SONY

コードレビューとコードメトリクスを活用したコード品質改善活動

～開発エンジニアが高品質なコードを目指す～

SPI JAPAN 2018 (2018/10/11)

ソニーネットワークコミュニケーションズ株式会社
クラウド＆アプリ事業部門 1部 1課
広石 達也 <Tatsuya.Hiroishi@sony.com>

Agenda

- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

Agenda

- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[はじめに] 自己紹介

- 広石 達也 (ひろいし たつや)
  - Tatsuya.Hiroishi@sony.com / GitHub: @t-hiroishi
  - ソニーネットワークコミュニケーションズ株式会社
    クラウド＆アプリ事業部門　1部　1課
  - 2013 年に ソニー株式会社 入社
    主にコンシューマ向けアプリケーション設計・開発を担当

↑ 趣味は弓道

News Suite
クラウド管理だから提供できる解析データ
スマートフォン向けアプリ & クラウドサーバー
Skin Viewer & Cloud Server

Music Center for PC

ELECTRON

[はじめに] 部署紹介

- ソニーネットワークコミュニケーションズ株式会社
  クラウド＆アプリ事業部門　１部
  - ハードウェアやクラウドと連携したサービスを担当

↑ 部のメンバ (一部)

会社紹介

Agenda

- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[背景] 2014年：我々のコードは高品質なのだろうか？

