - ソフトウェアの動作環境は多次元的であり，これらを全て検証することは実現不可能（無限大×無限大）
- ソフトウェアの全ての要件を動作させるテスト？
- ソフトウェアの規模に見合ったテスト項目数？
  （そもそもソフトウェアの規模とは一体何なのか？）
- バグを百発百中で抽出するテスト？
- テストの質評価の観点は，バグの抽出能力だけか？
  （他にもあるはず）

テストの質を **予め** 定量的に評価できる方法はあるだろうか？

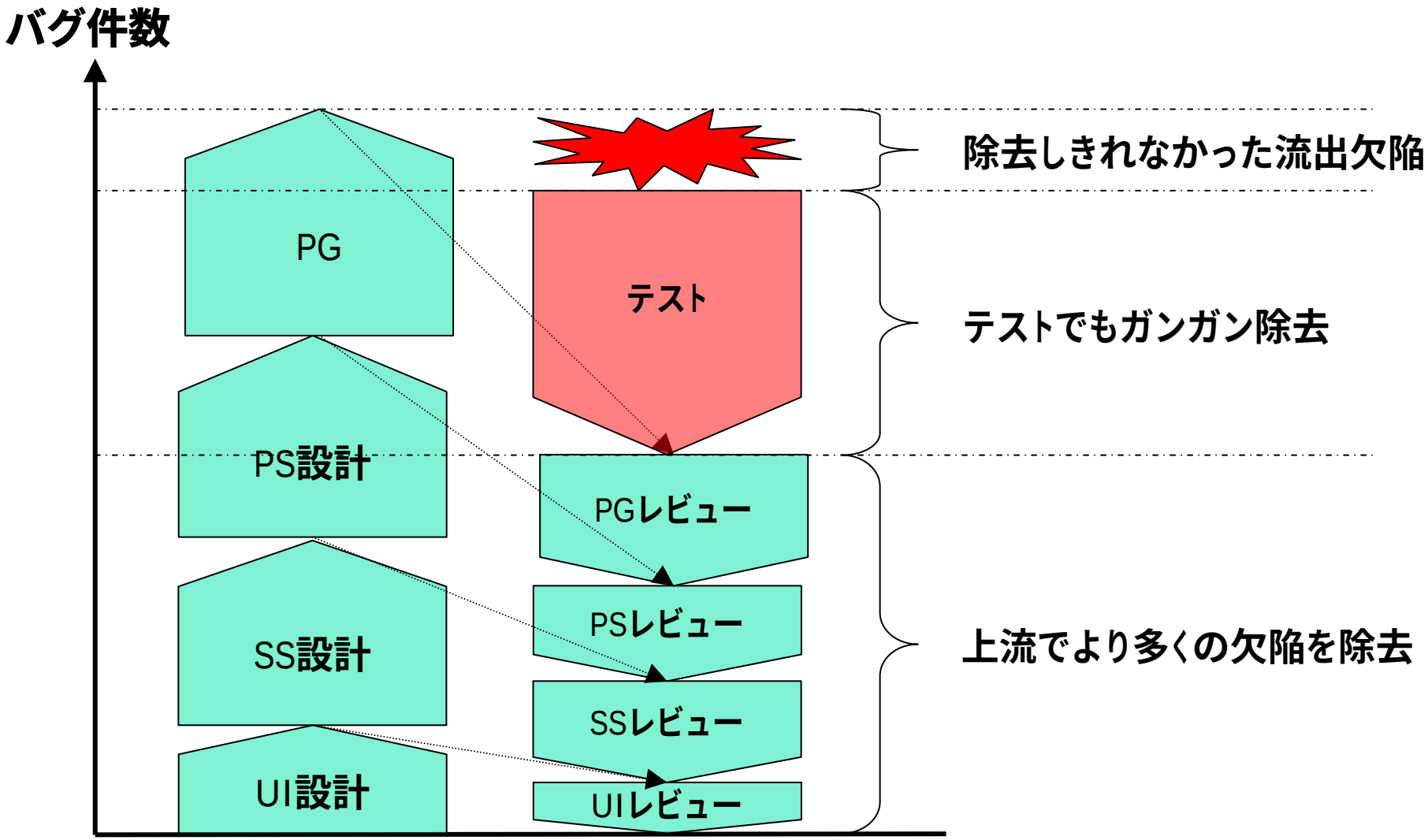
テストで抽出するバグはどこから来るのか？



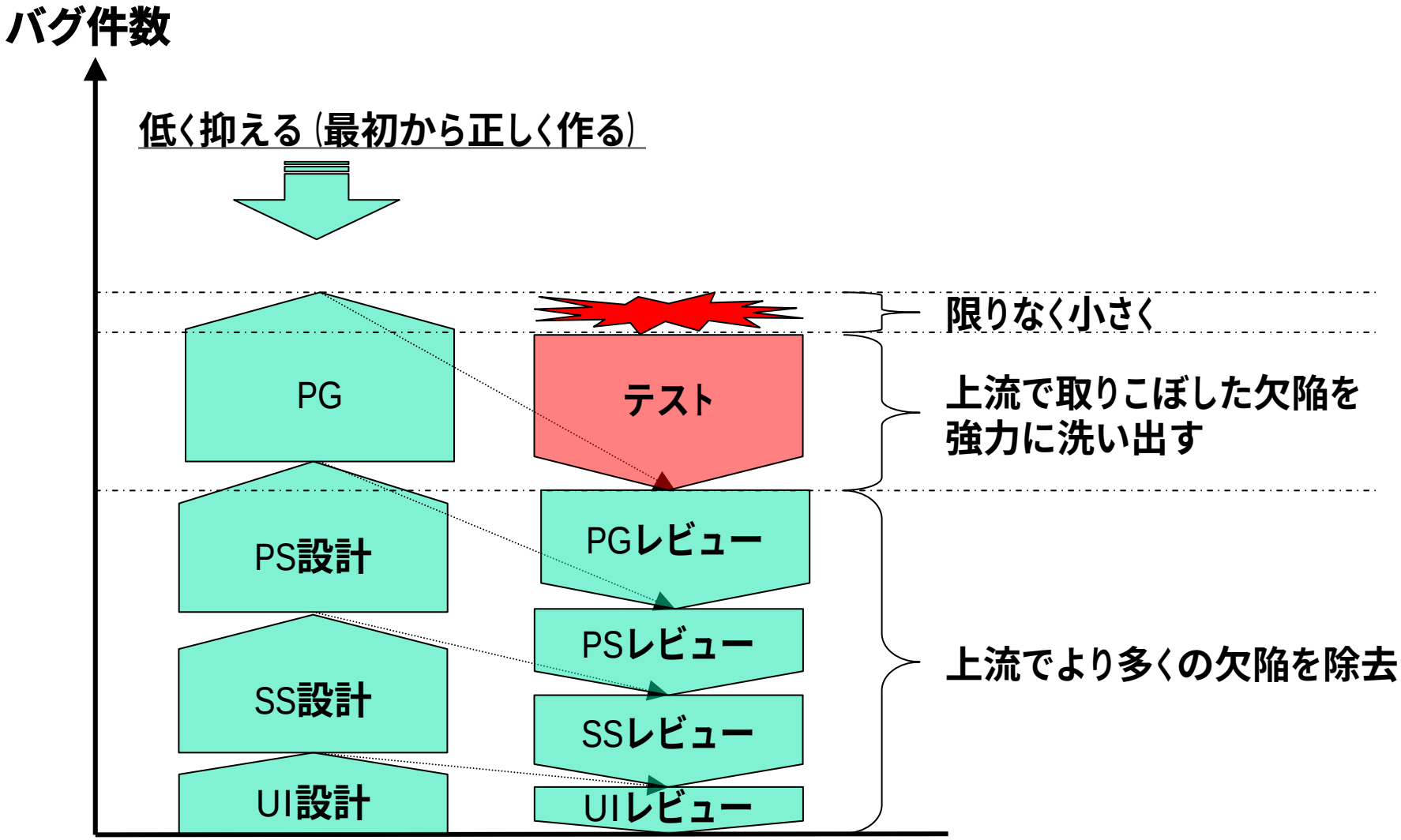
