ビジネスアプリケーション開発における生産性バラつき要因の分析事例

(株)オージス総研
技術部ソフトウェア工学センター
藤井　拓、木村めぐみ

発表のアウトライン

- 研究背景
- COSMIC法とは
- 生産性と生産性に影響する要因
- 機能部分と作業の分類
- プロジェクトへの適用結果
- まとめ
- 今後の課題

研究の背景

- 開発の多様化
  - 開発ライフサイクル、ツール、プラクティス
  - 新技術の適用などで開発
- プロジェクトの分析が困難
  - 各要因の生産性や品質に及ぼす影響を大きさを把握するのが困難

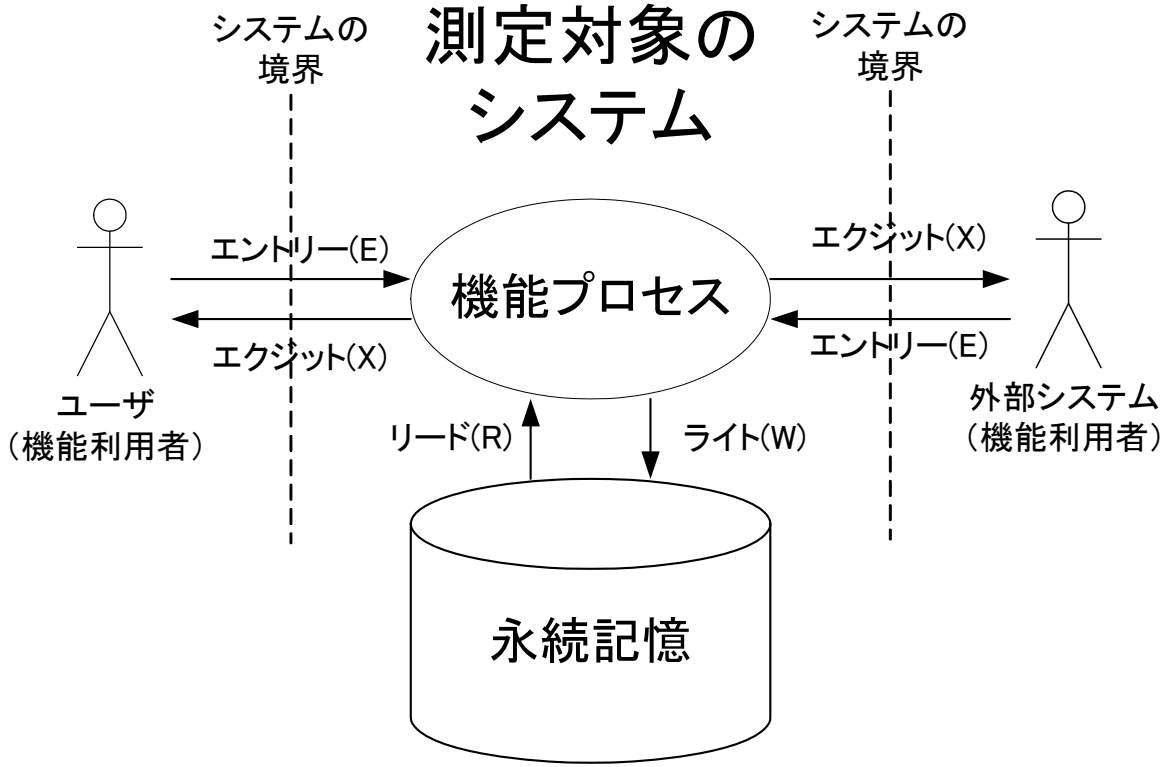
特に受託開発で開発の形態が多様化している

研究の狙いとアプローチ

- プロジェクトの生産性に大きな影響を与える要因を特定する
- そのために、以下の2つに基づく分析手法を研究している
  - 機能規模測定手法COSMIC法
  - 作業部分と作業種別に基づく労力の記録

COSMIC法とは

- データ移動に基づいて機能規模を測定する手法



データ移動とデータグループ

- データ移動の種類
  - エントリー(E):システム外からの入力
  - エクジット(X):システム外への出力
  - リード(R):永続ストレージのリード
  - ライト(W):永続ストレージのライト
- データのまとまり(データグループ)を単位として
  データ移動の数を数える(単位はCFP)

機能規模の算出方法

- 正味機能規模
  - システム全体でのデータ移動数
    - E, X, R, Wの総数
- 変更機能規模
  - 変更されたデータ移動数
    - 追加/変更/削除されたデータ移動数

新規開発も改修も開発規模は「変更機能規模」で測定可能！

COSMIC法の特徴

- 要求や既存システムに基づいて機能規模を測定できる
- データ移動に注目しているので、データベースアクセス以外に通信も測定可能
- 測定方法が比較的シンプル、かつ一般に公開されている

COSMIC法はISOやJIS標準の機能規模測定手法！

COSMIC法の長所と短所

- 長所
  - ソフトウェアの規模を実装言語や実装/設計構造と独立して測定することが可能になる
    - 結果的に実装言語や実装/設計構造と独立に開発生産性などを測定することができる
- 短所
  - 開発途上の規模の測定には使いづらい

開発途上の規模の測定はコード行数を併用したほうがよい

COSMIC法と開発生産性

- COSMIC法の使い道
  - 基本的には、測定するための「ものさし」
  - 測定値の応用範囲は様々である

- 例えば、開発生産性…

$$\text{開発生産性} = \frac{\text{変更機能規模}}{\text{実績労力}}$$

開発生産性に影響する要因

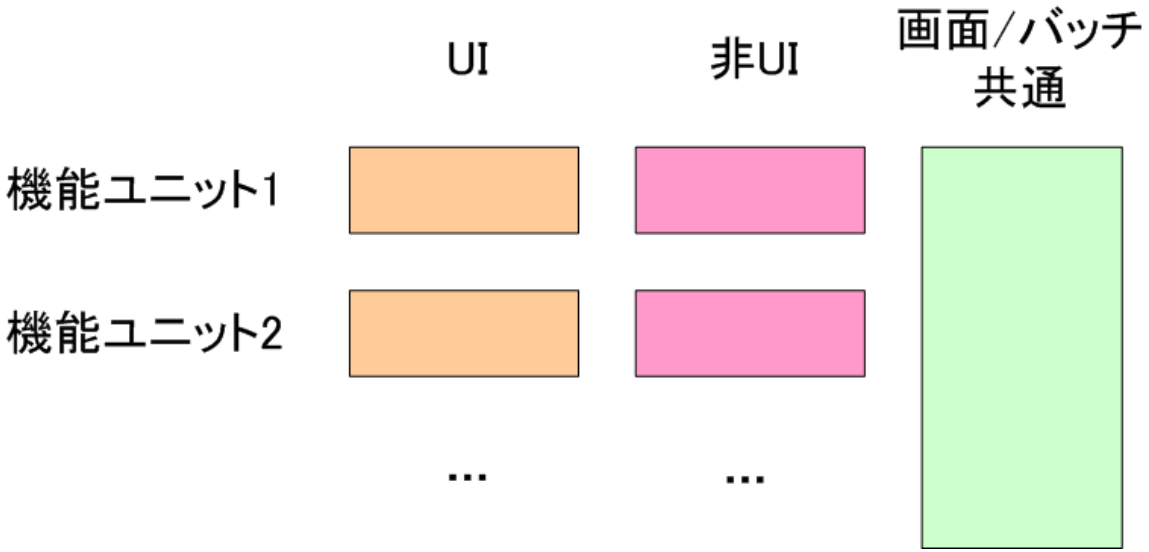
- 以下が本研究で注目している要因
  - 共通機能の括りだしの度合い
  - ユーザインターフェイス(UI)の難易度
  - 開発プロセスの重さ(ドキュメント量)
  - 協力会社の作業スキルや適合性
  - 指摘事項/欠陥の検出密度
  - 開発途上での開発依頼者からのフィードバックの量
  - 単体テストが自動化されている割合

機能ユニット

- 機能ユニットとは、画面、バッチ、ユースケースなどの大きなレベルでソフトウェア機能を分割したものである
- 機能ユニット単位で機能規模を求めることができる
  - COSMIC法の機能規模は、機能ユニットと同じ利用者機能要件（FUR）単位で求める

