

SPI Japan 2016

現場から始める アジャイルの技術プラクティス

～ ユニットテストから勝手に始めよう ～

2016年10月12日

富士通株式会社

岡本卓也

- 自己紹介と背景
- アジャイルについて
- 技術プラクティス
- 導入のためにやったこと
- 導入のときに悩んだこと
- 導入の効果
- 導入したプラクティス
- まとめ

■自己紹介と背景

■アジャイルについて

■技術プラクティス

■導入のためにやったこと

■導入のときに悩んだこと

■導入の効果

■導入したプラクティス

■まとめ

自己紹介

■ ソフトウェア開発者 (19年目)

- 開発チームのマネージャ

■ アジャイルとの関係

- 2002年、XPと出会う

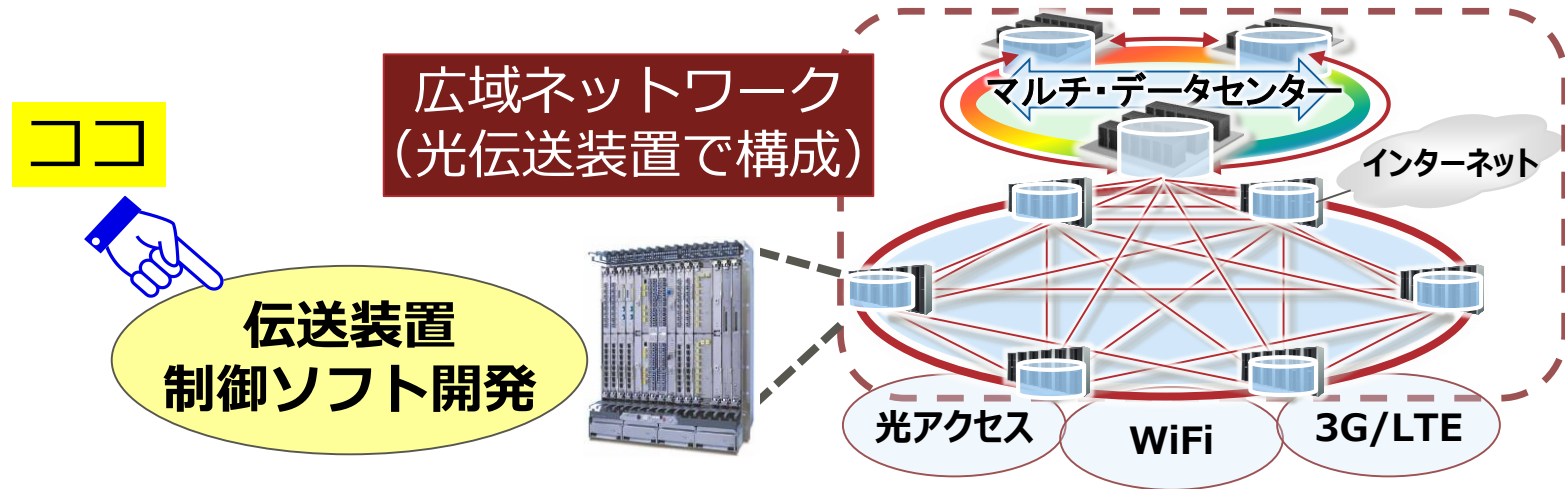
(長い間、一人で悶々とし続ける)

- 2013年からAgile Japanに参加(聴講者)

- 現在は、アジャイルを実業務へ適用/推進する為に奮闘中



■ 伝送装置の制御ソフト開発



■ 特徴

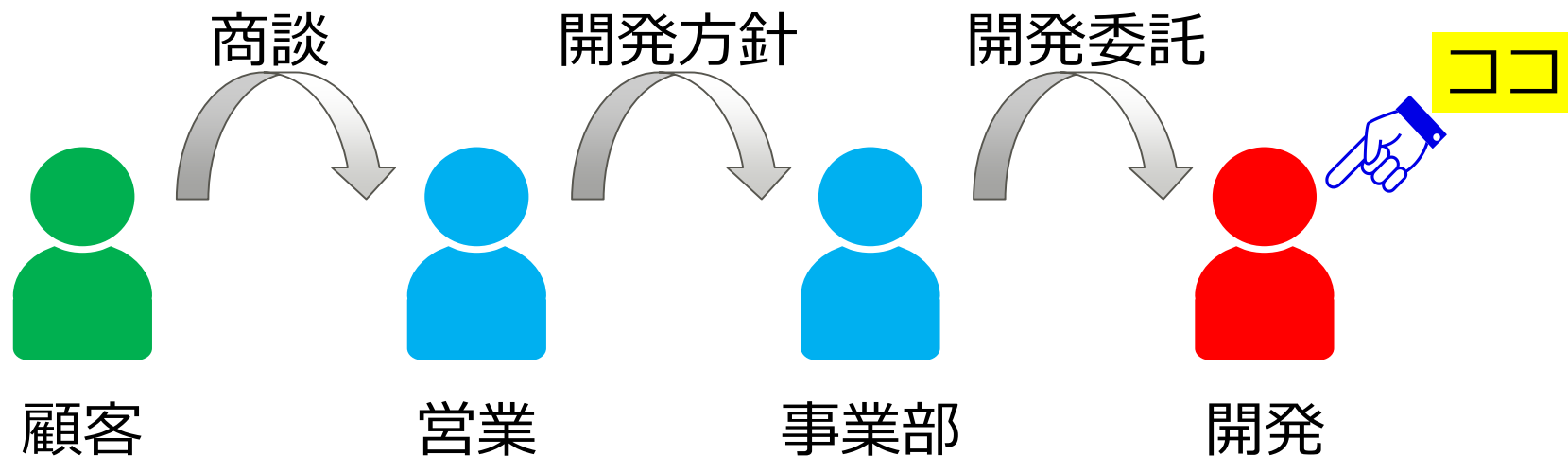
- 大規模 (数十人、数カ月)
- 厳格なプロセス (基本はWF)
- HW/SW 同時開発 (組み込み的)

