ODC分析の適用からえた気づき
～テスト工程からのアプローチ～

ODC分析研究会　運営委員長　佐々木方規

2019/3/20(水)

アジェンダ

- What's　ソフトウェアテストプロセス！？
  - ソフトウェアテストプロセスのあらまし
  - （目的別）テスト技術
  - テストプロセス設計（テストレベル）
  - V&Vモデル、代表的な（変則）Wモデルでのテスト活動
  - テスト計画、その他
- ソフトウェアテストをデザインする（参考）、テスト設計の例
- テストレベルを再考する
- テストプロセスの効率化（参考）
- 不具合解析・報告の流れ
- 付録
  - テスト実行時のノウハウ情報
  - 不具合連絡票（サンプル）

ODC分析分析 … その前に、

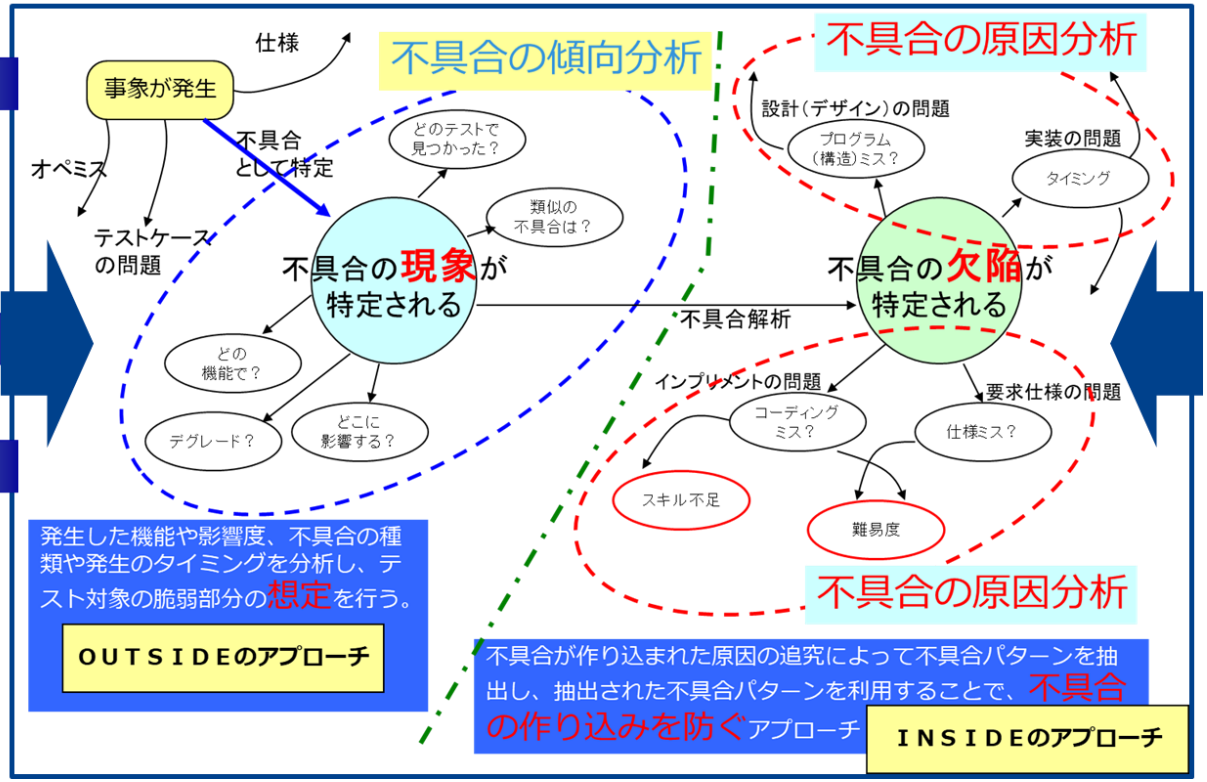
- （背景として）ODC分析を適用するまでの「不具合分析」は、ＩｎＳｉｄｅ／ＯｕｔＳｉｄｅ Ｖｉｅｗで分析していました。