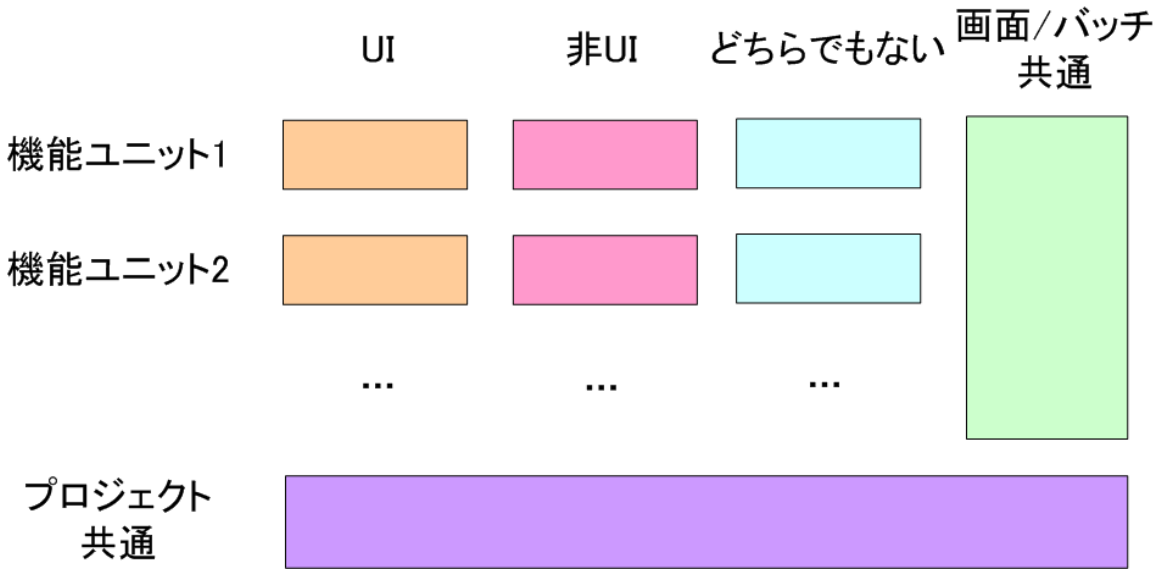
機能ユニットの分解

- UIの難易度や共通機能の括りだしなどの因子の影響を明らかにするため、機能ユニットを3つの部分に分解する



発生しうる作業の考慮

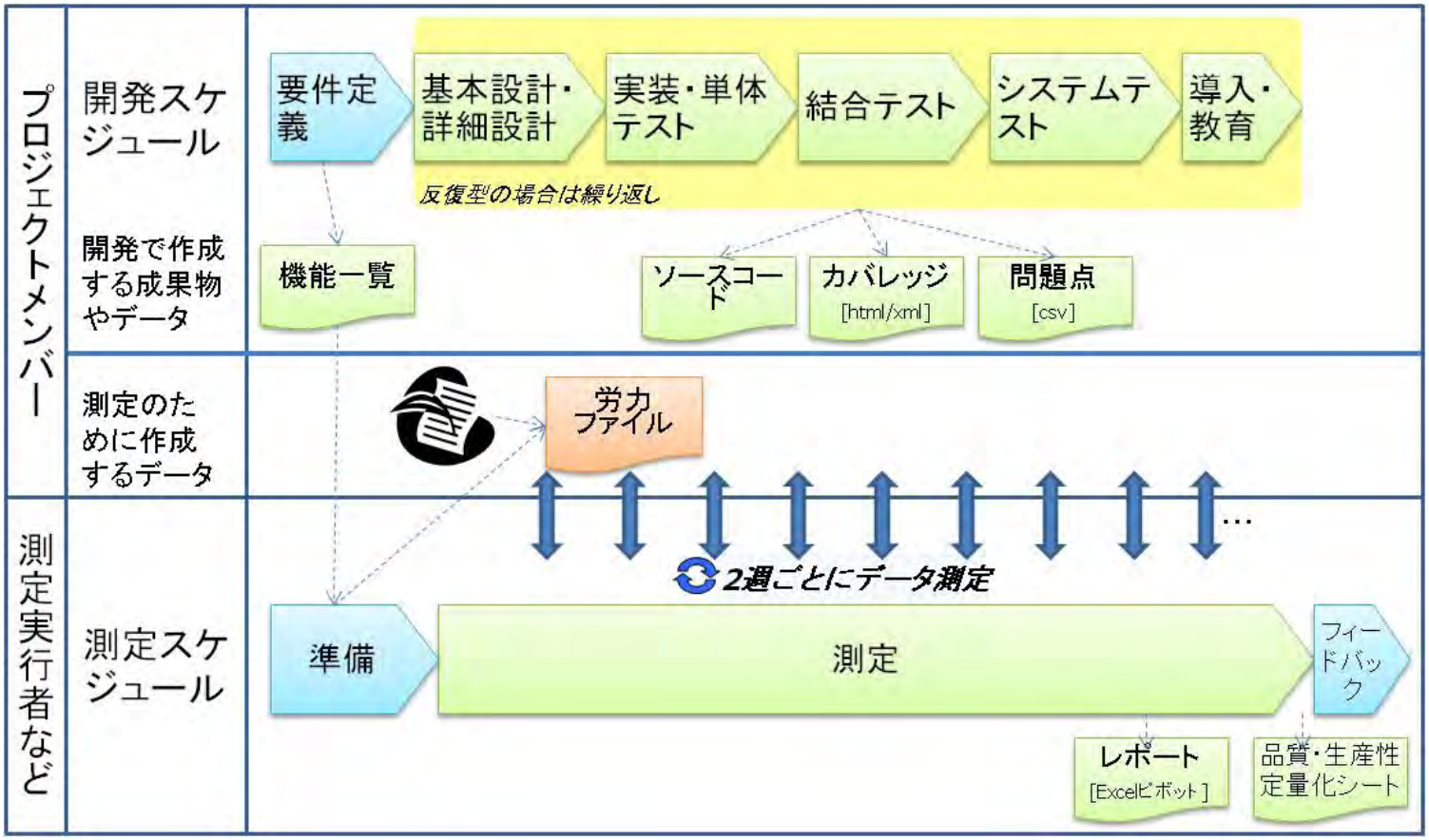
- 開発作業にはUI/非UIのどちらかに分類しづらい作業や機能ユニット（共通）に分類しづらい作業も発生する



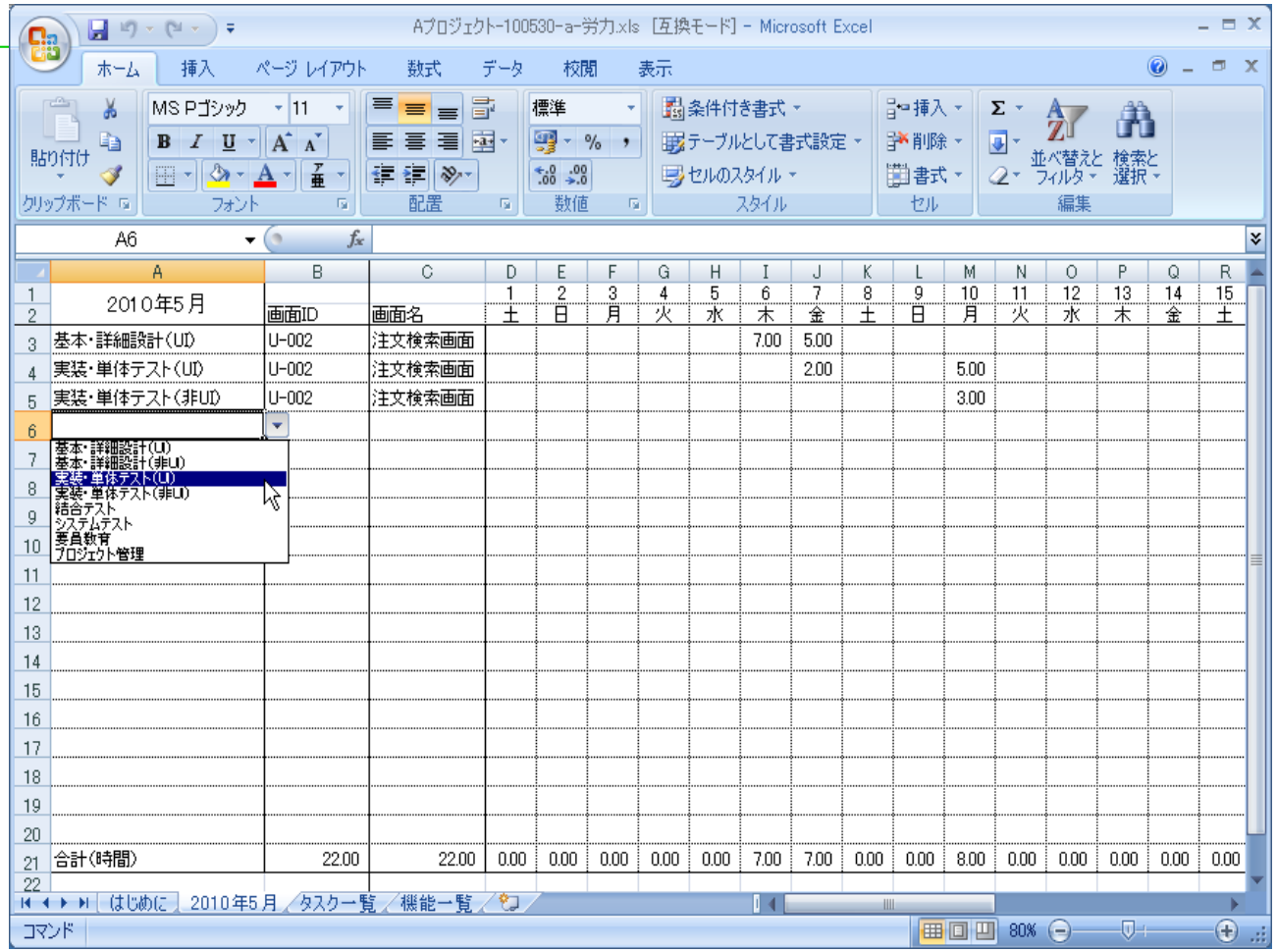
作業種別の定義

		UI/非UI分類		
		UI	非UI	どちらでもない
機能ユニット	画面/バッチ固有	● 詳細設計(UI) ● 実装・単体テスト(UI)	● 詳細設計(非UI) ● 実装・単体テスト(非UI)	● 基本設計
	画面/バッチ共通		● 詳細設計(非UI) ● 実装・単体テスト(非UI)	● 基本設計
	プロジェクト共通			● 基本設計 ● プロジェクト管理

測定の実行形態


プロジェクトメンバー
開発スケジュール
開発で作成する成果物やデータ
測定のために作成するデータ
測定実行者など
測定スケジュール
要件定義
基本設計・詳細設計
実装・単体テスト
結合テスト
システムテスト
導入・教育
反復型の場合は繰り返し
機能一覧
ソースコード
カバレッジ [html/xml]
問題点 [csv]
労力ファイル
2週ごとにデータ測定
準備
測定
フィードバック
レポート [Excelピボット]
品質・生産性定量化シート

労力の記録ファイル


Aプロジェクト-100530-a-労力.xls [互換モード] - Microsoft Excel
ホーム
挿入
ページ レイアウト
数式
データ
校閲
表示
貼り付け
MS Pゴシック
11
標準
条件付き書式
テーブルとして書式設定
セルのスタイル
挿入
削除
書式
並べ替えとフィルタ
検索と選択
クリップボード
フォント
配置
数値
スタイル
セル
編集
A6
2010年5月
画面ID
画面名
基本・詳細設計(UI) U-002 注文検索画面 7.00 5.00
実装・単体テスト(UI) U-002 注文検索画面 2.00 5.00
実装・単体テスト(非UI) U-002 注文検索画面 3.00
基本・詳細設計(UI)
基本・詳細設計(非UI)
実装・単体テスト(UI)
実装・単体テスト(非UI)
結合テスト
システムテスト
要員教育
プロジェクト管理
合計(時間) 22.00 22.00 0.00 0.00 0.00 0.00 0.00 7.00 7.00 0.00 0.00 8.00 0.00 0.00 0.00 0.00 0.00
はじめに
2010年5月
タスク一覧
機能一覧
コマンド
80%