- 部署内の複数の開発プロジェクトに対して第三者による外部レビューが実施され、ソースコードの品質に関するいくつかの課題を認識した

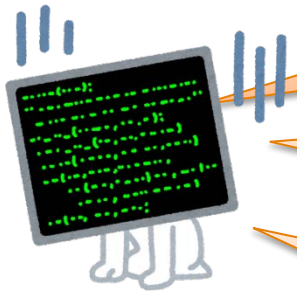
外部レビュー

「そちらの部署のソースコード品質はこんな感じです」

危機感

部門長

「文化を育てよう。
基本的な事から固めて、技術者の底上げを地道にやろう。」



開発エンジニア達

「費用対効果が見えにくいからあんまりやりたくない」

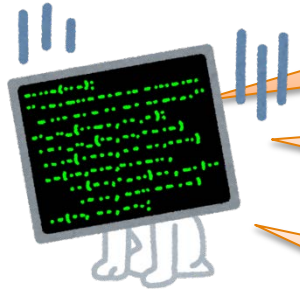
「ちゃんと品質向上に工数を割こう！」

「そもそも第三者のレビューで俺たちのレベルを測れるの？」

部署内でコード品質に対する意識に差があった

[背景] 2015年：とりあえずコードレビューをやってみた

- 各ソフトウェア開発プロジェクトにコードレビューのプロセスを導入

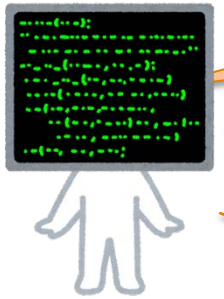


2014年

「費用対効果が見えにくいからあんまりやりたくない」

「ちゃんと品質向上に工数を割こう！」

「そもそも第三者のレビューで俺たちのレベルを測れるの？」



2015年

「レビューされるとなんとなくコードが良くなった気がする」

「ちゃんと品質向上に工数を割こう！！！」

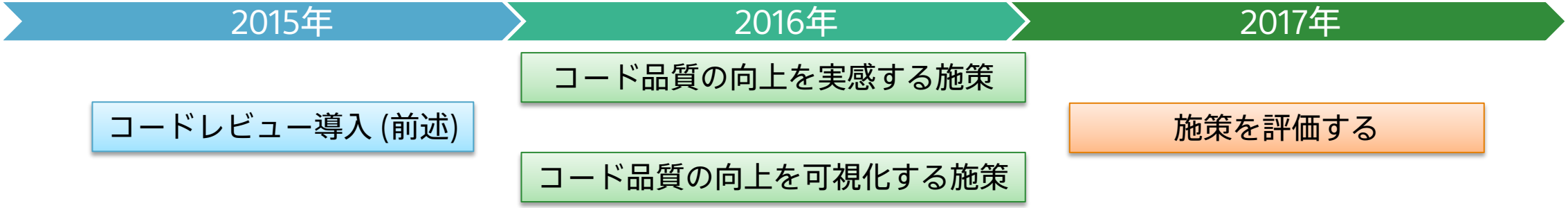
「俺たちのレビューはどのくらい品質向上に貢献したの？」

開発エンジニア達はレビューの効果をうまく説明できなかった！

Agenda

- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[改善策] 施策の目的と対象プロジェクト



- 目的
  - コードレビューが品質向上に対してどのくらい効果があったのかを知りたい

- 対象プロジェクト
  - コンシューマ向けスマートフォンアプリ開発チーム
    - 開発メンバ : 3名 (アーキテクト 1 名、メンバ 2 名)
    - 開発期間 : 約 1 年間
    - 開発言語 : TypeScript (JavaScript)
    - 開発体制 : GitHub Enterprise を利用して Pull Request ごとに他メンバがコードレビューを実施
    - 開発サイクル : 2 weeks / 1 Sprint

Agenda

- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[改善策] 2016年：コード品質の向上を実感してみた

- GitHub 上の開発プロセスで 2 つのルールを導入

1. Pull Request のタイトル

prefix	種類
feat:	新機能
fix:	修正
docs:	ドキュメント
style:	コーディングルール
refactor:	リファクタリング
test:	テストコード
chore:	ビルド系

KARMA (OSS) の Git Commit Msg ルールを参考にした
http://karma-runner.github.io/1.0/dev/git-commit-msg.html

2. コメントのリアクション

reaction	意味
	[+1] 良いと思ったコードやコメントへの賞賛
	[laugh] コードの意図などの補足説明
	[heart] 改善部分などの指摘

GitHub には他にも reactions が存在するが今回は未使用

[改善策] 2016年：コード品質の向上を実感してみた

- ルールその１

  「Pull Request は種類に応じてタイトルに prefix を付与する」

- 主な狙い
  - PR による変更の意図を明確にしたい
  - 後の振り返りで集計しやすくしたい

- PR の例

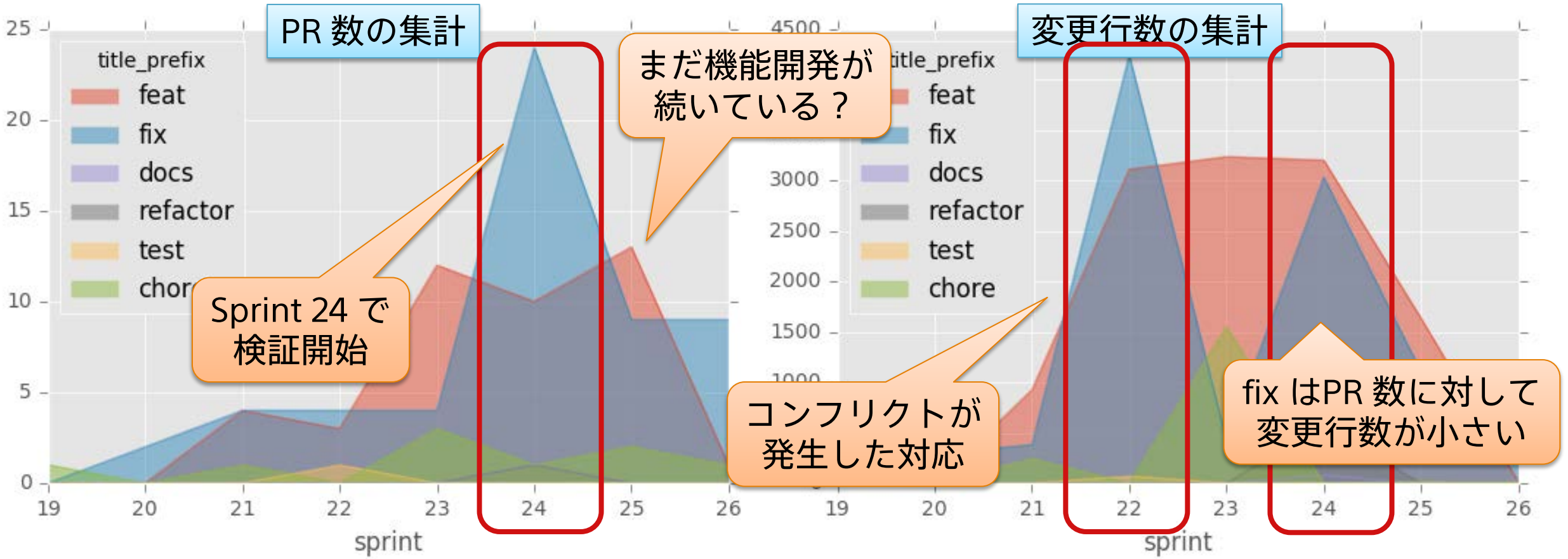
1. Pull Request のタイトル

prefix	種類
feat:	新機能
fix:	修正
docs:	ドキュメント
style:	コーディングルール
refactor:	リファクタリング
test:	テストコード
chore:	ビルド系