2007/10/26のセミナー資料より

現象の分析
- 機能に依存
- コンディション（条件）に依存
- コンフィグレーション（環境）に依存

タイミングの分析
- 不具合発生時期

デグレードの分析
- 同じ不具合の調査
- 不具合の修正による問題の調査

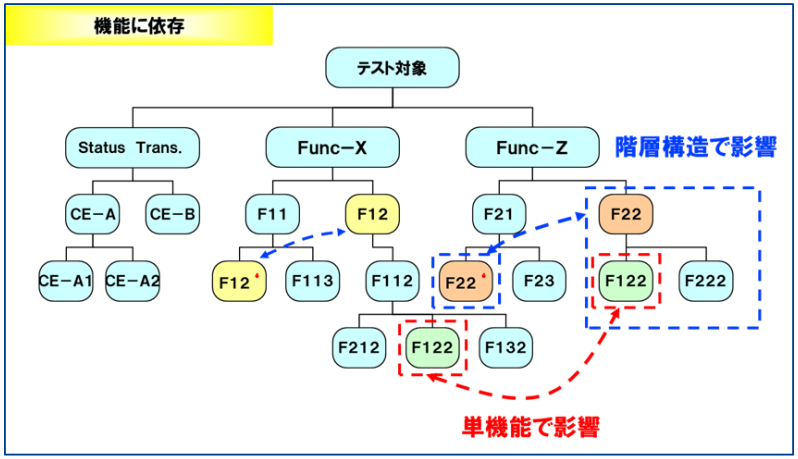


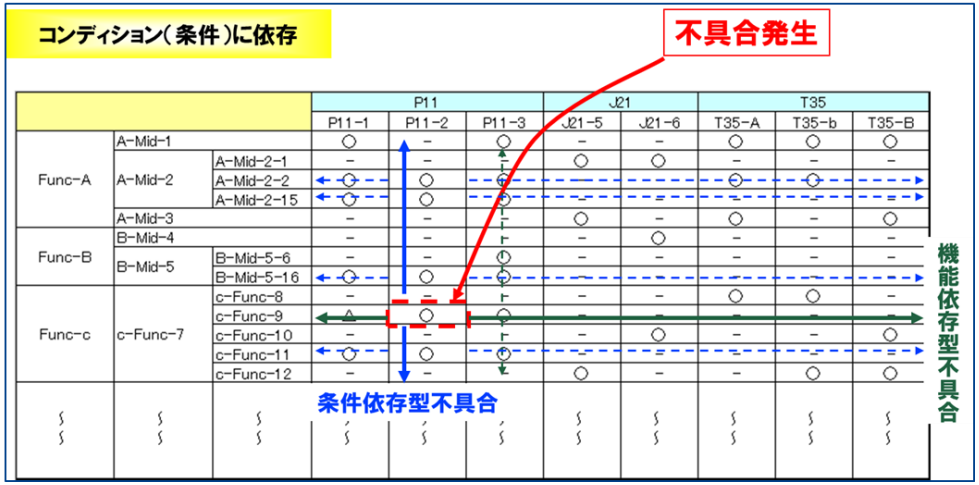
設計工程
- 仕様漏れ
- 仕様の間違い
- 仕様不備

製造工程
- コーディングミス
- コーディング漏れ
- コーディング想定外
- デグレード

OutSide View（動的傾向分析）

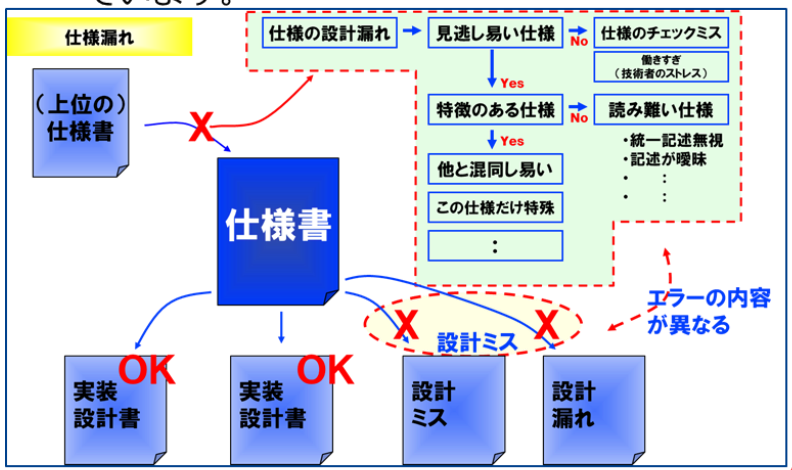
- テスト対象のCFT（機能樹形図）で機能間の影響を確認したり、テスト因子/水準（テストマトリクス）から依存関係を分析し、テスト対象の弱点に見当をつけてテストの実行を動的に行っています。主に動的傾向分析は、テストの実行工程期間中に実施します。

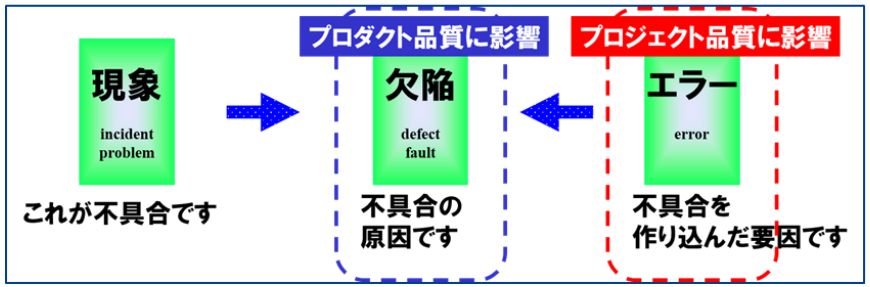


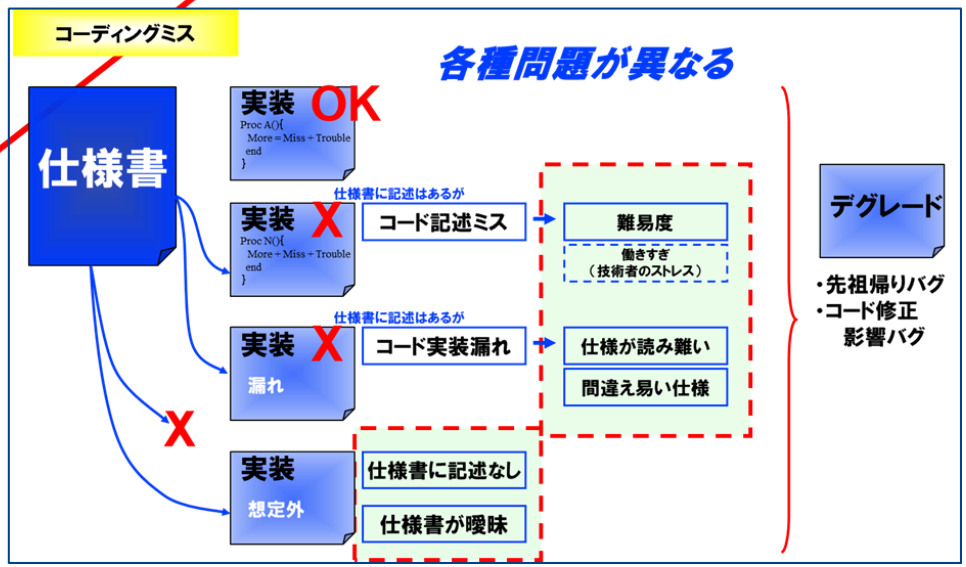


InSide View（静的欠陥解析）

- 検出した欠陥を作りこんだ要因を、「設計工程」と「製造（実装）工程」それぞれでエラーの分析を静的に行っています。







こんなプロジェクト（仮名）に遭遇

- あるプロジェクトで、不具合が収束しない状況に陥っていました。

プロジェクトの背景や前提条件
- ３ｒｄパーティのテストベンダとして、機能テスト（結合テスト）からシステムテストを担当。担当テスト工程以前の情報はない。
- 機能テスト（結合テスト）の工期は終了し、システムテストのテスト実行を実施中。
- テスト対象は、組込みシステムで、（一応）WF開発。