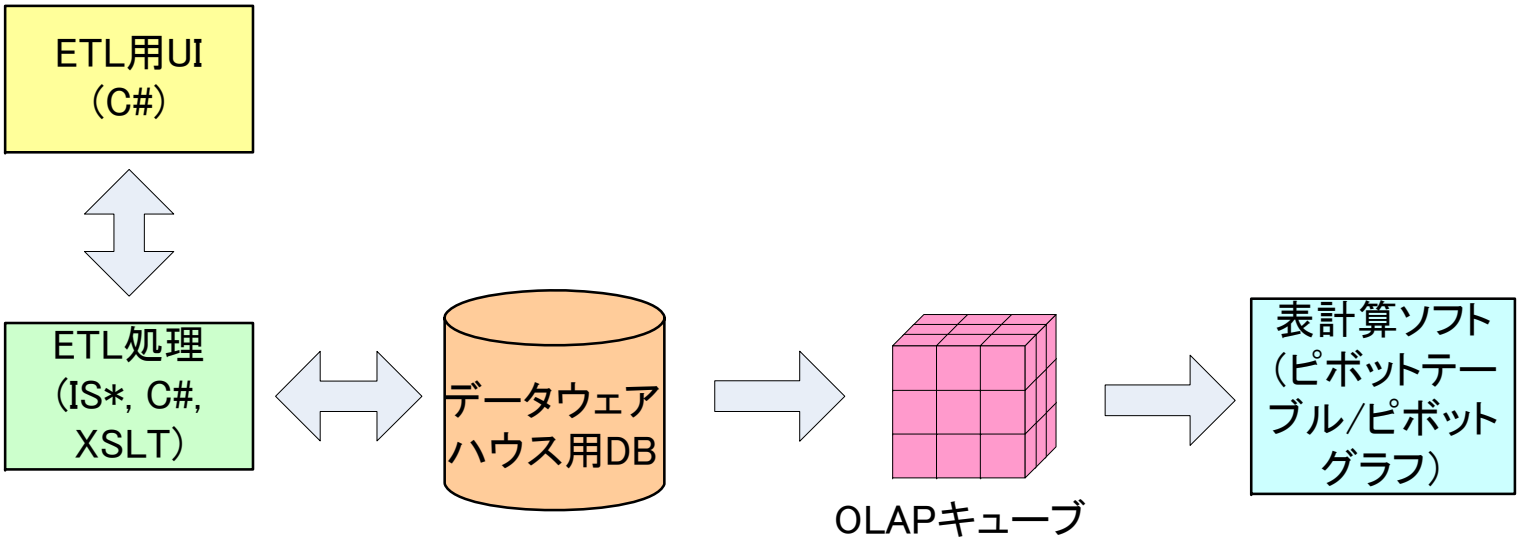
機能規模以外の主な測定データ

入力データ種類	ファイル形式	データの作成者
測定期間	CSVファイル	ソフトウェア工学センター
コード行数	javancss等(*)のXML出力	ソフトウェア工学センター
労力	Excelファイル	プロジェクトメンバー
問題点管理データ	CSVファイル	プロジェクトメンバー
コードカバレッジ	Cobertura 等 の XML 、HTML 出力	プロジェクトメンバー

*: ActionScript、JavaScript、C#の測定データも取り込み可能

IDeAnとは

- プロジェクトの測定データを、BI機能とExcelのピボットテーブル(グラフ)を利用して多面的に分析するためのシステム



*: Integration Serviceのビジュアル言語

今回分析結果を紹介する プロジェクトのプロフィール（1）

プロジェクトID	開発種別	処理の種類	開発プロセス	協力会社さん	フィードバックの有無
B	新規	オンライン+バッチ	インクリメンタル	国内	開発途上での追加・変更あり
C	新規	オンライン	ウォーターフォール	国内	
A2（第2弾）	改修	オンライン	進化型	国内	リリース毎にフィードバック
D	新規	オンライン＋バッチ	ウォーターフォール	国内＋オフショア	
E	新規	バッチ	ウォーターフォール	社員のみ	

以降、比較のためAの第1弾(A1)のデータも示す

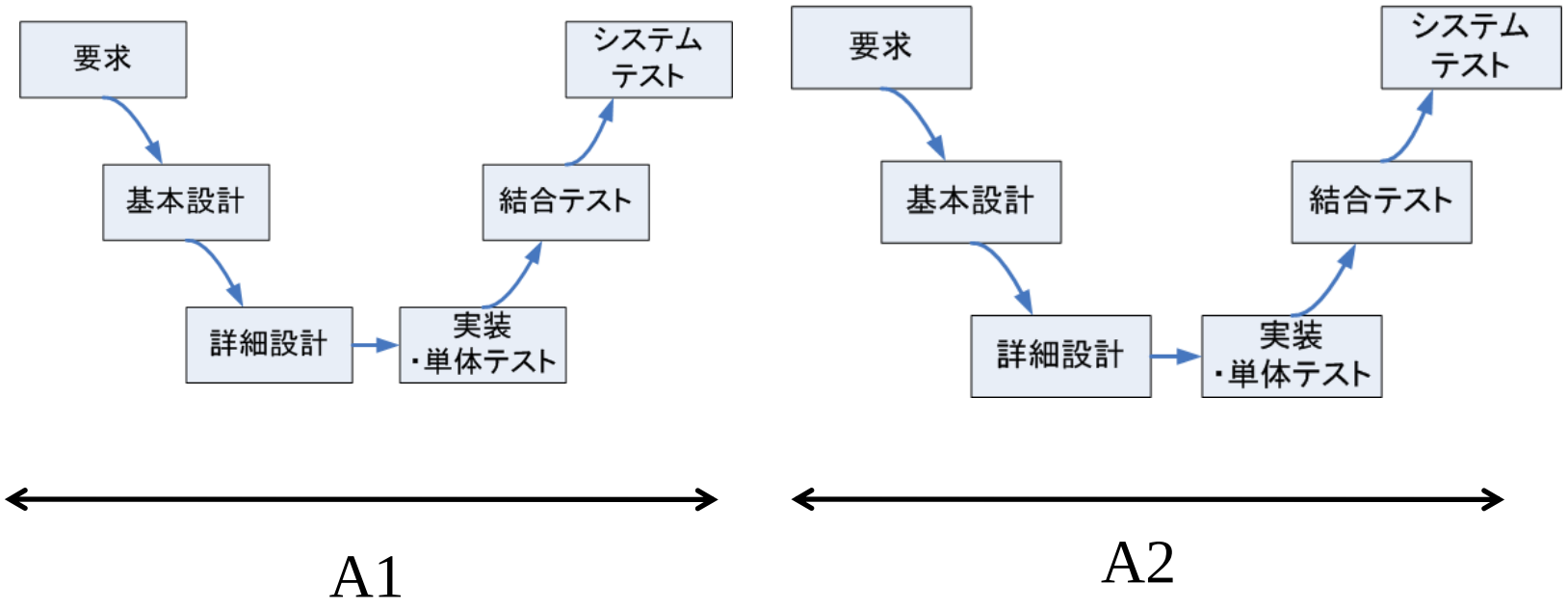
今回分析結果を紹介する プロジェクトのプロフィール(2)

変更

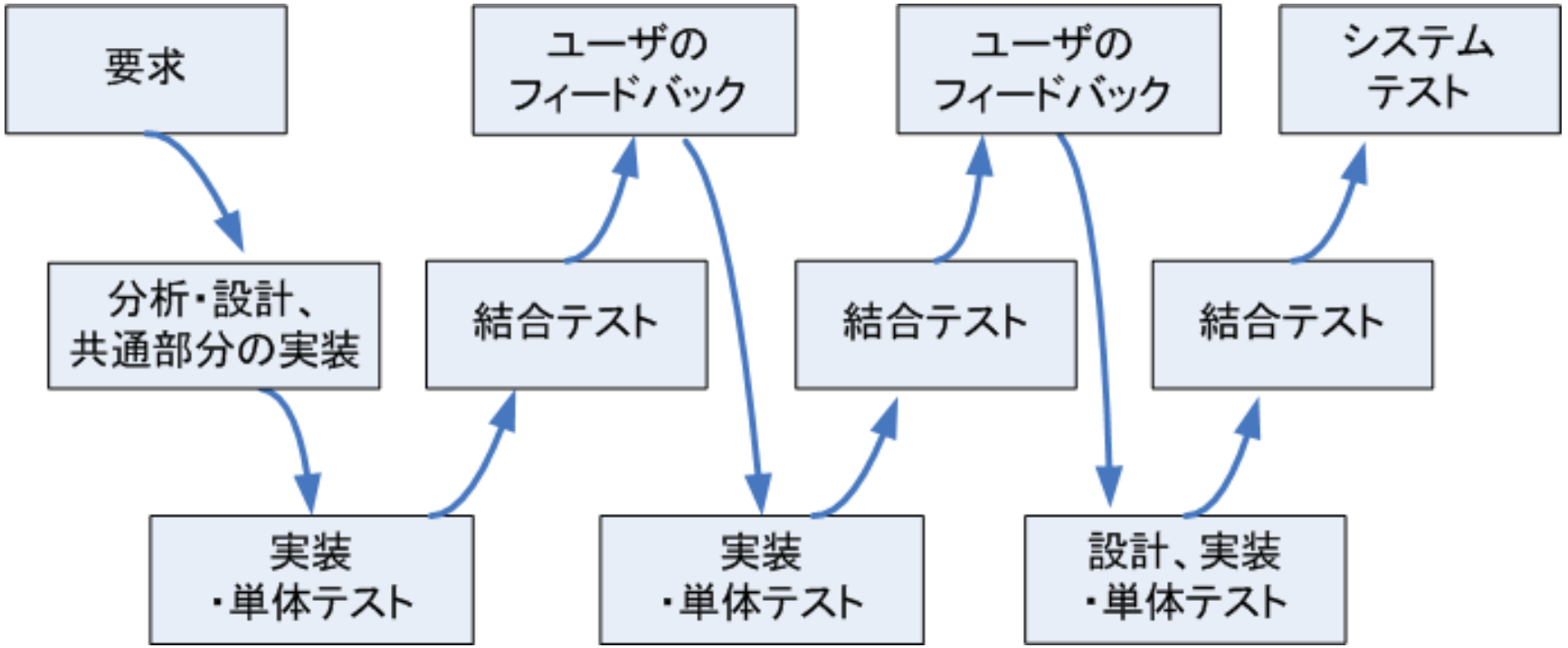
プロジェクトID	プロセスの重さ	単体テストの自動化	UI	備考
B	軽量	ほぼ実装	リッチ	Seaser2+Flex2、バッチ処理はJavaで実装
C	通常		シン	Seaser2,バッチはPro Cで実装
A2（第2弾）	軽量	部分実装	シン	Seaser2、詳細設計作業を削減
D	通常		シン	オンラインはSeaser2
E	軽量			Javaで実装