KARMA (OSS) の Git Commit Msg ルールを参考にした
http://karma-runner.github.io/1.0/dev/git-commit-msg.html

[改善策] 2016年：コード品質の向上を実感してみた

- ルール１「Pull Request は種類に応じてタイトルに prefix を付与する」



出来事や傾向の振り返りは次へ活かす材料となる

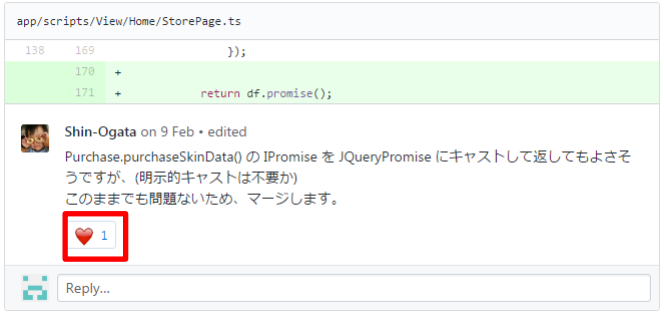
[改善策] 2016年：コード品質の向上を実感してみた

- ルールその 2

  「レビューコメントは意味に応じてリアクションを付与する」

- 主な狙い
  - ➢ レビューが役に立ったのか反応が欲しい
  - ➢ コメントの意図を一目で理解したい

- リアクションの例



2. コメントのリアクション

reaction	意味
	[+1] 良いと思ったコードやコメントへの賞賛
	[laugh] コードの意図などの補足説明
	[heart] 改善部分などの指摘

GitHub には他にも reactions が存在するが今回は未使用

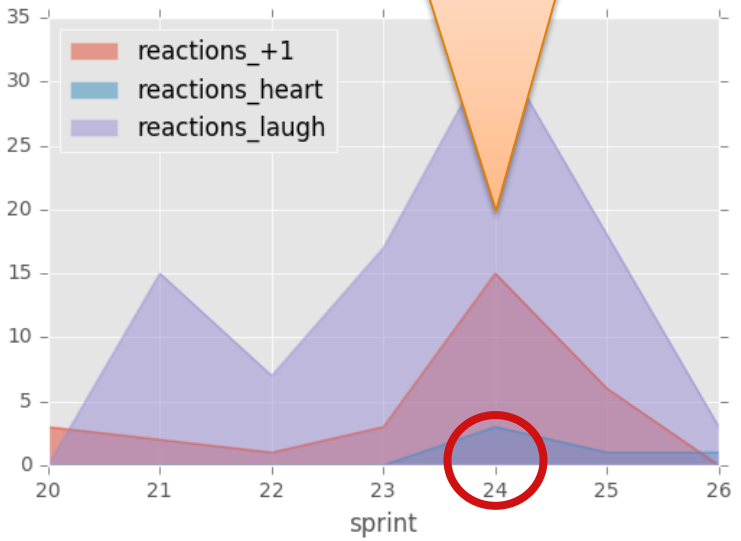
[改善策] 2016年：コード品質の向上を実感してみた

- ルール２「レビューコメントは意味に応じてリアクションを付与する」

- 👍 [賞賛]：良いレビューを振り返られるようになりモチベーションアップにつながった
- 😄 [補足]：コミュニケーションコストを抑えることができた
- ❤️ [指摘]：設計・開発段階で品質向上を意識してレビューできた

↓ [指摘] のレビューコメントの例 ↓

[指摘] が増加
⇒ 検証開始で品質を意識？



[改善策] 2016年：コード品質の向上を実感してみた

- 開発エンジニアが実感した効果
  - ➢ レビュールール１の効果
    - レビュー前：変更内容をあらかじめ推測できた
    - レビュー後：変更内容の傾向を振り返ることができた
  - ➢ レビュールール２の効果
    - する側：どのように役立ったかフィードバックを得られた
    - される側：品質向上に対するモチベーションアップ