上流での除去，下流での除去



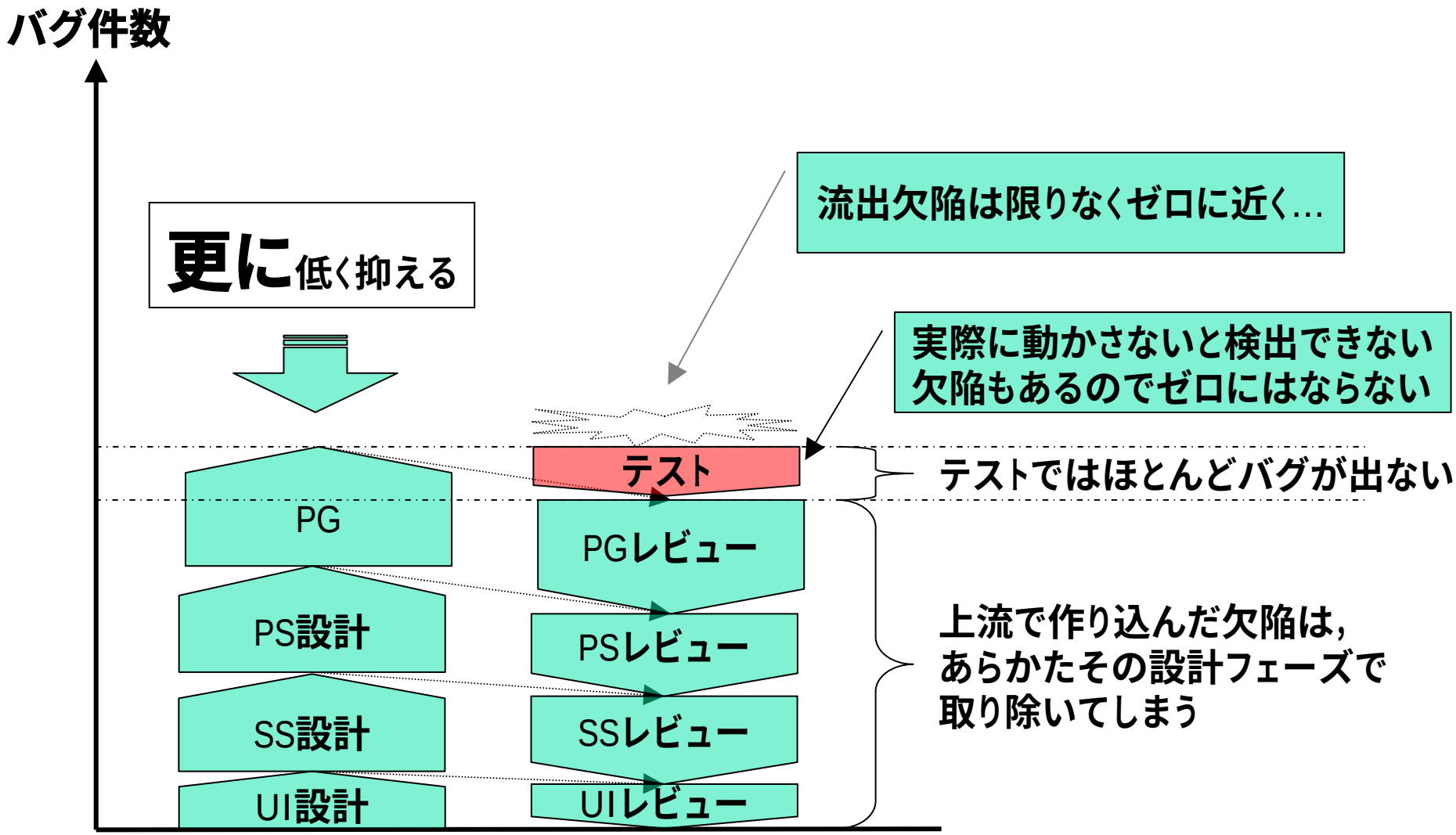
上流での除去，下流での除去(1st step)



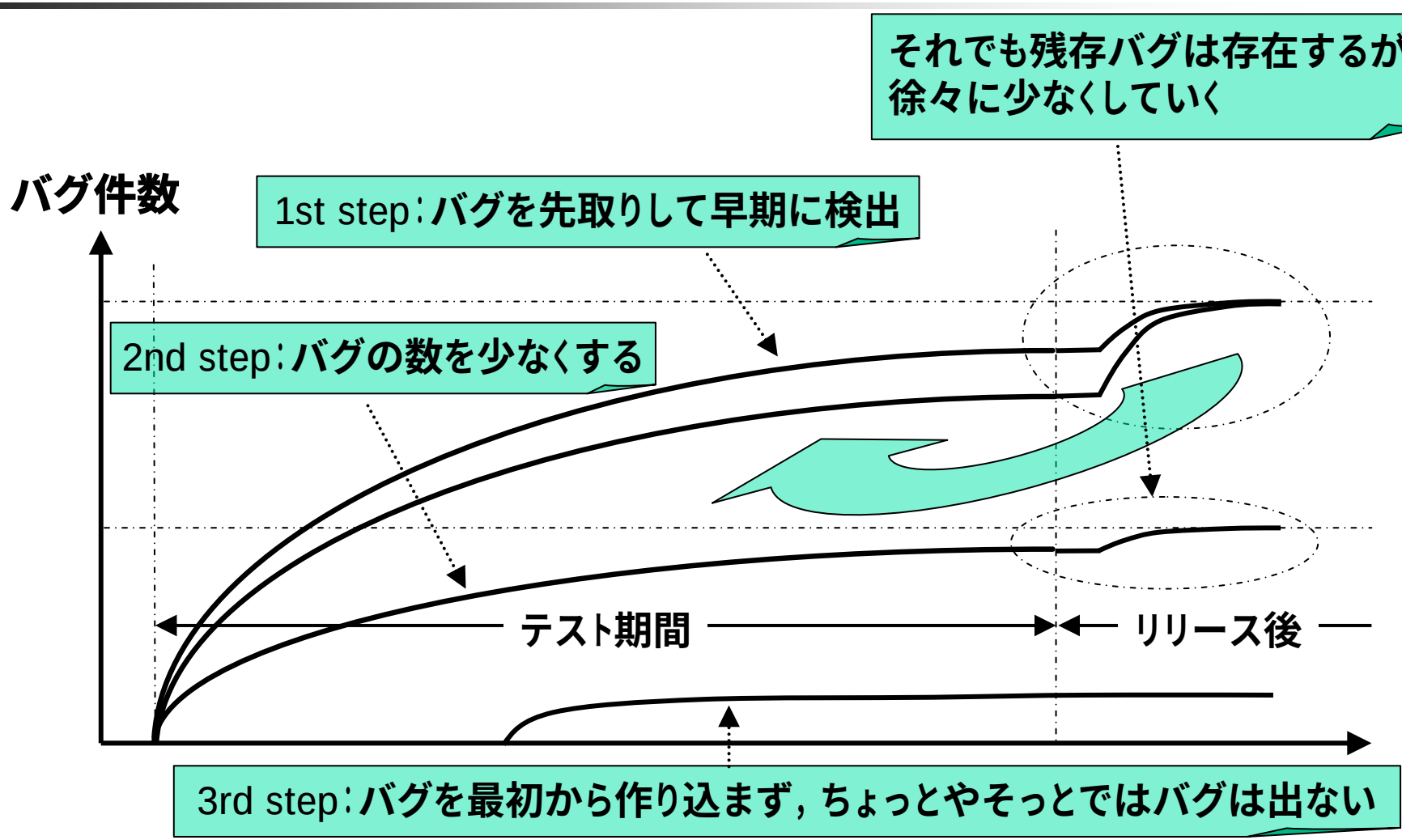
上流での除去，下流での除去(2nd step)