以降、比較のためAの第1弾(A1)のデータも示す

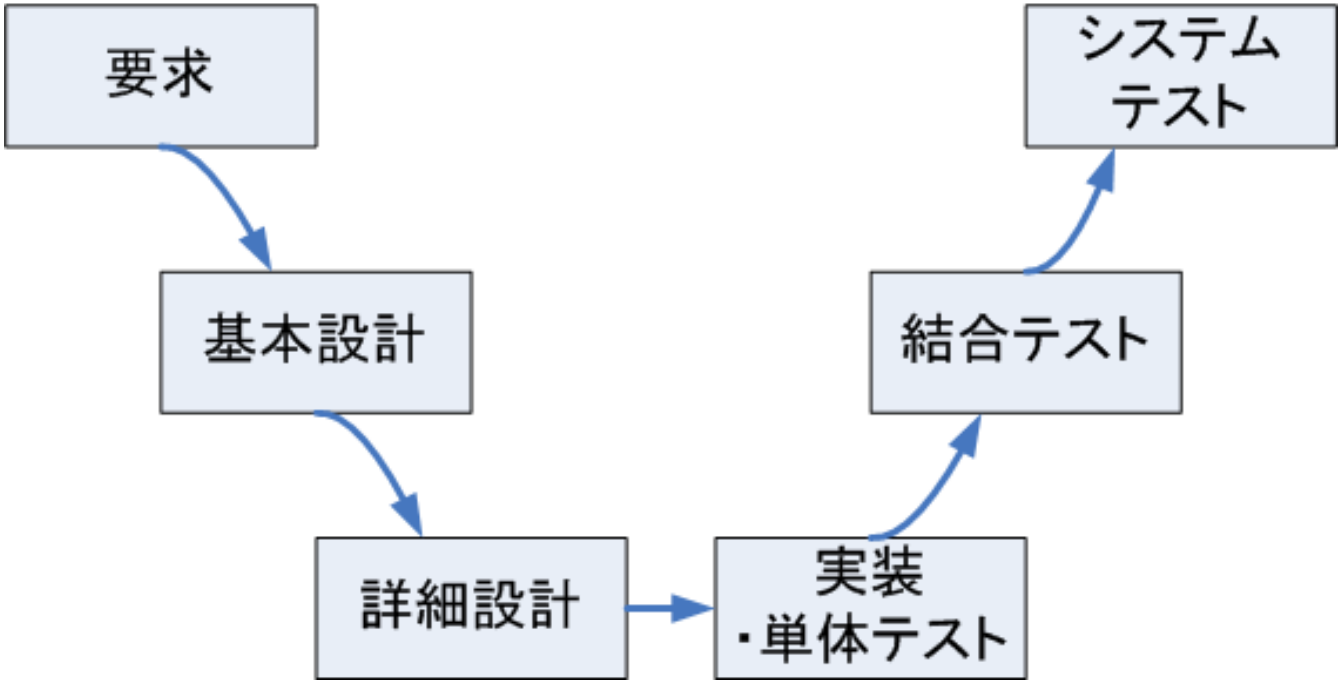
A1, A2の開発ライフサイクル



Bの開発ライフサイクル



C, D, Eの開発ライフサイクル



分析のポイント

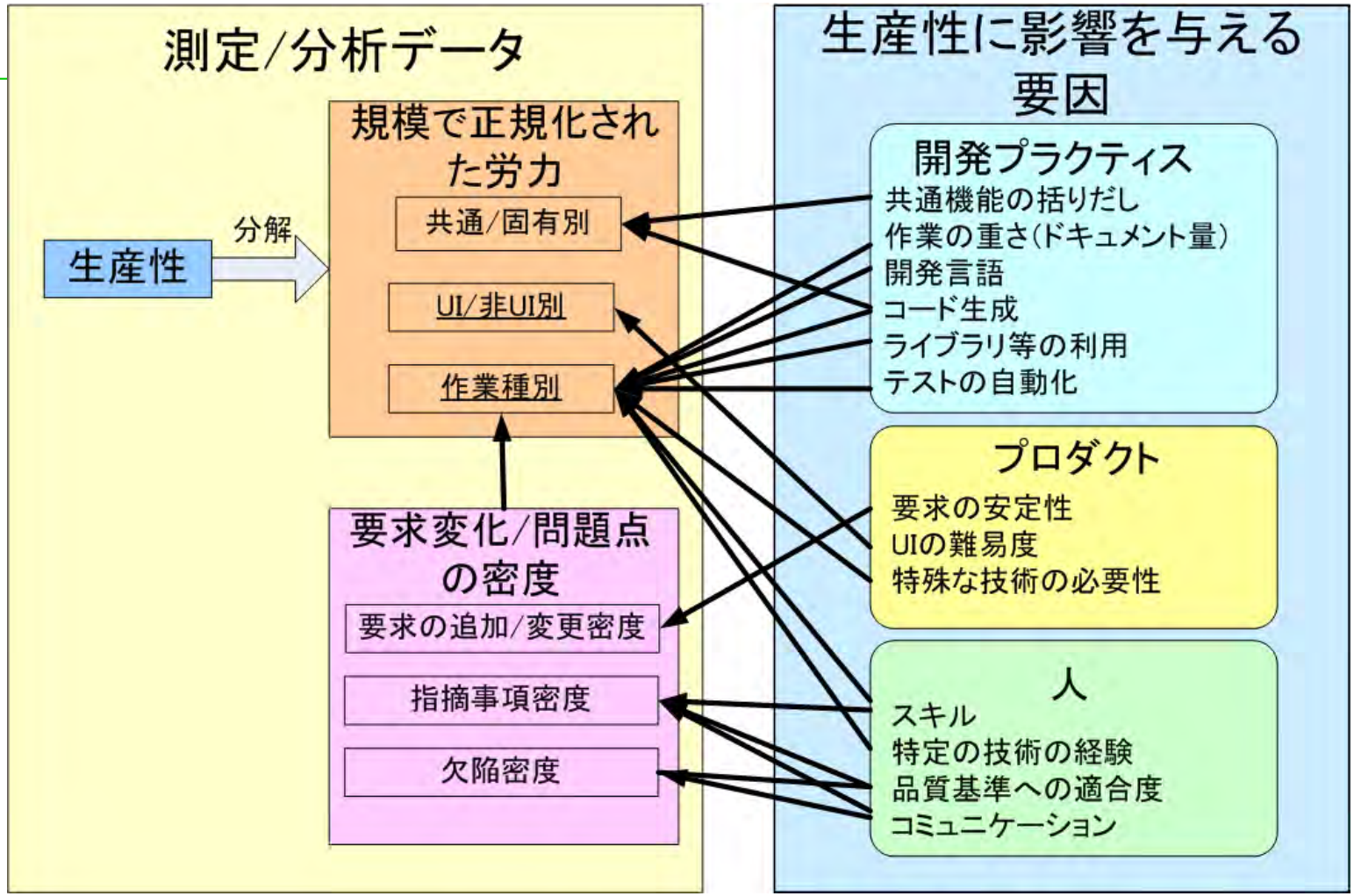
- 以下の違いは生産性にどう影響するか？
  - 開発種別の違い：新規/改修
  - 処理の種類の違い：オンライン/バッチ
  - 協力会社さんの違い：国内/オフショア
  - 開発プロセスの違い：通常/軽量
- 生産性のバラつきの要因を特定できるか？

生産性のバラつきの要因分析方法

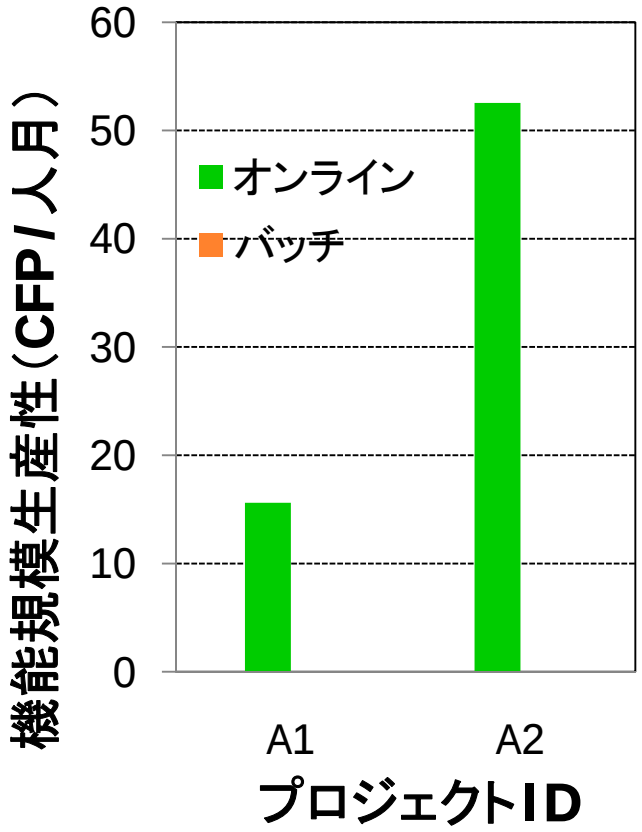
複数の分析値を異なるグループ間で比較することで、バラつきの要因を特定する

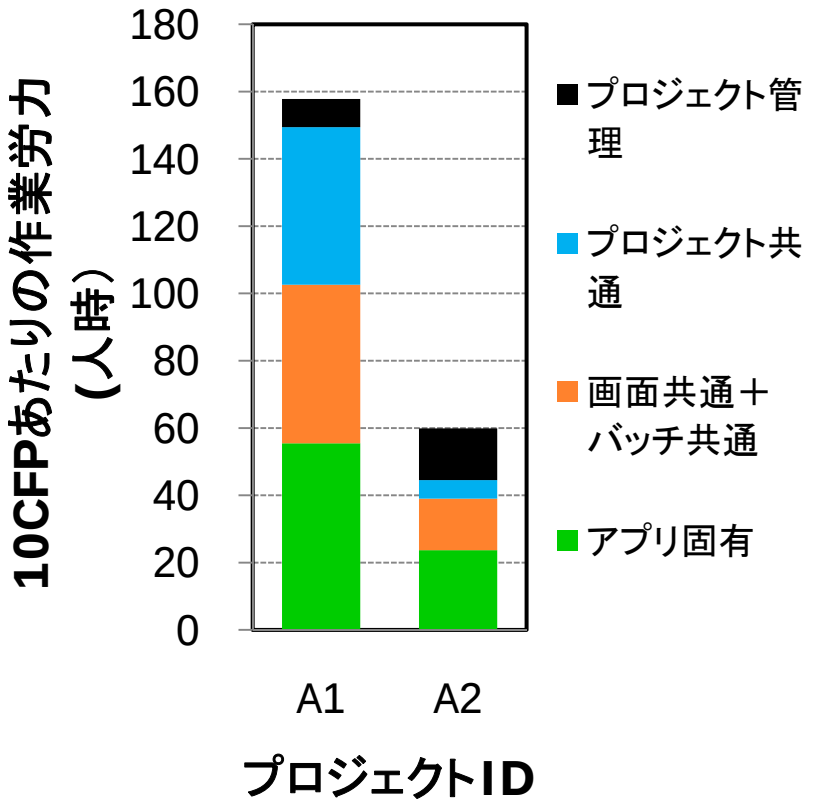
- 分析値
  - 機能規模生産性
  - 労力比
  - 機能規模で正規化した労力
  - 機能規模で正規化したコード量
  - 指摘密度/欠陥密度
- グループ
  - プロジェクト
  - アプリ固有/アプリ共通/プロジェクト共通
  - バッチ/オンライン
  - 協力会社さん
  - 機能ユニット

