IT Holdings

ソフトウエア開発における
定量品質管理の実践方法

平成24年12月21日

ITホールディングス株式会社

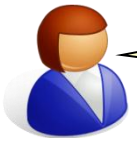
第一部：ソフトウエア開発における定量品質管理とは

内容

1－1　はじめに

IT Holdings

ソフトウエアの定量品質管理は何のために実施するのでしょうか？



・上司やお客様に対する品質報告資料を作成するため
・プロジェクト報告書に品質データを記載しなければならないから
・PMや上長に指示されるし、社内ルールで決まっているから

実施するプロジェクトの本音は？

・できればやりたくない。その時間やコストを開発活動に使いたい
・品質状況はデータなど無くても過去の経験や勘で分かるし
・見積にコストを上乗せできない。お客様にコストの説明ができない

きちんとやっているつもりが形式的になっている場合も ・・・

レビュー時間や不具合数を計測し、基準値と比較することによって品質が確保できているかどうかを判定する活動でしょ？

1－1 目的とねらい

IT Holdings

昨今のソフトウエア開発は短納期で複雑、いろいろな技術を組合せ、メンバーのスキルもばらばらです。

経験や勘では品質悪化を見逃したり、気付くのが遅くなってしまうこともあります。

本来の目的は、品質問題を早期発見し、余裕を持って手が打てるようにすることですが、多くのプロジェクトが定量データを上手く活用できていません。

ソフトウエアの定量品質管理の本質を理解し、プロジェクト活動の一部として定着させていきましょう。

しかし、定量品質管理というと ・・・

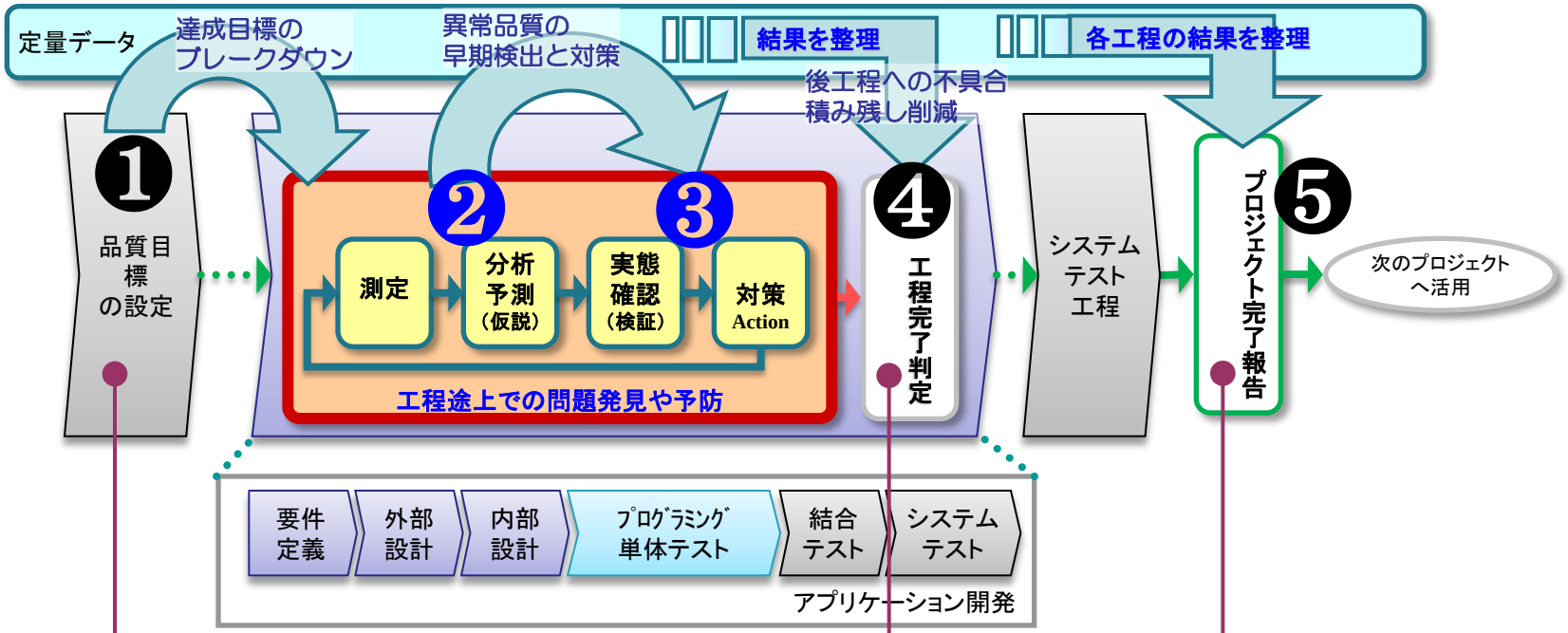
品質目標を設定し、品質計画を立てる ▶ 品質指標を設定し、必要なデータを計測する ▶ 指標値を基準値と比較し、品質状況を見極める ▶ 次のプロジェクトのためにデータを提出する

という4つのプロセスを強調しがちで、現場には分かりにくい

肝心の「データを開発の途上で活用する」ということが抜けている！

1－1　対象範囲

ここでは、「データを開発の途上で活用する」という定量品質管理の本質にスポットを当てます（下図の②と③）。



品質目標の設定

②③を着実に行うための目標設定でなければ意味がない

→形式的な目標設定からの脱却

工程完了判定

②③を実施した結果としての指標で判定することが重要

→判定だけのデータ集めの見直し

プロジェクト完了報告

②③でデータ活用できてこそ、データを残す意義が理解できる

→自分が利用するために行う

1－2　ソフトウェア品質管理を考える上での前提（１）

IT Holdings

ソフトウェアにバグはつきもの、でもどのくらいあるかが分からない

・人が作るものには必ずバグが混入してしまいます
　→ 問題は、どれくらい混入しているのか計測する手段がないこと
・なので、「バグの検出が少ないから品質が良い」と考えるのは危険です
　→ むしろ「検出できずに残っているバグがあるのだ」と考えるべき
・プログラムのロジックとして正常でも、仕様に合わなければバグになります
　→ この種のバグは、設計段階で発見できなければ結合テスト以降まで残存しがち

ではどうするのか？

バグが出なくなるまでテストしまくる	✕	非効率すぎる！
過去の定量データを使い、混入バグ数を推測する	○	

一般的な混入量が分かれば、自分のソフトウエアの品質が良いのか悪いのか、どのくらいのバグを潰せば安心できそうか、といった見通しが立てられます。

1－2　ソフトウェア品質管理を考える上での前提（２）

IT Holdings

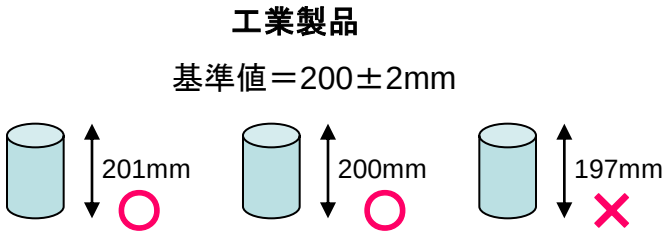
ソフトウェアの定量品質指標のほとんどは代替指標でしかない

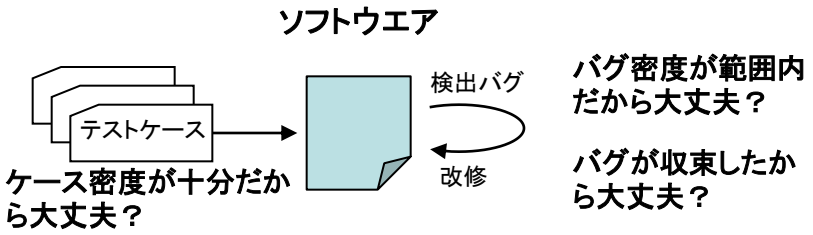
・言い換えれば、ソフトウエアの品質は直接計測できない、ということです

→ 品質を直接計測できないために代替指標を使うのですが、

・レビュー時間が長いからといって、レビュー品質が良いとは限らない（レビュー密度）
・バグを大量に検出して潰したからもう大丈夫だとは言えない（バグ密度）

といった課題があります





代替指標では品質について断定や保証ができないため、

・標準よりレビュー密度が低ければ、レビュー不足が懸念される
・バグ密度が標準より高ければ、何か問題があることが疑われる

即ち、品質そのものではなく、問題を早期に察知するための手段と考えるべきです。

1－2　ソフトウェア品質管理を考える上での前提（参考）

IT Holdings

このデータは、あるプロジェクトで混入されたバグと、その除去率を計測したものです

混入工程	混入されたバグ数（Kstep当り：C言語）	引渡しまでに除去できなかったバグ数	除去率
要求定義	7.8	1.8	77%
設計	9.7	1.5	85%
プログラミング	13.7	0.7	95%

※ Capers Jones 著「ソフトウエア品質のガイドライン」より抜粋

これは結果論としてのデータですが、

類似開発において、このような過去データがあれば目安として大いに役立ちます

1－3　定量品質管理概要(１)

IT Holdings

ソフトウエア品質に関する次の3つの報告のうち、どれが適切だと思いますか？

①不具合は想定より少な目で、出ているバグにも特に重大なものもありません。
　対応もほぼ出来ているので品質に問題はありません。

②幾つかのサブシステムでバグ密度が基準値を若干上回っていますが、テストケース密度を高くしているためなので問題はありません。
その他のサブシステムは全て基準値内に収まっており、品質に問題はありません。

③サブシステム毎にバグ密度の成長トレンドを監視していますが、テストケース消化率50%の時点でサブシステムAだけが標準値の上限に近づいていたので点検を行いました。
その結果、経験の浅いSEによるイージーバグが多いことが分かったので対応し、同じSEが担当した他のサブシステムについても点検を行いました。
それ以外はトレンド上も異常はなく、今のところ品質は安定していると評価しています。

1－3　定量品質管理概要(２)

IT Holdings

前述の3つの報告について解説します。まず①についてです。

①不具合は想定より少な目で、出ているバグにも特に重大なものもありません。
対応もほぼ出来ているので品質に問題はありません。

感覚的に判断しているだけで、何の裏付けもありません。
重大なバグが無ければよい、ちゃんと潰せば品質に問題ない、これではいけません。

②
その他のサブシステムは全て基準値内に収まっており、品質に問題はありません。

③サブシステム毎にバグ密度の成長トレンドを監視していますが、テストケース消化率50%の時点でサブシステムAだけが標準値の上限に近づいていたので点検を行いました。
その結果、経験の浅いSEによるイージーバグが多いことが分かったので対応し、同じSEが担当した他のサブシステムについても点検を行いました。
それ以外はトレンド上も異常はなく、今のところ品質は安定していると評価しています。