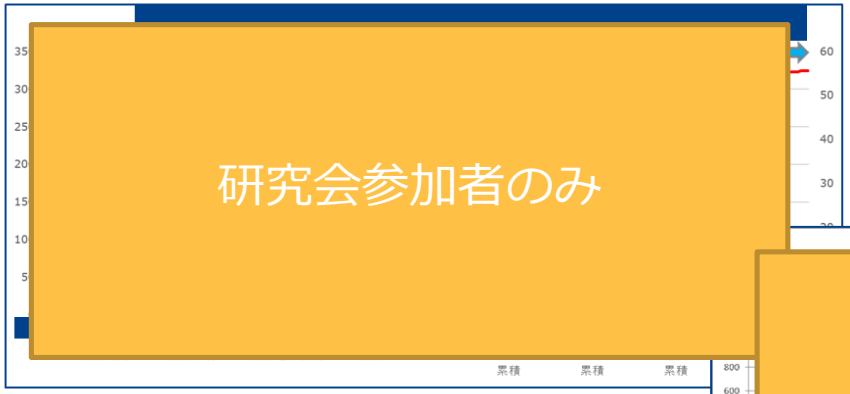
プロジェクトの症状
- テストベースの品質が悪く、システムテスト工程なのに新規開発以外の既存部分にも不具合が多数発生。
- ブロックバグ（テストを阻害するバグ）が多発して、未実行テストケースの積み残しが多く発生。そのため、テスト実行工程の後半で不具合発生にリスクが高まっている。
- （システムテストとして）目的の不具合が検出されているか不明の状態。
- 不具合の修正が適切に行われているか不明。エンバグの混入リスク。

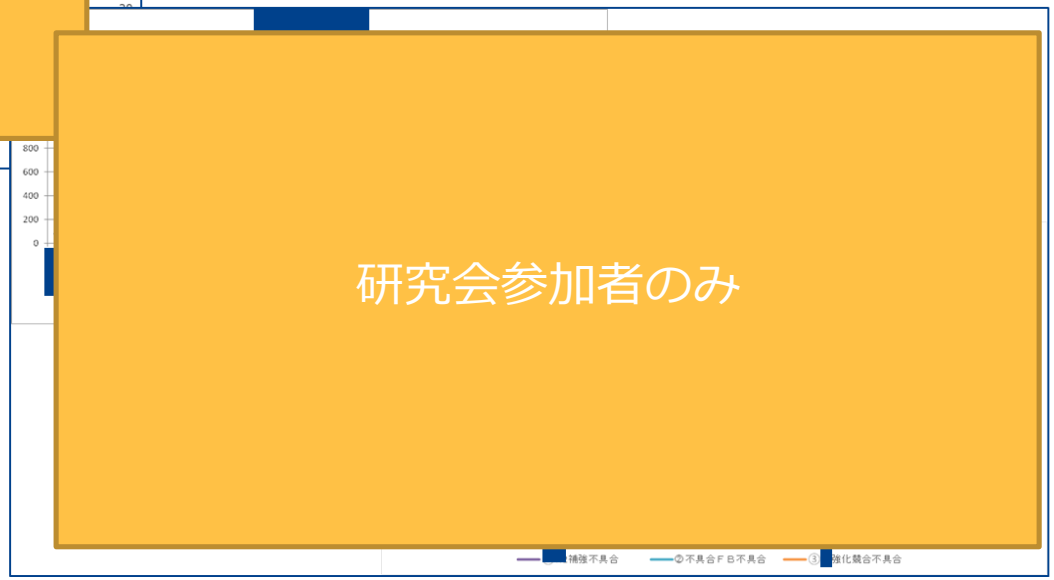
これらに対して

テストチームへの要求（ほとんどクレーム）
- そもそもテスト設計が不完全じゃないの？（テスト観点を網羅していない）
- テストのマネジメントが悪いんじゃないの？（効率的にテスト実施していない）
- 機能テストが終わっているのに基本的な不具合が発生しているのは、機能テストが不十分だったんじゃないの？
- で、いつ終わるの？

プロダクトの状況を分析を試みる

- 不具合の発生傾向や、タイミング、機能別、インパクトなど、さまざまな角度で分析を試みるが





ここから導き出せる状況は、「システムの品質が悪い」としか言えない。
多少、機能別で品質状況は分かるが、
テストを終わらせれるかわからなくなった。

うーん、困った？！

ここでＯＤＣ分析登場！！（道のり紹介）

Step1	・ODC分析の紹介、利点（おすすめポイント）など ・プロジェクト状況を背景に絶対にやるべきと熱弁
Step2	・プロジェクトの状況や開発プロセスなどから、適用までのアプローチを計画
Step3	ステークホルダー毎に説得 ・プロジェクトオーナーは、状況が打開するなら「いけいけGoGo」 ・開発リーダーは、 「入力工数がかかるよね」や「ちゃんと属性付与できるかな」など不安材料の説明を要求。（プロジェクトオーナーを後ろ盾に）準備や説明、フォローを約束
Step4	・適用するODC属性の抽出や簡易的なマニュアル、バグトラッキングシステムに属性フィールドを追加
Step5	・検出(Opnener)から属性付与を開始し、次に改修(Closer)を開発者が付与
Step6	・やっとODC属性の分析開始。ここまで約２Weekで。

適用した属性紹介①-検出(Openner)

