ソフトウェアプロダクトラインにおける コア資産評価の仕組み確立

オムロン ソフトウェア株式会社
原田 真太郎、筒井 賢

オムロン株式会社
赤松 康至

会社紹介

自動改札機、券売機等

制御機器、FAシステム等

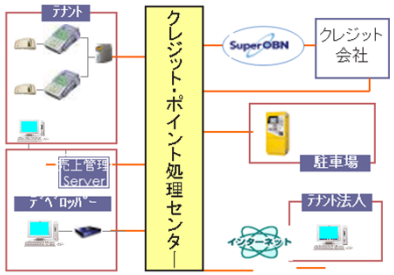
健康機器

オムロン ソフトウェア株式会社

卓越したソフトウェア技術で
「安心・安全・環境・健康・快適」の高い価値を創造する。

－つかむ・つなげる・創り出す
人と地球の明日のために－

決済ソリューション



監視・運用サービスソリューション

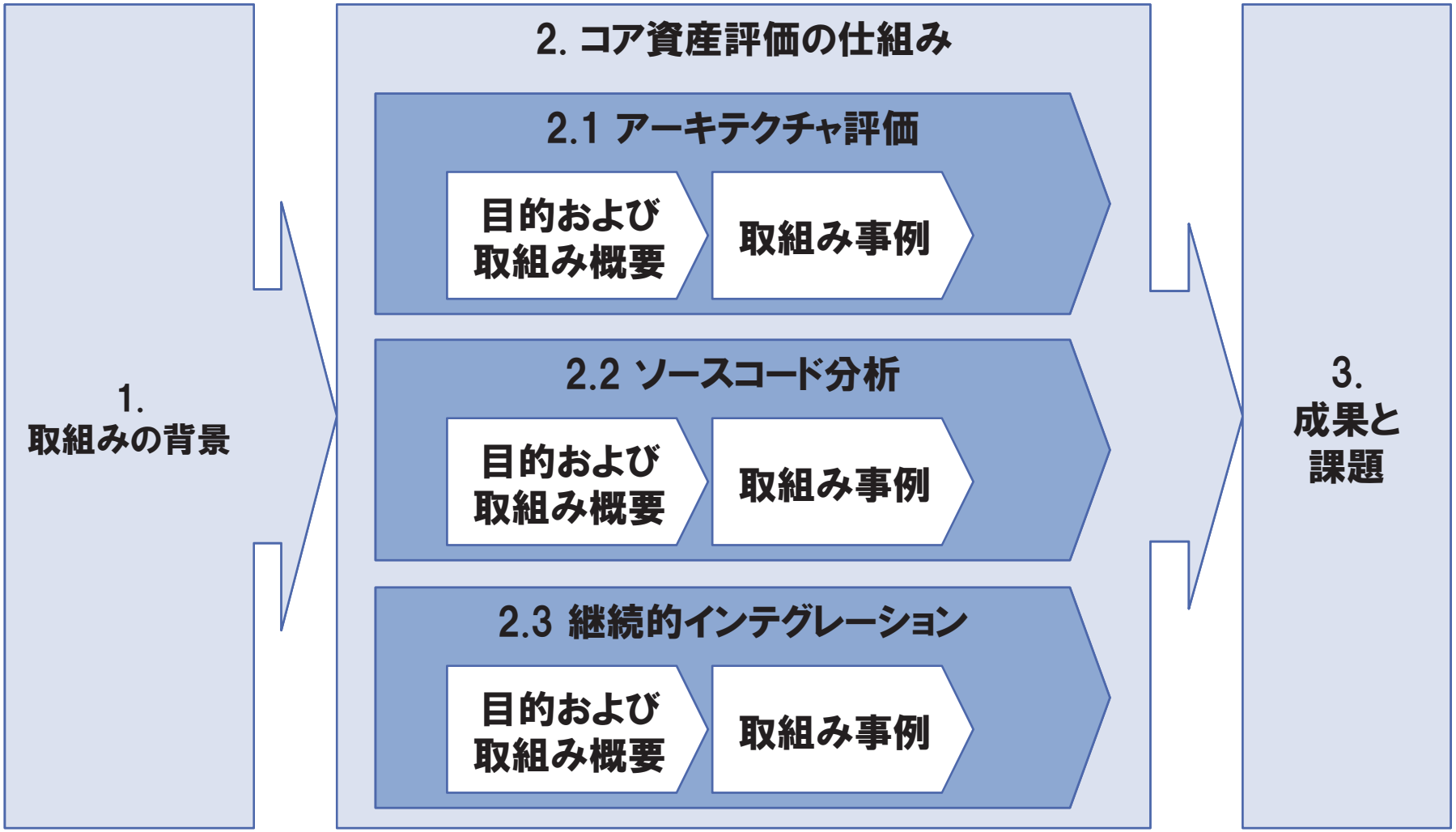