1－3　定量品質管理概要(３)

IT Holdings

②はどうでしょうか？

①不具合は想定より少な目で、出ているバグにも特に重大なものもありません。
　対応もほぼ出来ているので品質に問題はありません。

②幾つかのサブシステムでバグ密度が基準値を若干上回っていますが、テストケース密度を高くしているためなので問題はありません。
その他のサブシステムは全て基準値内に収まっており、品質に問題はありません。

定量指標を基準値と照らし合わせて判断している点は一歩前進です。

前提で述べた通り、バグ密度は代替指標であり、これが基準内に納まったからといって品質に問題無いと結論付けることはできません。
ましてや「テストケースを増やしたらバグも多く出た」という事実を「問題無い」と評価してよいのでしょうか。むしろ潜在バグがもっとありそうだと疑うべきです。

1－3　定量品質管理概要(４)

IT Holdings

では、③はどうでしょうか？

①不具合は想定より少な目で、出ているバグにも特に重大なものもありません。
　対応もほぼ出来ているので品質に問題はありません。

②

定量指標を問題発見の手段に使い、品質について現物確認している点がポイントです。
代替指標は品質を直接示しませんが、問題があればそのトレンドに表れます。②のように、最終計測値と基準値とを結果論で評価しても意味がありません。
また、定量データに不穏な動きがあれば、何が原因なのか、成果物やプロセスに異常は無いのかを掘り下げ、最終的には現物を見て確認することが重要です。

③サブシステム毎にバグ密度の成長トレンドを監視していますが、テストケース消化率50%の時点でサブシステムAだけが標準値の上限に近づいていたので点検を行いました。
その結果、経験の浅いSEによるイージーバグが多いことが分かったので対応し、同じSEが担当した他のサブシステムについても点検を行いました。
それ以外はトレンド上も異常はなく、今のところ品質は安定していると評価しています。

1－3　定量品質管理概要（まとめ）

IT Holdings

（１）ソフトウエアの定量品質指標は代替指標であり、その値で品質を評価するのではなく、品質問題が発生していないか開発の途上で察知するために活用します。

その方法には、トレンドを監視する、過去のデータと比較する、サブシステム間など同じプロジェクトで相互評価する、などがあります。

（２）過去の同類プロジェクトの品質データを目安として活用することが重要です。

即ち、混入されている潜在バグ数の推測や、トレンドが異常な状態になっていないかどうかの評価に活用します。

（３）品質指標に少しでも異常があれば、その箇所を特定し、原因を掘り下げます。問題があれば対処するだけでなく、同様の問題が他に波及していないかも確かめます。

指標値の異常が常に品質問題に起因しているとは限りません。難易度の高いテストが初期段階に集中していれば、バグが最初から多く検出されることもあります。

原因をどう掘り下げればよいのかは、第二部で説明します。

定量品質データは、第三者への品質報告のために実施するものではありません
自分達の開発を健全に進めるために実施するものです

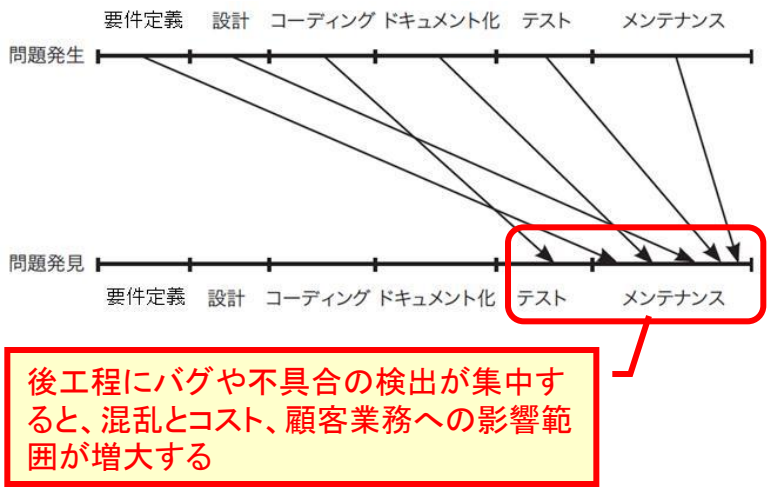
1－4　ソフトウェア品質管理実施上の原則(１)

IT Holdings

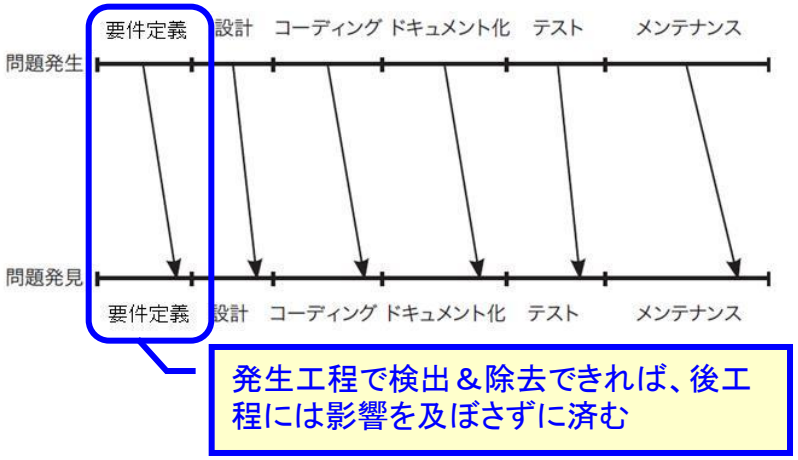
バグは作り込んだ工程で検出し、除去する

各工程で作り込んだバグは、原則としてその工程内で除去することを目指します

→ 後工程になるほどバグの検出と除去にかかる手間やコストが増加し、改修を繰り返すことで新たな品質劣化を招いてしまう危険性が高まるからです



後工程にバグや不具合の検出が集中すると、混乱とコスト、顧客業務への影響範囲が増大する



発生工程で検出＆除去できれば、後工程には影響を及ぼさずに済む

JaSST '08 TokyoでのCapers Jones氏基調講演資料から

特に上流工程(要件定義や基本設計)で混入されたバグは、下流に進むにつれて波及範囲が広がって除去するのが難しくなります。

1－4　ソフトウェア品質管理実施上の原則(ご参考)

IT Holdings

バグをその工程内で除去するには、混入しているバグの量を知る必要があります

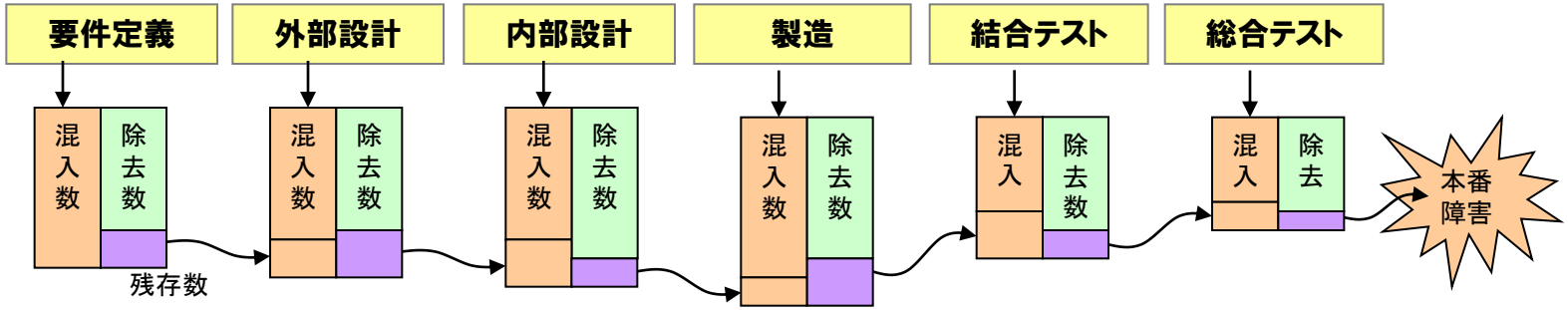
→ **過去の類似プロジェクトにおけるバグ密度が分かれば、対象となる成果物の規模を掛け合わせることで算出できます。**

■ 20KStep規模の成果物に対し、　　※ 1-2節の表より

設計工程:	20 × 9.7 = 194	→ レビューで検出すべき指摘数
プログラミング:	20 × 13.7 = 274	→ 単体テストで検出すべきバグ数

実際には対象プロジェクトの特性を考慮する必要がありますが、推測値に過ぎないので、厳密に算出することに注力しすぎないようにしましょう。

→ **混入バグ数の推測値から工程毎のバグの残存数を管理し、工程審査などで活用します**



完全除去は困難なので、残存バグをコントロールすること、前工程から引き継がれたバグが加わっていることを常に認識することが重要です。

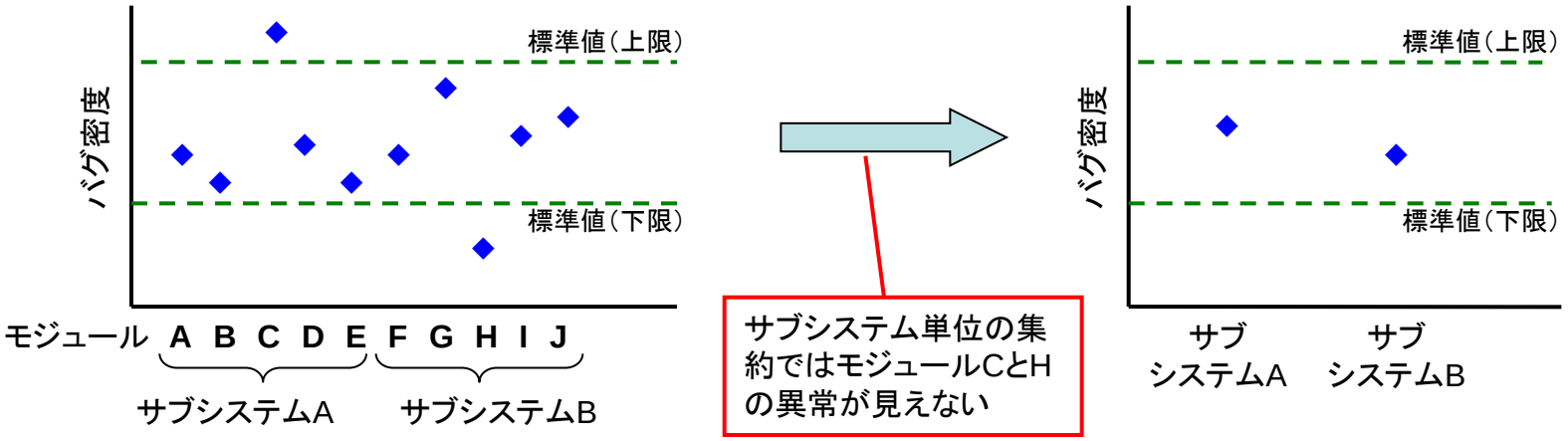
1－4　ソフトウェア品質管理実施上の原則(２)

