FUJITSU THE POSSIBILITIES ARE INFINITE

アジャイル開発の得失

ー代表9プロジェクトの開発体験を元に考察ー

テストシンポジウム　2006
2006年1月30日

馬屋原　篤雄（富士通株式会社）
t-umayahara@jp.fujitsu.com

目次

1. 背景

私の歩み
- 1968～　汎用OS開発（第二次オンライン（科目間連動）I／Oドライバ）
- 1972～　汎用OS開発（JOB管理担当）
- 1991～　汎用OS開発（ストレージ製品）
- 1996～　汎用OS開発（JOB/JES管理）
- 2005～　汎用OS開発（JOB/JES),GS統合

04年下期　経営幹部よりアジャイル開発プロジェクト募集
　　9プロジェクトが参画
05年6月　成果発表会
　　品質の是非について興味
05年10月　品質面での評価（ヒヤリング取材）

2.目的

**アジャイル開発に対し、
以下の点について考察する**

1. 著しく効果があった点
2. 作り込まれやすいバグ
3. アジャイル開発でのテスト法

3.著しく効果があった点

1）テストファースト→高品質

従来

華麗さ、自動制御の作り込み等

設計に専念 → 作成に専念 → テストに専念

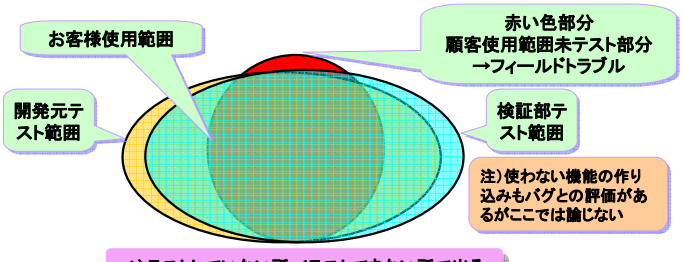
結果　テスト出来ないコードもあり→将来バグ

アジャイル

設計＋テスト法 → 作成＋レビュー → テストに専念

結果　テストルートを通過するコード作成→残留バグ0
単純バグも早めに検出可能

3.参考　関連して　フィールドバグは何故出る？



1）テストしていない所／テストできない所で出る
→ 実はテスト出来ないコードを作成している
→ アジャイル：テストできるコード作成→画期的

2）思い込み、考慮不足部分で発生

3.著しく効果があった点

2）操作性／運用性評価

従来

マニュアルで説明（数百ページ）

結果 実感が沸かない。提供後に改善要望あり

アジャイル

実際に使って貰う→改版し再度使って貰う（百聞は一見にしかず）

結果 実感に伴うさらなる改善が可能

3.著しく効果があった点

3）開発スピード（平均的に3～4倍）

従来

工程毎に作業内容を詳細に記述

結果 工数圧迫。改版漏れがあり再利用頻度少

アジャイル

必須要件のみ記述。動作を優先。

結果 プログラム開発に専念。工程短縮

3.著しく効果があった点

4）シンプル

従来

凝ったコード作成

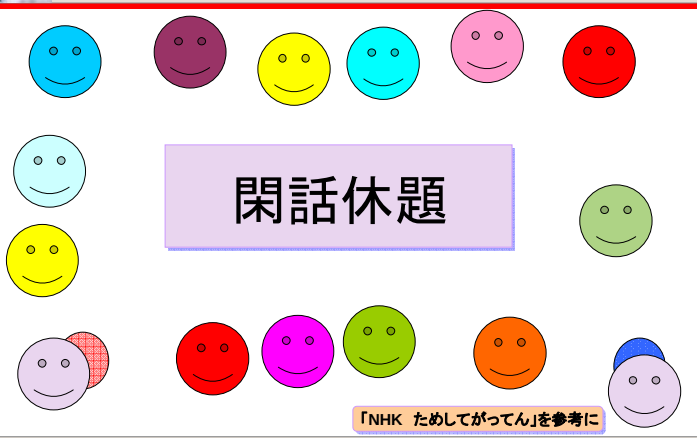
結果 その人しか分からない。改造ミス誘発。

アジャイル

シンプルが要件

結果 誰でも理解できる。改造が容易。

レビュー実験（●のパス数は?）



「NHK　ためしてがってん」を参考に

なぜ、見つからない

原因：人の習性、視覚、盲点

注意力を高めると見落とす

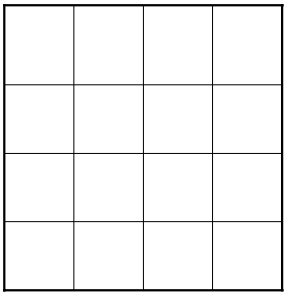
①パス数が正しい人
②パス数が誤っている人
③他のものに気がついた人
④他のものに気がつかなかった人
⑤ルール違反数まで気づいた人