コードレビューに対する納得感が向上した！

1. Pull Request のタイトル

prefix	種類
feat:	新機能
fix:	修正
docs:	ドキュメント
style:	コーディングルール
refactor:	リファクタリング
test:	テストコード
chore:	ビルド系

KARMA (OSS) の Git Commit Msg ルールを参考にした
http://karma-runner.github.io/1.0/dev/git-commit-msg.html

2. コメントのリアクション

reaction	意味
	[+1] 良いと思ったコードやコメントへの賞賛
	[laugh] コードの意図などの補足説明
	[heart] 改善部分などの指摘

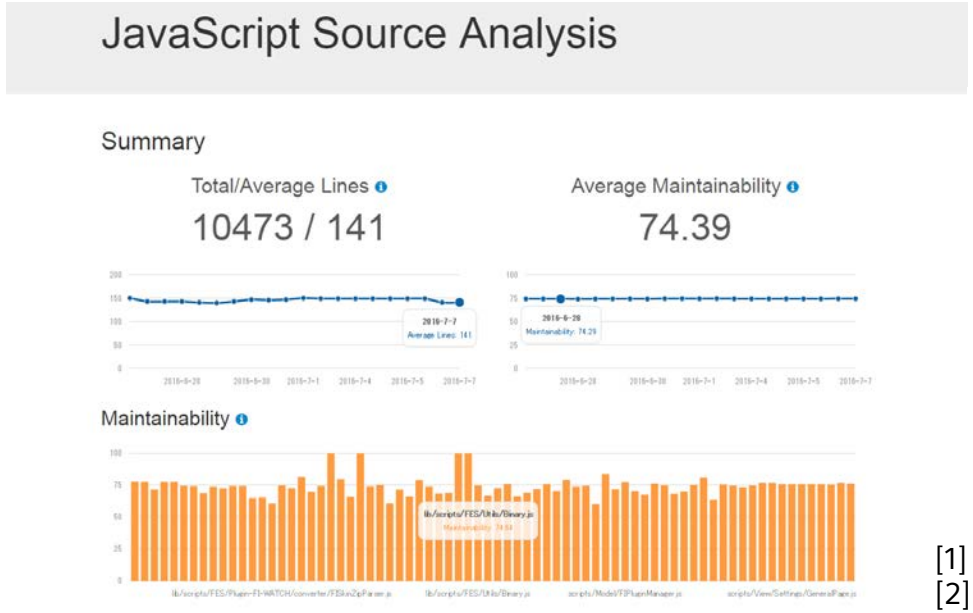
GitHub には他にも reactions が存在するが今回は未使用

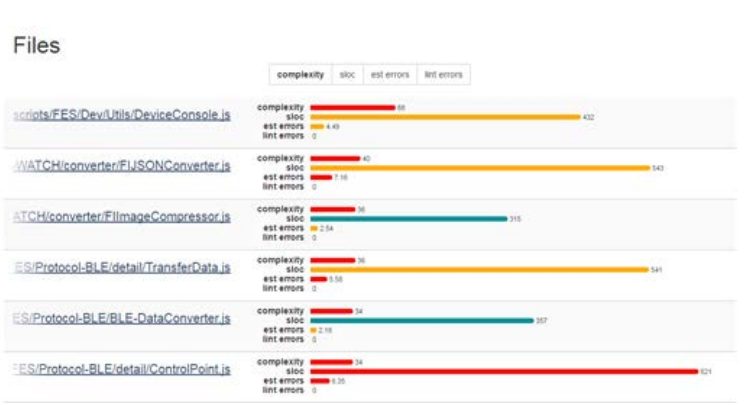
Agenda

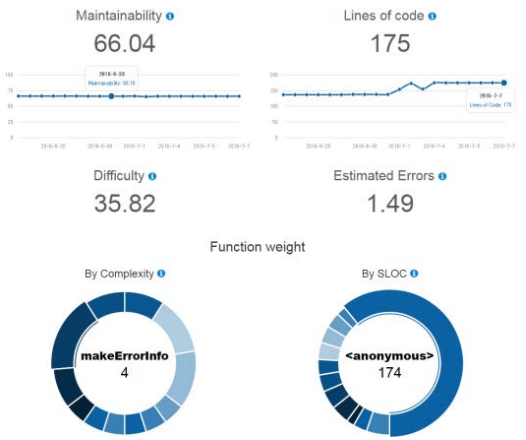
- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[改善策] 2016年：コード品質の向上を可視化してみた