上流での除去，下流での除去(3rd step)



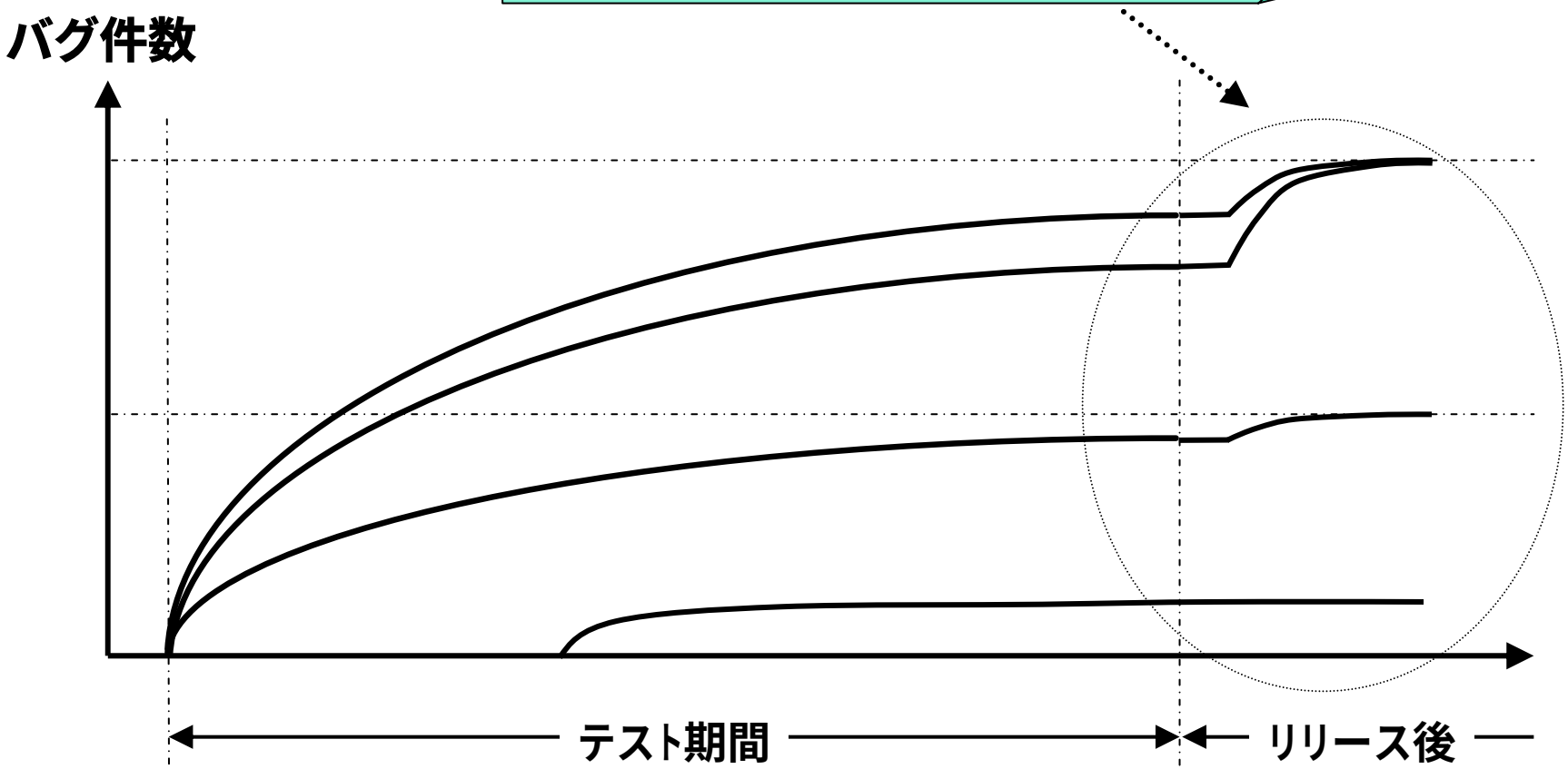
もう一回信頼度成長曲線



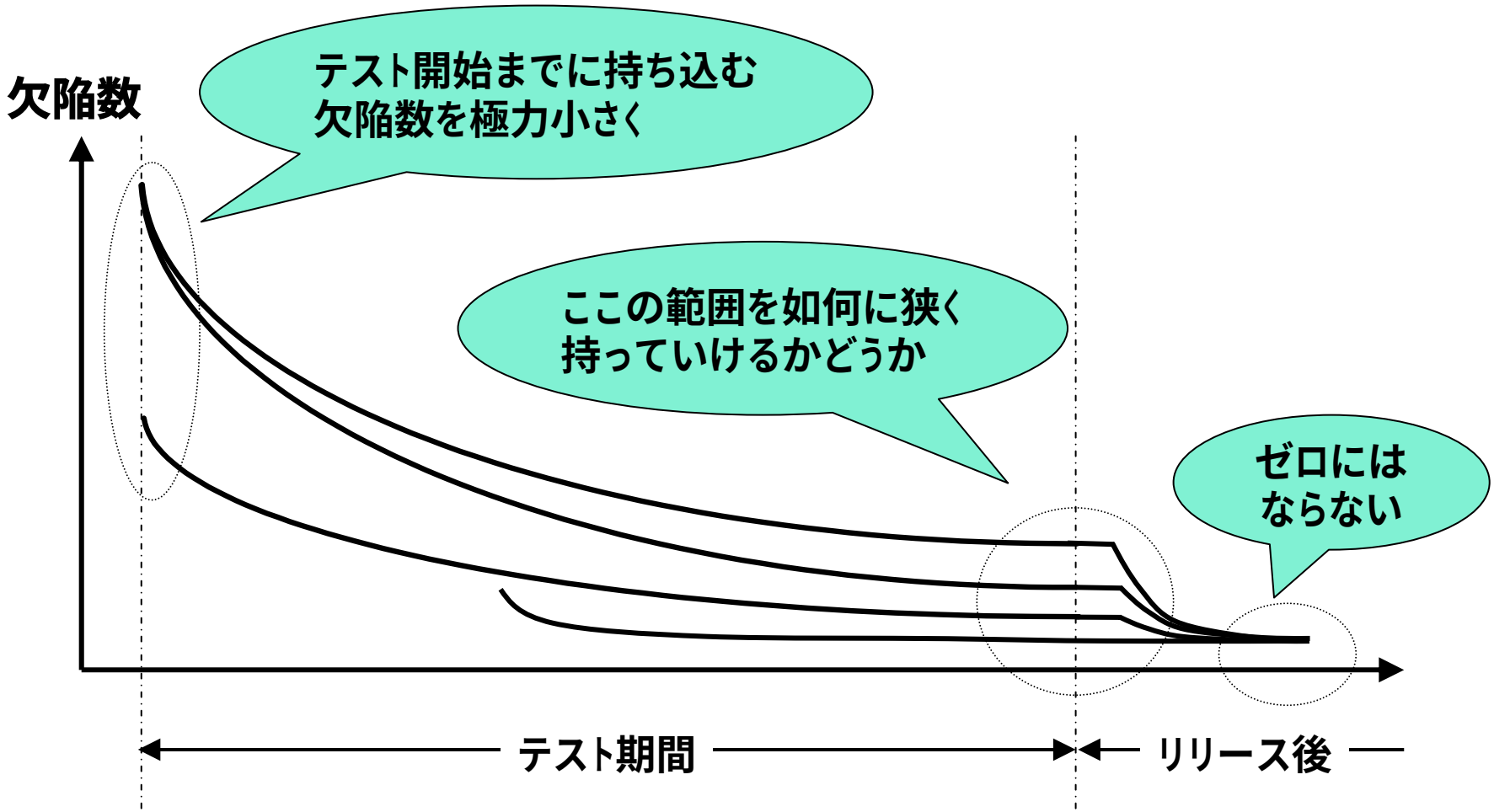
＜理想像＞　最初から正しく作り，バグは早期に叩き出す。
強力なテストを施し，乾いたタオルを更に絞りに絞ってバグを抽出する。
リリース後はその製品寿命が尽きるまで正常動作し続ける。