分かってて
つまづいてこける
踏み外す

だから、単純設計が必要。

参考：レビュー視点実験

正方形はいくつ？



なぜ、思い込む

原因：人の習性として、訳の分からない
ことが嫌い／脱出したい⇒安心感
思い込み→誤り→エラー

対策：「それは本当か」を繰り返す
①忌憚なく指摘してくれる人のレビュー
②再考の機会を持つ（回顧）
③短期リリース／ユーザテスト

XPプラクティスが有効

4.作り込まれ易いバグ

1）テスト要因不足

従来
要件、関連要件も合わせて抽出
結果 必要十分なテスト項目。テスト漏れ微小

アジャイル
要件重視で要件の必要性条件に偏りあり
結果 十分性（ソフト／ハード環境、関数）条件漏れ

4.作り込まれ易いバグ

2）レビュー漏れ

従来
各工程で対面レビュー
結果 網羅性あるレビュー

アジャイル
ペアプロ、気になる所のレビュー。気づき資材（仕様書等）不足
結果 レビュー漏れ

4.作り込まれ易いバグ

3）仕様／正否判断の揺らぎ

従来
一貫した仕様書
結果 判断根拠明確

アジャイル
要件仕様を元に複数人で実装
結果 要件仕様に書かれていない部分で微妙な差異

4.作り込まれ易いバグ

4）要望の受け入れ過ぎ

従来
開発当初に要件確定
結果 提供まで殆ど仕様変更なし。

アジャイル
数量的、提供間際まで新規要件が来る
結果 レビュー／テスト不足によるバグ

5.アジャイル開発でのテスト法

通常は正常動作をしているが、実運用で問題が発生する可能性があるもの

1）例外処理と後始末

従来
例外処理まで仕様確定
結果 異常メッセージを出力し終了

アジャイル
異様種別毎の振る舞いが曖昧（アジャイルに限らないが）
結果 いつの間にかプロセス不在

参考：異常は必ず発生する

異常は必ず発生する。→異常時考慮必須。

論理ミス（他からの破壊含む）
マシン障害
関数（システムコール）異常
CANCELコマンド投入
クラスタダウン
ページングI／O失敗
資源不足（ファイル容量，メモリ等）
人為ミス

→ リカバリ論理必須
［ポイント］
顧客被害の影響範囲の局所化。
（異常処理論理の局所化ではない）
→異常発生を局所化できる構造設計要。

しかも、いつどこで発生するか分からない（ソフトウェア開発の宿命）。

→ それらを踏まえたリカバリ論理構築 → チェックリスト化

5.アジャイル開発でのテスト法

2）性能

従来

走行ステップを考慮した設計

結果：設計通りの性能

アジャイル

簡易な言語でミスの介入は少ないが性能は疑問

結果：大量データだと極端に性能劣化

5.アジャイル開発でのテスト法

3）セキュリティ

従来

最終段階でセキュリティ確認

結果：設計通りの保護

アジャイル

短期リリースで思わぬセキュリティホールの介入

結果：セキュリティ問題発覚

5.アジャイル開発でのテスト法

4）その他考慮すべき点

- 大量データと使用資源量、キュー制御
- 大量イベント／リトライとメッセージ
- 排他制御とデッドロック
- 連続運転
- タイミング絡みの検証
- 過負荷試験
- 時間監視、イベント監視
- 人為ミス
- タイムアウト
- 例外事象
- プロセス間同期
- 新規使用関数

最後に

ソフト開発技術者として謙虚に

それで十分か、自らの限界はどこか、
を常に振り返り日々挑戦。

ご静聴ありがとうございました。

FUJITSU

THE POSSIBILITIES ARE INFINITE