WF: Water Fall

HW: Hardware
SW: Software

■特徴 (続き)

■顧客までの遠い距離



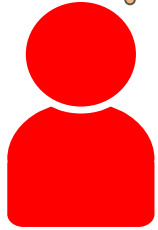
いろいろとアジャイルには不利な条件

- 自己紹介と背景
- **アジャイルについて**
- 技術プラクティス
- 導入のためにやったこと
- 導入のときに悩んだこと
- 導入の効果
- 導入したプラクティス
- まとめ

アジャイル導入あるある

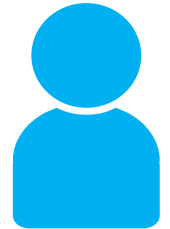
一括請負契約だから...

開発プロセスが...



開発

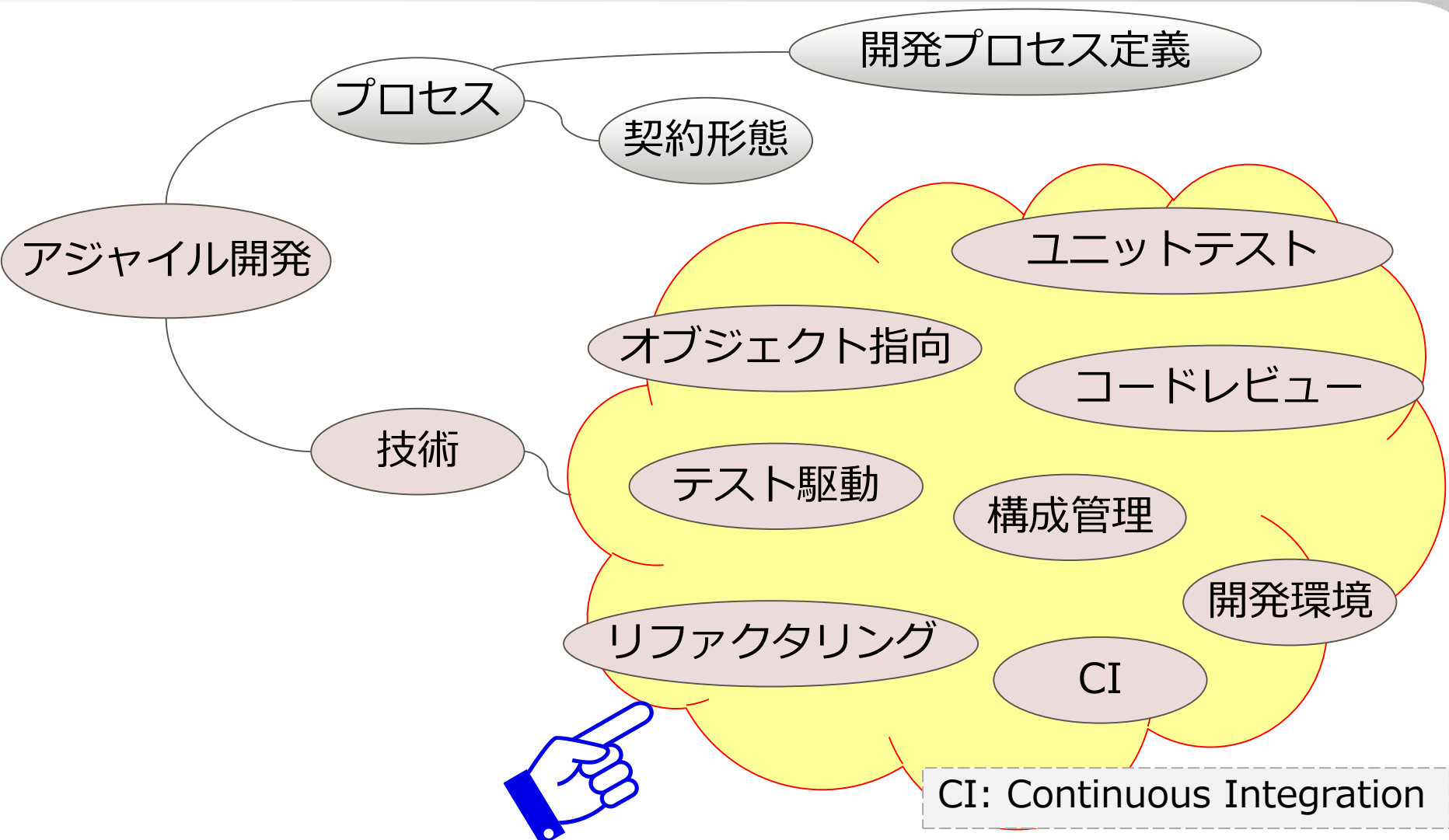
アジャイルでやって良いよ。
出来るよね?



(仮想)
偉い人

え!?

アジャイル導入の壁



CI: Continuous Integration

技術の壁は現場で解決しないとダメ

- 自己紹介と背景
- アジャイルについて
- **技術プラクティス**
- 導入のためにやったこと
- 導入のときに悩んだこと
- 導入の効果
- 導入したプラクティス
- まとめ

アジャイルの技術プラクティス

ペアプログラミング

テスト駆動開発

回帰テスト

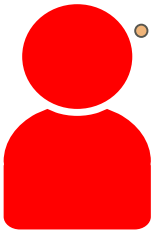
コードの共同所有

ユニットテスト

リファクタリング

シンプルデザイン

継続的インテグレーション



いったいどれから手を付ければ...

岡本

とあるセミナーにて (2013年頃)

アジャイルってどうすれば良いですか?

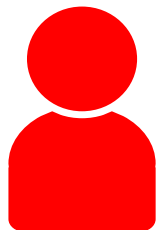
ユニットテストからやるのが
オススメ

なんでですか?