- ソースコードのメトリクスを計測して可視化する
  - ➢ OSS のメトリクス計測ツール “plato” [1] を導入して
    保守性に関するパラメータである “maintainability” [2] の推移を継続的に観測
    - ファイル単位でメトリクスを計測
    - [保守性 低] 0 – 100 [保守性 高]



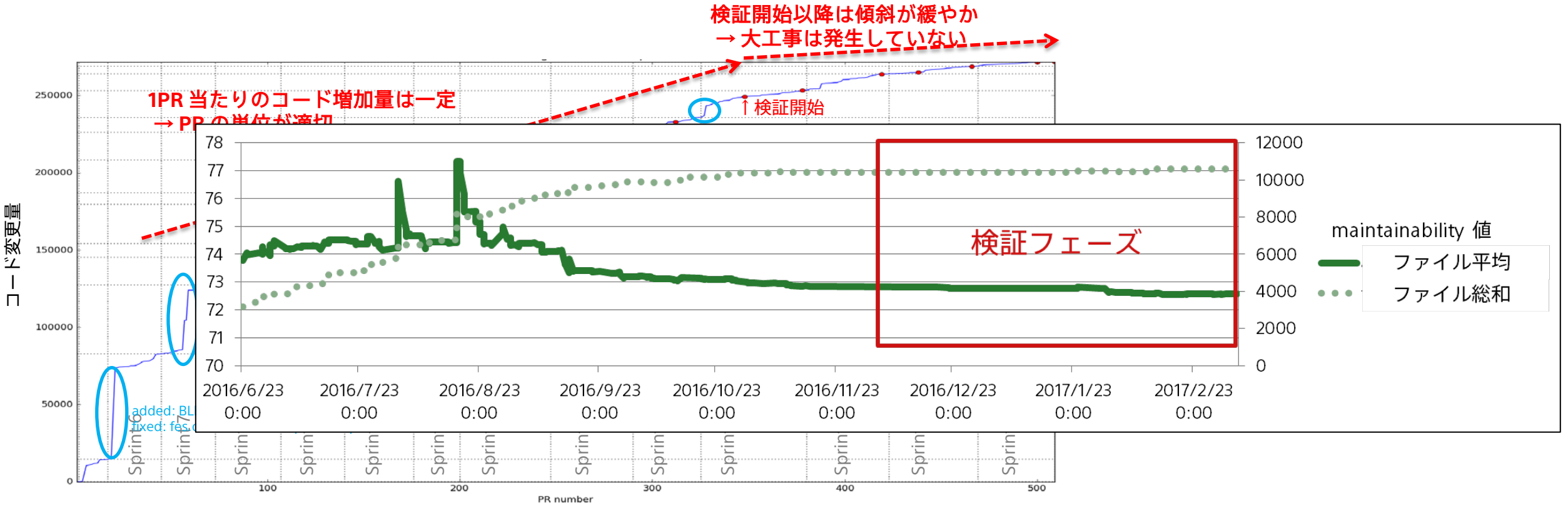




[1] https://github.com/es-analysis/plato
[2] https://blogs.msdn.microsoft.com/codeanalysis/2007/11/20/maintainability-index-range-and-meaning/