IT Holdings

定量品質指標の監視は、その粒度を考慮する

品質データの異常を的確に把握するには、データを集約する粒度が重要です

→ 粒度が大きすぎると異常値が平準化されてしまい、細かすぎると監視しきれません



また、プログラム、モジュール、サブシステムという階層だけでなく、機能別、チーム別など色々なカテゴリーで見ることが有効です。

→ 層別については第二部で説明します

1－4　ソフトウェア品質管理実施上の原則（３）

IT Holdings

収集データの基準を統一しておく

せっかくデータを収集しても、その基準がバラバラでは何も見えてきません

→ **例えばレビュー指摘密度の計測において、**

チームA：「てにをは」や表現の不備までを指摘数に数えている
チームB：設計不備や検討漏れなどの本質部分のみを指摘数として数えている
チームC：同時に複数の指摘が為された場合は別の指摘として分離して数えている

これではチーム毎のレビュー指摘数を比較しても意味がありません

きちんと傾向が見えなければモチベーションも下がり、計測が更にいい加減になるという悪循環に陥ってしまいます。

統一しておくべき項目には次のようなものがあります

- ステップ数のカウント方法（空行の扱い、セミコロン単位、言語の違いなど）
- ドキュメント量（頁数／行数／バイト数など）
- テストケース数のカウント方法
- レビュー時間：実施時間に人数分を掛けるか、工数にするか

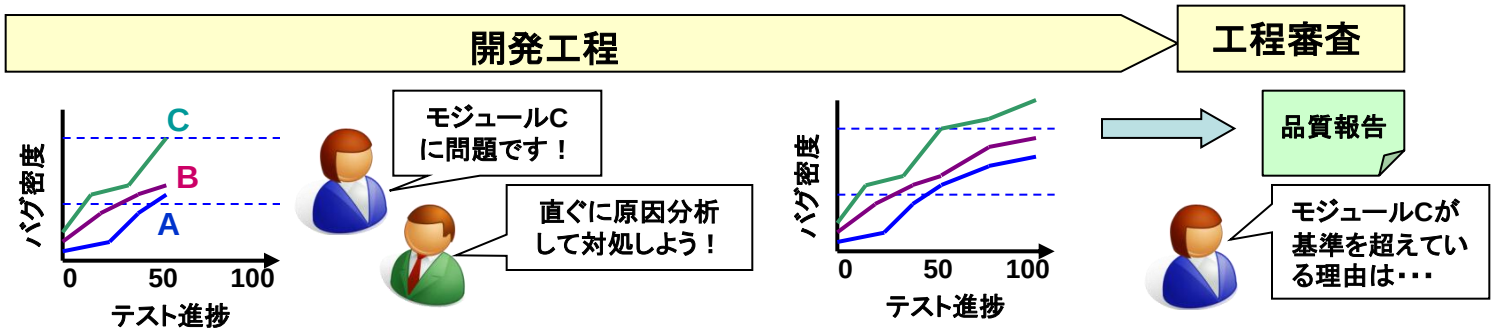
1－4　ソフトウェア品質管理実施上の原則（４）

IT Holdings

定量データは開発途上で活用することに意味がある

報告だけのために集めたデータでは説得力がありません

→ 開発途上で活用した結果としての品質データを報告することに本当の意味があります。



テストが全て終了してから、「データ集計したらバグ密度が標準値を大幅に超えていた」というのでは手遅れです。

せっかく集まったデータを、途上管理で是非活かしてください！

１－５　第一部のまとめ

IT Holdings

（１）ソフトウエア品質管理を実践する上での前提

→ ソフト開発にバグはつきもの。ただ、どのくらい作り込んだかを計測する手段がない

→ 定量品質指標のほとんどは代替指標。品質が直接計測できるわけではない

（２）定量指標を使って品質の悪そうな箇所に目星を付けることが重要

→ 過去の類似プロジェクトのデータやトレンドと比較して異常がないか点検する

（３）ソフトウエア品質管理における原則

→ バグはできるだけ作り込んだ工程内で除去する（下流に行くほど除去が大変になる）

→ 定量データによる監視は、その粒度を考慮する

→ データの収集基準を事前に決めておく

→ 定量データは工程の途上で利用することに意味がある

上記に加え、「結果を現場にフィードバックする」ことも重要です。データを入力する人達に、どう活用されているかを知らせることでモチベーションが向上するからです。
また、後続プロジェクトのためにデータをきちんと蓄積することも重要なポイントです。

第二部：ソフトウエア定量品質管理の実践

内容

2－1　定量品質管理を実践する前に

IT Holdings

そもそも定量品質管理が何の助けになるのか？（第一部のおさらい）

・品質の悪そうな箇所を推定し、早い段階で問題を発見して手が打てる
・品質を悪化させている根本原因の発見プロセスを効率化できる
・今後の品質の見通しが立てられ、何を優先すべきか判断材料が得られる

では、どんなデータを集め、何を指標にすればいいの？

定量品質指標には、基本指標と導出指標とがあります

基本指標：直接計測して得られるデータで、それ自体では指標として有効でないものもあります
導出指標：基本指標から算出されるデータで、情報をより有効に得るために工夫されたものです

（例）バグ密度

- バグ数は、「テストにより発見されたバグの数」という基本指標です
  →これ自体でもバグが多いか少ないかという指標にはなります
- でも、過去の事例や他のチームの状況と比べて評価するには不十分です
  →バグの量は規模や複雑に影響されるからです
- そこで「規模が2倍あればバグも2倍になる」という仮説の基に導出指標を考えます
  →バグ密度 ＝ バグ数 ／ プログラムの規模
- 「複雑ならバグも多くなる」という仮説を立てば、次のような導出指標も考えられます
  →バグ密度 ＝ バグ数 ／ プログラムの複雑度

2－1　定量品質管理を実践する前に（主な指標）

IT Holdings

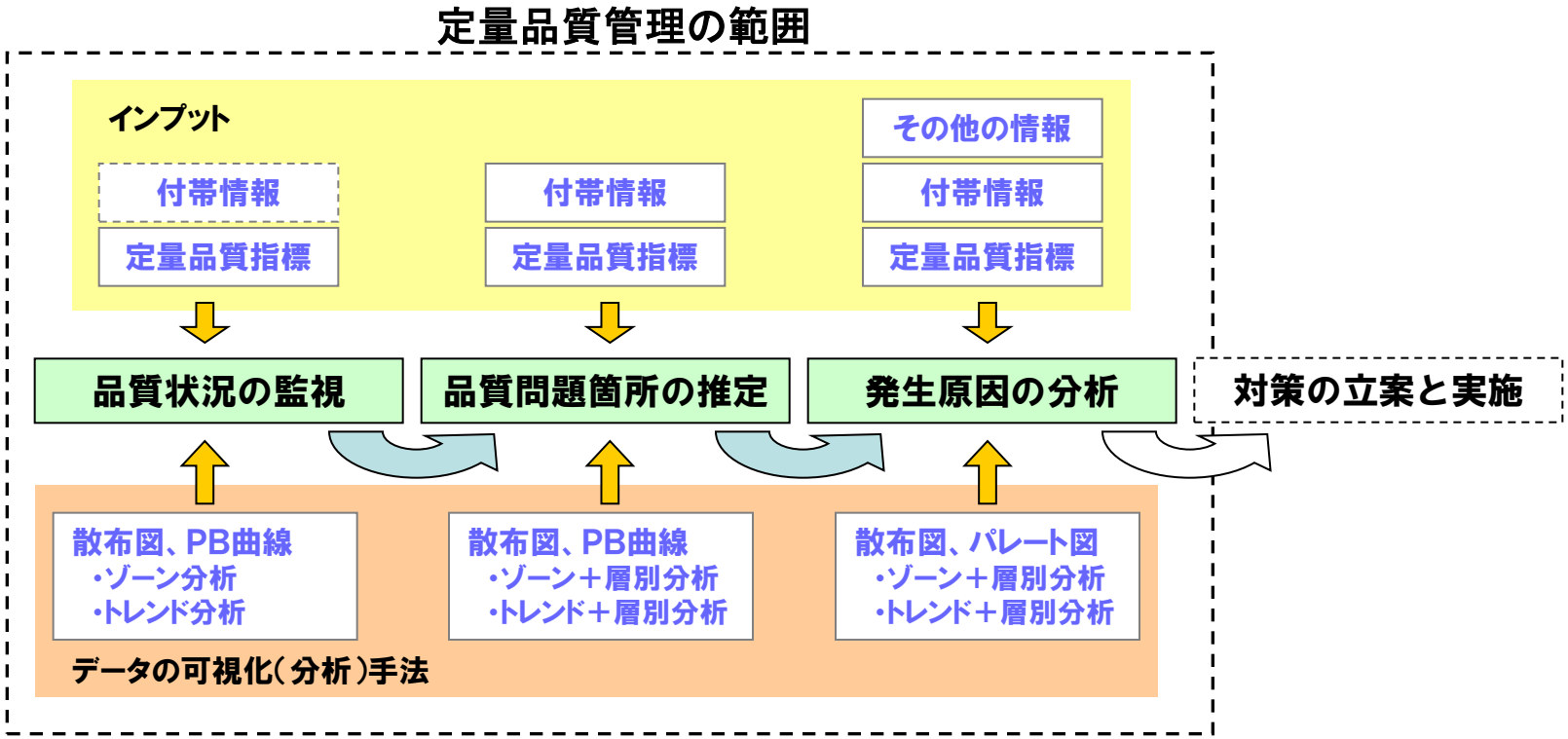
■ 主な基本指標

指標	単位	収集方法（例）
規模	FP、LOC	ファンクションポイント（FP）、LOC
作業工数	人時	－
レビュー回数	回数	レビューの実施回数
レビュー工数	人時	各レビューアのレビュー実施時間
レビュー対象規模	ページ数	レビュー対象のドキュメント量（A4換算ページ数）
レビュー指摘件数	件数	レビュー記録に記載された指摘件数
バグ数	件数	バグ票の数
テスト項目数	項目数	テスト仕様書の項目数

