システムテストフェーズから始めるソフトウェア品質マネジメント

Software quality management to start from the system test phase

中嶋　良秀
yoshi@noritz.co.jp
株式会社　ノーリツ
制御ソフトウェア開発部　品質技術推進室

発表要旨:

我々の組織では、給湯器等の製品を設計・性能評価を実施する製品開発部門と、仕様化から結合テストまでを実施する制御ソフトウェア開発部門があり、「製品発売」を判定するためのシステムテストを実施するのは、前者の製品開発部門である。

製品として品質を確保した上で、発売するためには、システムテストが重要になるが、各々の部門が専業化しているため、どうしてもシステムテストとしての取り組みが弱くなっていた。(システムテストを専門的に考える部門がおらず、効果的なシステムテストに取り組めていなかった。)

そのため、従来は、網羅的にテストケースを設定し、それぞれの重要度は決めているものの、効果的なシステムテストの進め方になっておらず、外部流出につながるケースがあった。

今後、我々の製品にも高機能化の要求が強まる中で、従来どおりのやり方では品質を確保できなくなる可能性も否定できない。そこで、効果的に品質を確保するために、システムテストの部分にもっと注力していく必要が出てきた。

本発表では、システムテストを起点として、派生開発における問題の兆候を掴むための基準作り等、定量的なソフトウェアの品質マネジメント実現のために、工夫した内容とその結果について述べる。

キーワード:

システムテスト、メトリクス、マネジメント、プロセス改善、派生開発

想定している聴衆

これからソフトウェアメトリクスを活用してマネジメントに役立てたい方

派生開発が中心で、開発部門がシステムテストを実施している組織の方

発表者の紹介（全角１００文字）:

１９９４年株式会社ノーリツに入社　風呂給湯機器等の組込みソフトウェア開発に従事
２００２年から３年間、派生開発のコンサルを受け、ＸＤＤＰによる開発へシフト
２０１３年からＳＱｉＰ研究会での活動を機に、現部署にてＳＥＰＧ／ＳＱＡを担当