欠陥分析で考慮すべき点

出荷後のバグ発生状況も
開発プロジェクト全体の分析に組み込み，
次のプロジェクトにつなげる

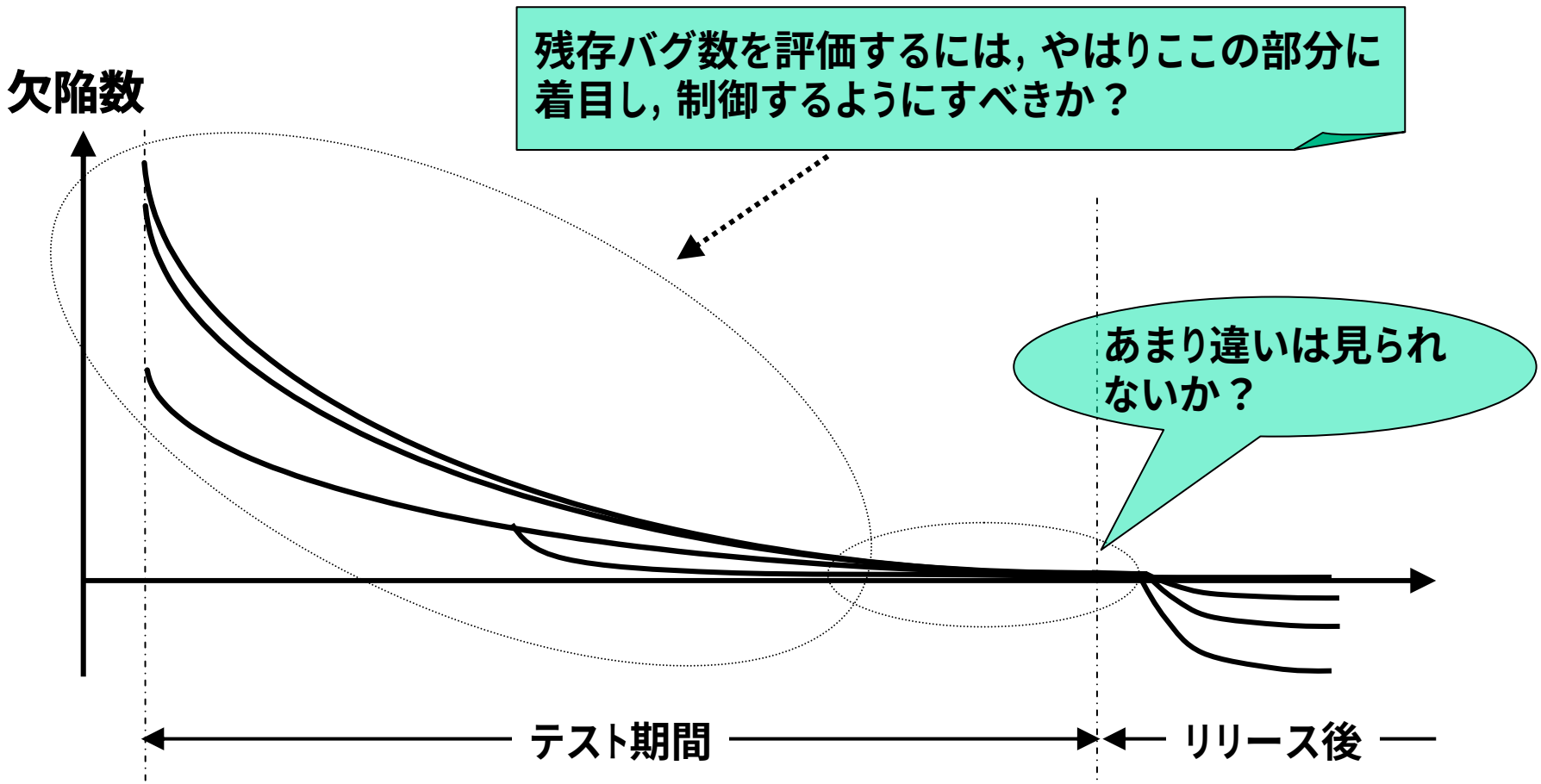


信頼度成長曲線を反転させると…



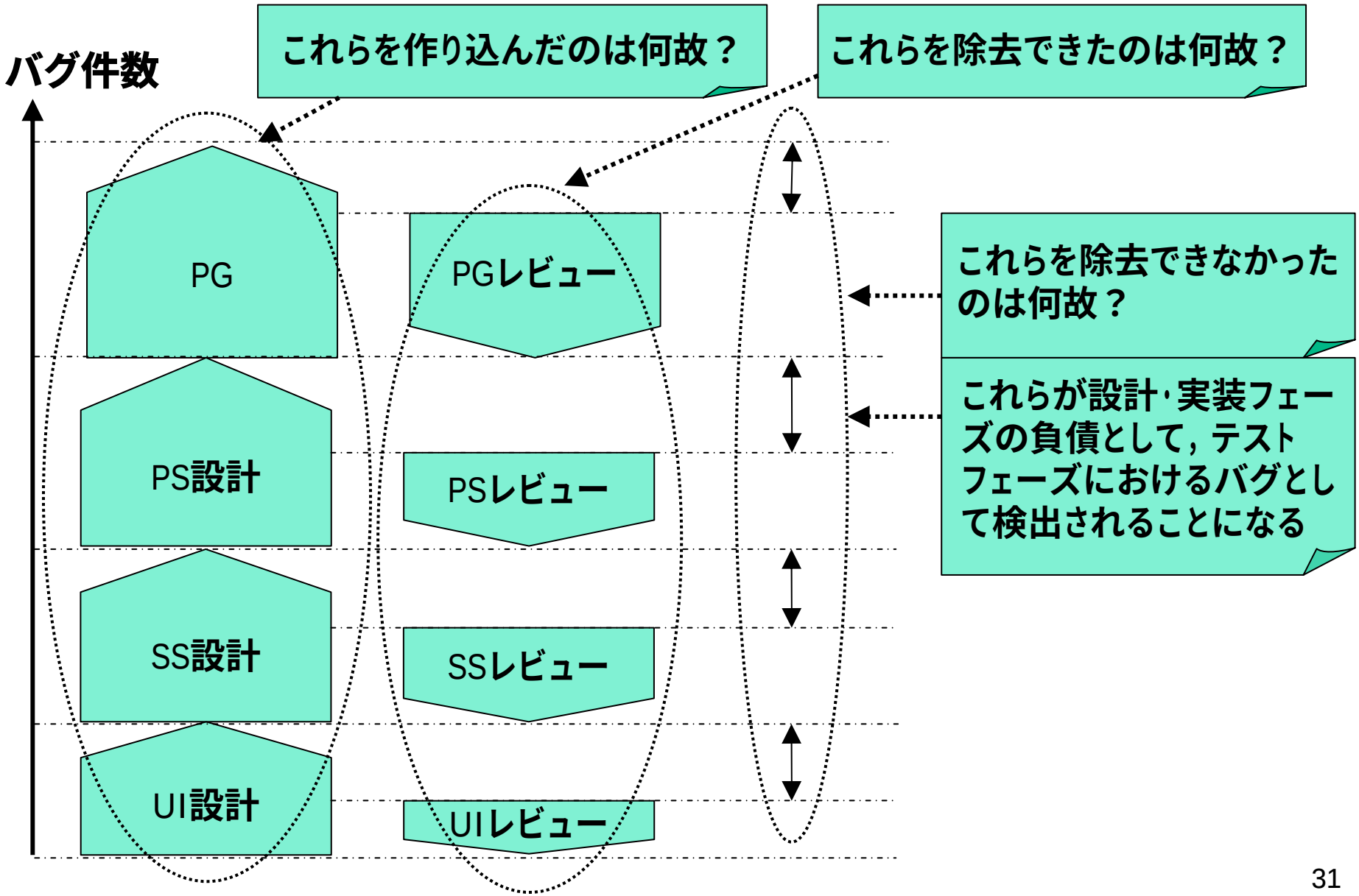
過去の実績データと比較して現在の状態を類推したいが…

しかし収束箇所を重ね合わせてみると…



過去の実績データと比較するのはテストの序盤から中盤にかけてか？

欠陥の作り込みパターンと除去パターン(その1)



欠陥の作り込みパターンと除去パターン(その2)

5W1H	欠陥除去編	欠陥作り込み編
What	どんな欠陥を検出したのか	どんな欠陥を作り込んだのか
When	何時検出したのか	何時作り込んだか
Where	どこのテストで検出したのか	どこに作り込んだか
Why	なぜ事前に検出できなかったのか	なぜ作り込んだのか
How	どのようにして検出したのか	どのようにして作り込んだのか
Who	誰が検出したのか	誰が作り込んだのか

これらを組み合わせて分析すると，ある一定のパターンが見えてくる

これら分析結果を蓄積し，欠陥除去とバグの作り込み防止のノウハウに昇華

欠陥の作り込みパターンと除去パターン(その3)