データと要因の関係

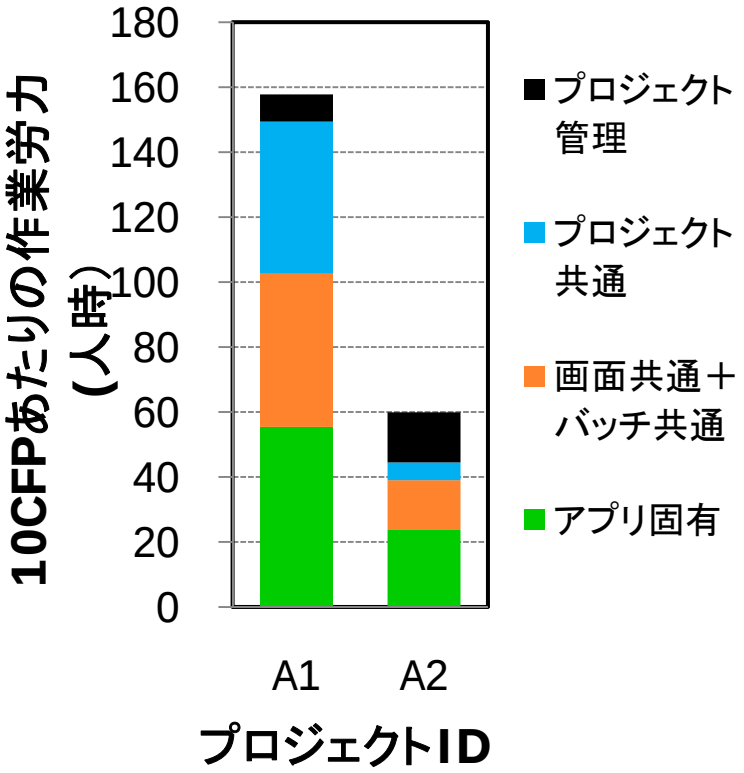

測定/分析データ
規模で正規化された労力
共通/固有別
UI/非UI別
作業種別
分解
生産性
要求変化/問題点の密度
要求の追加/変更密度
指摘事項密度
欠陥密度
生産性に影響を与える要因
開発プラクティス
共通機能の括りだし
作業の重さ(ドキュメント量)
開発言語
コード生成
ライブラリ等の利用
テストの自動化
プロダクト
要求の安定性
UIの難易度
特殊な技術の必要性
人
スキル
特定の技術の経験
品質基準への適合度
コミュニケーション