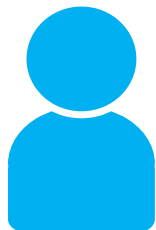
自動化して怒る上司はいないから

(上手いこと言うな~)

しかし、本当は**深い示唆**があった

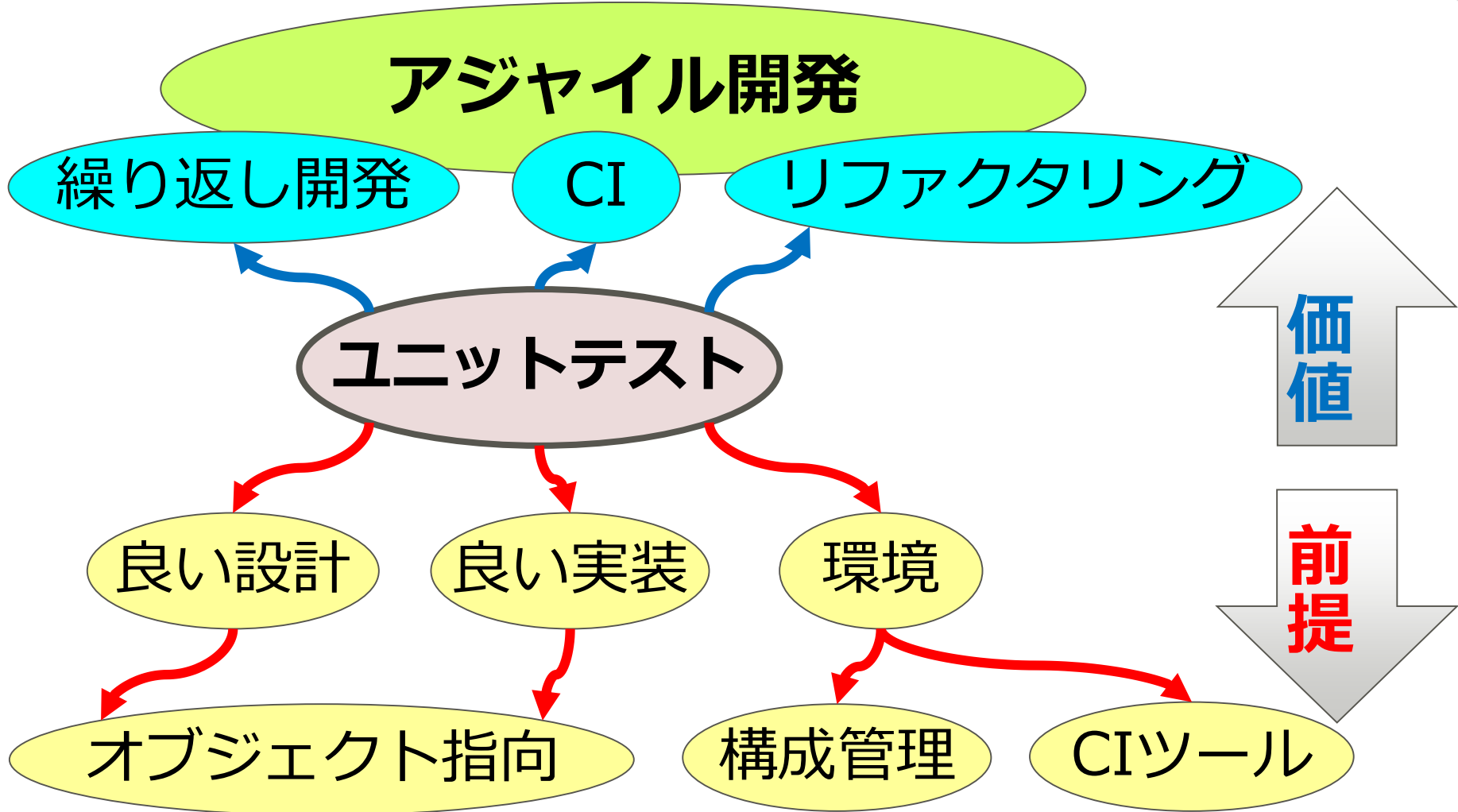


岡本



先達

ユニットテストの前提と価値

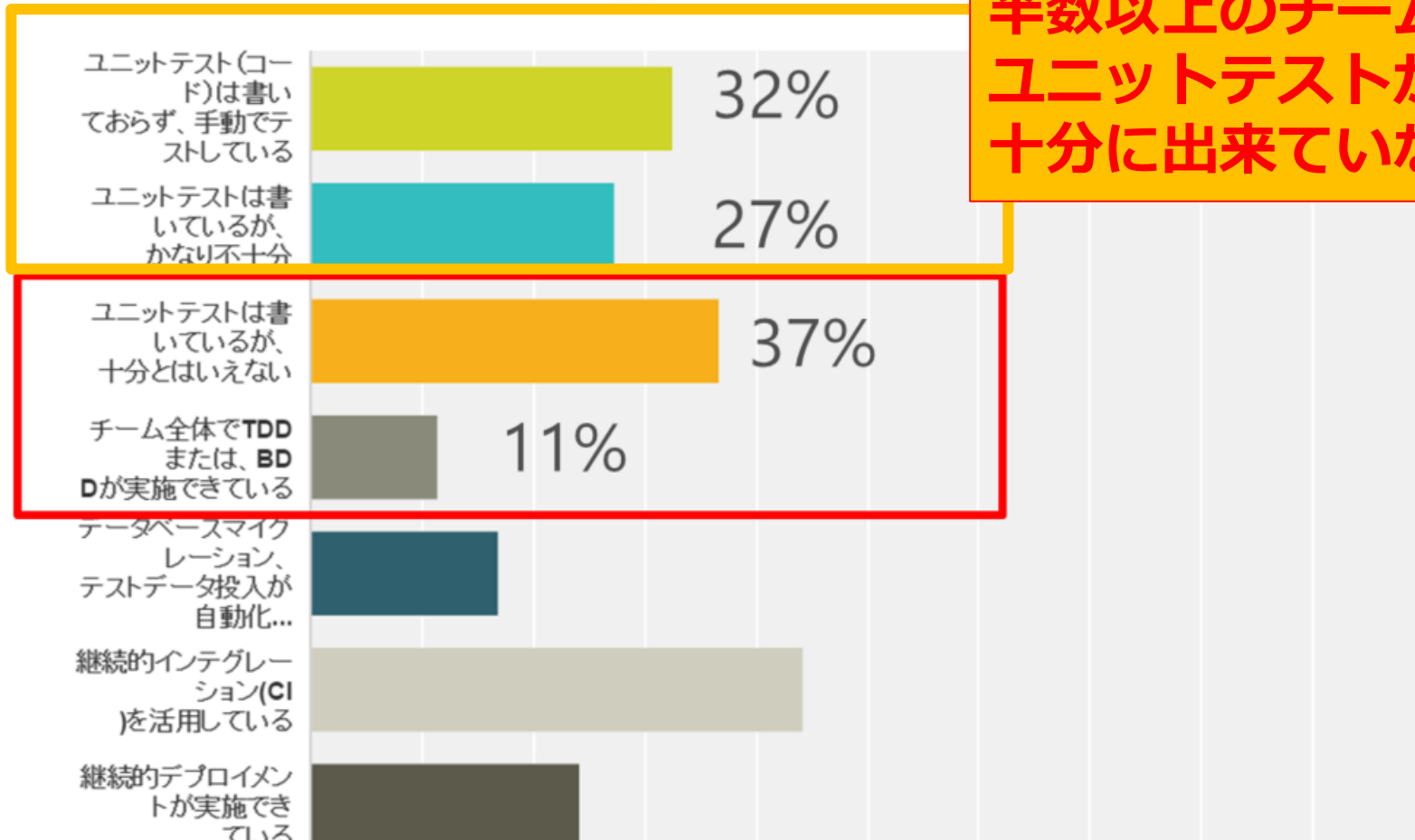


ユニットテストはアジャイルの肝

(参考)アジャイル実践企業の実態調査

御社のチームの平均のチームでテストの自動化
はどの程度実施されておられますか？

回答: 95 スキップ: 5



半数以上のチームで
ユニットテストが
十分に出来ていない

(出典:牛尾剛「アジャイル・DevOps 実践企業サーベイ(2016)」)

- 自己紹介と背景
- アジャイルについて
- 技術プラクティス
- **導入のためにやったこと**
- 導入のときに悩んだこと
- 導入の効果
- 導入したプラクティス
- まとめ

■従来の単体試験(手動テスト)を ユニットテストに置き換えてみた