今日は、そんなに重要でない話

- 全部のＯＤＣ属性は適用していません。まず、最低限の分析を行うために属性を設定。（その当時の情報なので、多少理解不足な点があります。）

検出(Opnener)		
Defect Removal Activiteis(検出工程)	Trigger（欠陥を発見した要因・観点）	Impact（重要度）
どの工程で検出したのか？	どういう操作・観点で検出したのか？	どれくらいの影響か？
バグトラキングシステムを流用	新規入力	バグトラキングシステムを流用
01_システム要件分析	レビュー/静的解析	致命
02_システム方式設計	デバッグ	重
03_XXXXXX	基本機能（標準）	軽
04_XXXXXX	基本機能（オプション・パラメータ変更）	参考程度（サジェスチョンなど）
05_ソフトウェア要件分析	特定の操作/手順	
06_ソフトウェア方式設計	エラー/例外処理	
07_ソフトウェア詳細設計	状態遷移	
08_ソフトウェア構築	複合動作	
09_単体テスト	負荷/ストレス	
10_ソフトウェア結合テスト	大容量	
11_ソフトウェア適格性確認	高頻度、繰り返し	
12_XXXX	通常状態復帰（電源断・通信断・再起動など）	
13_XXXXXXX	構成変更	
14_システム結合	性能	
15_システム適格性確認テスト		
16_出荷後		
90_構成管理		
98_混入工程不明		
99_XXXXXXX		

ほとんど機能テストですね

テスト観点や、テストタイプの情報で満足できそう

Triggerは新規の属性付与だけど、その他は今までの属性を流用できるので、あまり違和感がない。

XXは、プロジェクト固有の内容なので隠匿
属性付与は、既存のバグトラキングシステムのFieldを流用や新規に追加した

適用した属性紹介②-改修(Closer)

今日は、そんなに重要でない話

- 続き

改修(Closer)				
Target（発生トリガー）	Defect Type（障害タイプ）	Qualifier（障害実装タイプ）	Age（履歴）	Source(出所)
どこが悪かったのか？	何を間違ったのか？	どのように間違ったのか？	どの時点で間違ったのか？	発生源はどこか？
バグトラキングシステム流用・追加	新規入力	新規入力	バグトラキングシステム流用	バグトラキングシステム流用
正常系	値の代入、初期化ミス	実装間違い	新規	新規モジュール
異常系	値の条件チェック	変更ミス	既存	改造モジュール
境界条件[設定上の境界値、最大値など]	プログラムロジックミス	実装し忘れ	仕様変更	流用モジュール
誤操作・特殊操作	共有リソースなどのアクセス制御ミス		デグレード	COTS(プラットフォームなど)
機能競合	モジュール間のインターフェースミス			
高負荷	クラスのメソッド、インスタンスミス			
特定条件	データ構造の定義ミス			
特定のタイミング	GUI			