* 副題は不要であれば行ごと削除してください

システムテストフェーズから始める定量的ソフトウェア品質マネジメント

★Ａｇｅｎｄａ

1．事例背景
2．事例概要
3．問題解決のアプローチ
4．問題と解決結果
5．まとめと今後の課題

株式会社ノーリツ
制御ソフトウェア開発部　品質技術推進室

○中嶋　良秀　　yoshi@noritz.co.jp

事例の対象製品

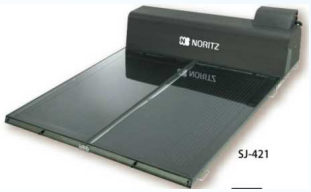
1．事例背景

給湯器

ファンヒータ

コージェネレーション

太陽熱利用



ハイブリット給湯器

ミスト付浴室暖房乾燥機

床暖房

おそうじ浴槽

マイクロバブル

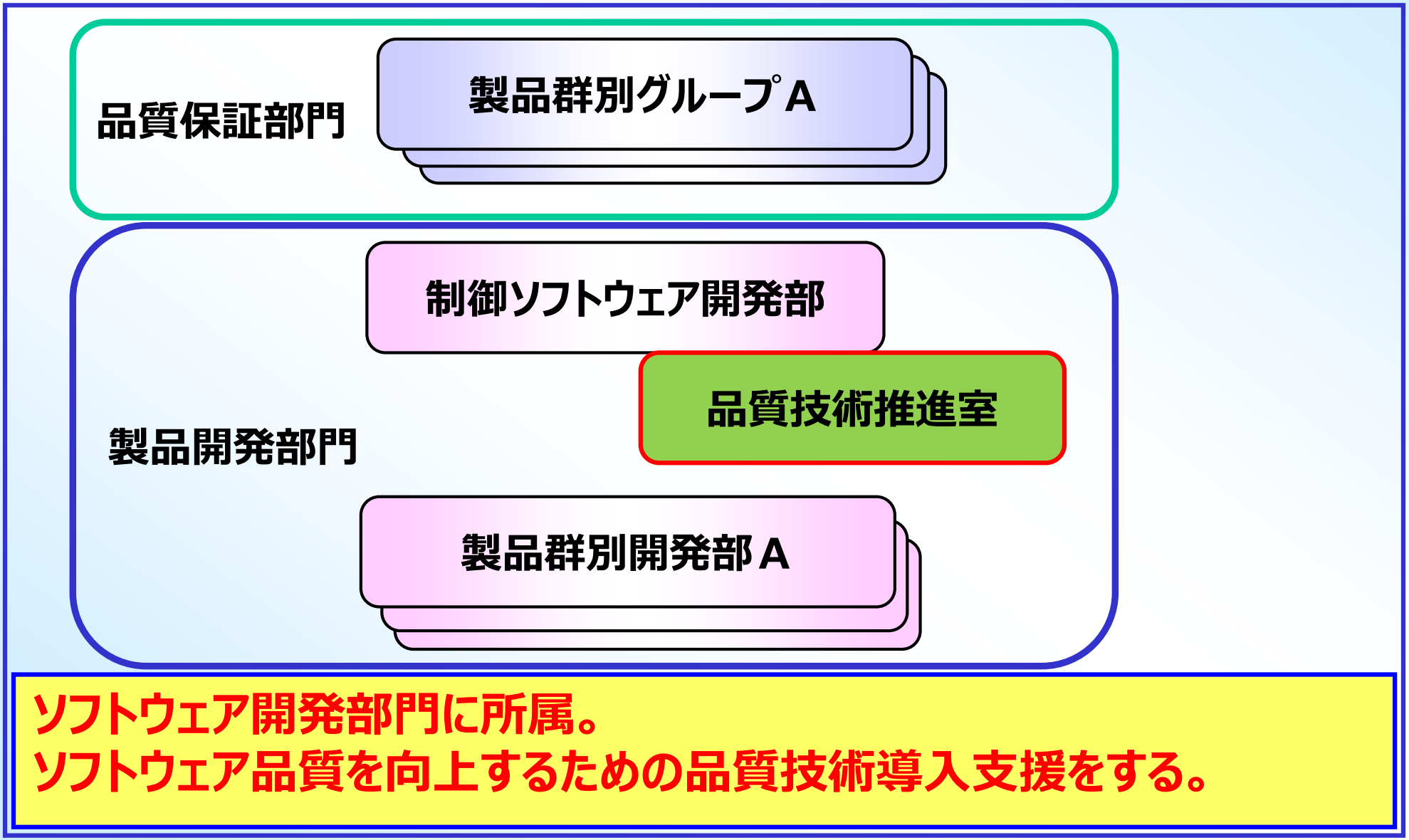
浴室ＴＶ

ネット接続

ガスコンロ

住宅設備は、家庭内に代替品もなく、高品質が求められる。
さらに、機器同士から外部との接続等、システム化が進む。

組織の構成



ソフトウェア開発部門に所属。
ソフトウェア品質を向上するための品質技術導入支援をする。

組織構成における問題

品質保証部門

製品群別グループＡ

製造プロセス及び部品に関する品質保証が中心。

製品開発部門

制御ソフトウェア開発部

ソフトウェア開発部門は、ＸＤＤＰによる派生開発

品質技術推進室

製品群別開発部Ａ

仕様書の記載内容に沿ったシステムテスト中心。

製品開発部門は、システムテストを担当。
ソフトウェアテストの専門部隊ではなく、取り組みも弱い。
品質保証部は、製造工程移管後の品質保証を行う。

高機能化に対する問題

① 派生開発のため、今回の変化点を中心に、
テストの重みづけを行って、システムテストの効率を上げている。

② 機能数の増加と共に、システムテスト工数は、
開発終盤の大きなウェイトを占めている。