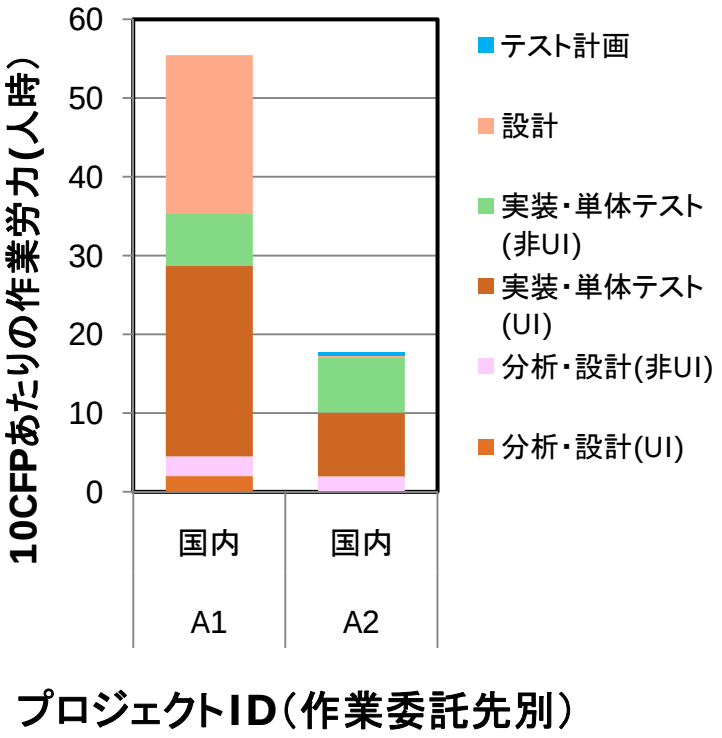
生産性
→単位規模あたりの労力（共通/固有）





単位規模あたりの労力
共通/固有 → アプリ固有

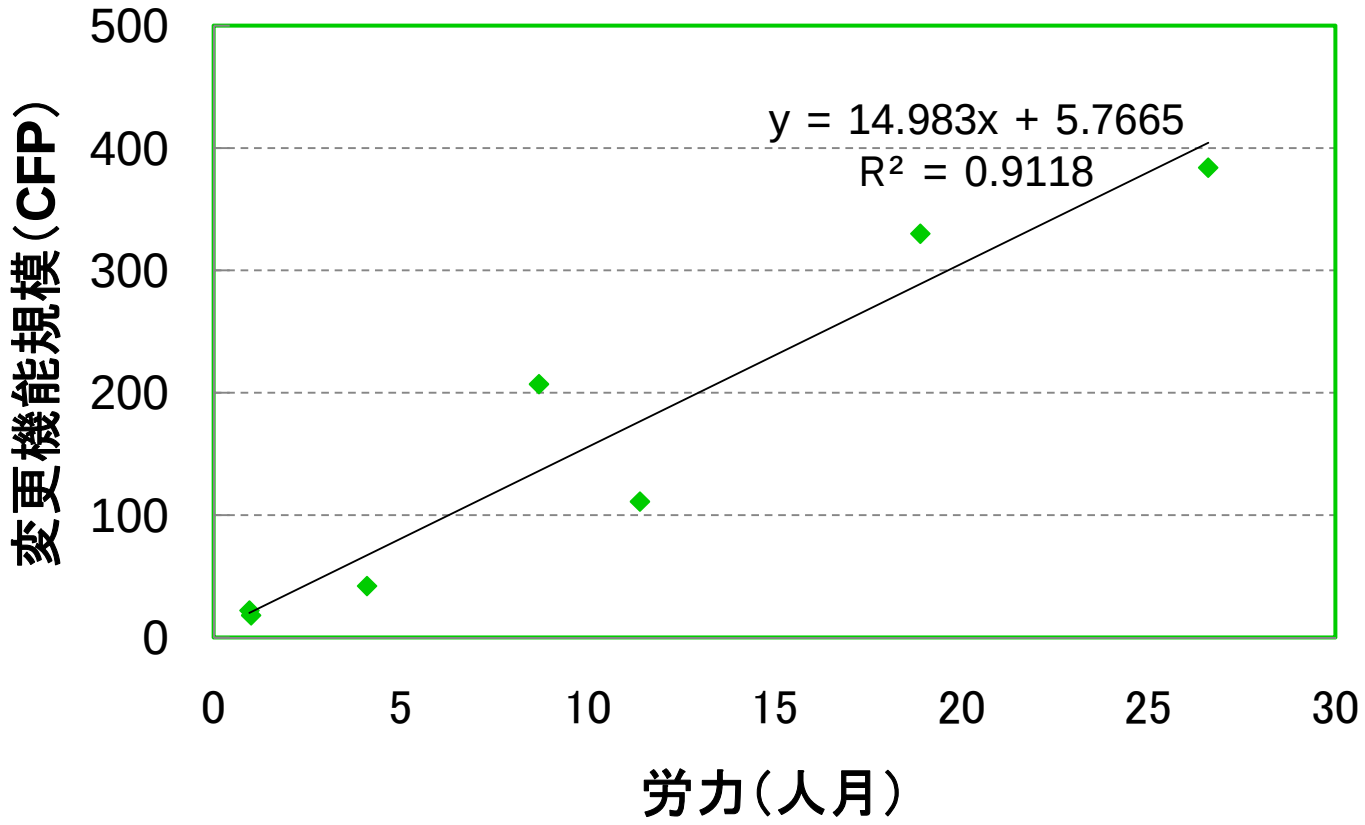




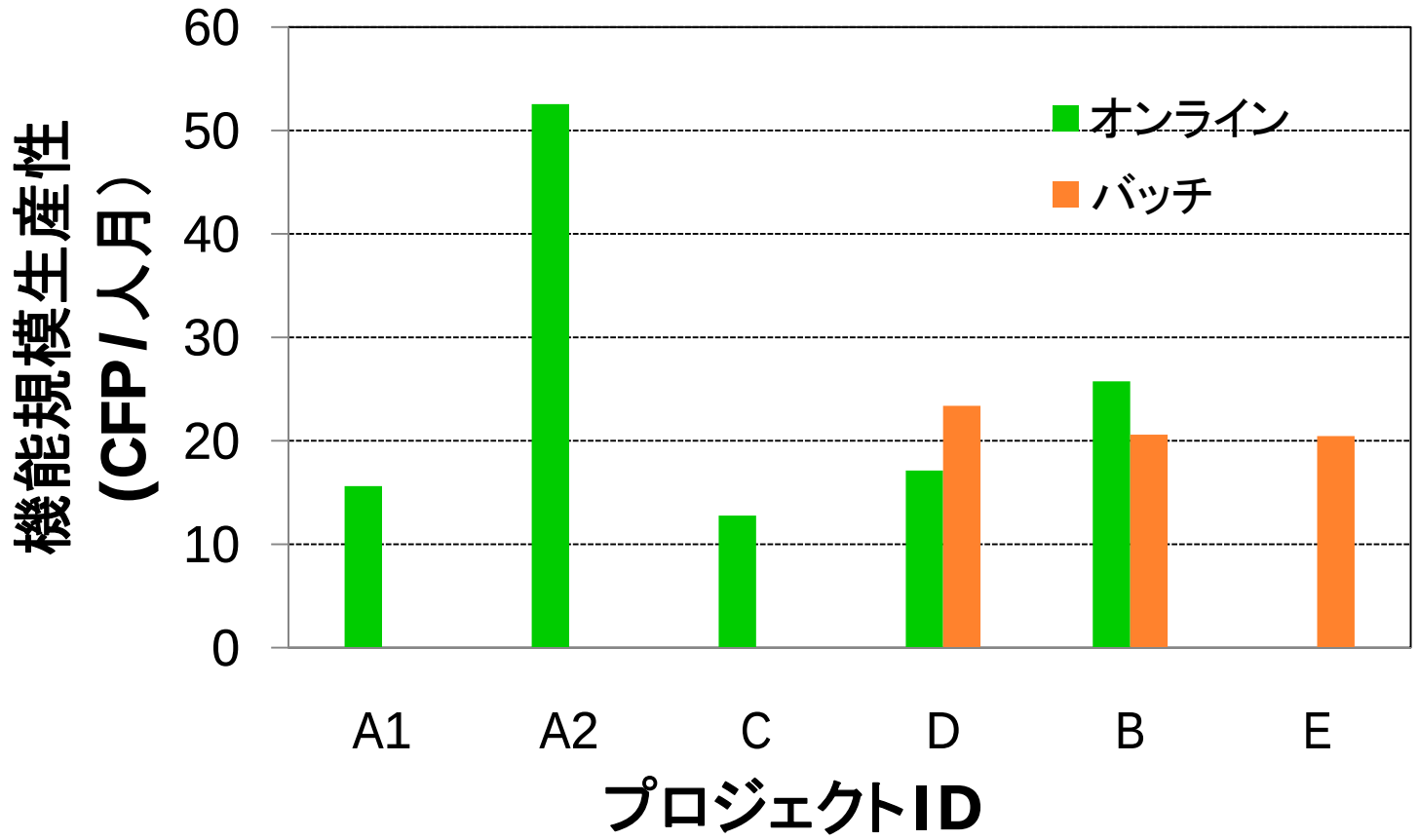
プロジェクトの基本データ
スケジュール、機能規模、労力

プロジェクトID	プロジェクト終了日	開発期間（ケ月）	正味機能規模(CFP)	変更機能規模(CFP)	分析対象労力（人月）	その他の労力（人月）
A1	09/3/31	6	111	111	11.4	0.44
B	09/9/30	4	330	330	18.9	
C	09/10/31	5	42	42	4.1	0.00
A2	10/2/3	6	274	207	8.7	0.91
D	09/12/29	6.4	384	384	26.6	1.94
E	10/3/12	1.2	18	18	1.0	0.2

開発労力と機能規模の相関



機能規模生産性
オンライン/バッチ別

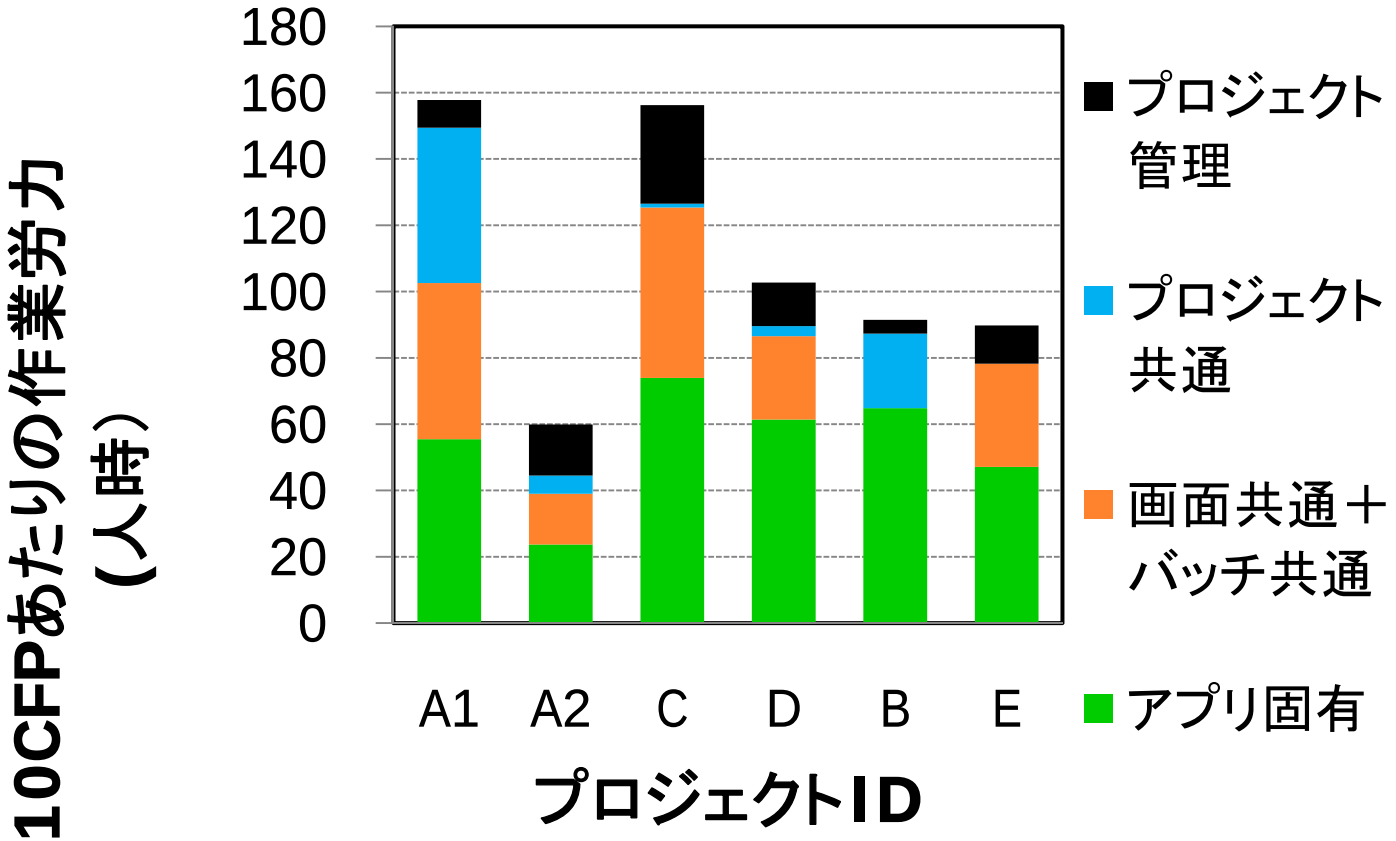


生産性の比較から分かること

- 生産性高→A2
- 生産性中→B, D, E
- 生産性低→A1, C

- オンラインとバッチの生産性はほぼ同じ
  - バッチ処理の生産性測定にも機能規模は有望である！

10CFP当たりの作業時間
(アプリ固有/共通、プロジェクト共通)