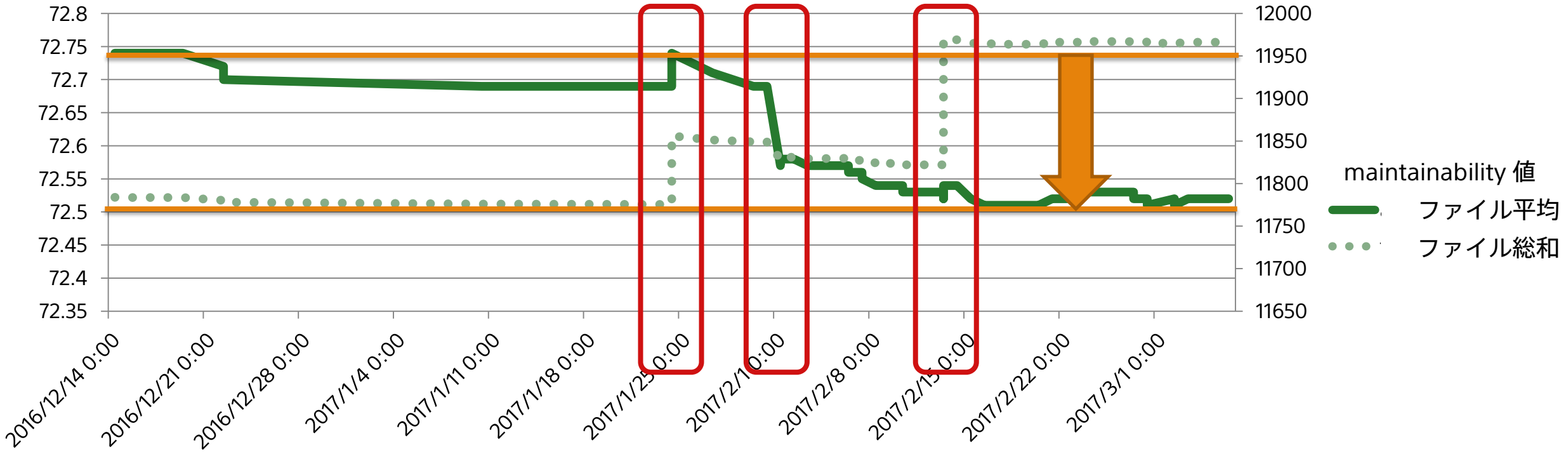
[改善策] 2016年：コード品質の向上を可視化してみた

- 気づき 1：コードメトリクスの変化を時系列で観測できた
  - コード変更量：開発フェーズでは一定の割合で増加、検証フェーズでは収束
  - maintainability：開発フェーズでは不規則に増減、検証フェーズでは収束



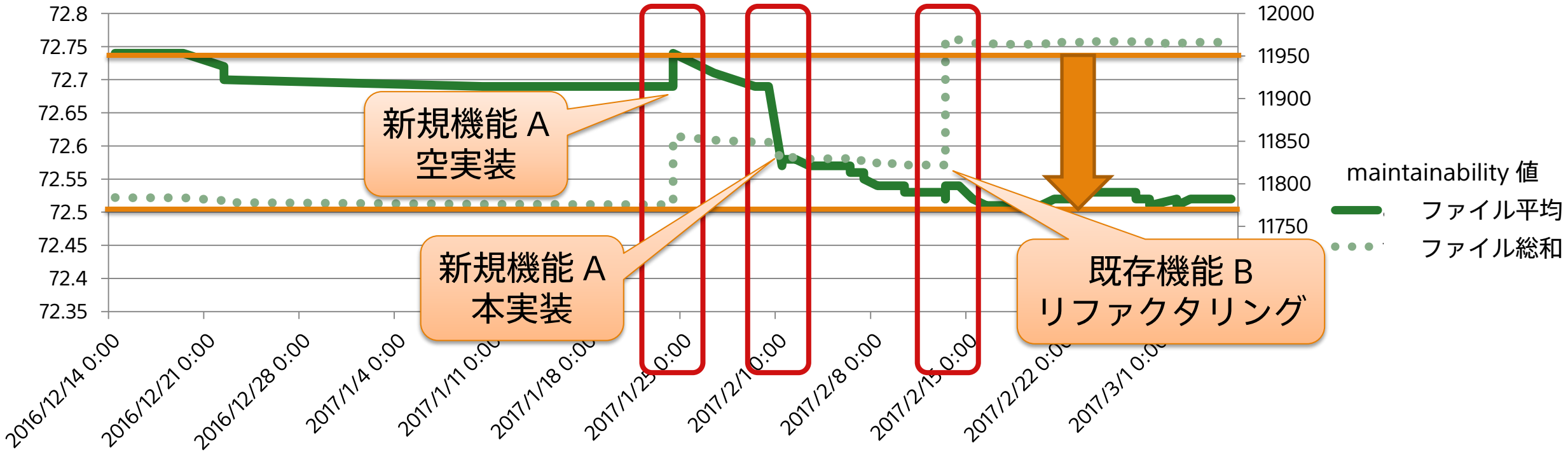
[改善策] 2016年：コード品質の向上を可視化してみた

- 気づき 2：メトリクスの相対値の変化から大きなコード変更を観測できた
  - 「下図の赤枠のタイミングで何か大きな変更があったかも」
  - maintainability 値が急落したときに何か不都合な変更が入った可能性がある