iWnn

MobileOmCR

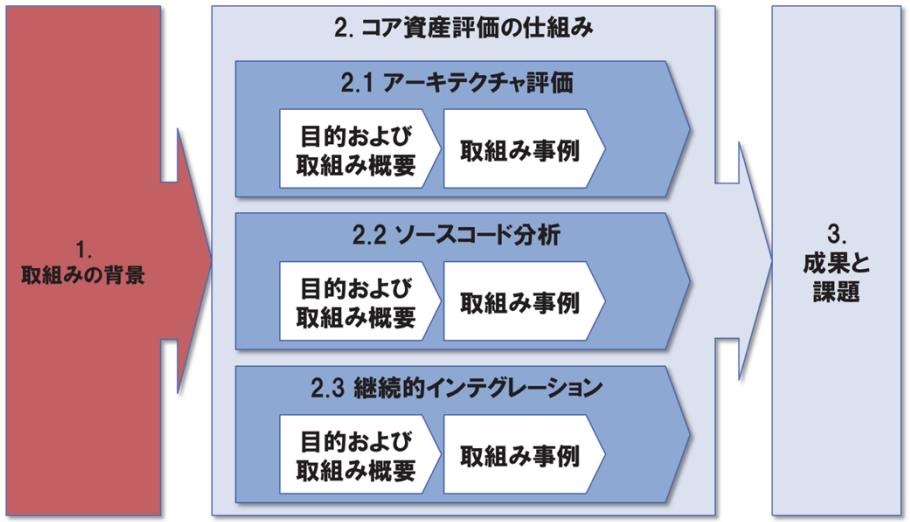
Bizcaroid

モバイル ソリューション

目次



1. 取組みの背景



1. 取組みの背景 - 現状

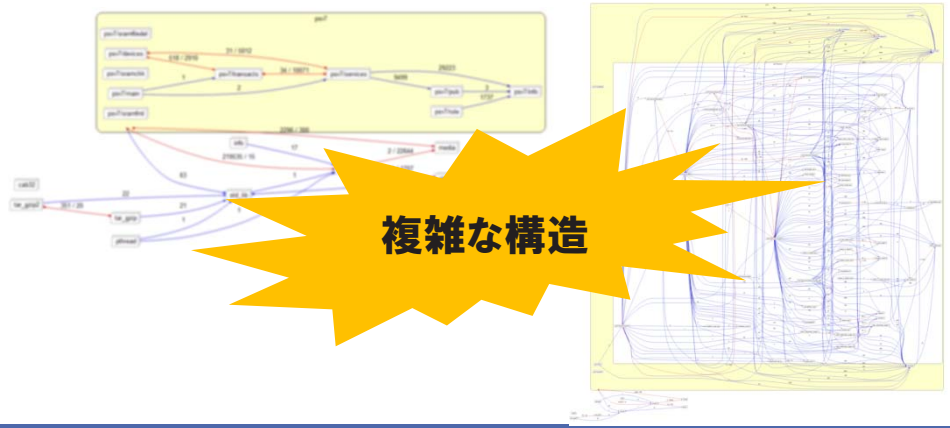
目指す姿

開発効率を大幅に向上し、顧客ニーズに応えられるQCDを確保する

ソフトウェアプロダクトライン開発の導入

現状

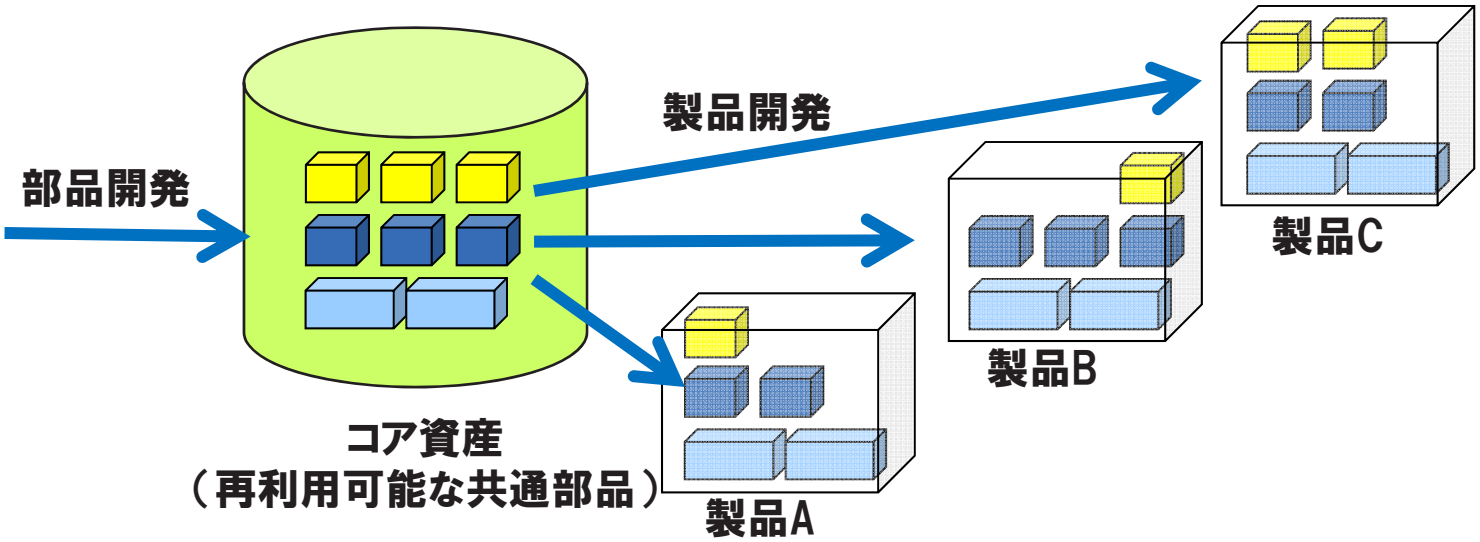
変更に強いソフトウェアを構築したつもりだったが、
徐々に劣化(複雑化・肥大化)しQCD確保が困難になっている