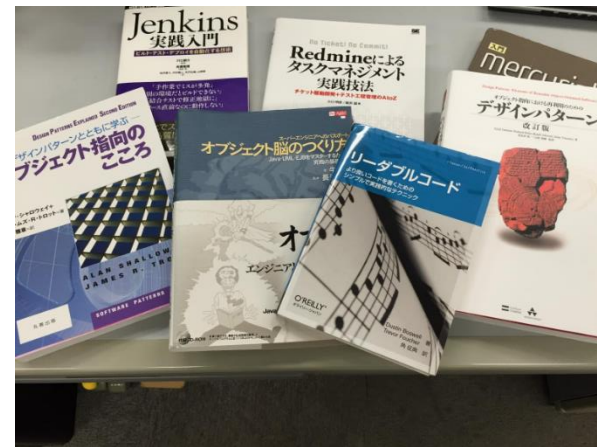
従来	今回
試験項目書	テストコード
試験手順書	テストコード
試験項目/手順レビュー	テストコードのレビュー
試験消化作業	“make check” 叩く or CIで自動的に実施

■これらの内容を開発計画に明記した

- 現実：初めてユニットテスト書く人が大半
- 勉強する
 - 勉強会の開催（エース/キーマンを講師に）
 - テストコード/ノウハウを共有
 - 1h/週程度を、業務と別枠で確保



勉強会



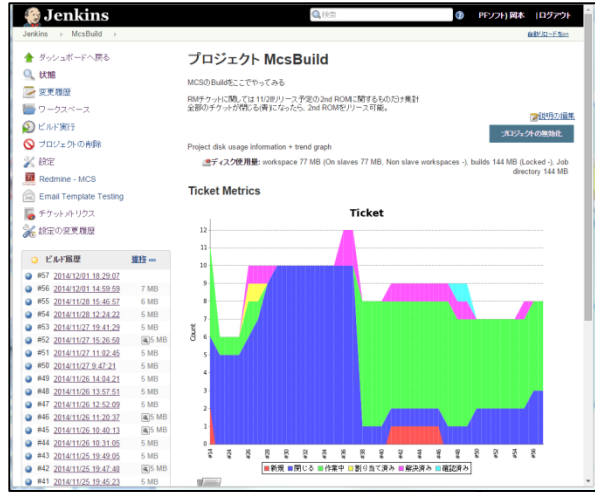
読書
(押し売り)

- 開発マシンの管理者になる
 - スピード感のために自分で動く
- 各種環境/ツールの導入

目的	導入環境/ツール
CI環境	Jenkins
メトリクス測定	gcov/SonarQube
コード管理(VCS)	git/RhodeCode
チケット管理(ITS)	Redmine
テストフレームワーク	Google Test