[改善策] 2016年：コード品質の向上を可視化してみた

- 気づき 3 : 品質を評価する指標として単純にメトリクスを使うことは難しい
  - 複雑な機能の実装は保守性が高いコードでも maintainability 値が下がりやすい
  - 偉い人「このコードの maintainability は低いから改善しなさい」 ← 絶対ダメ！



[改善策] 2016年：コード品質の向上を可視化してみた

- 開発エンジニアが考えたコードメトリクスの使いどころ

使いたい/使ってほしい	使いたくない/使ってほしくない
同じプロジェクトで メトリクスの変化を観測する	違うプロジェクトで メトリクスの値を比較する
開発プロジェクトの目標値として メトリクスの値を設定する	開発プロジェクトのノルマとして メトリクスの値を設定する
コードの変化に気づくきっかけとして	人事評価の指標として

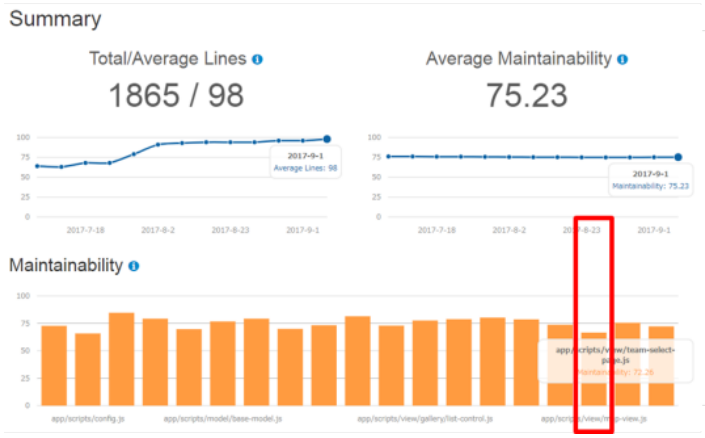
開発エンジニアのモチベーションに大きく影響するため
メトリクスは前向きな気づきを得るために活用すべし

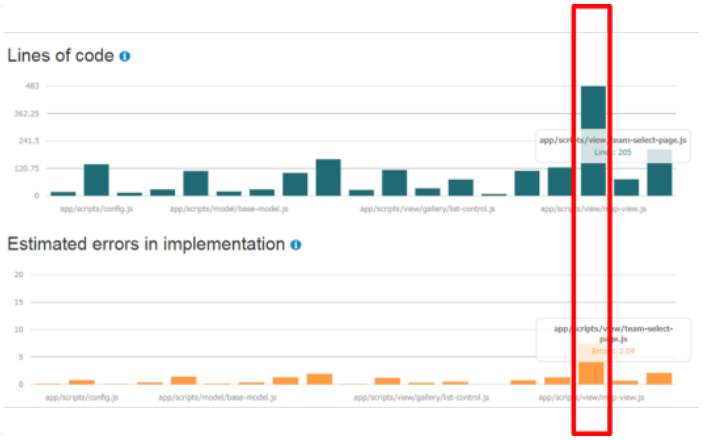
Agenda

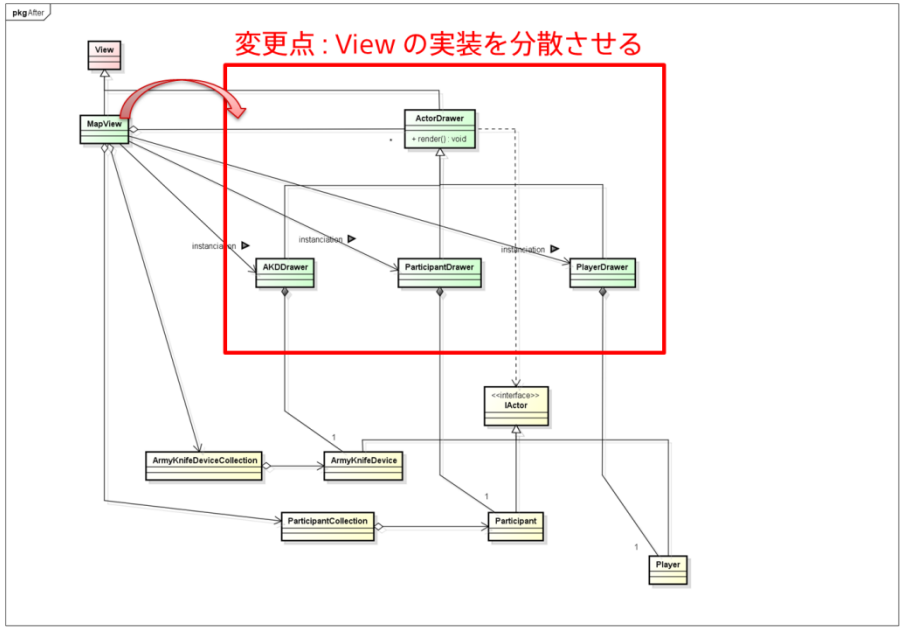
- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : これまでの施策を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[評価] 2017年：これまでの施策を評価してみた