1. 取組みの背景 – ソフトウェアプロダクトラインとは

ソフトウェアプロダクトライン（以降SPL）とは？

場当たり的な再利用 → **戦略的・計画的再利用**



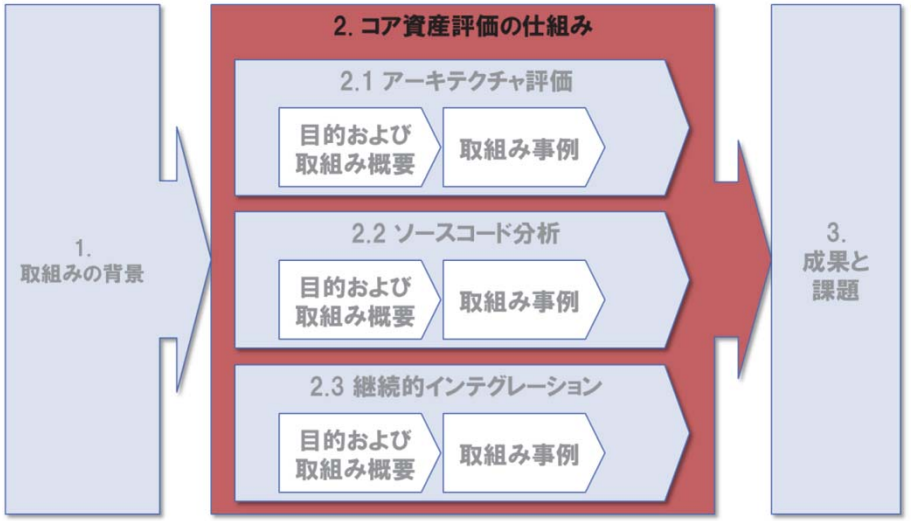
従来も、共通化・再利用を意識してソフトウェアを作成していたはず
なぜ、劣化してしまったのか？

1. 取組みの背景 – 課題

- 元の木阿弥にならないように、
  継続的にソフトウェアの**課題を見える化**する仕組みが必要

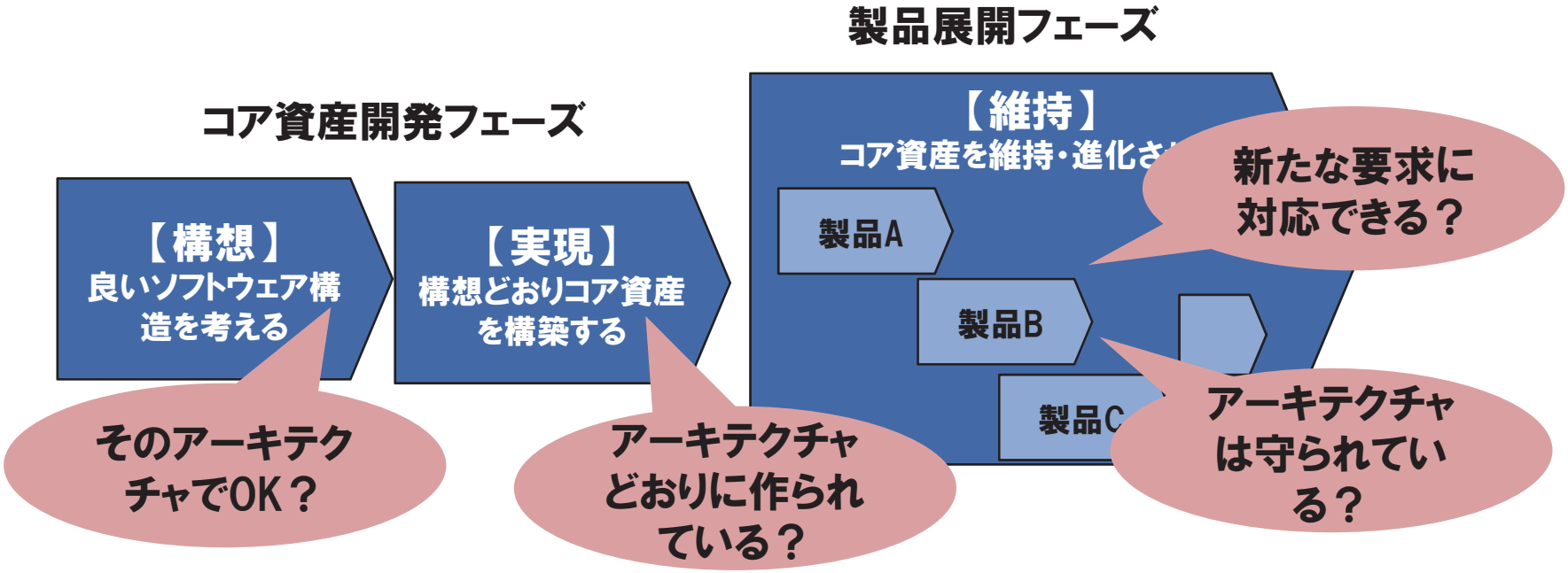
	課題	方針
①	**「課題の見える化」** ソフトウェアの良し悪しを見える化し、開発者に是正を促す	**「とことん見せる化」** • 設計からソースコードまで • さまざまな観点で
②	**「続ける仕組み」** 劣化を防ぐため、継続的に評価可能な仕組みを用意する	**「無理なく続ける」** • 軽量で繰り返し実施可能な仕組み **「現場が嬉しい」** • 開発者が効果を実感し、使い続けたいと思う仕組み