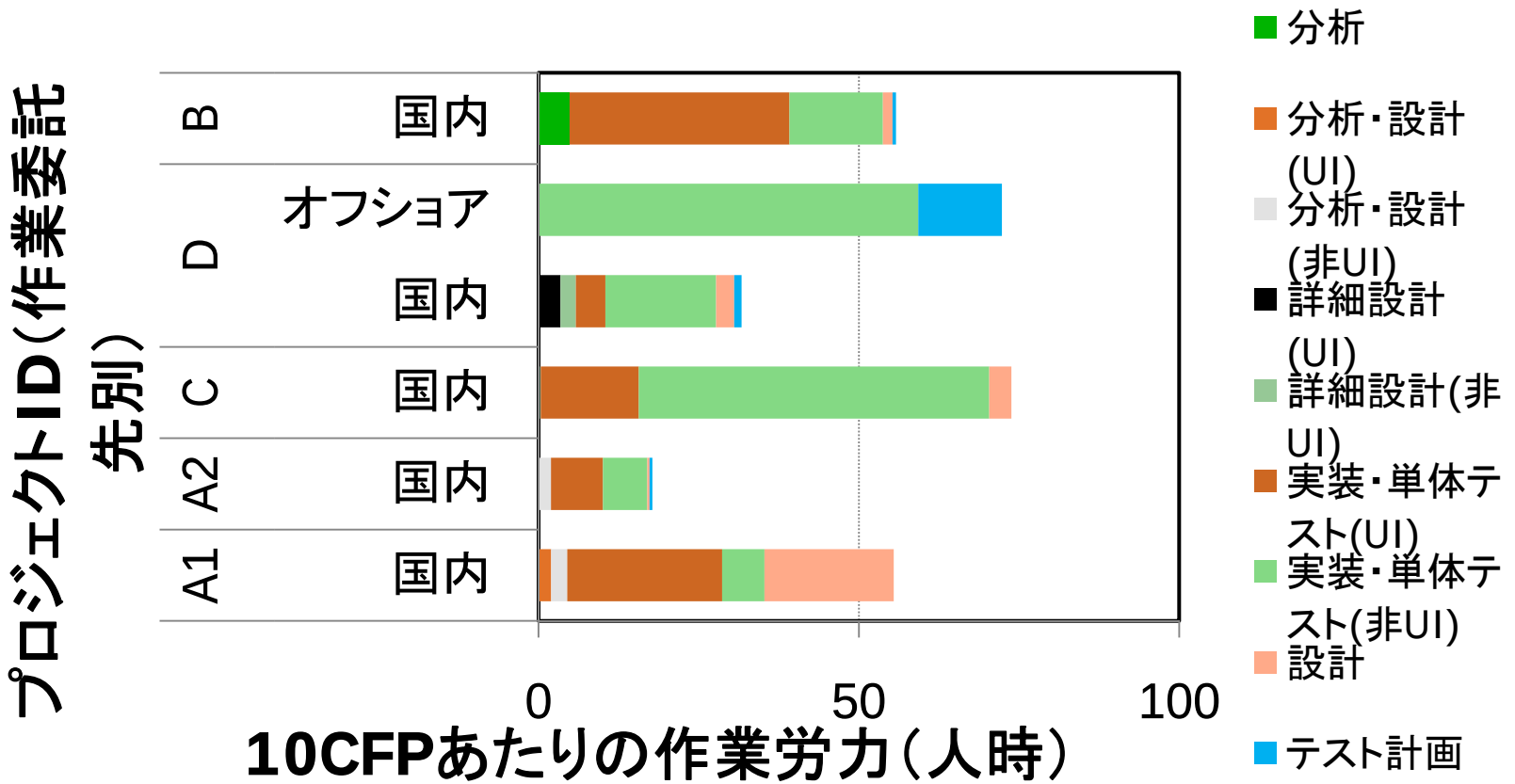
10CFPあたりの作業時間
（アプリ共通/固有、プロジェクト管理）

- アプリ固有が50%以上：B, D, E
- アプリ固有が50%未満：A1, A2, C
- プロジェクト共通が30%以上：A1

アプリ固有小かつ生産性高
→共通化等による生産性の向上等の可能性あり

アプリ固有大かつ生産性低
→共通化等が不十分で生産性が低い可能性あり

10CFPあたりの作業時間（アプリ固有、オンライン）

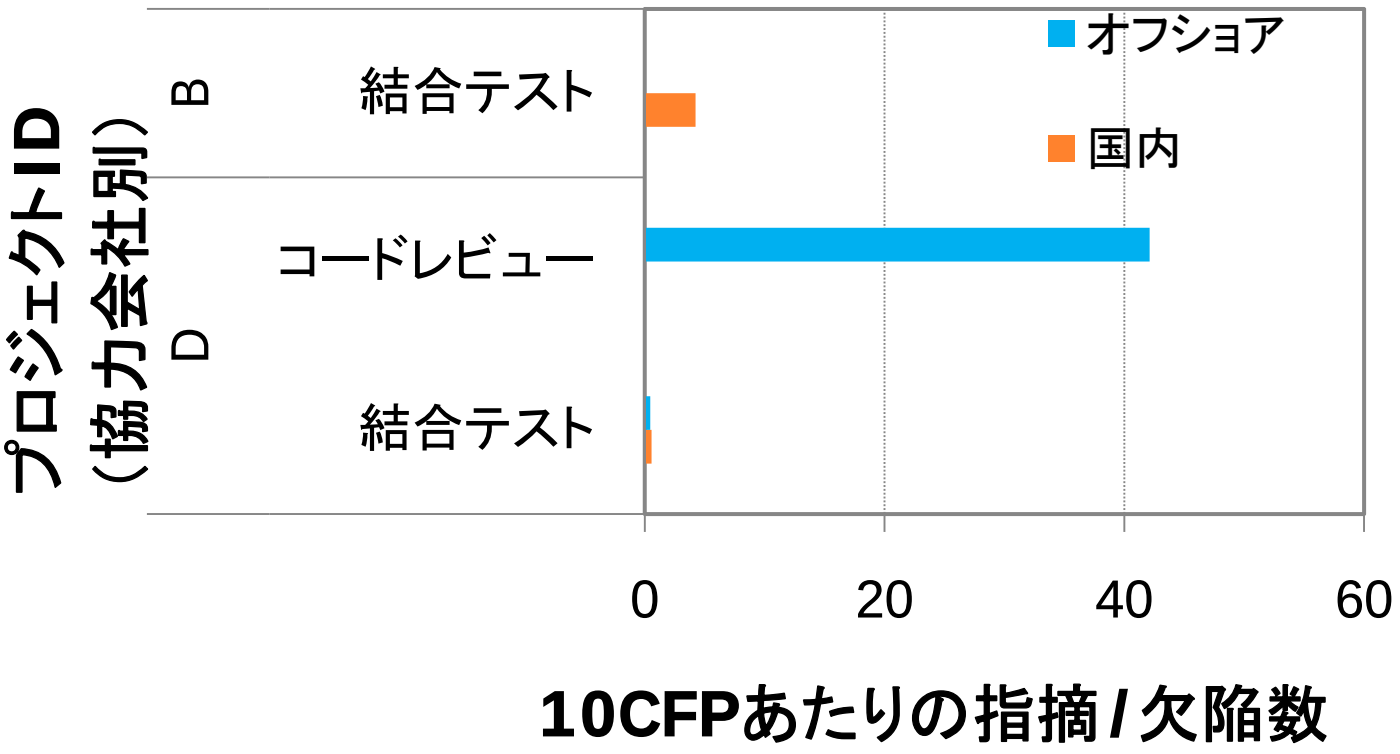


10CFP当たりの作業時間から分かること

- A1→設計と実装・単体テスト(UI)大
- A2→設計と実装・単体テスト(UI)小
- C→実装・単体テスト(非UI)大
- D(オフショア部分＊)→実装・単体テスト大
- B(オンライン)→実装・単体テスト(UI)大

＊:オフショア部分はUIと非UIを分けて労力を記録していない

10CFPあたりの欠陥/指摘数



欠陥密度から分かること

- B
  - Dと比較すると欠陥密度がやや高い
- D
  - 指摘事項密度:オフショア部分がとびぬけて大きい
  - 結合テストの欠陥密度:オフショア部分と国内部分が同等

BとDでは(画面テストを含むか否かという点で)単体テストの定義が異なる

測定/分析結果から分かったこと

プロジェクトID	機能規模生産性	アプリ固有の作業労力	アプリ共通の作業労力	プロジェクト共通の作業労力	設計の作業労力	実装・単体テスト(UI)の作業労力	実装・単体テスト(非UI)の作業労力	コードレビューの指摘事項密度
A1	低	中	大	大	大	大	小	未実施
B	中	中	中	中	中	大	中	未実施
A2	高	小	小	小	中	中	小	未実施
C	低	中	大	大	中	中	大	未実施
Dの国内	中	中	中	中	中	小	中	未実施
Dのオフショア					中	大		大

生産性低下の主要因

プロジェクトID	プロジェクトメンバーのコメント	要因種別
A1	設計作業に不慣れなメンバーで作業効率が悪かった	スキル
	特殊なUI仕様の実現で試行錯誤(再作業)があった	UIの難易度
C	DI(Dependency Injection)コンテナを使った開発が初めてであった	特定技術の経験
Dのオフショア委託分	コードレビューの指摘事項への対応でコードの再作業が多く発生した	品質基準への不適合

生産性向上の主要因

プロジェクトID	プロジェクトメンバーのコメント	要因種別
A2	特殊なUI仕様の実現で試行錯誤(再作業)がなくなった	特殊技術の経験
	（設計作業を軽量化した）	作業の重さ(ドキュメント量)

A2プロジェクトで生産性が向上した可能性が高いと考えるが、新規開発と改修の生産性の比較にはさらにデータの蓄積が必要

開発段階と生産性のバラつき要因

	開発初期	開発初期以降
開発作業	•基本設計 •共通機能と少数のアプリ機能の詳細設計と実装・単体テスト	•残りの機能の詳細設計、実装・単体テスト •結合テスト、システムテスト
バラつきを生む要因	•技術的経験 •試行錯誤 •作業の重さ	•要求の追加/変更 •作業の類似性 •作業の重さ •UIのリッチ度 •品質（基準）

開発段階と生産性

追加

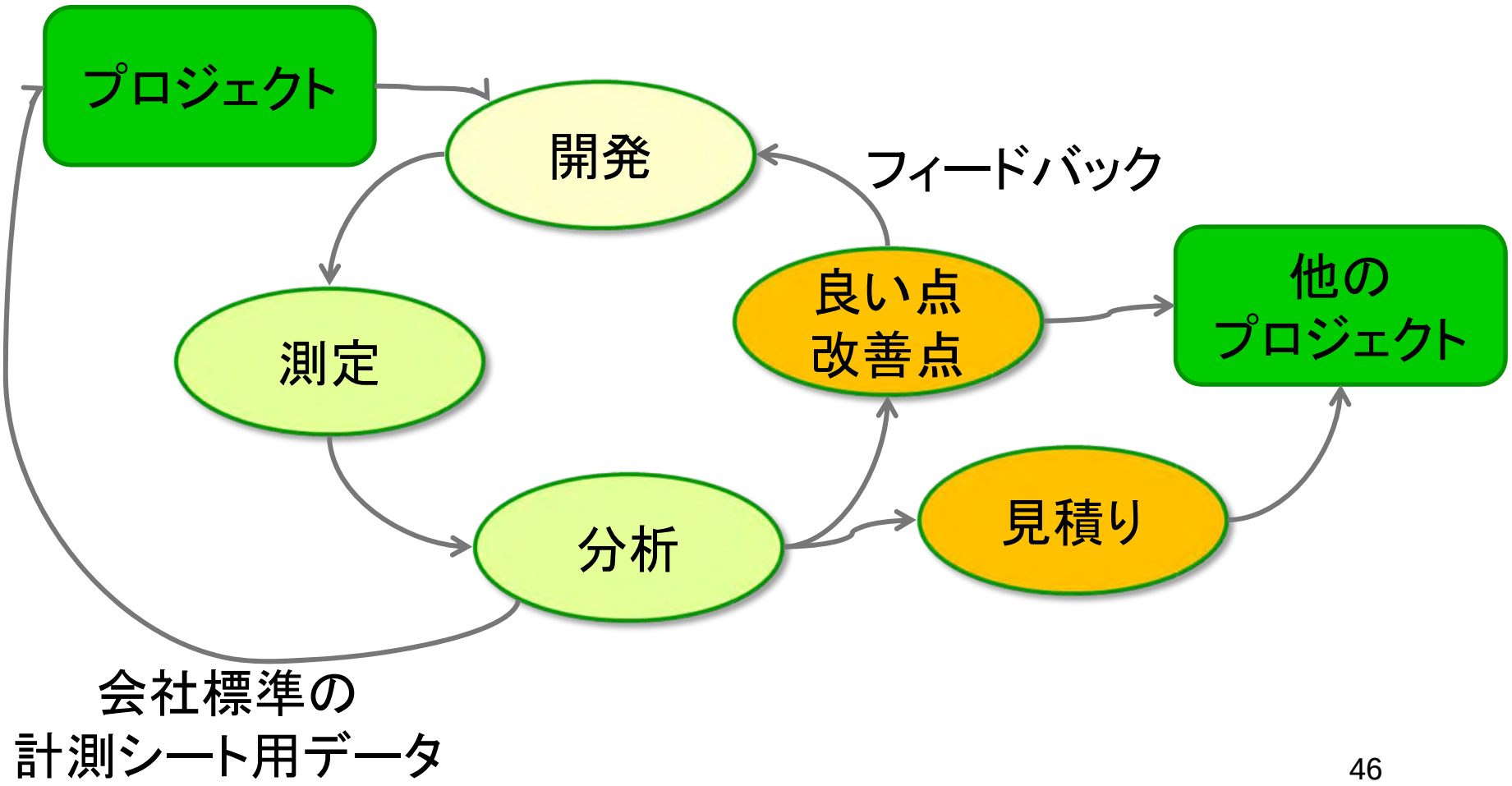
生産性	開発初期 （立ち上がり）	開発初期以降 （定常状態）
高	（A2）	A2
中	B, D（国内）	B, D（国内）
低	A1, C	A1, C, D（オフショア）