③ 通信システムとの連携など、機器単体の機能増加に加え、
機器同士の組み合わせテストも増加傾向にある。

**今まで以上に、効果的且つ効率的に、
品質を確保できるようになる必要がある。**

なぜシステムテストフェーズから始めるのか？

① 「組織構成における問題」で、
ソフトウェア品質保証の知識がないメンバーが、
体験的にソフトウェアの品質保証を理解するタイミングとして、
システムテストフェーズが最適。

② 「高機能化に対する問題」で、
システムテストでは、複数人のメンバーが分業している。
マネジメントを機能させて、チームとして機能する風土が必要。

③ ②における、マネジメントを機能させるために、
工数を都度入力するのを習慣にしてもらう必要があり、
システムテストだけと限定することで、まずは実績を作る。

**ソフトウェア品質保証に必要な取組みが弱い組織構成において、
各部門が共通の認識で取り組めるタイミングとして、
システムテストフェーズを選択。**

ＳＱｉＰ２０１５の内容

３．問題解決のアプローチ

＜概要＞

★システムテストフェーズにける心配事

＜システムテスト開始前＞

- 無理な見積もりになっていないのか？

＜システムテスト中＞

- システムテスト、予定通りに進んでいるのか？
- 指摘内容がリアルタイムに確認できているのか？
- ソフトウェアの出来栄えは、問題ないのか？

＜システムテスト終了前＞

- システムテストを終了しても大丈夫なのか？

昨年度は、テスト工数の取得ができていないのが課題だった。
今回は、システムテスト工数の都度入力による効果も含めて、
問題解決に取り組んだ。

（参考）基準の作り方

<ソフト修正数の管理基準について>

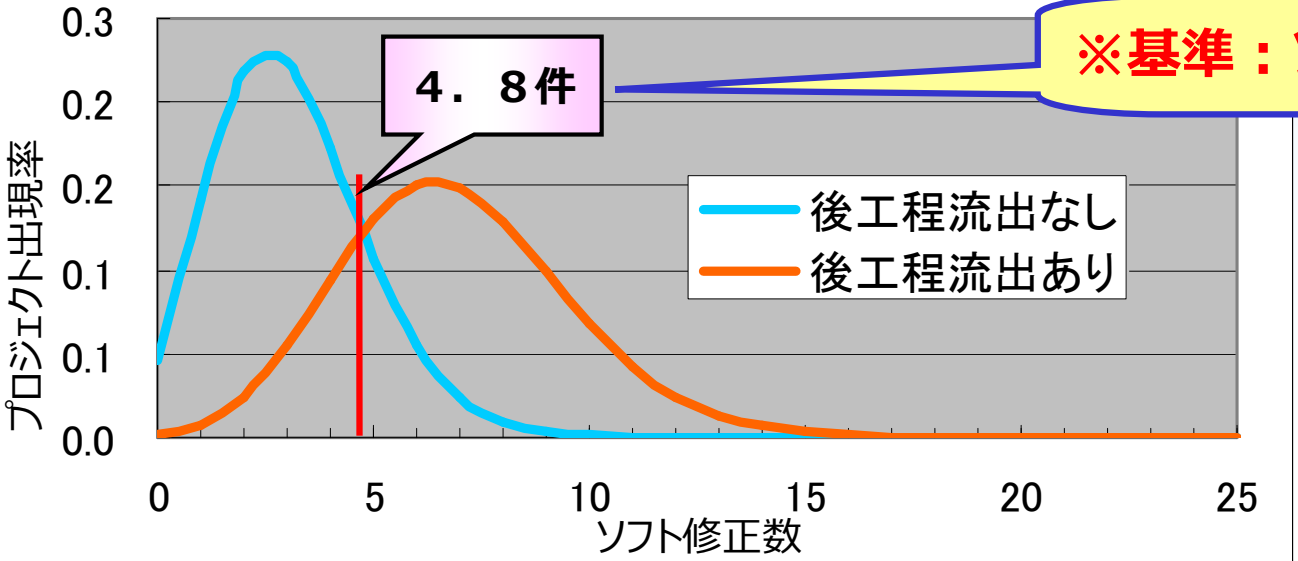
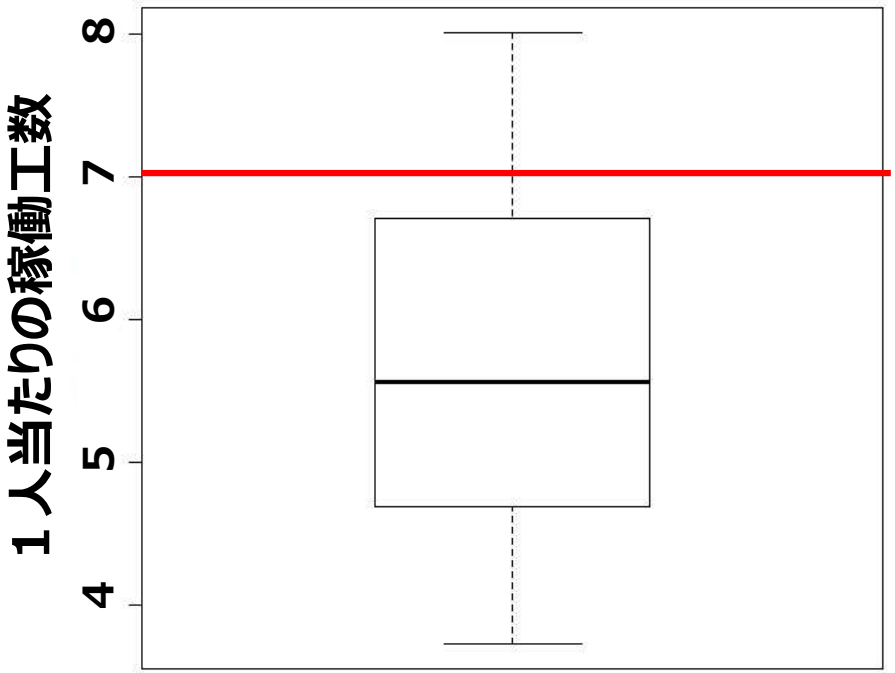