これ全然Targetでない。（参考にしないで下さいね。）
Targetの間違いに気付き、その後は訂正（なぜ間違ったのかは、分析していない）

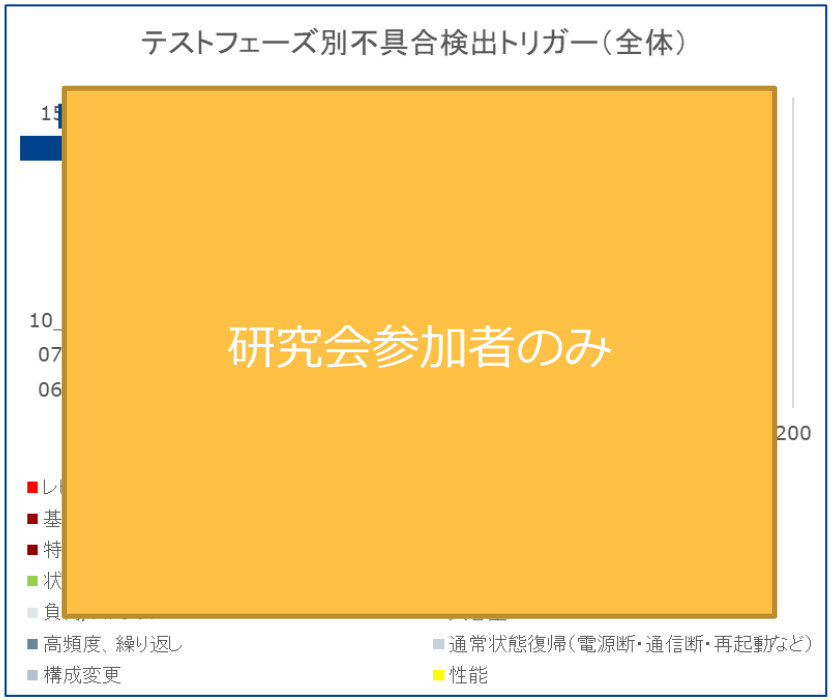
ここはスクリーニングで対応しようっと。
Inside　Viewが使えるかも。

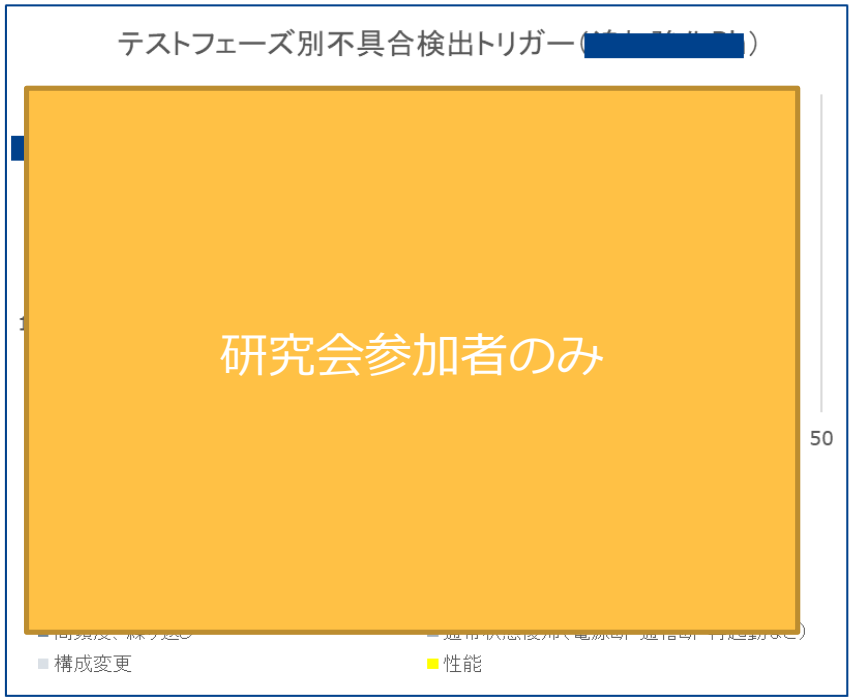
新規の属性付与は、2種類。ちょっとこの2つは、注意が必要かな。

で、結果はどうだったの？ ①（Openerの話）

今日は、そんなに重要でない話

- 分析が未成熟だなぁ





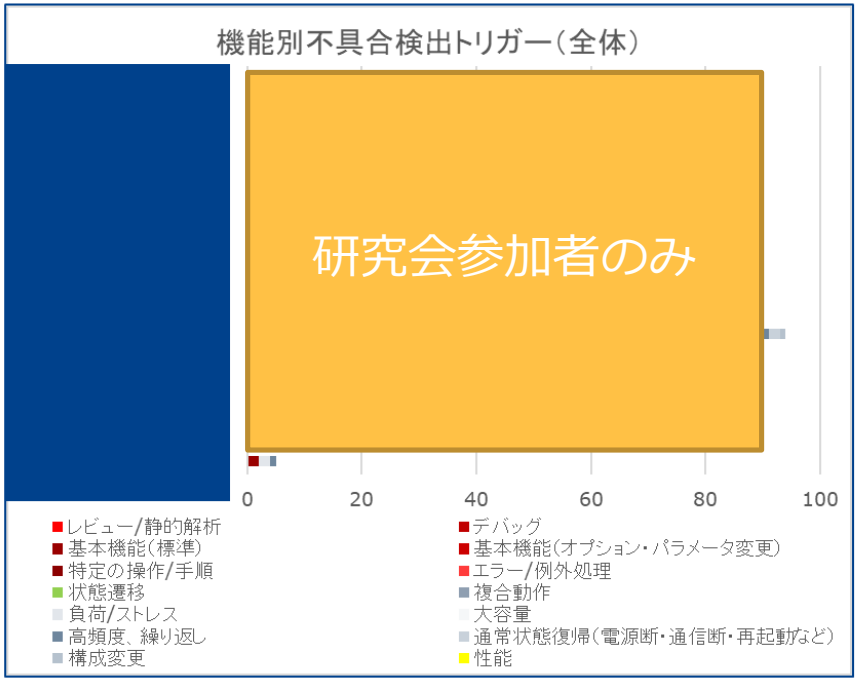
単純操作での不具合が減って、いじわる操作の不具合が多くなった。「いじわる」テストの効果が見えはじめる。
テスト少しステージが進んだ感あり。