■ 主な導出指標

指標	説明	算出方法
レビュー指摘密度	特定規模あたりのレビュー指摘件数	レビュー指摘件数 ÷ 規模 レビュー指摘件数 ÷ レビュー対象規模
レビュー工数密度	特定規模のレビューにかかった時間	レビュー工数 ÷ 規模 レビュー工数 ÷ レビュー対象規模
レビュー指摘効率	単位時間あたりに指摘した件数	レビュー指摘件数 ÷ レビュー工数
バグ密度	単位コードあたりのバグ数	バグ数 ÷ 規模
テスト密度	単位コードあたりのテスト数	テスト項目数 ÷ 規模

まずは定量品質指標を使った監視から



付帯情報　：コーディング者名、テスト者名、コーディング期間、原因分類など、定量指標に付帯する情報

その他の情報：体制図、プロジェクト計画、レビュー記録など、定量指標に直接付帯しない情報

2－3　データの可視化手法（散布図１）

IT Holdings

定量データの可視化方法として、散布図、PB曲線、パレート図の3つを説明します

■ 散布図

散布図は、集計したデータをプロットし、異常値を発見するのに使用します。

（１）ある時点での複数データから異常値を見つける（スナップショット）

①特定の範囲や傾向から外れたものを異常値と考える

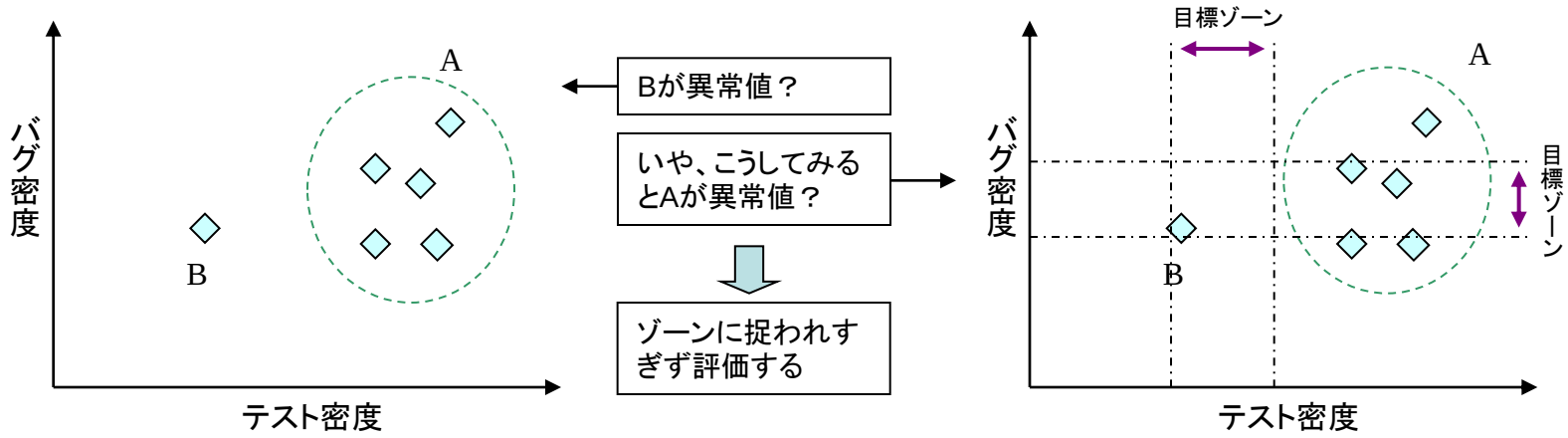
②目標値などで分割した領域（ゾーン）から逸脱しているものを異常値と考える（ゾーン分析）