2. コア資産評価の仕組み



2. コア資産評価の目的

- SPL成功のためには、あるべき姿と実態のズレを可視化し是正に繋げる仕組みが必要



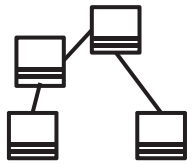
- アーキテクチャが製品群で中長期にわたって使えるか？
- ソフトウェアがアーキテクチャ構想どおりに作られているか？
- 製品開発時に、ソフトウェアが崩れずに維持されているか？

2. コア資産評価の3つの取組み

- 目的と評価対象の変化の頻度に応じて3つのサイクルで
  ソフトウェアを診断・フィードバックする仕組みを開発プロセスに導入した

そのアーキテクチャでOK？

新たな要求に対応できる？

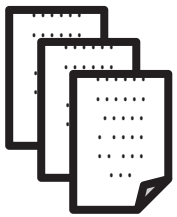


アーキテクチャ

① アーキテクチャ評価
（アーキテクチャ作成・変更時）

アーキテクチャどおりに作られている？

アーキテクチャは守られている？

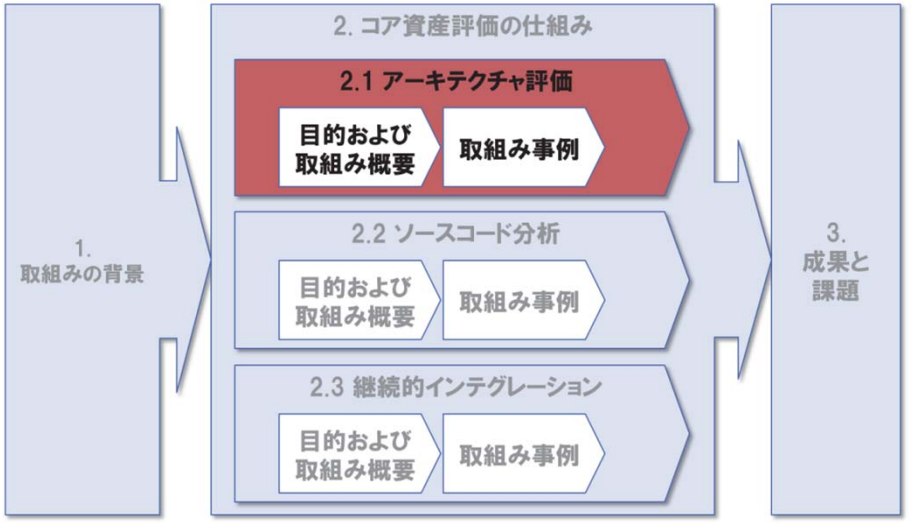


ソースコード

② ソースコード分析
（スプリント毎－3週間毎）

③ 継続的インテグレーション
（毎日/毎時）

2.1 アーキテクチャ評価

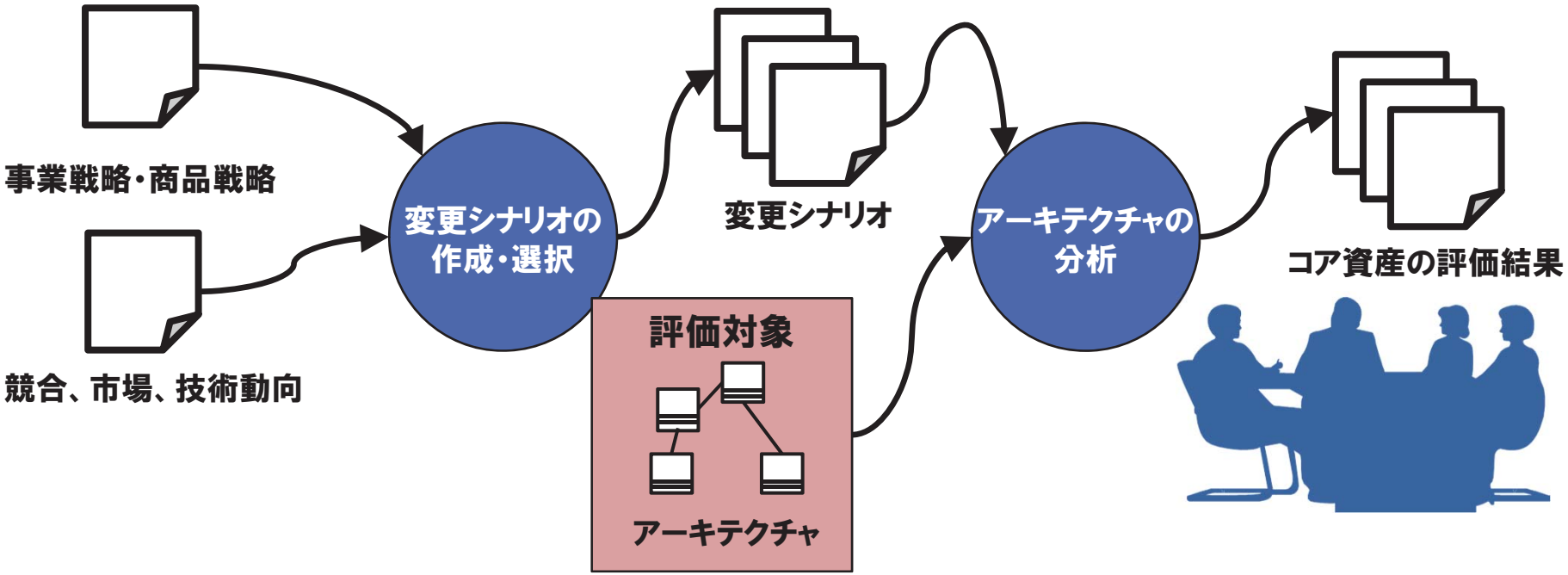


2.1 アーキテクチャ評価 － 概要

- アーキテクチャの良し悪しが、変更容易性に大きく影響
- SEIが開発したSAAMをベースとした手法を利用
  変更シナリオを用いて、アーキテクチャの変更容易性をレビュー形式で検証する