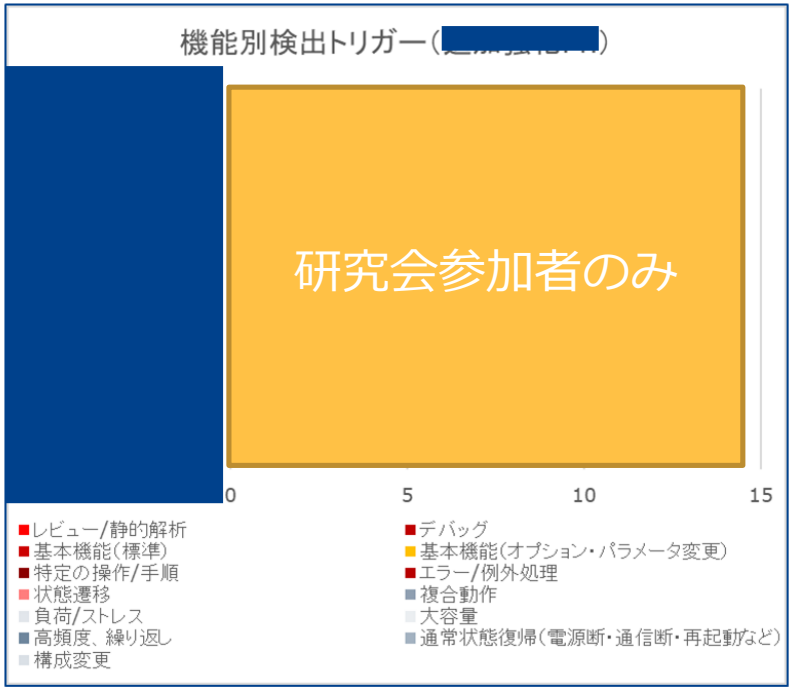
SQiP Software Quality Profession

で、結果はどうだったの？ ②（Openerの話）

今日は、そんなに重要でない話

- やっぱり、分析が未成熟だなぁ。



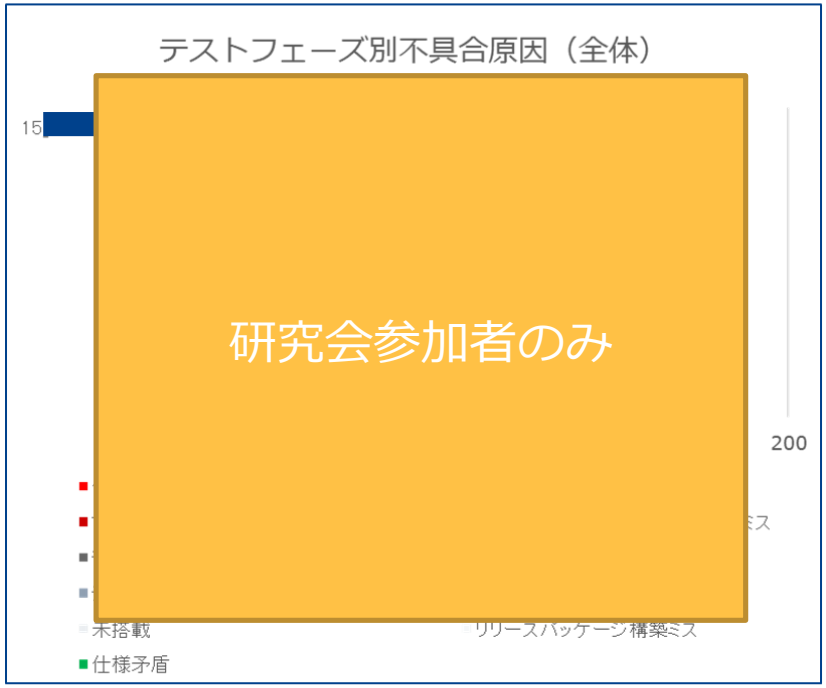


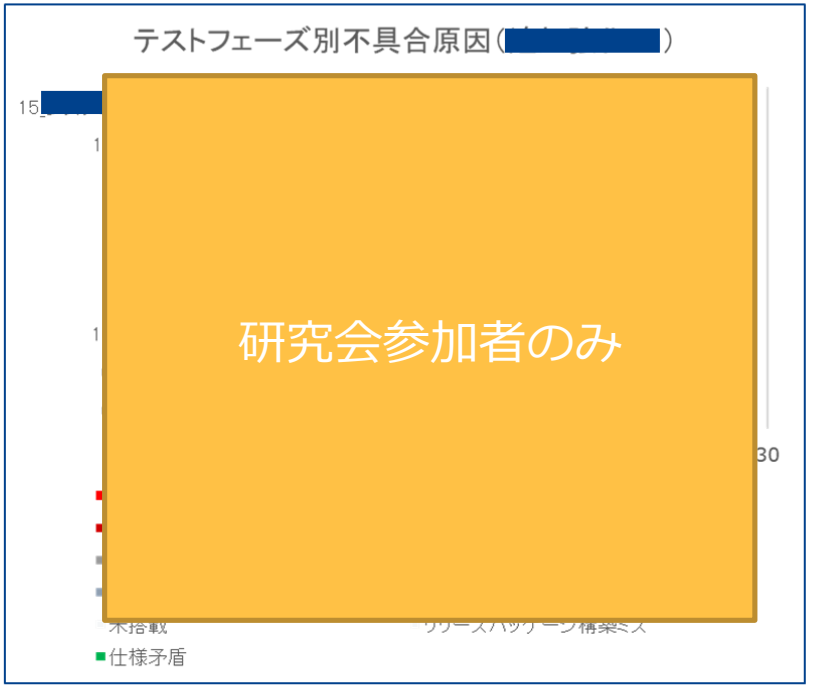
（機能別）非機能観点の不具合が検出されはじめるが、特定の機能は依然として機能要件の不具合の検出が続く。
対応処理の必要性あり。

で、結果はどうだったの？

今日は、そんなに重要でない話

- ぜんぜん、いけてない。



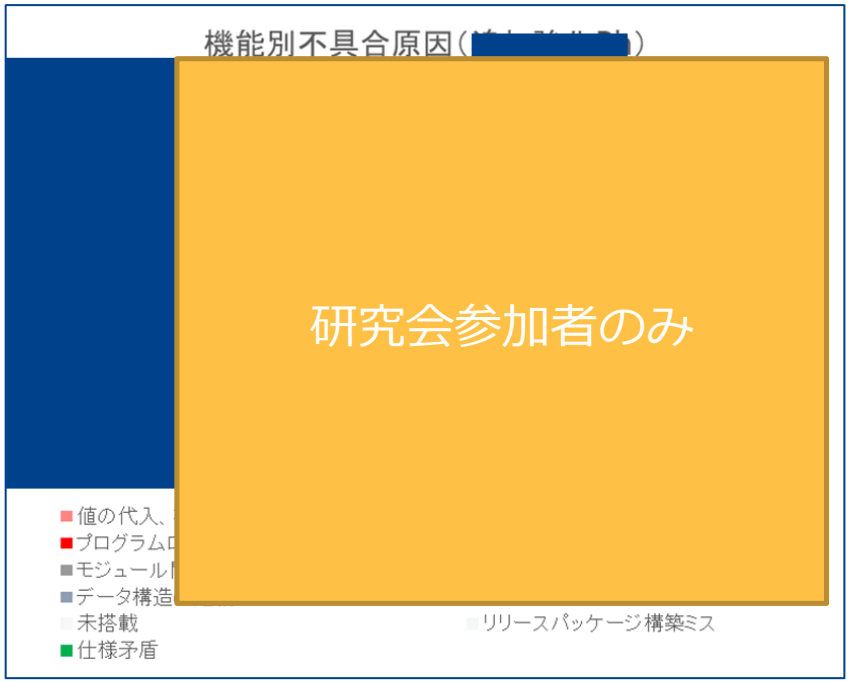


単体テストが弱いかも。（静的解析ツールでも、かなりの効果がありそう。）これでは結合テスト以降に影響出るのも当然ですね。

で、結果はどうだったの？

今日は、そんなに重要でない話

- ちょっと大胆に（ODC分析の）見直しが必要か。



（機能別）単体テストレベルの原因が多い。少し追加したテストは、複雑な不具合の検出もあり。まだまだ、予断を許さない。

本題の話をしよう！！

（「テストチームがODC分析を適用してみた」の大きな流れは説明の通り。）適用の流れから、ポイントを振り返ってみた。

■：背景　□：適用思考/過程　★：気づき

ポイント１：導入動機と切っかけ

■テスト対象の品質状況が説明できない。テストチームが、どの機能がどの程度の不具合を検出していること（事実）しか見えないため、少し客観的な説明が求められた。

□そういえば「ODC分析」って先進事例があったよな。開発チームは忙しいから（と言うかもしれない）、テストチームが主管にするとODC分析の提案は通り易いかも。適用準備はスモールスタートで。

★気づきとなった点は、あんまり無いかな。（責任をもってやりきる意思が重要）

ポイント２：省エネ作戦

■まともに適用を考えたら、時間と労力が多くかかりそう。忙しい（と言っている）人たちに新たな負荷は敬遠するから、なるべく既存の仕組みを利用してと。

□プロジェクトで利用している不具合管理票からODCで有効な属性を抽出してみよう。それで足りないものは、理解しやすいものと効果が大きいものに限定しよう。

★気づきとなった点は、

①「ODC属性」を全部適用するとしなかったので、なぜこの属性が存在し、また必要/不必要と考える機会となった。これが、プロジェクトのプロセスなどの理解を進める結果となった。