（２）同一データの時間的な変化から異常値を見つける（トレンド）

①複数データの変化傾向を比較し、特異な傾向を示すものを異常値と考える

②ゾーンと組み合わせ、目標に到達できそうもないものを異常値と考える

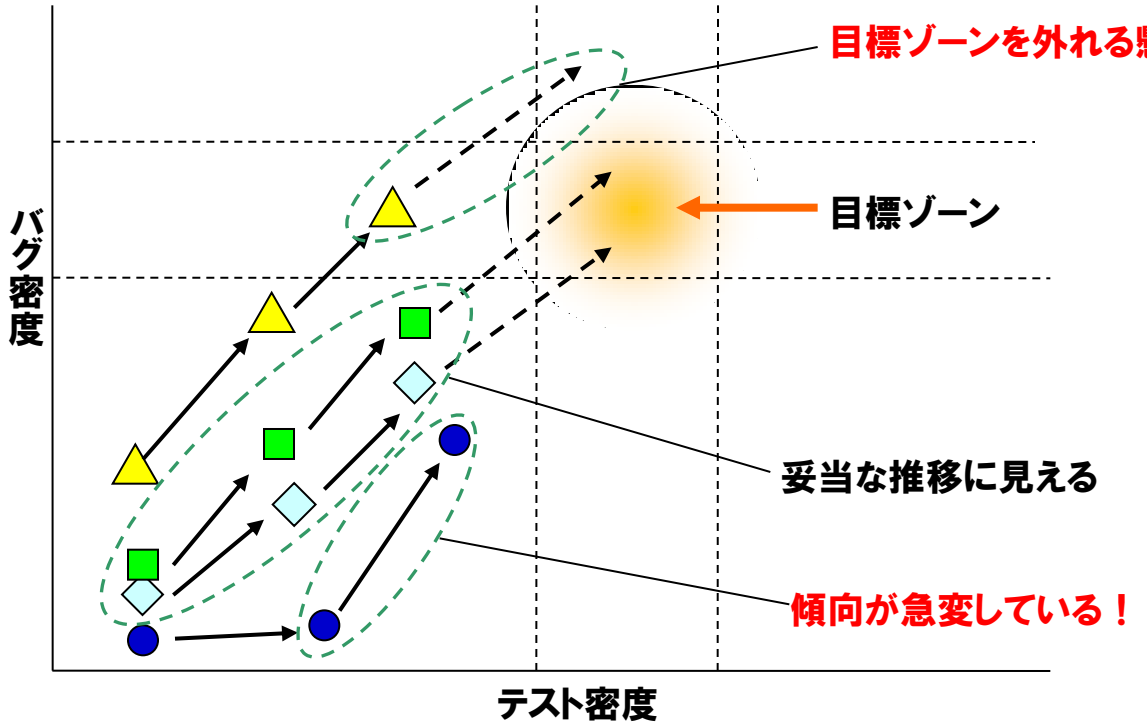
＜スナップショットの例＞



2－3　データの可視化手法（散布図2）

IT Holdings

＜トレンドの例＞



次のようなトレンドに注意する

- 他のものと比較して、明らかに異なる傾向を示すもの
- 1つのデータ推移の中で、傾向が大きく変化している箇所
- 目標値に到達できないと予測されるもの

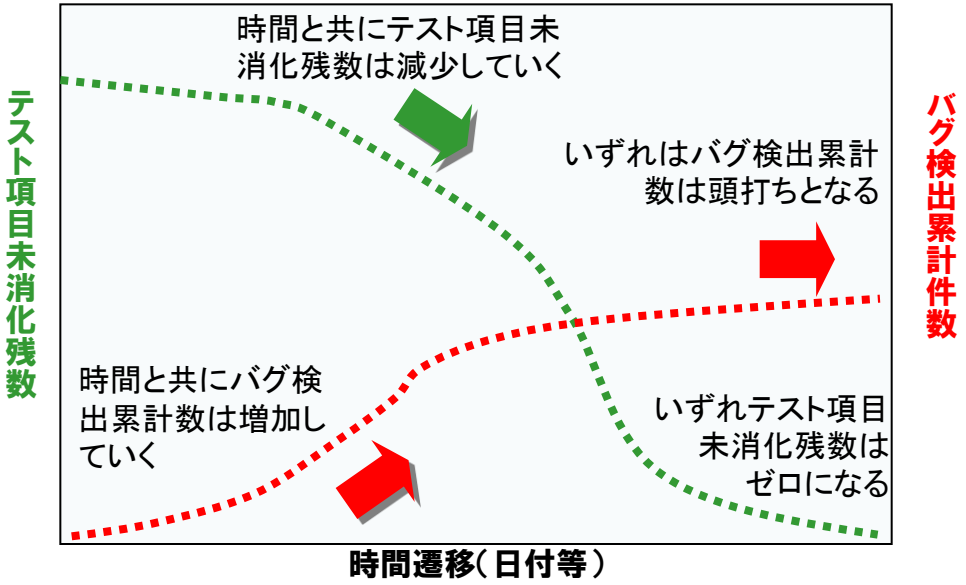
2－2　データの可視化手法（PB曲線１）

IT Holdings

■ PB曲線

PB曲線は、検出したバグの累積とテストの消化状況を表したもので、テストの完了見通しを判断するために使われます。

バグの発生や解消のトレンドは品質状況の見極めにも役立つため、工程の完了判断や出荷判定にも用いられます。

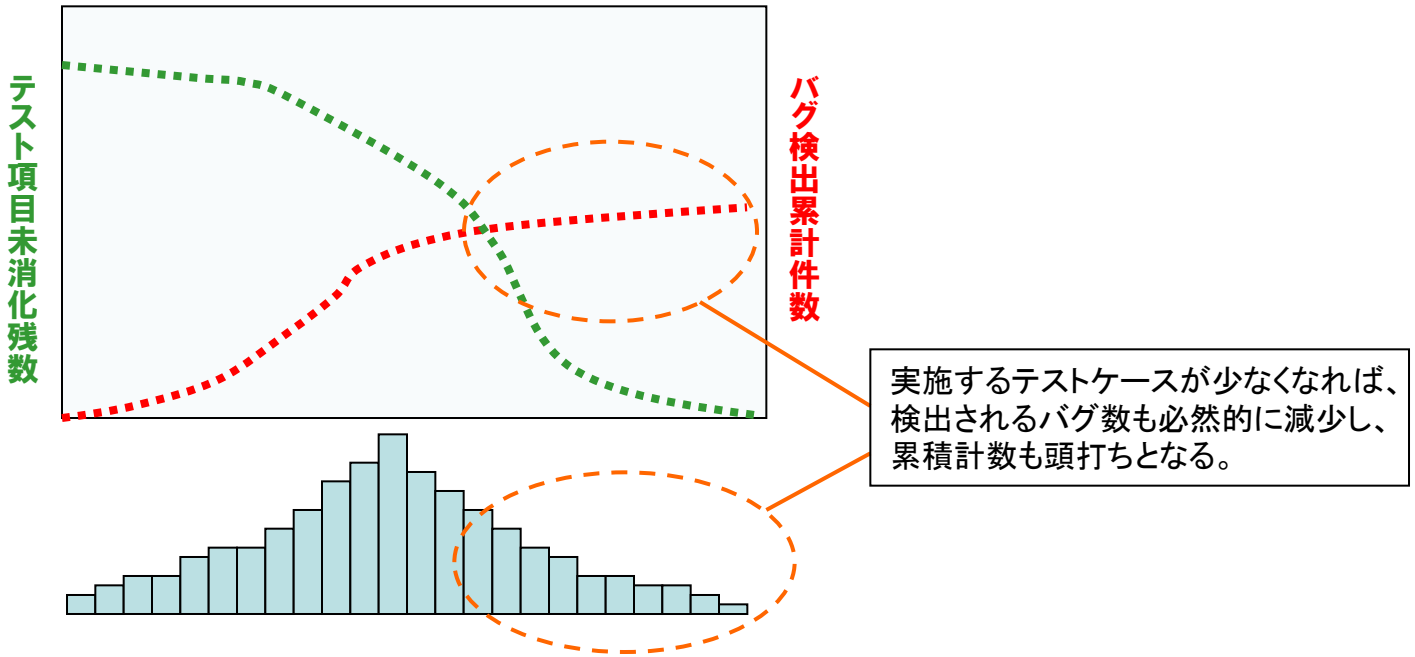


2－2　データの可視化手法（PB曲線２）

IT Holdings

（疑問）バグ検出累計数が頭打ちになれば品質が確保されたと言えるのか？

テストは徐々に実施するケース数を増やし、最後に向かって減らすのが一般的です。



即ち、累積が頭打ちになっただけで品質が安定したと断言はできません。

ただ、テストの内容に偏りが無く、網羅性も高いなら、品質安定の判断の１つとしてよいでしょう。

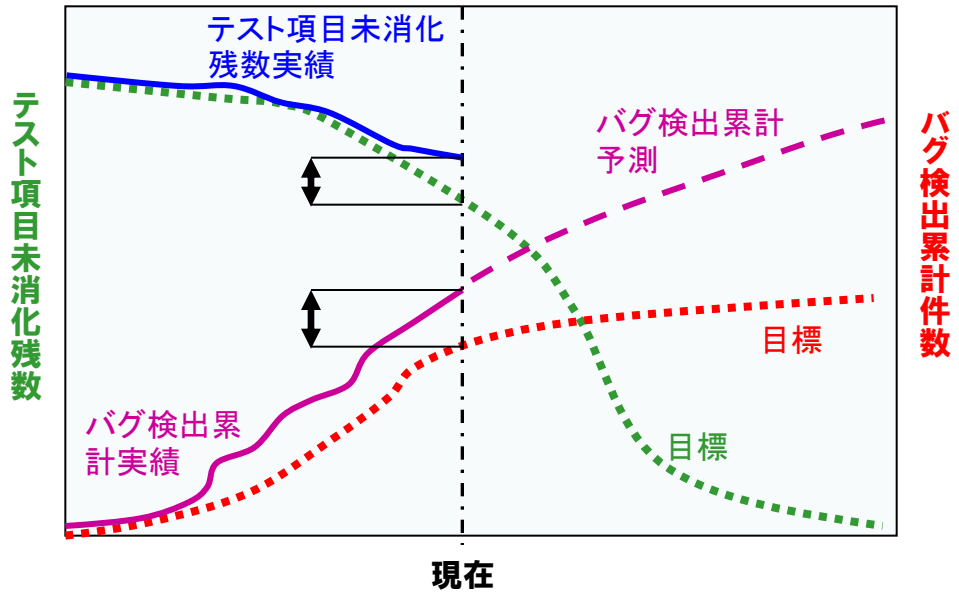
偏りがある： 最後に難易度の高いテストが残っている場合、最終段階でバグが多発する可能性もある

網羅性が低い： テストに抜けがある状態でバグが頭打ちになっても、品質が良いとは言えない

2－2　データの可視化手法（PB曲線３）

IT Holdings

PB曲線は、予測値（目標値）との差異を途上で監視するのに有効



この例では、テストの消化が目標より遅れているにもかかわらず、バグが目標よりも多く検出されているという状況で、品質状況に問題があると言わざるを得ません。

この状態を放置すれば、バグが目標を大きく上回り、モグラ叩き状態になる可能性もあります。

バグ検出目標： 過去の実績（ケース当たりの検出バグ数）から計算できる

バグ検出累計予測： ゴンペルツ曲線などを適用することで計算できる

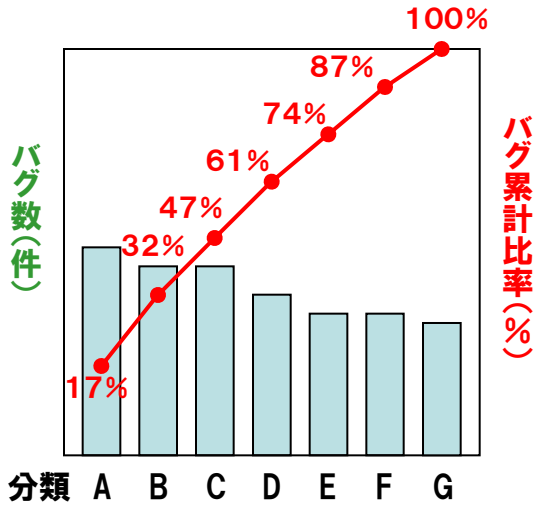
2－3　データの可視化手法（パレート図）

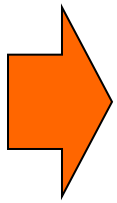
IT Holdings

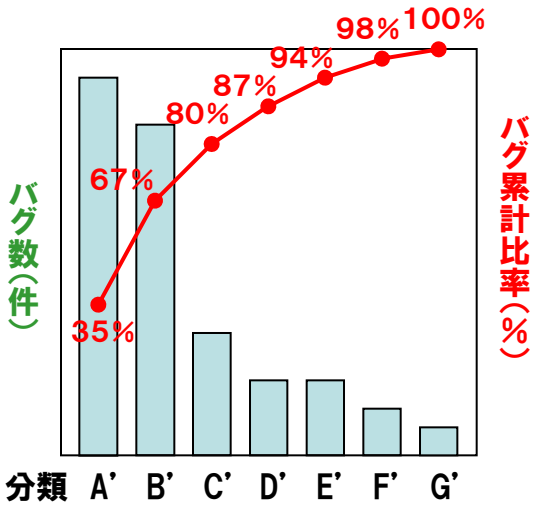
■ パレート図

バグを事象や原因別に分類し、バグ数の多い順に棒グラフで示したものに、バグの累積比率を示す折れ線グラフを重ねたものです。この手法には付帯情報が必要になります。

バグの元となる、どんな事象や原因の影響が大きいのかを見極め、最も有効な部分に手が打てるようにするために用います。







・この分類では重点課題が見えない
・分類方法を変えてみる

・この分類ではAとBで全体の2/3を占めている
・この2つに分析の重点を絞れる

例えば・・・
・要員別分類で、担当者AとBのスキルに問題
・原因別分類で、AとBが使用ツールに起因する問題

2－4　品質の監視と分析

IT Holdings

これまで解説した手法を使って品質状況を日常的に監視しましょう

定量データから品質状況を見極めるには、次の3つを意識することが重要です。

データの精度：

第一部で説明した通り、バグやステップ数の数え方が統一されていないなど、集めたデータに統一性が無かったり、エラーデータが混じっていたりしては分析ができません。

適切な粒度：

第一部で説明した通り、不適切な粒度で集計すると、データが平準化されてしまい、特異点が見えません。大きな粒度で問題が見えない時は、粒度を下げて分析し直すことも必要です。

付帯情報による層別：

全体集計では異常が無くても、付帯情報で層別して見ると特異点が見えてくることがあります。例えば、要員別やサブシステム別、バグの混入工程別などで層別して見ることです。

付帯情報は後付けで収集することが難しいので、予め定義してメンバーに周知しておくことが重要です。

例えば、バグ票に記入欄を設け、データと一緒に集めることです。

但し、何を集めるかは難しい判断で、あまり項目が多いと現場の負荷も重くなります。

2－4　品質の監視と分析（層別分析）

IT Holdings

収集すべき付帯情報は、「どんな層別分析をするか」として考える

成果物：機能（業務）別、サブシステム別、プログラム別などで層別してみる。

分かること： 難易度の高い特定の機能やサブシステム、外部連携が多い特定階層のプログラムに品質問題が集中している、といった事象。

要員：開発やテストの担当者別、パートナー別などで層別してみる。

分かること： 特定の開発者やパートナーが作った成果物に品質問題が集中している、テスト担当者によってテスト品質にバラつきがある、といった事象。

工程：バグを混入させた（と思われる）工程別に層別してみる。

分かること： 前の工程で本来潰しておくべきバグがどの程度あるのか、といった事象。あまり多い場合は前工程に戻ることも判断の１つとなる。

事象：アベンド、ハングアップ、誤出力など、予め定義した事象分類別に層別してみる。

分かること： クリティカルな問題がどの程度発生しているか、といった事象。これにより、最もクリティカルな問題から優先的に対応できる。予め設定する分類をどうするかが重要。

原因：単純ミス、仕様齟齬、理解不足など、予め定義した原因分類別に層別してみる。

分かること： どのような原因によるバグが多いか。これにより、どんな手を優先的に打つかが決まってくる。予め設定する分類をどうするかが重要。

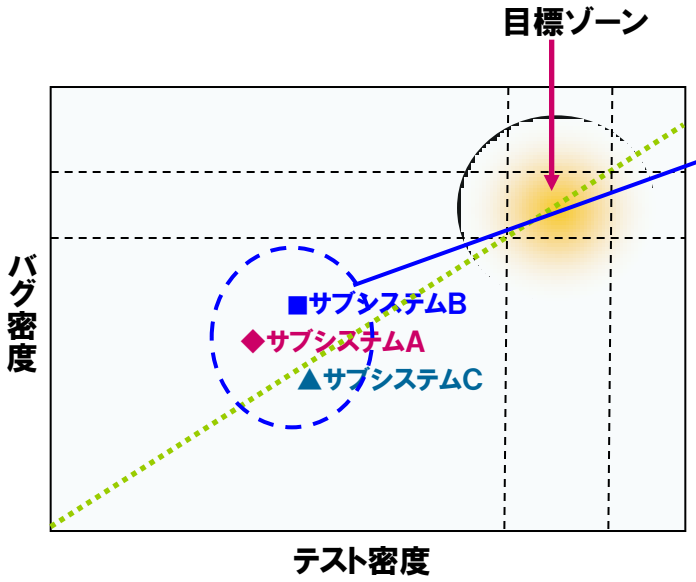
事例をあげて、実際の活動をイメージしてみましょう！

2－5　散布図から掘り下げる事例(１)