図１．　ソフト修正数－プロジェクト出現率（※）

派生開発であれば、ソフト修正０件が当たり前。
ソフト修正が滅多に起こらないものと考えると、
ポワソン分布との適合を検定してみる価値がある。
仮に適合しなかったとしても、自社モデルを構築し、そのモデルとの適合を証明すれば、
基準は簡単に作ることができる。

（参考）基準の作り方

＜１人あたりの稼働工数基準について＞



※上限・・・７ｈ／日

図２．　１人当たりの稼働工数 箱ひげ図

１人あたりの稼働工数は、全体の約８０％を占める７ｈ／日を上限とする。
実際、約半数の開発は、６ｈ／日に満たないため、
計画時点で７ｈ／日を超えるような場合は、そのペースを継続するのが困難とみなす。

問題解決の事例

４．問題と解決結果

- 以下の３つの視点で問題解決を実施した。
  - 進捗の視点
    - ①　工数進捗の管理
    - ②　オープンクローズ管理
    - ③　信頼度成長曲線
  - 開発時品質の視点
    - ④　ソフト修正数管理
  - 今後の開発に関する視点
    - ⑤　開発へのフィードバック

① 工数進捗の定量化

4．問題と解決結果

（１） テスト工数の進捗管理（改善前）

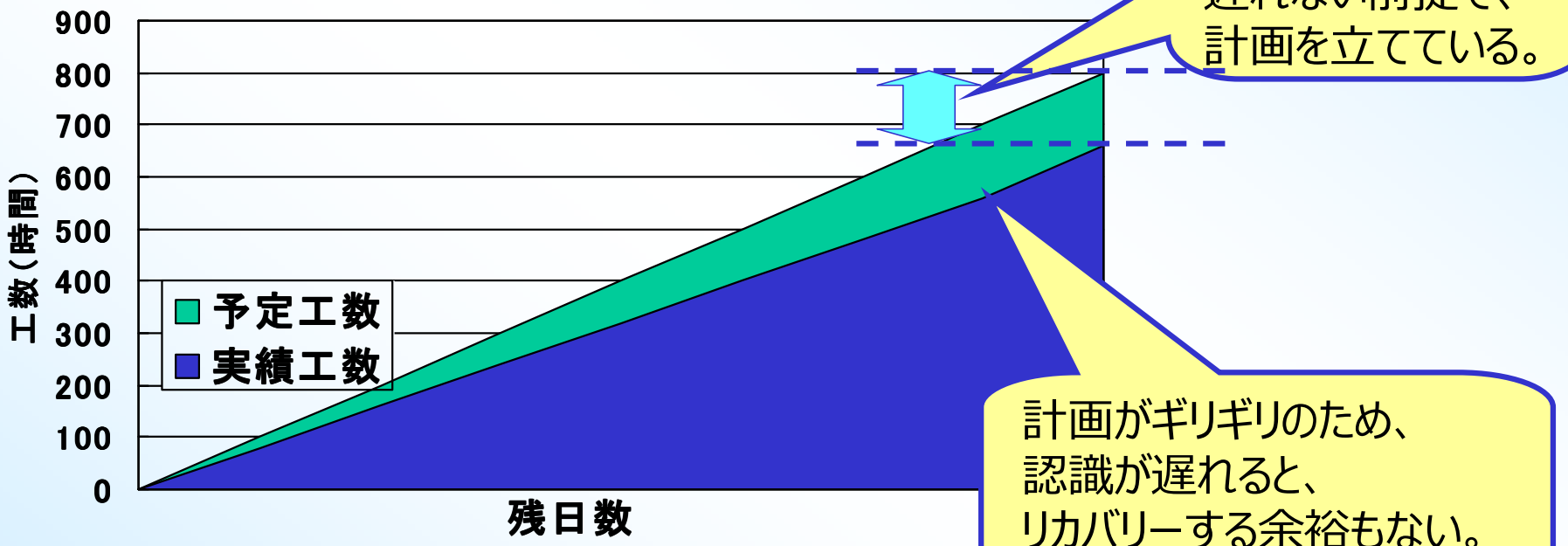


図３．テスト工数推移（製品Ａ）

計画は終了日に対して、目標に立てている。
遅れだすと、毎日テストばかりすれば、間に合う計画に変更する。

① 工数進捗の定量化

4．問題と解決結果