- 他の開発プロジェクトへ前述の施策を展開
  - ➢ レビューコメント「このファイルはコード量が多すぎるから分割しよう」







レビューの指摘がメトリクスに対してどのように影響するか

[評価] 2017年：これまでの施策を評価してみた

- リファクタリングの実施前後でコードメトリクスを比較してみた

- 対象ファイル単体で見ると
  - コード量は大きく減少した ← ファイル分割した
  - maintainability は低くなった ← 依存関係が増加した

- 全ファイルを結合 (PJ 全体) で見ると
  - コード量は増加した ← ファイル数が増えた
  - maintainability は高くなった ← 品質が向上！？

	Before	After
コード行数 (ファイル)	483	163
maintainability (ファイル)	73.75	73.16
コード行数 (PJ 全体)	1333	1433
maintainability (PJ 全体)	69.83	70.85

レビューの実感がメトリクスにも表れた！

[評価] 2017年：フィードバックをもらってみた

- 第三者や上長から好意的なフィードバックを得ることができた

協業者

「引き継いだソースコードは読みやすく保守しやすいです」

「具体的なアクションを起こして結果を出せたのは良かった」
「継続的に品質改善活動に取り組んでほしい」

部門長

開発エンジニア達

「コードレビューが役に立っていることを実感できた！」

「簡単な施策で品質向上に貢献できた！」

「コード品質を良くするための気づきを得ることができた！」

部署内の他プロジェクトへ施策を展開中

Agenda

- はじめに
- 背景
  - 2014年 : 我々のコードは高品質なのだろうか？
  - 2015年 : とりあえずコードレビューをやってみた
- 改善策
  - 2016年 : コード品質の向上を実感してみた
  - 2016年 : コード品質の向上を可視化してみた
- 評価
  - 2017年 : コード品質の向上を評価してみた
  - 2017年 : フィードバックをもらってみた
- おわりに

[おわりに] まとめ

- 背景
  - 開発エンジニアはコードレビューの効果を実感したい
- 改善策
  - Pull Request とレビューコメントのルール化 ⇒ 品質向上への貢献を実感
  - コードメトリクスの継続的観測 ⇒ コード品質の変化に対する関心が向上
- 評価
  - コードレビューの指摘がコードメトリクスに影響することを確認できた
  - 第三者やマネージャから好意的なフィードバックを得ることができた

[おわりに] ROCRO の紹介

- 開発者向けサービス群
  - https://rocro.com
  - GitHub / Bitbucketと連携

様々な言語でコードメトリクスの取得が可能！

Rocro とは

- 開発者の定常的・共通的タスクを自動化する CI 分野の SaaS 群
  - Inpecode : 自動コードレビュー＆修正サービス
  - Docstand : API ドキュメント生成＆ホスティングサービス
  - Loadroid : 自動負荷試験サービス
- ソニーのクラウド開発から始まる
- 回転させながらモノを作るプロの道具：轆轤（ろくろ）とソフトウェア開発の類似性から命名

ROCRO

Inspecode：コードレビューを SaaS で手間なく廉価に利用

- GitHub.com/ Bitbucket のレポジトリと連携、プッシュごとに自動的に静的解析を実行
- 実行結果はInspecode による Web サーバでホスティング
- プルリクエストもレビュー
- 検出したIssue を自動修正，修正内容は Pull Request として発行

→ エンジニアの修正作業の時間を大幅に削減

```
if err != nil {
    return err
}
return nil
```

↓

```
return err
```

現在 ROCRO の INSPECODE を plato と併用して活用中