環境構築の**主導権**を握る

構築した環境の全体像



```
1 file changed: 1 inserted, 1 deleted
Browse files | Expand All | Collapse All

src/ac_tapio/interfaces.proto
Show File | Unified Diff | Side-by-side Diff | Raw Diff | Download Diff
Ignore whitespace | Increase contrast | Hide comments

... 351 351 message acInband {
352 352     required uint32   tlf_idx       = 1; // inbandの着る tlf index (1..128)
353 353     required enum@PrintType grow     = 2; // 着る 種類を赤字(1)で示す
354 354     optional enum@PrintType fa_type  = 3;
355 355     optional uint32   outer_vid     = 4; // 1..4094
356 356     optional uint32   inner_vid    = 5; // 1..4094
357 357 }
```

0 Commit Comment

Create a comment on this Commit

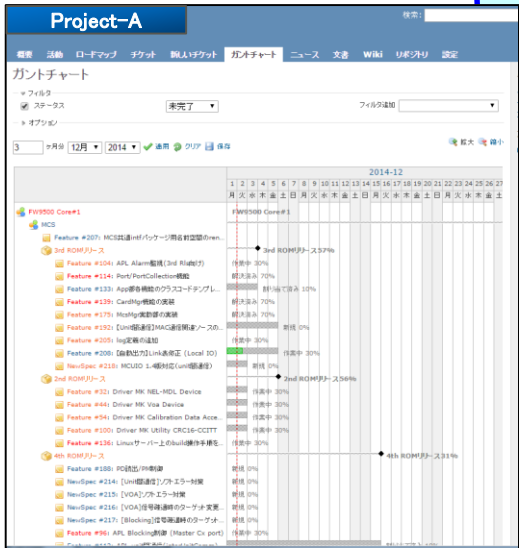
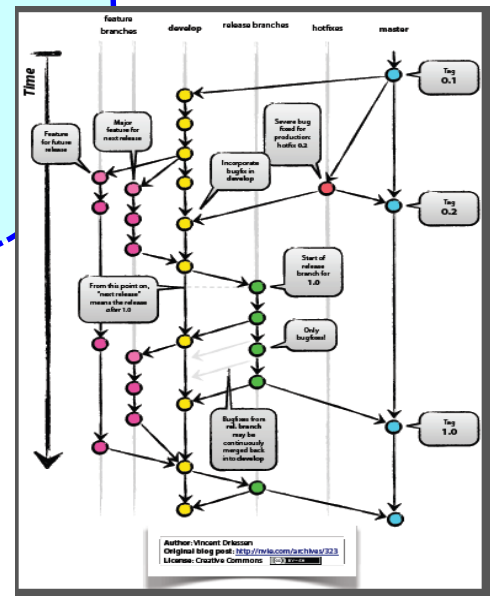
Jenkins (CI)

RhodeCode (コードレビュー)

REDMINE (チケット管理)

git (コード管理)

googletest (ユニットテスト)



- 自己紹介と背景
- アジャイルについて
- 技術プラクティス
- 導入のためにやったこと
- **導入のときに悩んだこと**
- 導入の効果
- 導入したプラクティス
- まとめ



テストコード書く時間ある？

手動でテストした方が早くない？

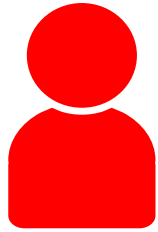
■解決法

- 自分でテストコード書いて試してみた

■結論

- 想定範囲内でテストコードは書ける
- 逆に試験の実行時間は激減する

(具体的な結果は後述)



手動テストと同じ品質出せる?

■ 解決法

- (なし)

■ 結論

- 最初のイテレーションでやってみる
- ダメならそこで手動テストに戻せば良い

(具体的な結果は後述)

ブラックボックステストについて



ブラックボックステストで良い?

従来はホワイトボックスだけど

■工夫したこと

- 実装の中身はコードレビューで担保
- テストカバレッジを測定して安心感を得る
- 心配な所は手動でホワイトボックスもやる

■結論

- ブラックボックステストでOKとする

LCOV - code coverage report

Current view: top level

Test: fl_pon.info

Date: 2016-09-19

	Hit	Total	Coverage
Lines:	742	990	74.9 %
Functions:	201	293	68.6 %

Directory	Line Coverage	Functions
pon	18.4 % 7 / 38	50.0 % 6 / 12
pon/handler	91.4 % 412 / 451	73.0 % 111 / 152
pon/thread	56.8 % 231 / 407	63.4 % 59 / 93
tlv	97.9 % 92 / 94	69.4 % 25 / 36

Generated by [LCOV version 1.10](#)

例) カバレッジデータ

```

68      :
69      : /**
70      :  * @brief Validate if we should go Ramping State.
71      :  */
72      3844 : bool StateInit::meetsConditionForRamping() const
73      : {
74      3844 :     return false;
75      : }
76      :
77      :
78      : /**
79      :  * @brief Validate if we should go PID State.
80      :  */
81      3844 : bool StateInit::meetsConditionForPID() const
82      : {
83      3844 :     return false;
84      : }
85      :
86      :
87      : /**
88      :  * @brief creates Shutdown State.
89      :  * @return ShutdownState object for the next state.
90      :  */
91      0 : VoaCp::State* StateInit::createNextShutDownState()
92      : {
93      0 :     return NULL;
94      : }
95      :
96      :
97      : /**
98      :  * @brief creates ALD State.
99      :  * @return ALDState object for the next state.
100     :  */
101     0 : VoaCp::State* StateInit::createNextALDState()
102     : {
103     0 :     return NULL;
104     : }
105