IT Holdings

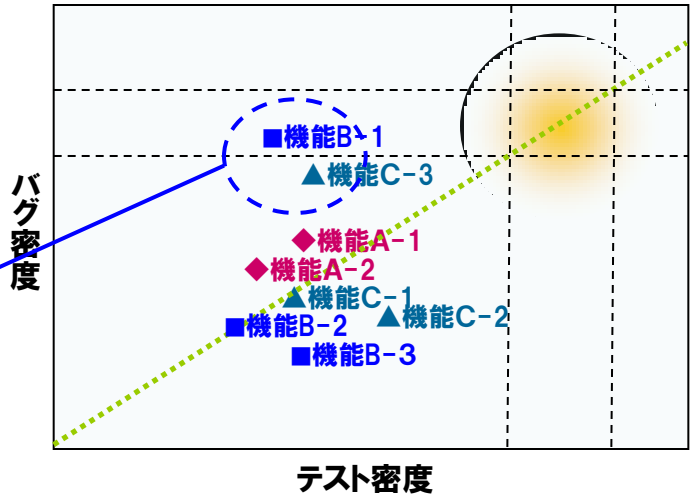
■ 散布図による途上管理

結合テストの途上で、ゾーンを意識した散布図上にデータをプロットしてみる



・データが特に突出していることも無く、各サブシステムに問題は無さそうに見える

念のため、粒度を機能レベルに落としてプロットし直してみると・・・



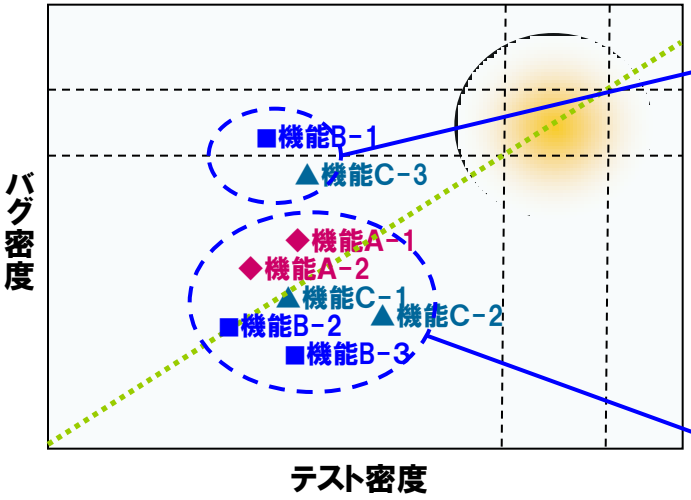
・テストの消化がまだ50％程度の段階で、既にバグ密度が目標ゾーンに達している

この状況について仮説を立てて考えてみる

2－5　散布図から掘り下げる事例（２）

IT Holdings

深堀する前に仮説を立て、それを検証するようにブレークダウンしてゆく



B-1とC-3についての仮説

- 何らかの理由により、本当に品質が低い
- 不具合数が正しく収集できていない（重複カウントなど）
- 難易度が高く、そもそも目標ゾーンの設定が低すぎる（他の機能よりもバグ密度が高くても正常）

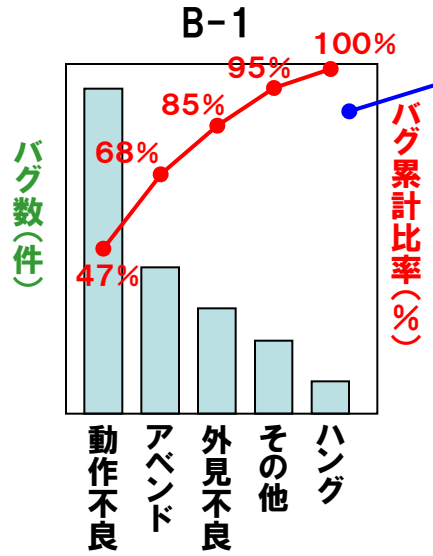
こちらのグループは大丈夫か？

- システム全体の難易度が平均よりも高く、目標ゾーンの設定が間違っているとすると、品質に問題があるのはこちらかも知れない
- バグ密度としては正常に見えても、非常に重いクリティカルなバグが多く含まれているかも知れない
- もちろん、想定内の品質が確保されているかも知れない

但し、この状況ではB-1とC-3を深堀する優先度の方が高いと考えるのが普通である

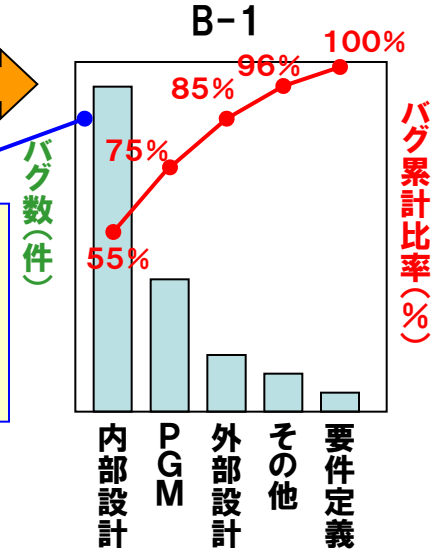
B-1 と C-3 について掘り下げてみる

2－5　散布図から掘り下げる事例（３）



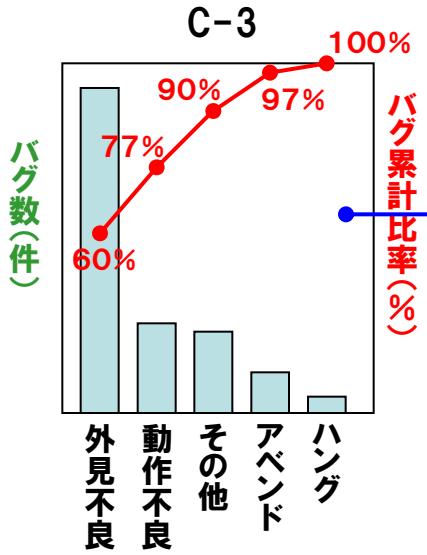
・動作不良やアベンドが多く、続行不能な状況が起こっている

原因工程別に分析してみる



・内部設計起因のバグが異常に多い

- ・内部設計のレビュー記録を確認
- ・仕様リーダ多忙により、レビュー指摘の対応確認が十分できていないことが判明
- ・内部レビューの対応結果の再確認を実施



・外見不良にバグが集中している

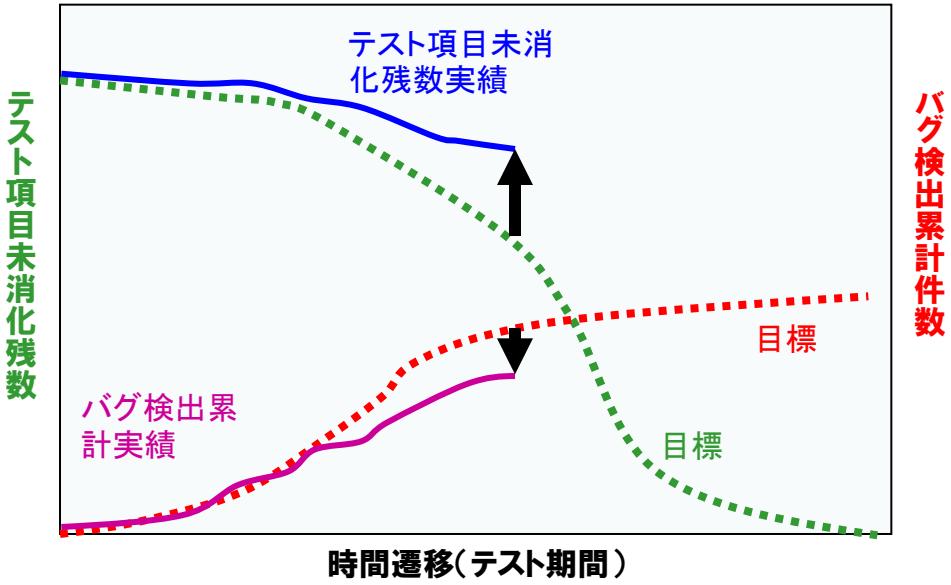
- ・不具合の内容を確認すると、テスト担当者の認識不足により、異なるテストで同じ画面の同じ箇所のバグを重複してカウントしていた
- ・これを除いたところ、バグ密度が正常化したので、とりあえず問題なしと判断した

2－6　PB曲線から掘り下げる事例（１）

IT Holdings

■ 傾向①：末広がり

テスト項目未消化残数は目標の外側、バグ検出累計数は目標の内側を推移している状態
（テストが目標通り進行しておらず、結果としてバグも思うように検出できていない）



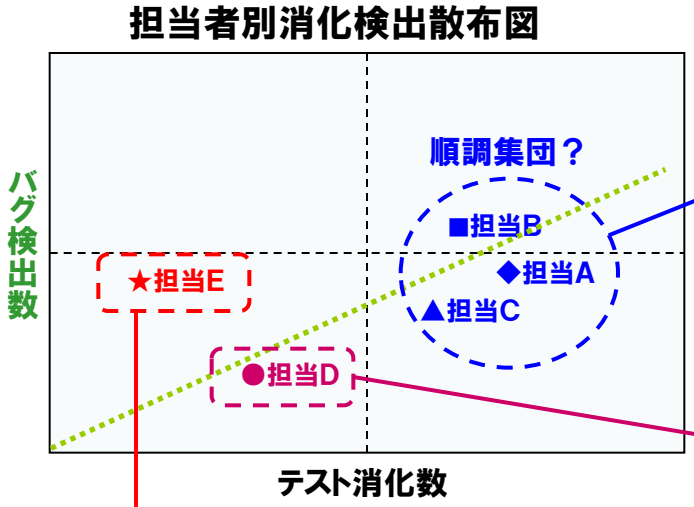
①テスト担当者別、サブシステム別のテスト消化状況を見てみる
　→ 全体的な傾向ではなく、特定領域の問題という可能性がある

②サブシステムや機能、モジュール別の散布図を見てみる
　→ いくつかのバグが全体進捗を阻害している可能性がある

2－6　PB曲線から掘り下げる事例(１)－１

IT Holdings

テスト担当者別に、散布図(テスト消化数vsバグ検出数)を見てみると ・・・



・担当A、B、Cは5年以上の経験者でテスト仕様作成にも関わっている
・テスト消化状況も順調で、バグもそれなりに検出している
・データ上も同じような傾向を示している