SAAMとは？
Software Architecture Analysis Methodの略で、カーネギーメロン大学・ソフトウェア工学研究所が開発した、ソフトウェアアーキテクチャを評価する手法

繰り返し実施可能なように
ライトウェイトにカスタマイズ



2.1 アーキテクチャ評価 － 取組事例

- アーキテクチャ上の課題を特定し、対策につなげる
- 従来のアーキテクチャとの比較により、ビジネスゴール達成の確からしさを検証

変更シナリオ（例）

ID	変更シナリオ	変更が必要なコンポーネント	概算対応工数	従来対応工数
1	＊＊機能の追加	3（Aモジュール、Bモジュール、Cモジュール）	25人日	40人日
2	＊＊画面のレイアウトを変更	1	2人日	10人日
3	＊＊デバイスを＊＊デバイスに変更	2	40人日	37人日
4	・・・	・・・		

アーキテクチャ（例）

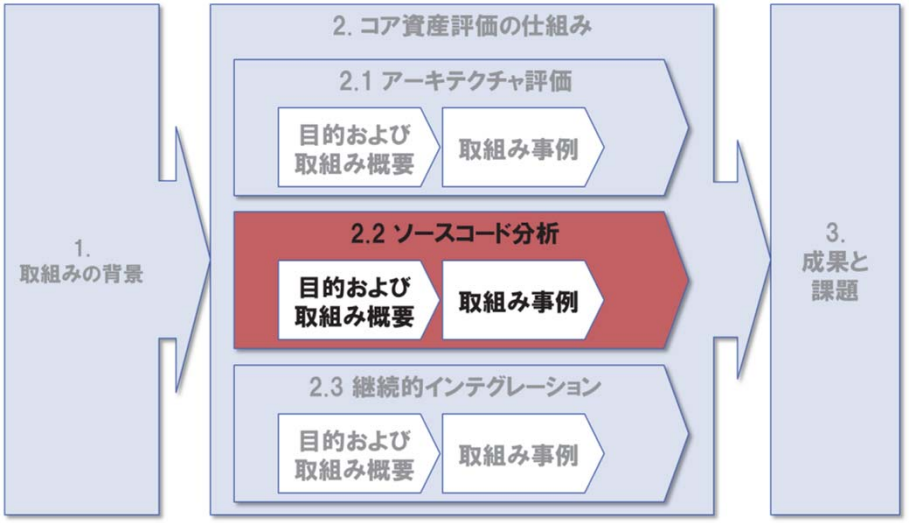
多くの関係者が参画することで、アーキテクトが想定していない変更シナリオを抽出することがポイント

アーキテクトが楽しめる（気付きを得れる）実りのあるレビューに！

変更シナリオ評価結果（例）

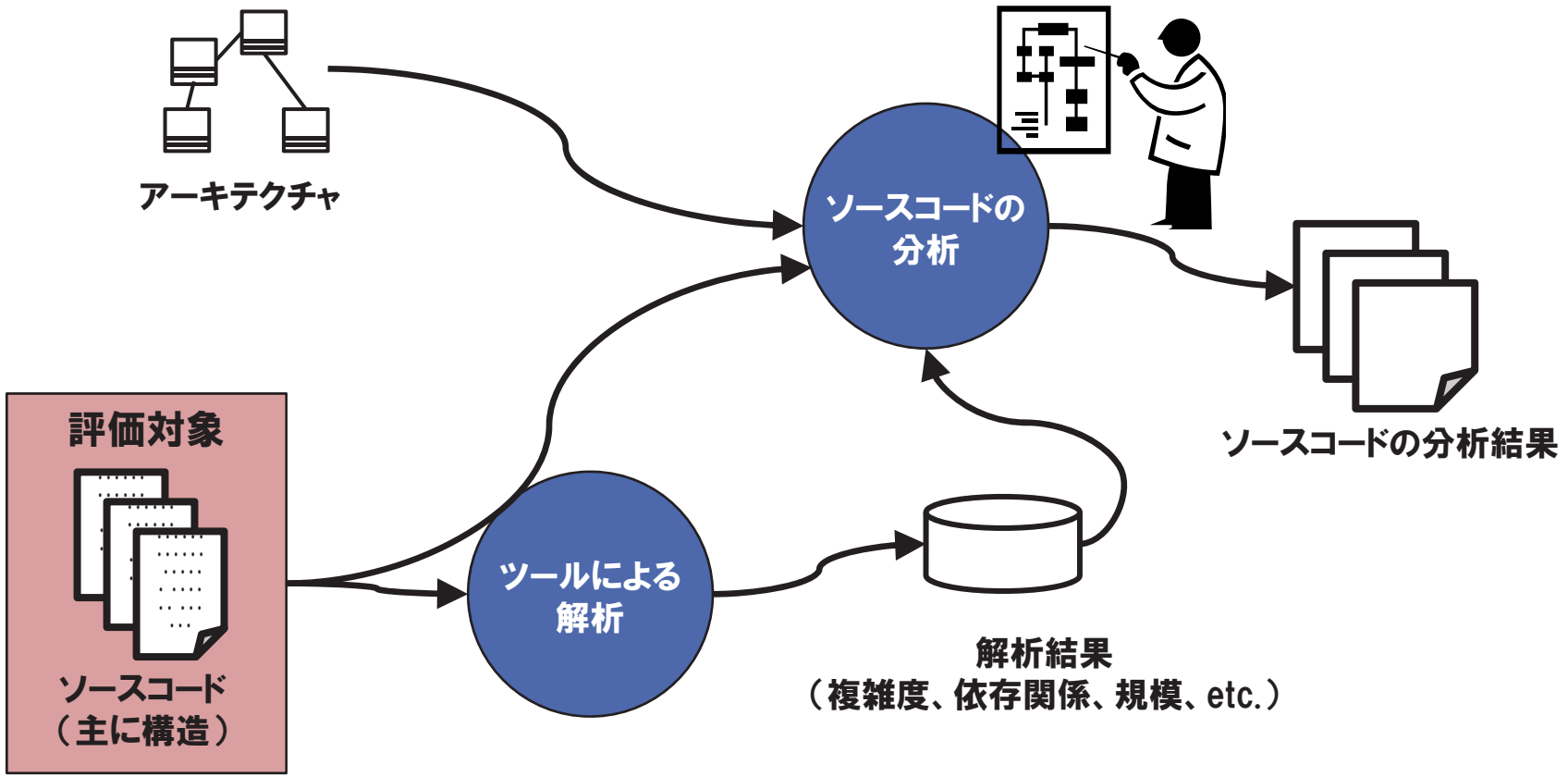
- 画面レイアウトに関する変更は比較的容易に実現可能
- ＊＊デバイスの対応は共通部に影響が発生するため該当部分のアーキテクチャの変更が必要
- ・・・