```

テスト済

テスト未

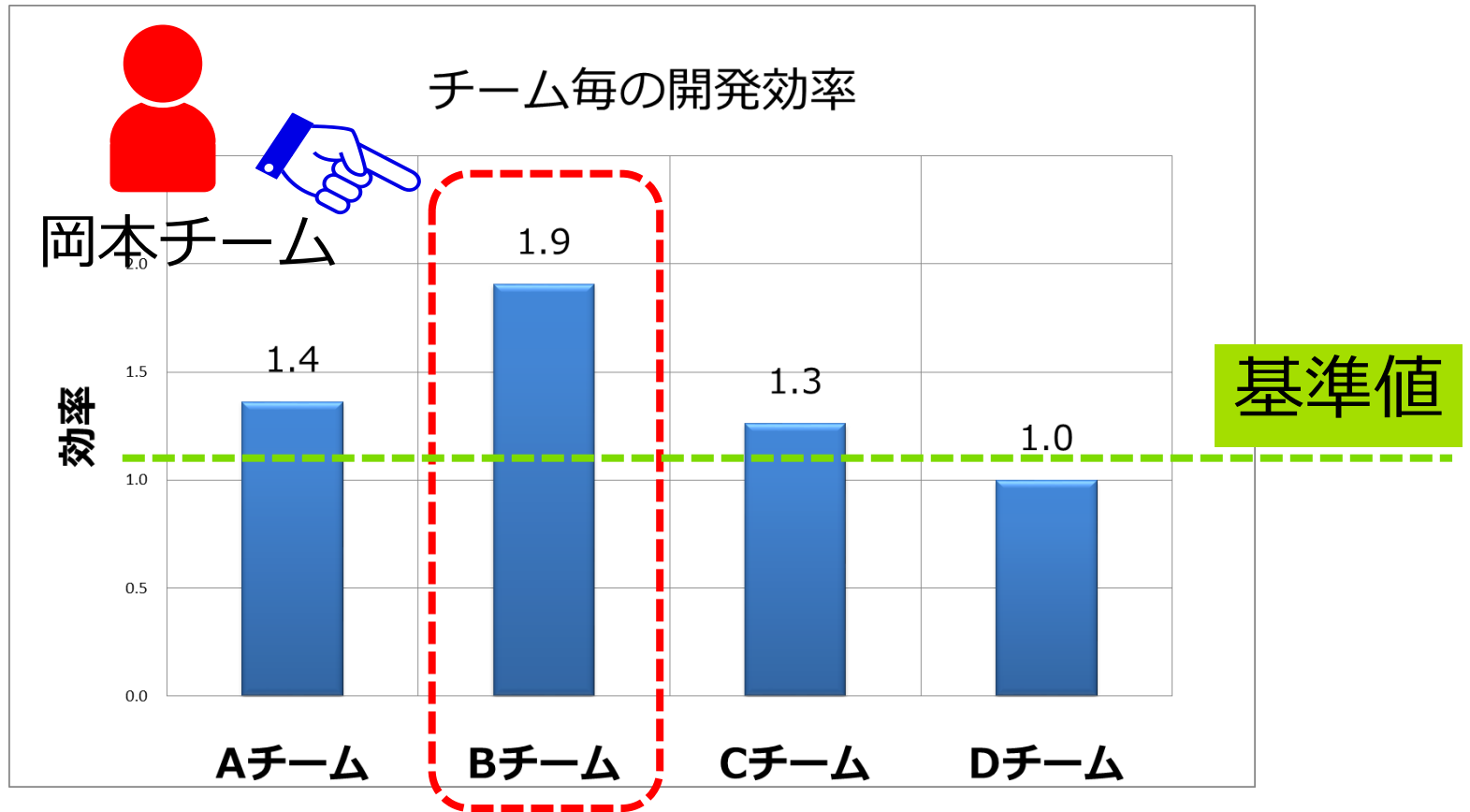
↓
 テストコード追加
 or
 手動でテストする

可視化して安心感を得る

- 自己紹介と背景
- アジャイルについて
- 技術プラクティス
- 導入のためにやったこと
- 導入のときに悩んだこと
- **導入の効果**
- 導入したプラクティス
- まとめ

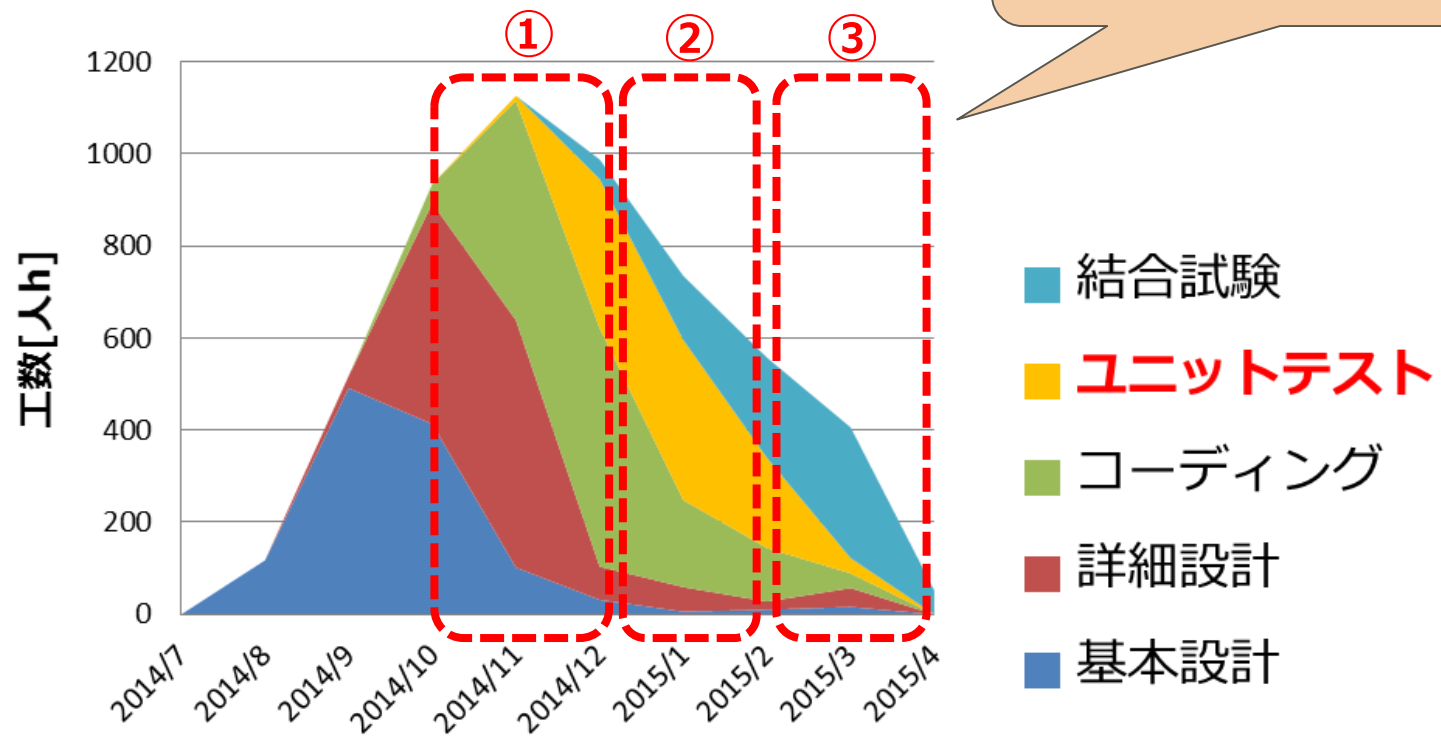
項目	内容	
開発装置	伝送装置の制御ファーム	
開発言語	C++	
開発メンバー数	13人	
開発規模	プロダクトコード	約30 KStep
	テストコード	約30 KStep
テストカバレッジ	約80% (ラインカバレッジ)	