A2は改修なので、開発初期に不確定要素は少ない

結論

- 新規開発
  - 機能規模、機能部分と作業種別に基づく労力の記録を行うことで、開発生産性に大きく影響した要因を特定できるようになった
- 改修
  - 変更機能規模等々を利用することで、開発生産性に大きく影響した要因を特定できるようになった
  - 但し、プロジェクトの測定数を増やして変更機能規模と生産性の関係をさらにはっきりさせる必要がある

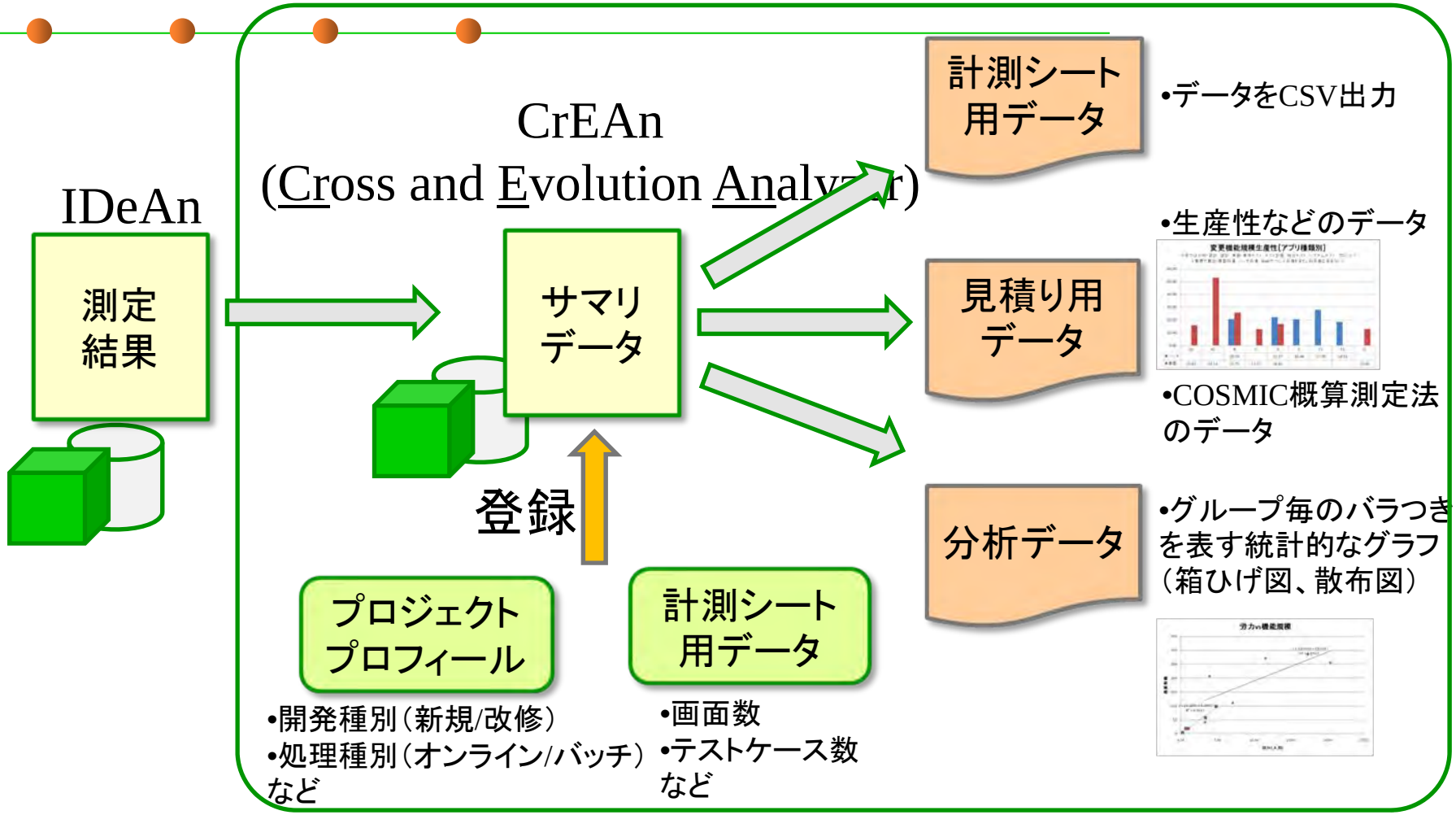
測定結果の活用のイメージ



今後の課題

- 改修の場合について、変更機能規模と生産性の関係についてさらに検証を行う
- 測定結果をさらに活用するためのシステムを開発する（プロジェクト間、世代間の比較）

測定結果の検索とレポート機能



参考文献（その１）

- COSMIC 機能規模測定法マニュアル第3.0版　ー 測定マニュアル, 2007年9月
  - 日本ファンクションポイントユーザグループ（JFPUG）のWebサイト（http://www.jfpug.gr.jp/cosmic/CFFP-index.html）から入手可能
- COSMIC-FFPによる業務アプリケーションソフトウェア規模測定の指針（第1版）, 2005
  - 業務系アプリケーションに即した例とともに機能規模の測定指針を解説
  - COSMIC ver2.2ベースなのでver3.0とは用語が少し異なる点に注意してください！
  - 上記JFPUGのサイトで提供中

参考文献(その2)

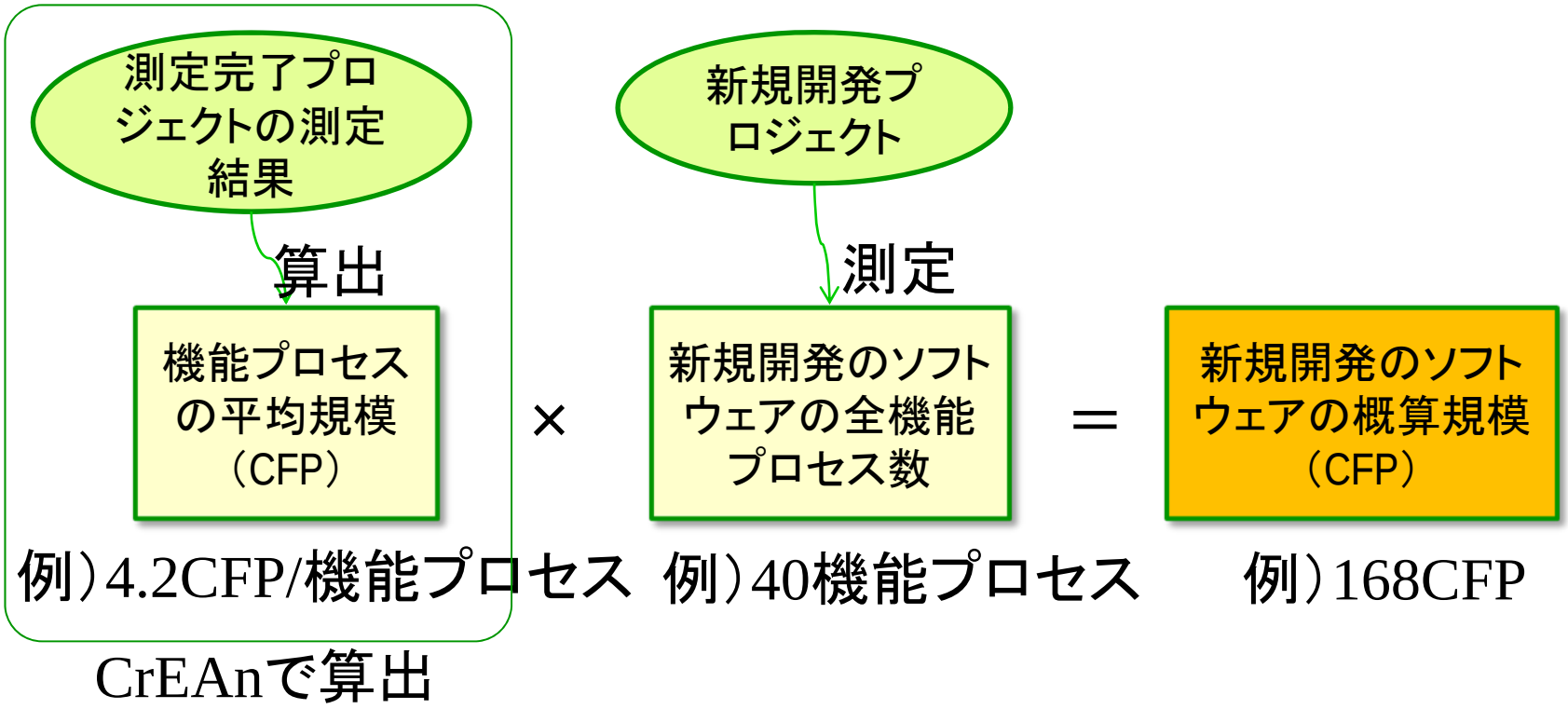
- オブジェクトの広場「ビジネスアプリケーション開発者のための機能規模測定手法 COSMIC 法入門」, 2010年6月から連載中
  - COSMIC法によるビジネスアプリケーションの機能規模測定方法のチュートリアル
  - http://www.ogis-ri.co.jp/otc/hiroba/technical/IntroCOSMIC/index.html
- 藤井拓, 木村めぐみ, COSMIC 法で求めた開発生産性バラつき要因の分析, 情報処理学会研究報告2010-SE-170, 2010

参考文献(その3)

- COSMIC 機能規模測定法 3.0版 応用編,2007年12月
  - COSMICとIFPUG機能規模測定結果の相関、COSMIC法の概算法
  - JFPUGのサイト(http://www.jfpug.gr.jp/cosmic/CFFP-index.html)から入手可能

COSMIC概算法の例

- 例）平均機能プロセス数法



※COSMIC概算法には4つの方法がある。上記の方法はそのうちの1つ。