2.2 ソースコード分析



2.2 ソースコード分析 － 概要

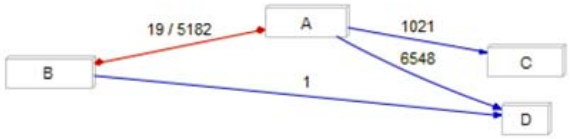
- ソフトウェア構造がアーキテクチャ構想どおりに実現されているかを確認
- ツールを用いて分析し、結果を開発者にフィードバック



2.2 ソースコード分析 － 取組事例

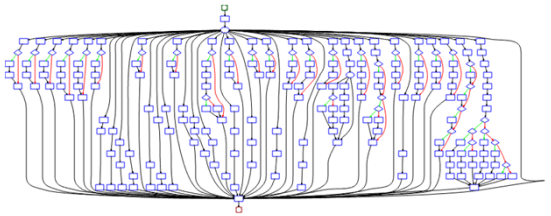
- スプリント振り返りミーティング（3週間毎）にて開発者に結果および改善案をフィードバックし是正を促す

依存関係分析



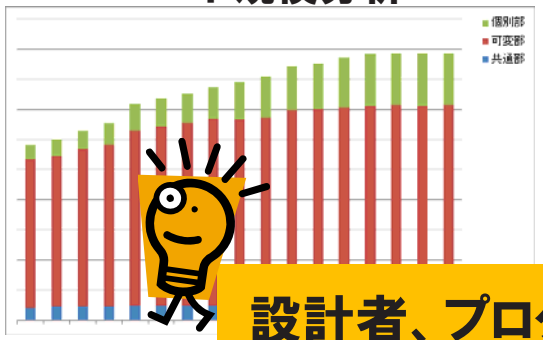
- アーキテクチャ設計通りか
- 逆参照・相互参照が発生していないか
- 特定のコンポーネントに依存関係が集中していないか