②既存の取り組み（テストの場合は、特に“Opener”）を極力残すことに注力したため、既存の取り組みも必要/不必要と考える機会となった。

本題の話をしよう！！

（「テストチームがODC分析を適用してみた」の大きな流れは説明の通り。）適用の流れから、ポイントを振り返ってみた。

■：背景　□：適用思考/過程　★：気づき

ポイント３：開発者とのコミュニケーション

■ODC属性のCloserは、開発者に依頼した。「Defect Type（障害タイプ）」「Qualifier（障害実装タイプ）」は、新しい項目なので、正確に入力してもらえるようにする工夫を検討した。

□あまり斬新的じゃないが、ODCの属性説明資料やサンプルを開発者（もしかしてポイント？）と作成した。

★ODC属性自体を勘違いしたこともあったが、付与されたODC属性について、どのような背景で属性選択し付加したのかを、思考の過程を確認することができた。複数の思考過程をモード化することで、 ODC属性付与についてのガイドラインのベースが構築できた。（Closerを理解できていないテストチームが担当したことが良かったかもしれない）

ポイント４：テストチームの再教育

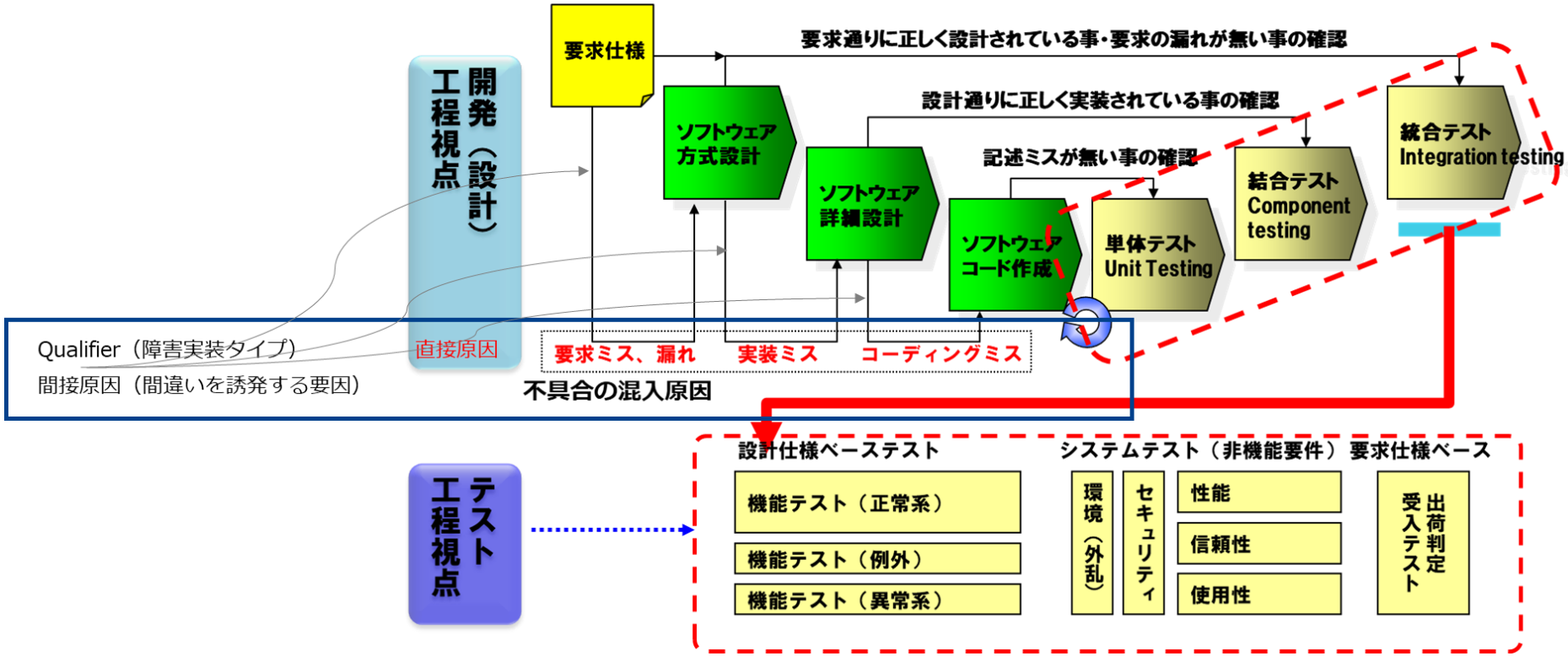
■ODC分析を適用するために、開発プロセスだけでなくテストプロセスを熟知する必要があった。プロダクトの開発期間や製品ライフサイクルも長いため、各プロセスに認識は形骸化や暗黙知化が進んでいた。

□ODC分析の目的としてテストチームの責務範囲で品質説明が求められていた。業務の定型化が進んでいたが、定性的な目的など再認識の機会となった。

★プロダクトの特徴やプロジェクトへの要求が定型化するなか、抜け漏れ業務の再確認ができ形骸化に歯止めがかかった。ODC分析がテストプロセス（の成熟度）も丸裸にした。