- 同時期に他の3チームでも類似の開発を行ったため、結果の比較を行う
- ユニットテストの実施は岡本チームのみ



- 基準の開発効率をクリア
- 他チーム比でも良好な結果

工程別稼働推移

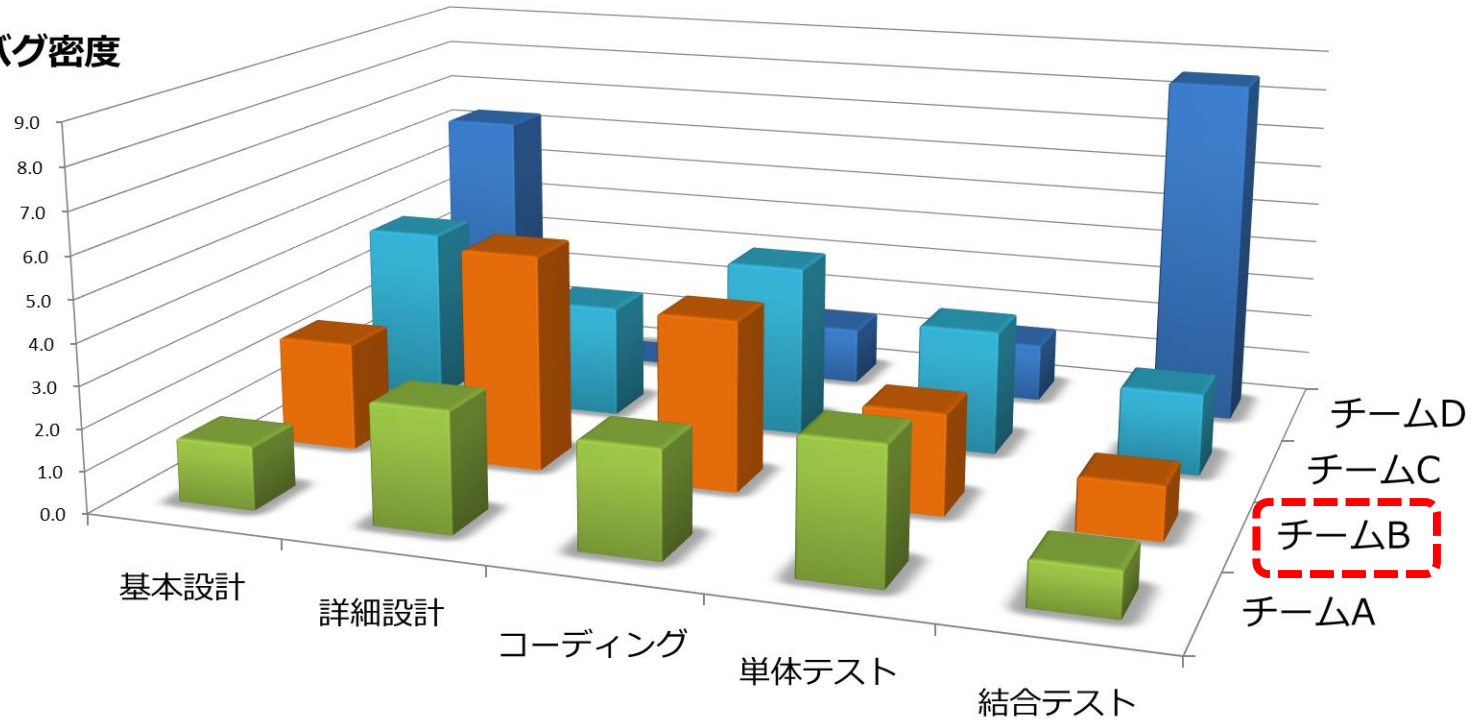


実際にはイテレーションを3回まわしている

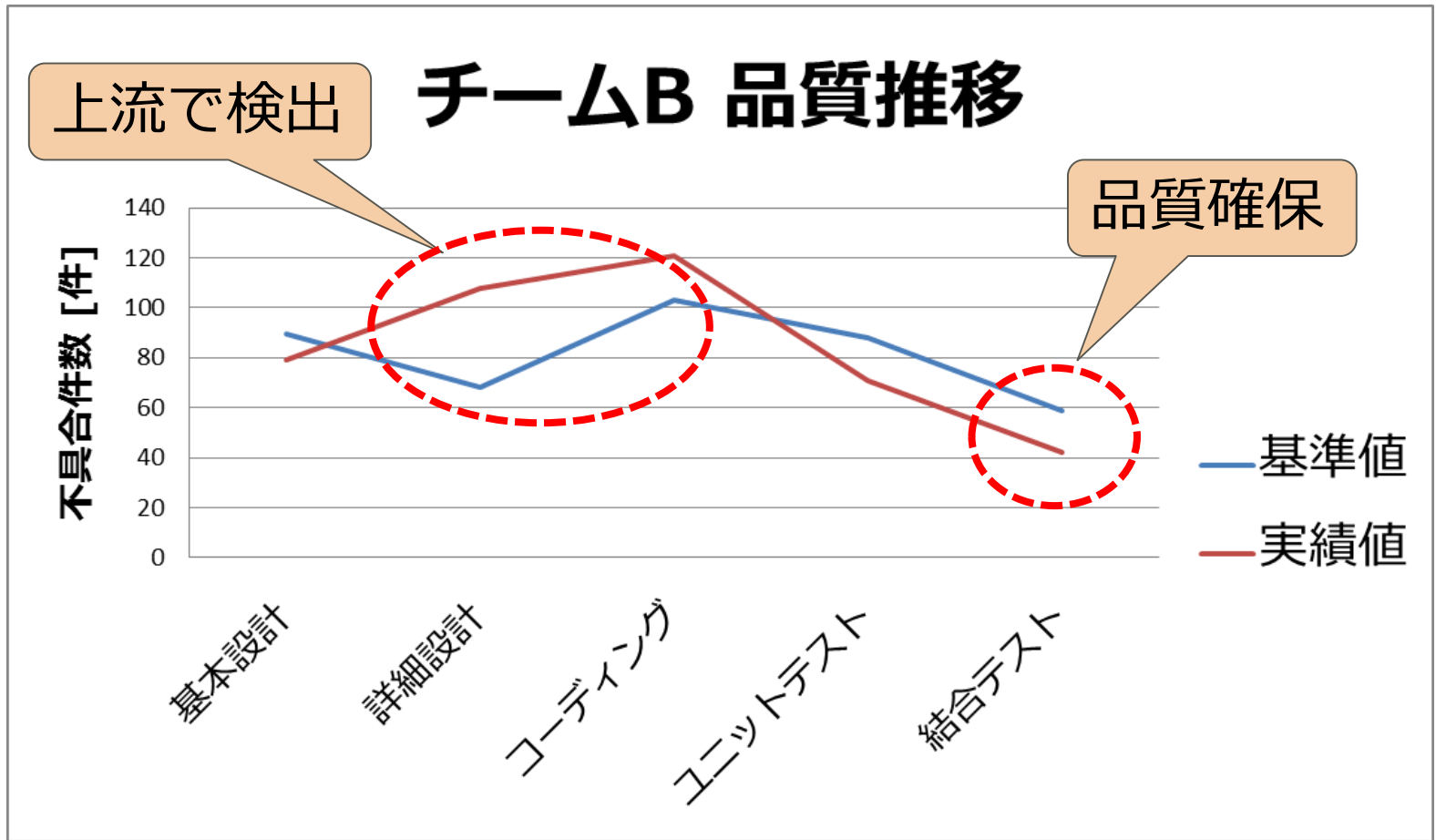
- 繰り返し開発でも試験工数は爆発せず
- 絶対値でも基準の範囲内(従来と同等)

工程毎の品質と推移

バグ密度



- 他チーム比でも良好な結果



- 不具合の検出時期が上流にシフト
- ユニットテストで品質を確保可能

その他の効果 (エピソード#1)

なんかテストがFailするんですが...

いつから?

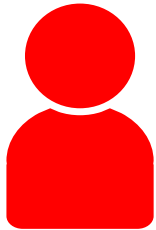
2週間前までは動いてました...

いや、毎日テスト流そうよ!!

CIの本当の価値を認識

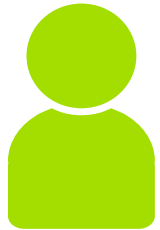
その他の効果 (エピソード#2)

なんかテストが書き難いんですが...



岡本

どうして?



メンバ

前準備とか与えるデータを用意するのが大変すぎて...

それ、ソフトの造りが悪いよね

良い設計と実装の価値を認識

- 自己紹介と背景
- アジャイルについて
- 技術プラクティス
- 導入のためにやったこと