（１） テスト工数の進捗管理（改善後）

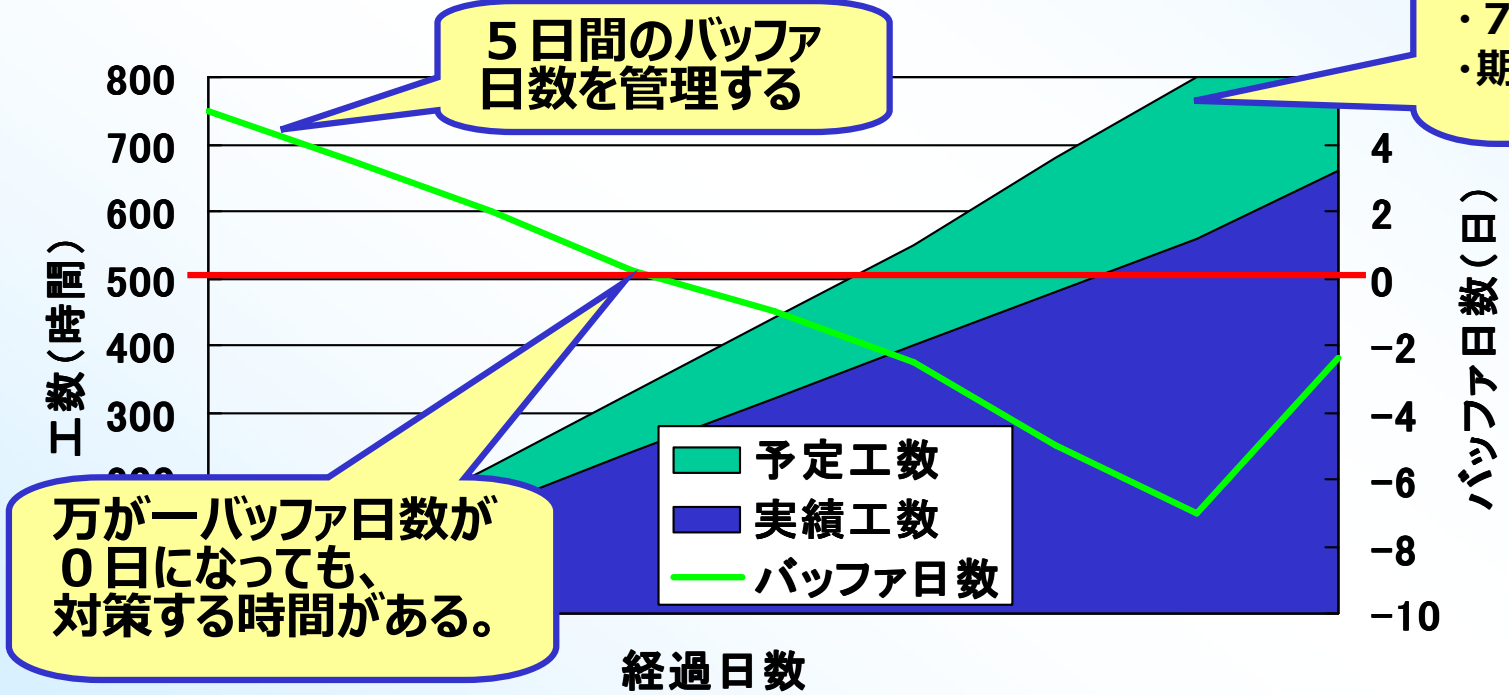


図４．テスト工数推移（製品Ｂ）

**バッファ日数の減少状況を監視しておけば、
当初計画の問題点が早期にみつかる。（マイルドＣＣＰＭ）**

① 工数進捗の定量化

4．問題と解決結果

（２） テストランク別進捗管理

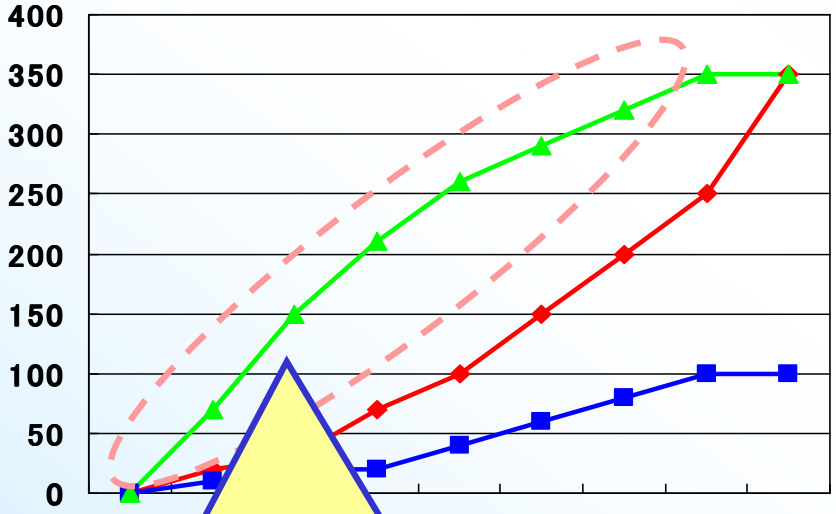


図５ 工数推移（製品Ｃ）

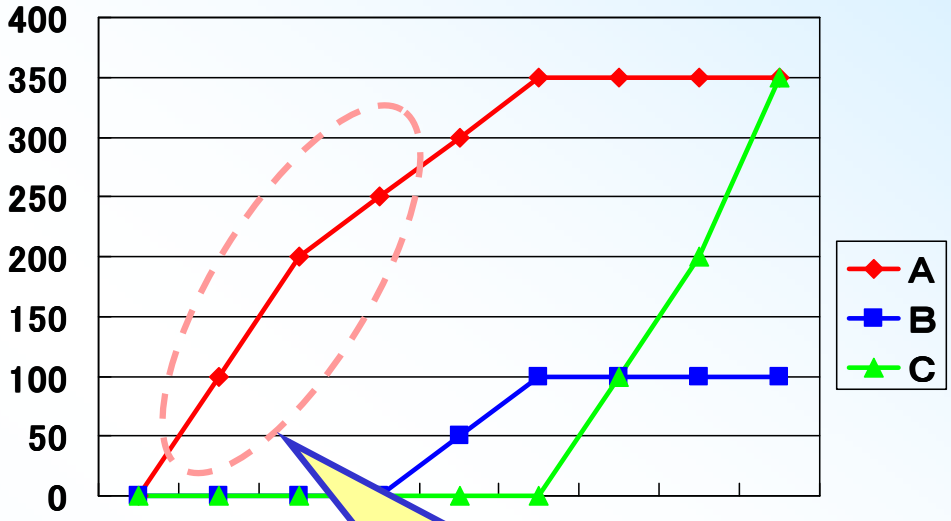


図６．ランク別 移（製品Ｄ）

Ｃ（変更なし）ランクが先行しており、
終盤にソフト修正が発生するリスクがある。
⇒是正が必要

Ａ（最重要）ランクが
先行している⇒問題なし。

投入工数が確保されていたとしても、中身が伴っていない場合、早期に問題を共有することで、リスクを回避する。

② オープンクローズ管理

4．問題と解決結果

（１） オープンクローズ管理（改善前）

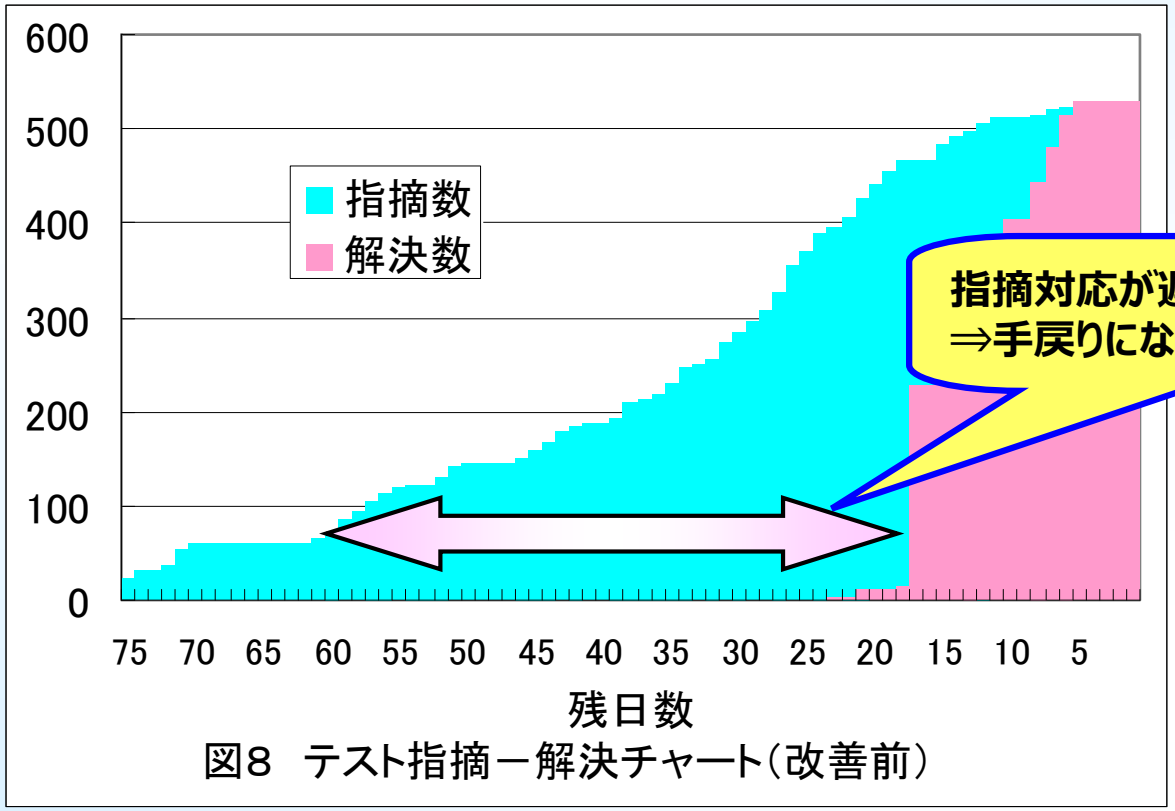


図８ テスト指摘－解決チャート（改善前）

クローズできていない案件が多くなると、
ソフト修正による手戻りや再確認のリスクが増大する。

② オープンクローズ管理

4．問題と解決結果

（１） オープンクローズ管理（改善後）

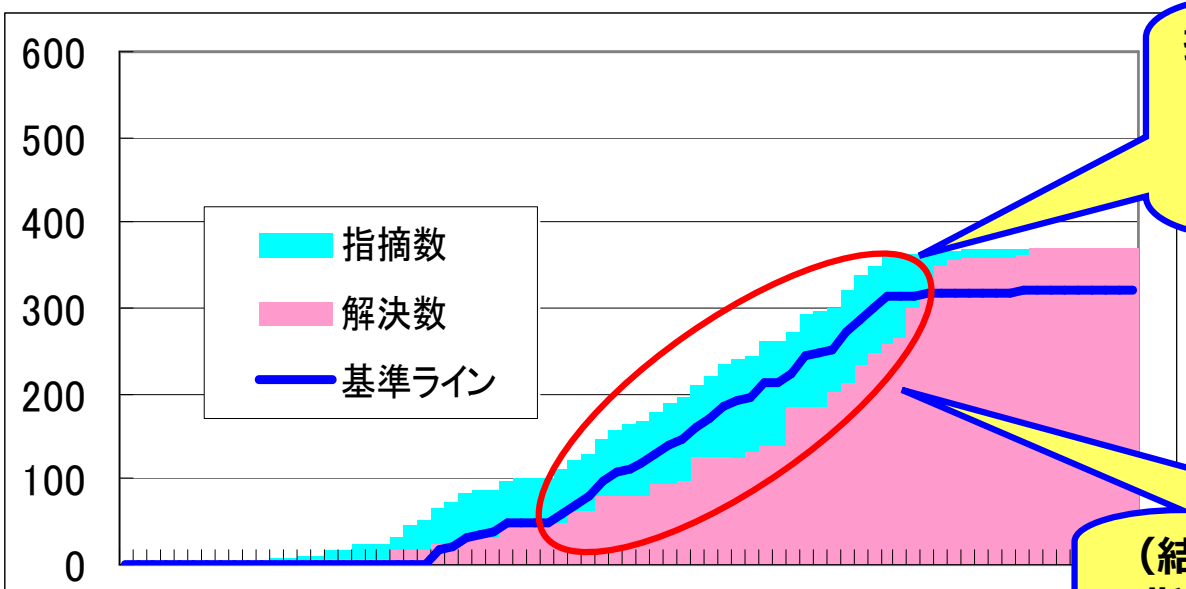


図9.　テスト指摘ー解決チャート(改善後)