ロジックの複雑度分析



- 極端に複雑なロジックは無いか（基準値を超えていないか）

コード規模分析

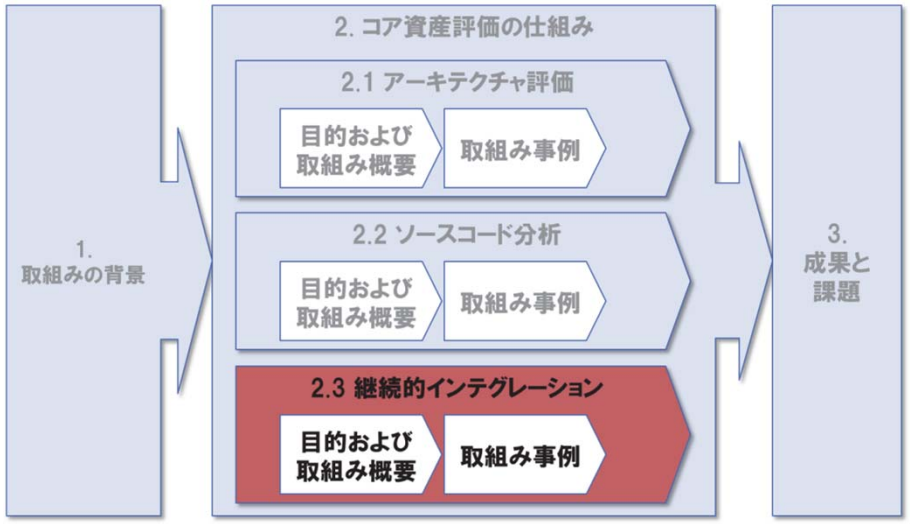


- 共通/可変/個別の規模が目論見どおりか
- コード開発量は想定通りか

地味に効いたと好評

設計者、プログラマが意図せずにコーディングしてしまった違反を可視化することがポイント

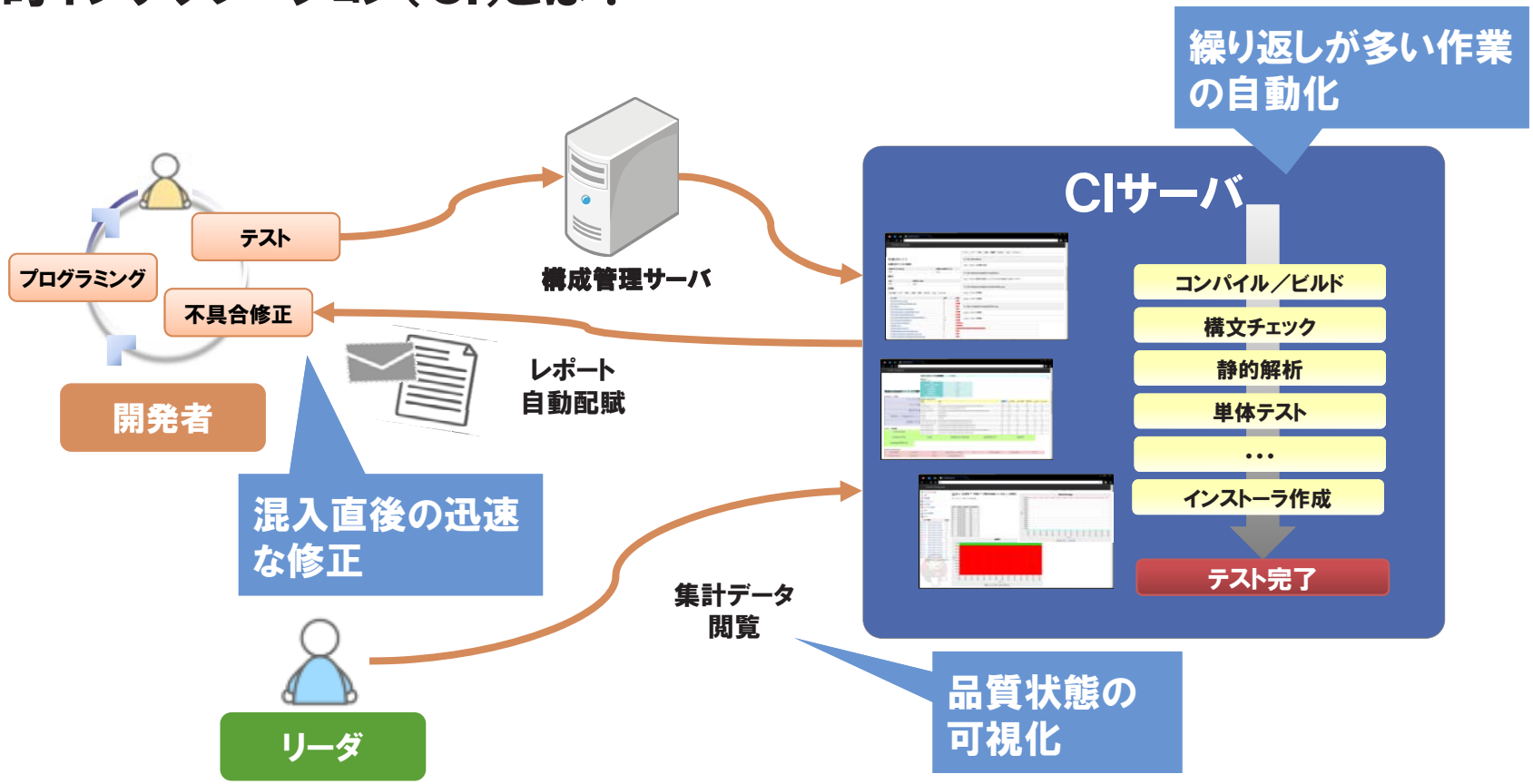
2.3 継続的インテグレーション



2.3 継続的インテグレーション ― 概要

- 正しくコーディングがされていることをツールで自動チェック
- コードの劣化を見える化し、開発者に気づきを与える仕組みとして活用

継続的インテグレーション(CI)とは？

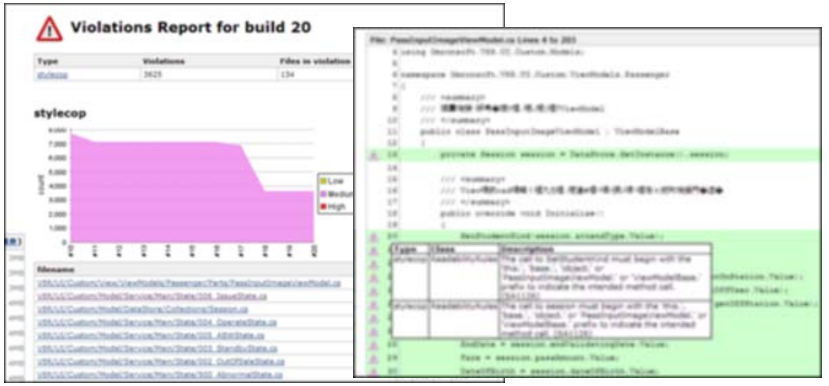


2.3 継続的インテグレーション ー 取組事例

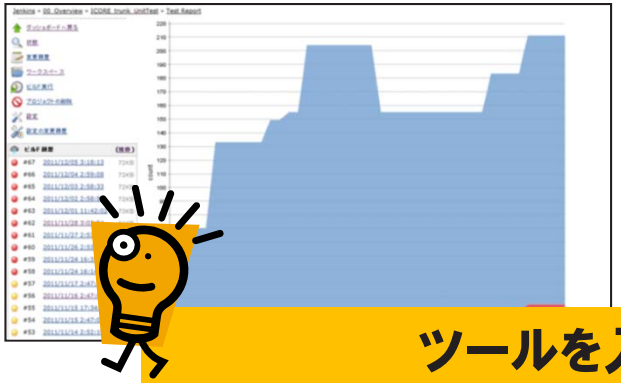
- ビルド、単体テスト、コーディング規約違反などを自動で実行
- 開発者が自主的かつ迅速に対応する習慣が定着した