- 導入のときに悩んだこと
- 導入の効果
- **導入したプラクティス**
- まとめ

カテゴリ	プラクティス
技術	コードの共同所有
	バージョン管理
	自動ビルド
	継続的インテグレーション
	シンプル設計
	リファクタリング
	インクリメンタル開発
	自動化された回帰テスト
	ユニットテスト

導入したプラクティス一覧

カテゴリ	プラクティス
プロセス	日次ミーティング (朝会)
	かんばん
	定期的なふりかえり
	ファシリテーション

- 自己紹介と背景
- アジャイルについて
- 技術プラクティス
- 導入のためにやったこと
- 導入のときに悩んだこと
- 導入の効果
- 導入したプラクティス
- **まとめ**


まとめ

- ボトムアップからのチャレンジは可能
 - 現場は勇気をもってやれば良い
- ユニットテストで従来の単体テストを代替することは可能
 - 効率/品質共に致命的な問題はない
 - 併用するという選択肢もある
- 継続と改善が大事
 - とある現場の単発事例で終わらせない
 - 見える形で実績を積み上げて定着させる

- アジャイルとかWFとか関係ない
 - 技術プラクティスはプロセスに依らない
 - WFでもやれば良い (やるべき)

- 技術は超大事
 - エンジニア/開発部門の根幹

- 人とチームを大事にする
 - 技術は人/チームに宿る
 - チームを作り上げるには手間と時間が必要



FUJITSU

shaping tomorrow with you