指摘数－５０件＞解決数
⇒指摘対応が滞っている
→危険
→喚起

（結果）
指摘対応状況が改善
⇒手戻りになるリスクが軽減

**クローズできてない案件数に基準を設けると、
テスト終盤のソフト修正による手戻りリスクを低減できる。**

③ 信頼度成長曲線

4．問題と解決結果

（１） 信頼度成長曲線（改善前）

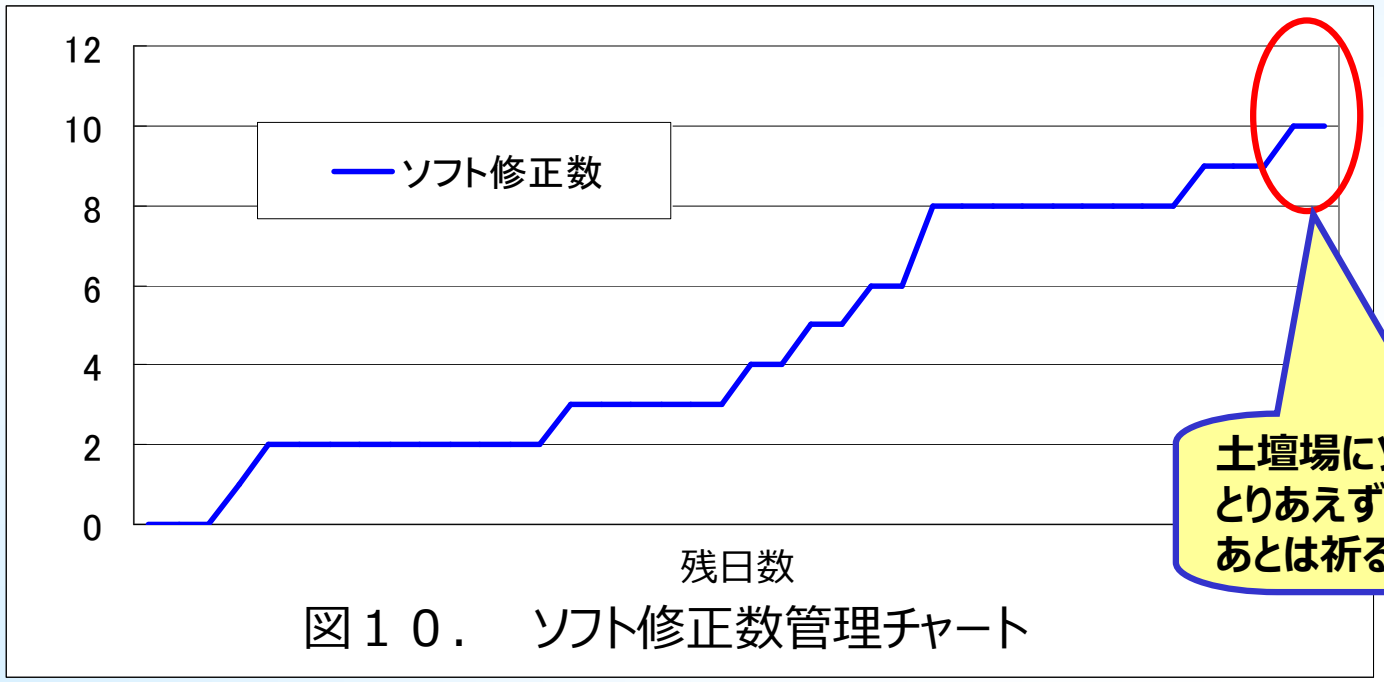


図１０． ソフト修正数管理チャート

派生開発では、ソフト修正数がそもそも少ない。
信頼度成長曲線だけでは、収束しているのか判断しにくい。

③ 信頼度成長曲線

4．問題と解決結果

（１） 信頼度成長曲線（改善後）

※ 不具合指摘率
＝指摘数／システムテスト工数（h）

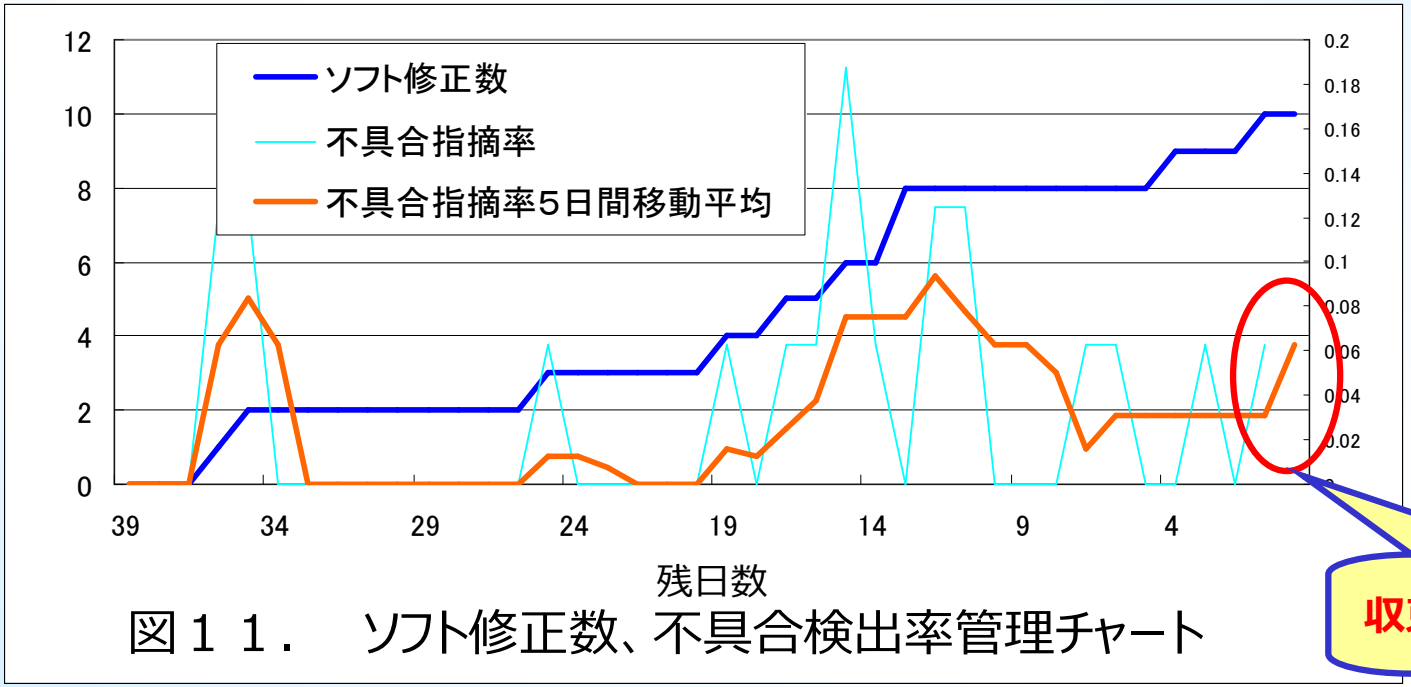


図１１． ソフト修正数、不具合検出率管理チャート

収束していないかも！？

不具合指摘率を併用し、テスト終盤でソフト修正が発生した時でも、客観的な視点で収束判断を可能にする。

③ 信頼度成長曲線

4．問題と解決結果

（２） 不具合指摘率の適用（改善後）

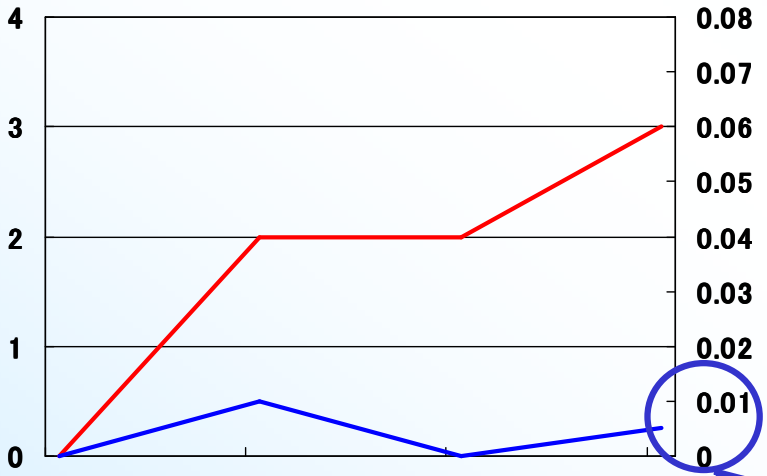


図１２．不具合修正率推移（製品Ｅ）

図13．不具合修正率推移（製品F）