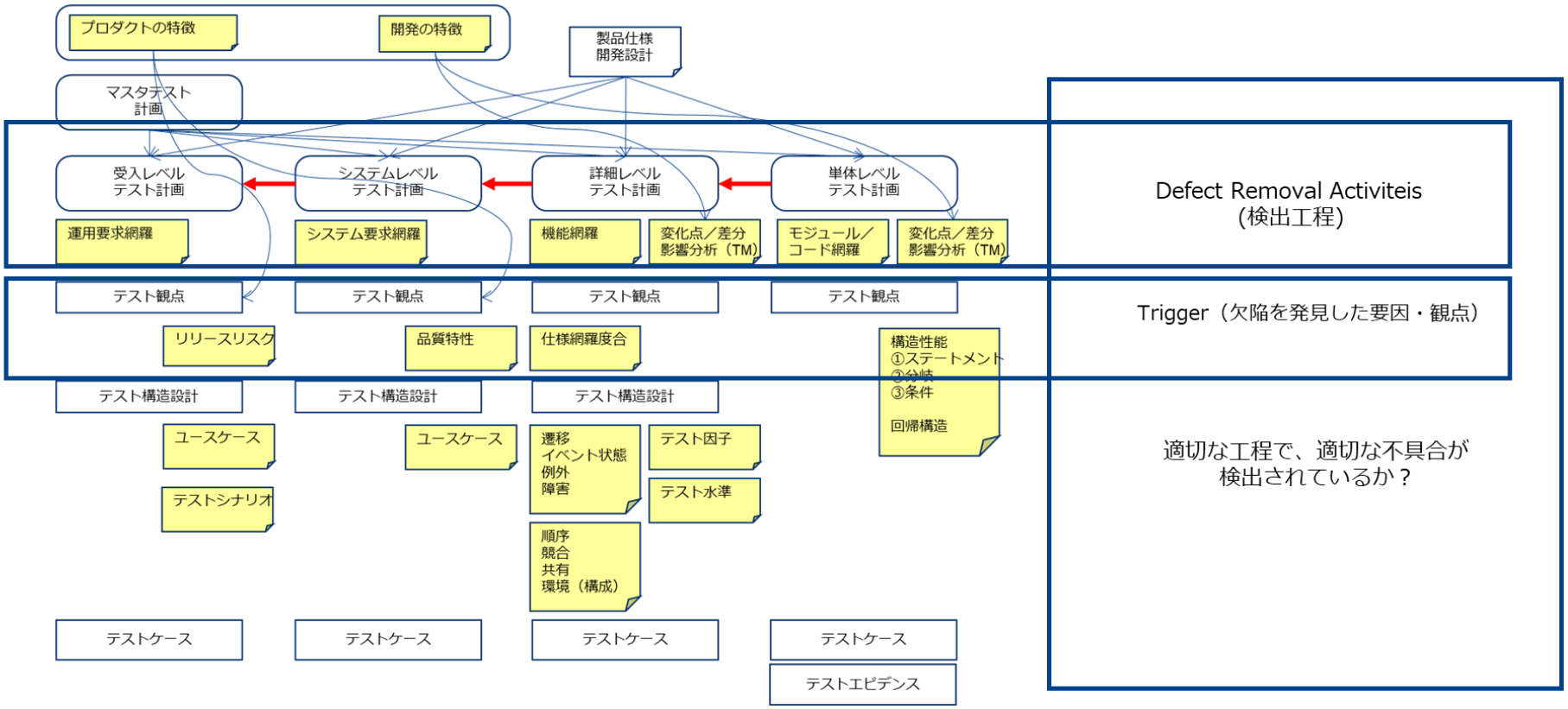
テスト工程からのアプローチ①

- テスト工程からのODC分析アプローチ①。開発工程に潜むエラーの存在を知る。



テスト工程からのアプローチ②

- テスト工程からのODC分析アプローチ②。テストレベル、テストタイプが目的を果たしているか。マスターテスト計画が妥当か、正しく実行されているかも確認できる。



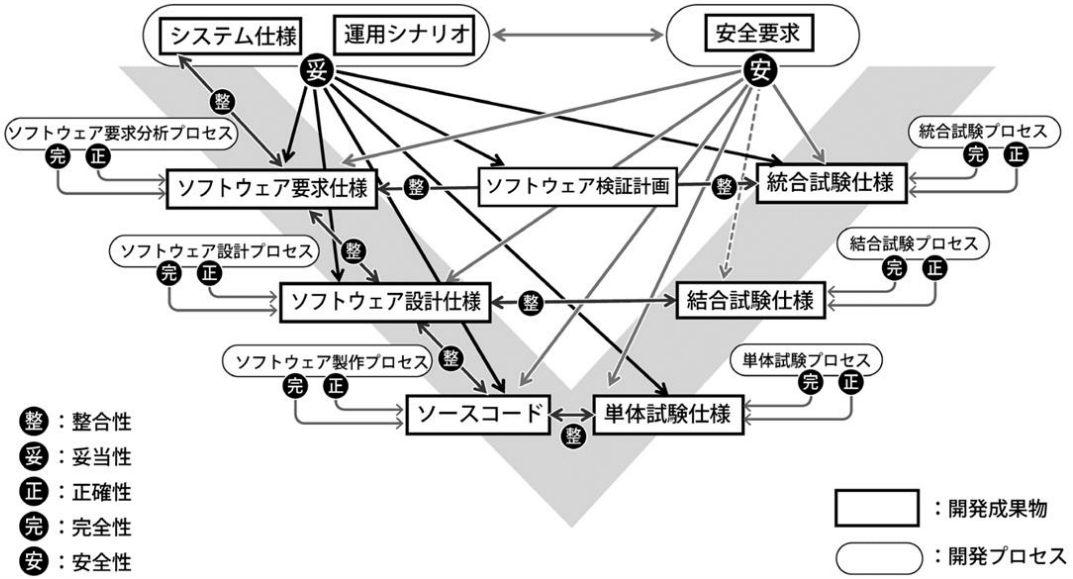
テスト工程からのアプローチ③

- （プロセス全体であるが、）Qualifier（障害実装タイプ）やDefect Removal Activiteis(検出工程)、Trigger（欠陥を発見した要因・観点）でもValidation（妥当性確認）ができる。

- マスターテスト計画や各レベルテスト計画の妥当性やテストプロセスの問題点も確認（けっこう大切）できる。

- 属性付与の思考過程を整理する段階で、見えていなかったプロセスが（少し）可視化できる。
これをきっかけにコミュニケーションが少なかった開発者とテスト担当者の議論の場になった。

・本論には記載していないが、1回目の分析は上手く行かなかった。しかし2回/3回と繰り返すことで、分析活動だけでなく開発プロセスのロール（責務）に認識が高まった。



JAXA IV&Vガイドブック【虎の巻】 発行:2013年3月
https://repository.exst.jaxa.jp/dspace/handle/a-is/19895

EOF

ご清聴ありがとうございました。