とりあえず、この集団は順調にテストが進んでいると判断

・担当Dはテスト経験が浅い
・テスト消化が遅れており、その分、バグの検出も少ない
・データ上は順調集団と同じような傾向を示している

・ヒアリングにより、テスト仕様の理解に時間を要していることが判明
・経験者をバックアップに付けたところ、テスト消化数とバグ検出数が順調集団に追いついてきた
・とりあえず、これで様子を見ることにした

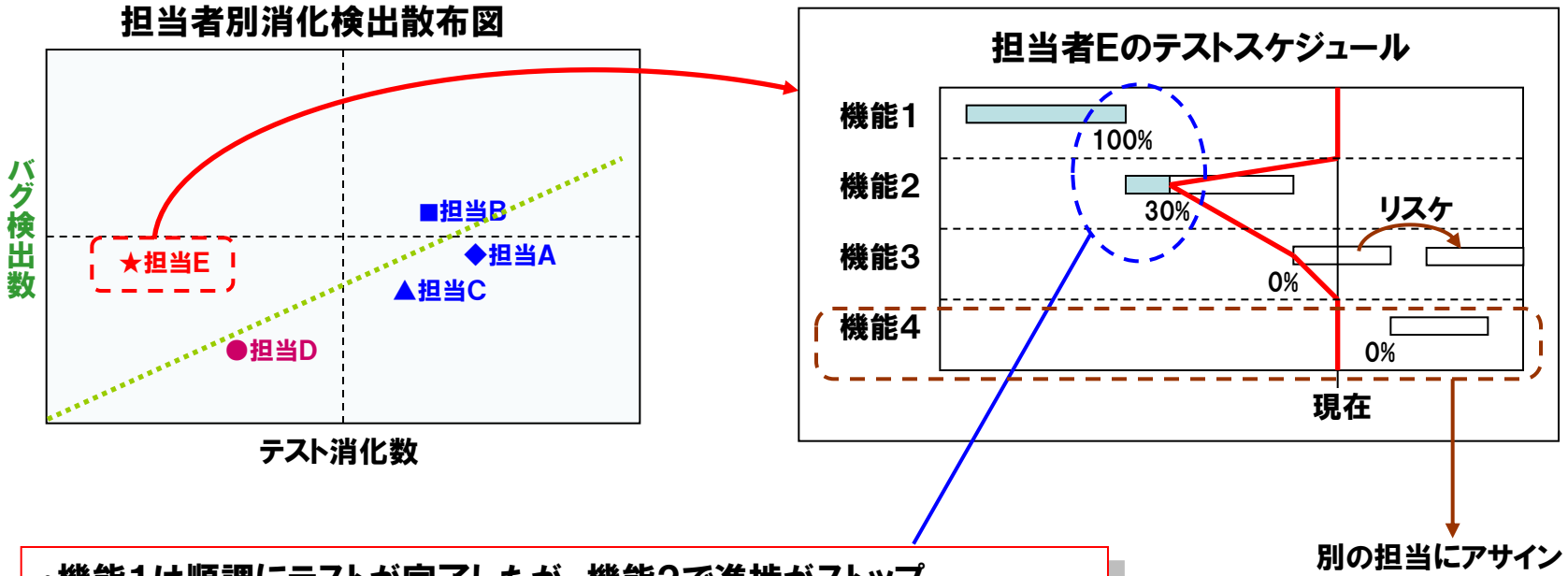
・担当Eは5年以上のテスト経験がある
・しかし、テスト消化が遅れ、バグの検出が異常に高くなっている
・データ上も順調集団と全く違う傾向を示している

担当Eについては早急に深堀すべき状況であると判断

2－6　PB曲線から掘り下げる事例(１)－２

IT Holdings

担当者Eについて深堀してみる ・・・



・機能１は順調にテストが完了したが、機能２で進捗がストップ
・内容を確認すると、システム全体に影響する致命的なバグが発覚、その対応に追われてテストが中断していることが判明

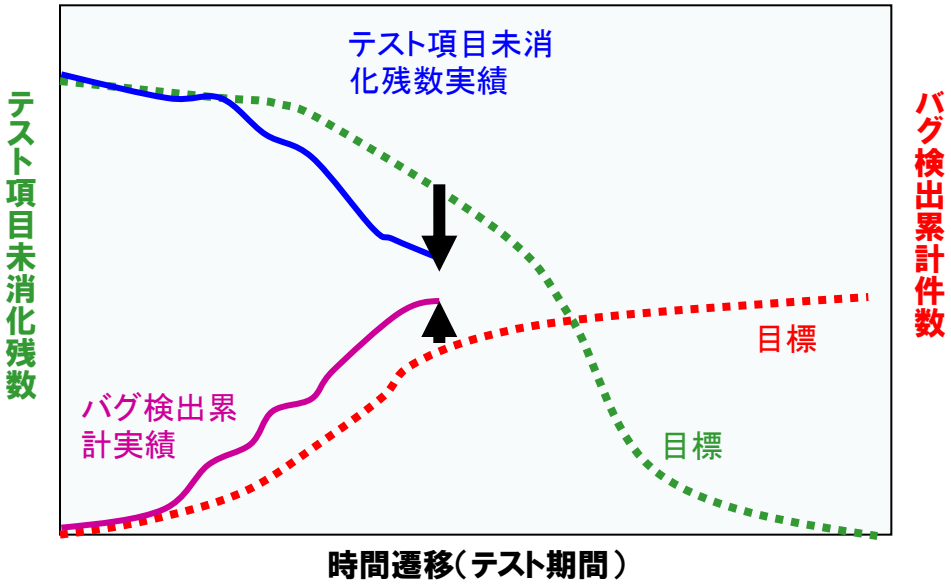
・その問題を担当Eから切り離し、担当Eには機能２のテストに専念してもらうことに
・更にスケジュールキャッチアップのため、機能３をリスケ、機能４は別の担当に割り振った

2－6　PB曲線から掘り下げる事例（２）

IT Holdings

■ 傾向②：末閉じ

テスト項目未消化残数は目標の内側、バグ検出累計数は目標の外側を推移している状態
（テストが目標よりも進んでおり、結果としてバグも早目に検出されている）

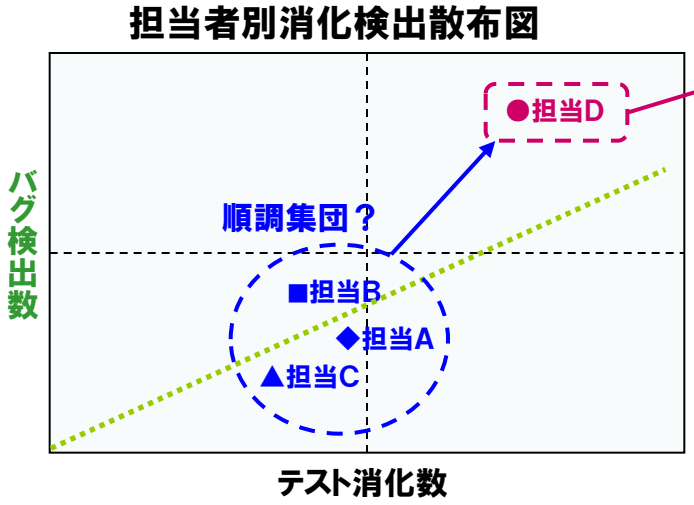


①サブシステムや機能、モジュール別の散布図を見てみる
　→ スケジュールが単に前倒しになっているだけなのかを確認する

2－6　PB曲線から掘り下げる事例(２)－１

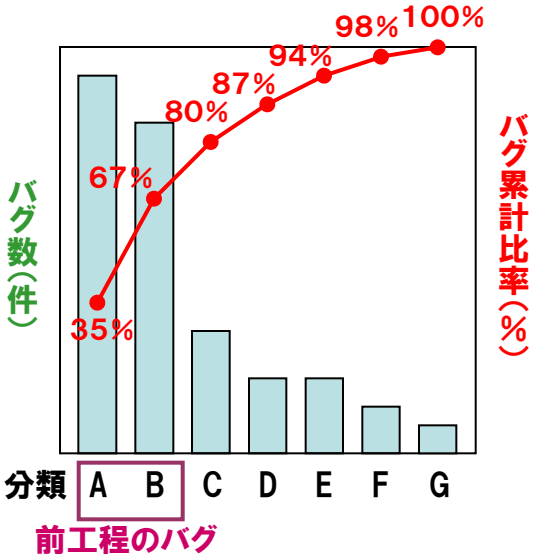
IT Holdings

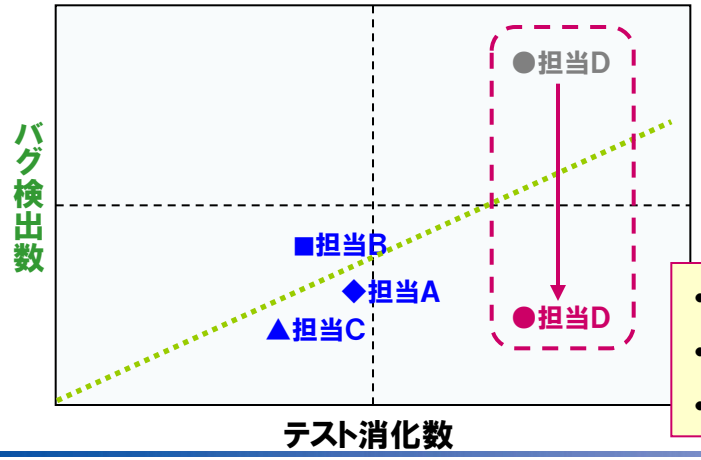
テスト担当者別に、散布図(テスト消化数vsバグ検出数)を見てみると ・・・



・担当Dだけが突出してテストが進んでいる
・バグ検出も多く、順調集団とは違う傾向が見える

・バグの内容をパレート図により分析してみる





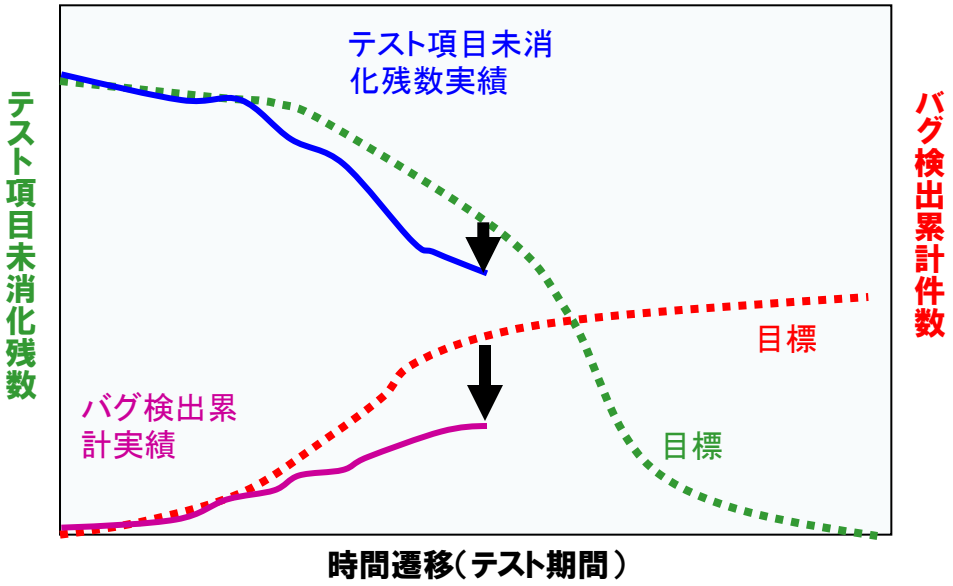
・本来、前工程で検出すべきバグが多いことが判明
・それらを差し引くと現工程の検出バグ数が激減
・PB曲線も「末閉じ型」から「下向き型」へと変化