このレベルなら、
様子を見よう

工数をかけたら、
まだ不具合が出そう

不具合件数の増え方は同じでも、
短時間で不具合が検出された“製品Ｆ”には、
本当に再確認必要な部分がないか問いかける。

④ ソフト修正数管理

４．問題と解決結果

（１） ソフト修正数による対応

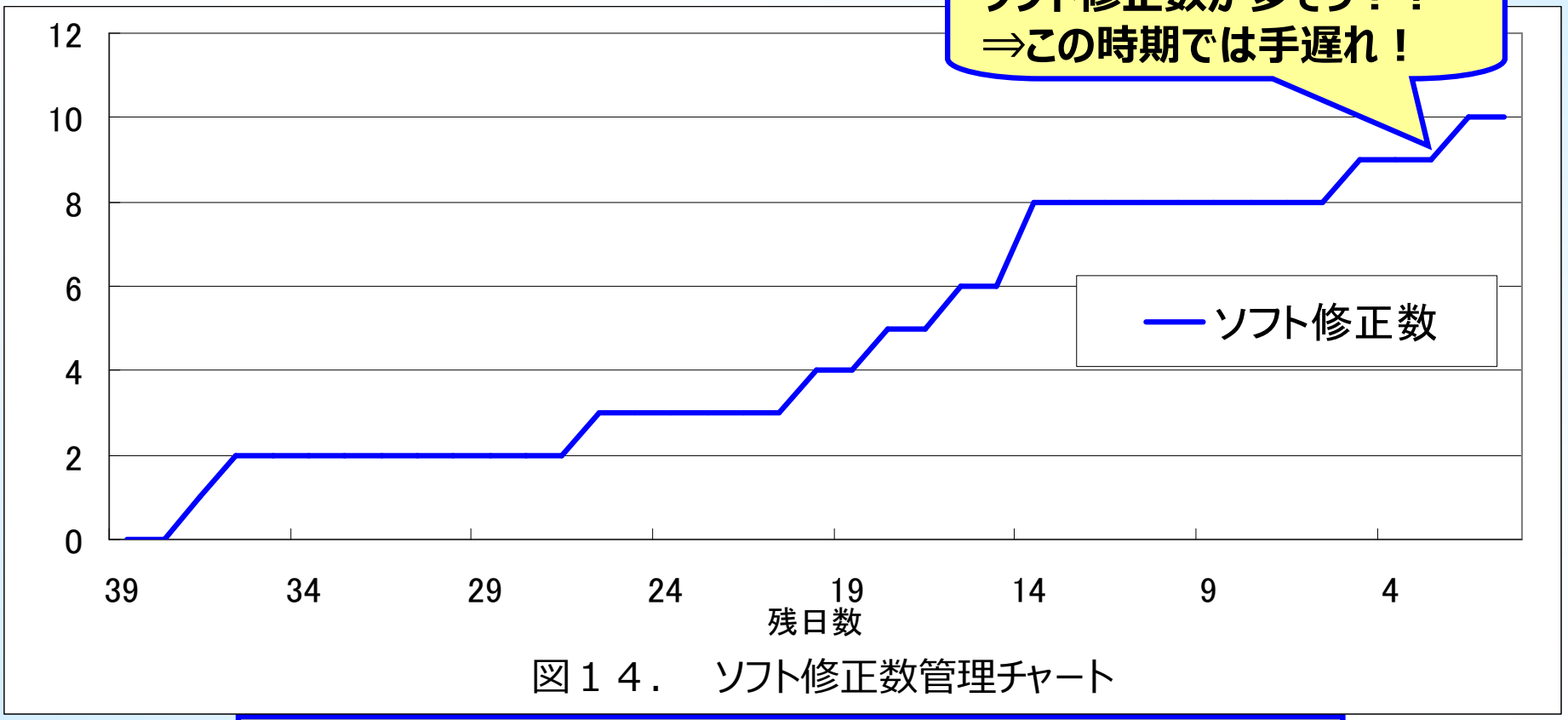


図１４．　ソフト修正数管理チャート

ソフト修正等、対応を続けている間は、
目の前の仕事をこなすのに一生懸命になる。

④ ソフト修正数管理

４．問題と解決結果

（１） ソフト修正数による対応

過去のデータから導いた基準線を設ける。
⇒早期かつ効果的に、
成果物等の点検を行う。

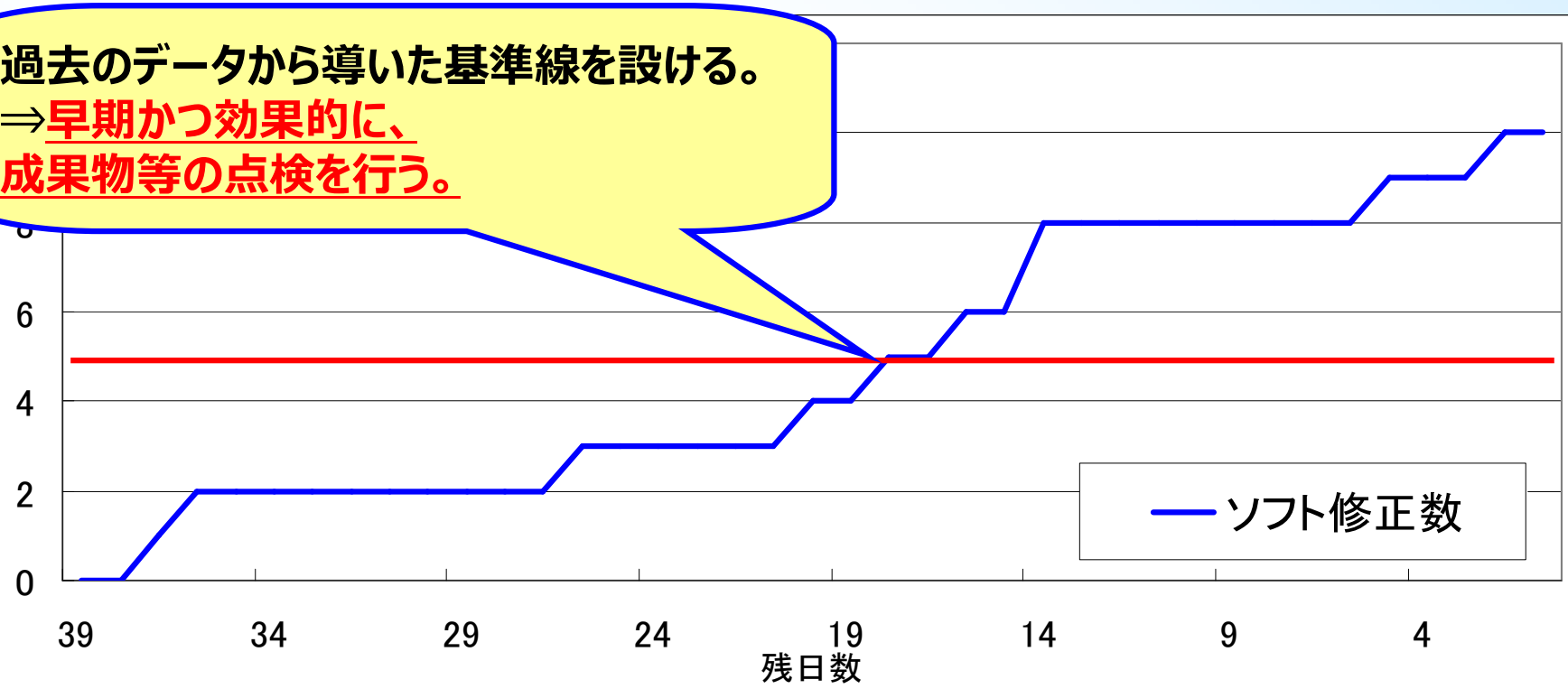


図１５． ソフト修正数管理チャート

※基準・・・ソフト修正数５件（ポアソン分布モデルを適用）

適切な基準であれば、早期に効果的な行動に移行できる。

⑤ 開発へのフィードバック

4．問題と解決結果

（１） フィードバック例

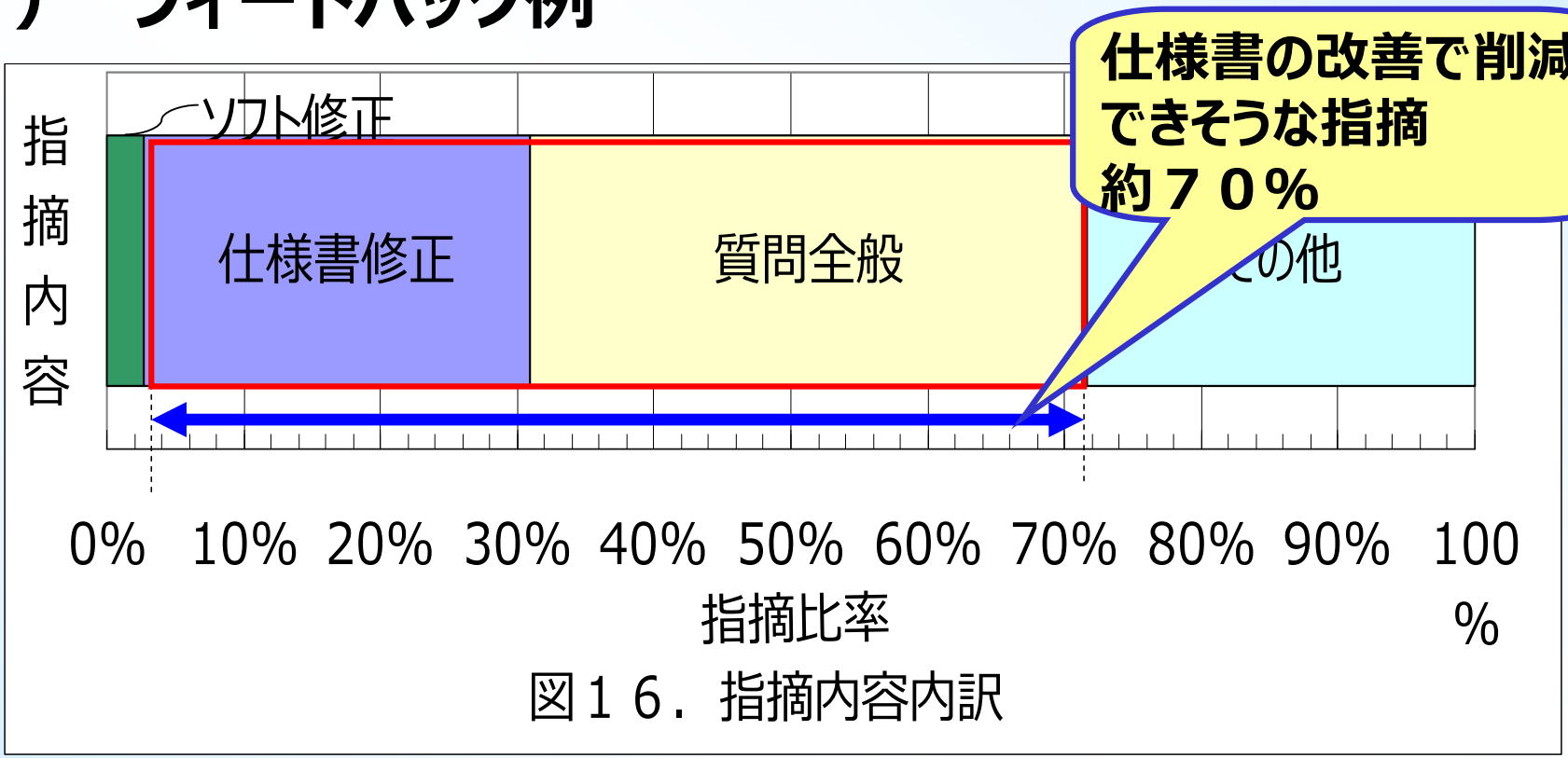


図１６．指摘内容内訳

システムテストで取得したデータを分析することで、次期開発へのフィードバックにつなげる。

まとめ