統合ビルド

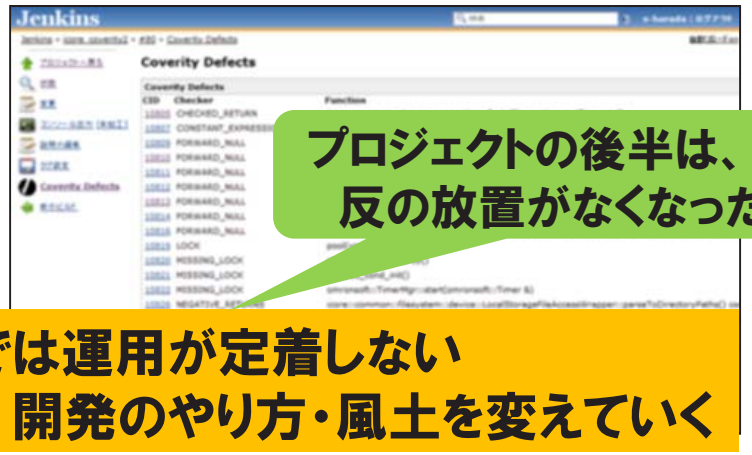
コーディング規約違反チェック



単体テスト



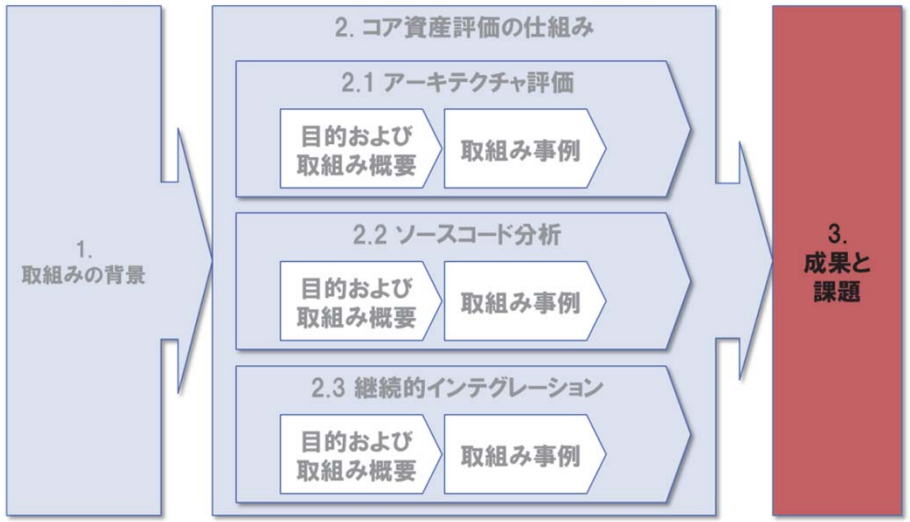
静的解析



プロジェクトの後半は、違反の放置がなくなった

ツールを入れただけでは運用が定着しない
開発者に意義を理解してもらい、開発のやり方・風土を変えていく

3. 成果と課題



取組みまとめ

ゴール	持続的な開発効率向上
取組み	ソフトウェアの良し悪しを見える化し、 開発者に気づきを与えて是正を促す仕組みを構築 ① アーキテクチャ評価 ② ソースコード分析 ③ 継続的インテグレーション
方針	• とことん見せる化 • 無理なく続ける • 現場が嬉しい

3. 成果と課題

- 目論見通りの開発効率向上を実現
- 「改造のしやすさ」に関するメトリクスも大幅改善

項目		従来	現在	改善効果
開発効率 （アーキテクチャ評価による想定効果）		―	―	29% 向上
規模 (KStep)		1,700	890	48% 削減
共通部割合		47%	81%	34% 向上
本質的複雑度	最大値	722	16	98% 削減
	基準値(10)を超えるモジュール数	1,664 個	18 個	99% 削減

3. 成果と課題

目指す姿と課題

持続的な
開発効率の向上

- ソフトウェアの良し悪しを見える化・是正
- 継続的に実施可能な仕組み
- さらには、

成果

設定した目標達成の見込みを得る

アーキテクチャとコードの一貫性を維持
複雑度などのメトリクスも大幅改善

軽量な３つの評価技術を確立・手順化
開発者が効果を実感し積極的に活用

開発者の意識・スキルの向上

3. 今後の取り組み

技術・仕組みはできた。
あとはやりきるのみ！
確実に組織で回せるように努力する！

継続的に使い続ける体制構築

- 開発チーム主体で実施できるように技術移転
- 変更／構成管理委員会の設置とコア資産評価の役割設定

組織展開

- SPLの展開とセットで導入（必要不可欠なアイテムに）

ご清聴ありがとうございました