- 欠陥の作り込みと欠陥除去のパターンを見る
  - 何時作り込み，何時検出したのか？
    - バグ分析で，「何時検出したのか」はよく分析されている
    - テストフェーズでのバグはデータベース化されていることは多い。
    - しかし欠陥を「何時作り込んだのか」は，意外にデータが残っていないのではないか？

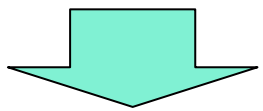
工程	UI	SS	PS	PG	PT	IT	ST
欠陥除去数	△件	△件	△件	△件	△件	△件	△件
欠陥作り込み数	△件	△件	△件	△件	△件	△件	△件

データを蓄積して欠陥作り込みと除去のパターンを見出し，見積もりに使う

仕様・設計変更，バグ修正に伴うデグレードも考慮に入れると，単純ではなさそう

欠陥の作り込みパターンと除去パターン(その4)

- 「バグやテストのパターンから弱点と思われる箇所に対処する」ことはどこでも実践されていること
  - 同じようなミスは，まだ他にもある
  - 同じようなミスで，他でも欠陥を作り込んでいる
  - 欠陥は大体，ある箇所にまとまって潜んでいる



- 後で対処できる手段は限られている
  - 設計の見直し
  - レビューのやり直し，追加
  - テストの追加

色々な手段で検証･分析しても，
後で出来る対処方法は意外に少ない

対処した内容と結果も，欠陥分析の対象に入れて全体を見る

これらのノウハウに通じた，Quality Profilerが必要なのかもしれない。

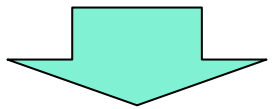
「品質リスクの予測」への展望

「作り込んだ欠陥の総数」 － 「検出・修正した欠陥の総数」 ＝ 0

- これから実施するテストの質を予め評価するには？
  - テスト戦略・テスト計画・テスト設計のノウハウを総動員する必要がありそう
- 欠陥の作り込みパターンと除去パターンを分析する仕組みは？
  - 除去パターンはテストだけでなく，上流でのレビューも含める
  - 更にこれらを使ってパターンを見積もる（予想する）方法は？
- リリース後に発露したバグを分析の対象に入れる仕組みは？
  - リリース後の欠陥はトップシークレットになって見えにくくなることもある

欠陥のパターンを見出し，予測すること

- 物作りは，未だこの世に無いものを新しく生み出すもの
  - 物作りそのものは非常に難しいものであるが，欠陥の作り込みパターンと除去パターンを予想することは，更に困難なものに思える



- しかし予測するということは，
  - 過去の実績を積み重ね，現在の状態を類推し，
  - それに基づいて未来を予想し，予測し，
  - 良い方向へ向けるように先手を打つこと。
  - 良くない兆候が見受けられた時には分析して原因を特定し，軌道修正をかけること。
- 予測し，計画して備えることで，本来なすべきことがきっと見えてくるはず。
- 決してあきらめず，注意深く，根気よく取り組みを継続することが肝要ではないか？

完全なソフトウェアウェアは存在し得るのか？

- 多分，答えはYES (私見)
  - しかし，"完全である"ことを証明できる手段が無い
  - また，　"完全である"状態を，はたして定義できるか？
- 完全さを追い求めるあまりに，やってはいけないこと
  - ノウハウを様々な設計規約等の形で蓄積し，共有することはもちろん良いこと
  - しかしエンジニアを「規則」でがんじがらめにするようなシステムになってしまわないようにしたい
  - ソフトウェアは人が作る。ソフトウェア開発は人の知的生産活動そのもの。そういう意味で属人性は避けて通れないものと，受け入れなければならないのかもしれない。
  - 規約やシステムは，エンジニアの知的生産活動を底上げし，イマジネーションを掻き立てるものでありたい。

S-open メトリクスSIGでの取り組み

- S-open (ソフトウェア技術者ネットワーク) は，ソフトウェアエンジニアのための開かれたコミュニティ
  - ここではエンジニア自身が主役
  - 誰に言われてでなく，自分の興味と信念で活動するフィールド
- メトリクスSIG (Special Interest Group) は2003年10月から活動
  - 既に丸3年以上が経過(2007年1月現在)
  - SIGメンバーは24名，意見交換のみのコミュニティは61名
- メトリクスを通じてソフトウェア開発に切り込む
  - 物事を定量的な視点で考えると，思わぬところで検討対象の根本を考えるきっかけになる
  - 何でもかんでも定量化するのが目的ではない。
    物事を客観的に把握し，関係者で意思疎通し，次の手を打てるようにするのが本来の目的と思う。
- 集まれ仲間！ 共に悩んで研鑽し，自分たちの仕事を素敵に変えていこうじゃないか。

S+open http://www.s-open.net/