５．まとめと今後の課題

① テストマネジメントの知識がないメンバーに対して、
現状報告とこの先に起こる問題提起に対して、
耳を傾けてもらえるようになった。

２０１５年度
外部流出０件達成

⇒ 単に厳しすぎる基準を設けるのではなく、
過去のデータを活用した基準作りが有効である。

② システムテストの計画やメンバーの役割が変更になっても、
その理由や問題点をチームで共有するきっかけができた。

③ システムテストの工数をリアルタイムに取得できるようになった。
また、工数入力に対する不満の声も少なくなった。

「精密に誤るよりも、漠然と正しくありたい」をモットーに、
適切な基準を作り、その基準を元に状況を判断した上で、
開発者同士のコミュニケーションを誘発する問いかけが重要

今後の課題

① 開発フェーズの各マイルストーン毎に、
製品開発全体の進捗状況を定量化できるようにする。

② 開発着手時に、各マイルストーン毎に、
達成すべき基準値を設定する。

③ システムテストのインプットとなる仕様書の改善と、
その効果について、定量的に表現する手段を検討する。

**ソフトウェアメトリクスの適用範囲を開発フェーズにも拡大して、
開発現場のコミュニケーションがさらに活性化する仕掛けをする。**

ご清聴ありがとうございました

新しい幸せを、わかすこと。

NORITZ