2－6　PB曲線から掘り下げる事例（３）

IT Holdings

■ 傾向③：下向き型

テスト項目未消化残数、バグ検出累計数ともに目標の内側を推移している状態
（テストが目標より進行しているが、バグの検出が目標よりも少ない）



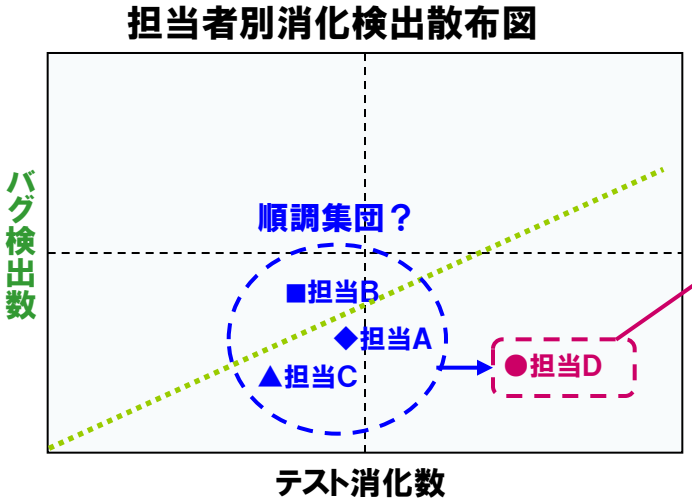
①サブシステムや機能別のテスト消化状況、散布図を見てみる
　→ 特定領域のテストが安易に進められている可能性がある

②テスト内容を確認する（妥当性、網羅性）
　→ 難易度の低いテストばかりが偏って実施されている可能性がある

2－6　PB曲線から掘り下げる事例(３)－１

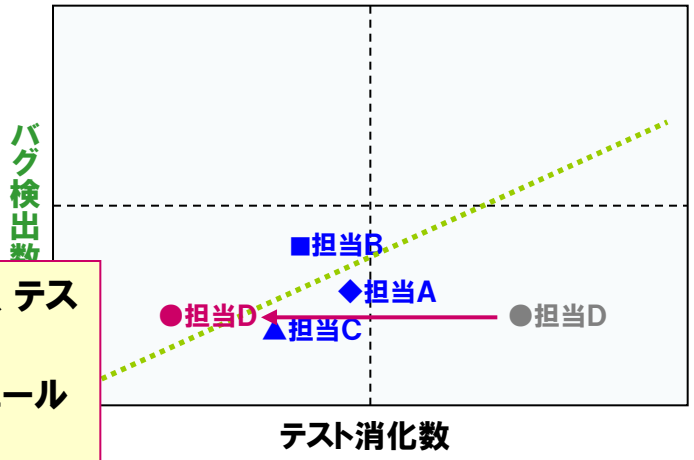
IT Holdings

テスト担当者別に、散布図(テスト消化数vsバグ検出数)を見てみると ・・・



・担当Dだけが突出してテストが進んでいる
・しかし、バグの検出は想定よりもかなり少ない
・データ上も順調集団とは違う傾向が見える

・テスト項目数は多いが、網羅性が低く、正常系が中心でかつ重複するものが多いことが判明
・テスト項目を精査し、網羅性を高めた上でテスト密度が確保されるよう項目を追加
・その結果、テスト消化数は大幅に減少



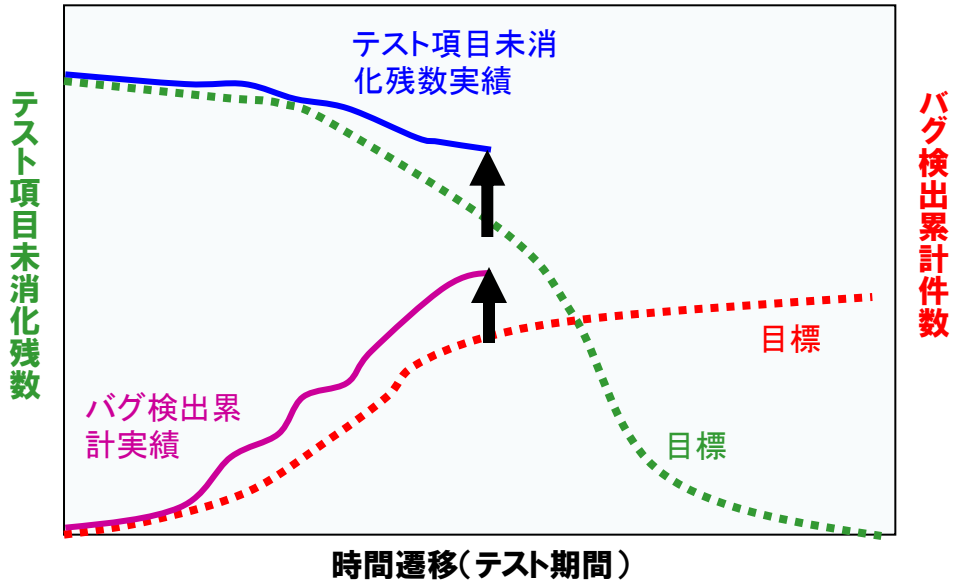
・担当者Dのテストは周回遅れ状態になったが、テストの内容は適正化された
・あとは別の担当者に振り分けるなど、スケジュールをキャッチアップする

2－6　PB曲線から掘り下げる事例（４）

IT Holdings

■ 傾向④：上向き型

テスト項目未消化残数、バグ検出累計数ともに目標の外側を推移している状態
（テストが目標通り進行していないにも関わらず、バグが目標以上に検出されている）



①バグ原因工程別のパレート図を見てみる
　→ 設計や要件に漏れや不良がある可能性が高い

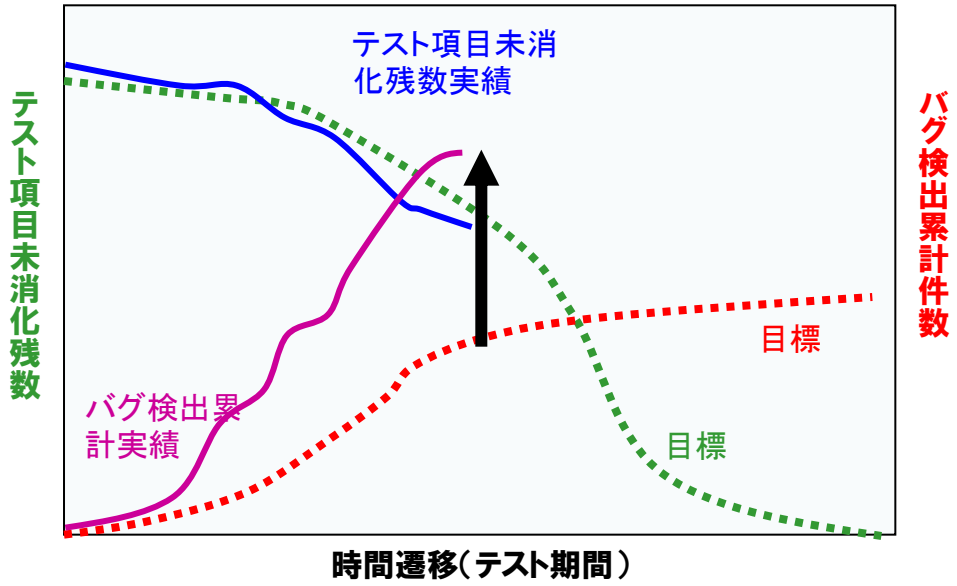
②サブシステムや機能、モジュール別の散布図を見てみる
　→ いくつかのバグが全体進捗を阻害している可能性がある

2－6　PB曲線から掘り下げる事例（５）

IT Holdings

■ 傾向⑤：バグだらけ型

テスト項目未消化残数は予定通りだが、バグ検出累計数が目標の外側を推移している状態
（テストの進行は目標通りなのに、想定以上にバグが検出されている）



①検出したバグの内容をパレート図で見てみる
　→ 軽微なバグ（前工程のバグ）が大量に出ている可能性がある
　　（その場合、単体テストや机上デバッグからやり直した方が良い）

②サブシステムや機能、モジュール別の散布図を見てみる
　→ 一部でモグラ叩き状態に陥っている可能性がある

2－7　第2章のまとめ

IT Holdings

（１）必要なデータを定期的に収集・集計し、色々な可視化手法（散布図やPB曲線）を使って異常がないかをチェックする。

→ 他のデータとの逸脱、異常なトレンド、目標値との乖離など

（２）特に異常が見られない場合でも、データの粒度や層別により集計単位を変えてチェックしてみる。

→ 粒度を細かくしたり、機能別、要員別などで層別してみる

（３）データに気になる変化があれば、問題と思われる箇所を特定するためにデータを層別に分類し、その変化が顕著になる部分を絞り込む。

→ 事象や原因の分類別に層別し、パレート図などを使って分析してみる

（４）体制図やスケジュール、残業状況、仕様変更の発生状況、会議議事録などの情報も必要に応じて参照し、発生原因を深堀する。

→ バグの作り込みが多い要員の残業状況など、因果関係を調べる

監視活動から何か察知しても、必ずしもそれが問題とは限りません。定量データで分かる範囲には限界があるので、問題をある程度絞り込んだら、あとは現場ヒアリングや現物チェックなどで確認することが重要です。

ご清聴ありがとうございました


IT Holdings