機能規模測定手法COSMIC法を用いた開発生産性の分析事例の報告

(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、細川　亮二*、永井　学、木村めぐみ

*: 現在の所属は、オージス総研BS3部J2T

報告のアウトライン

- 研究の背景
- 測定結果
- 測定結果の考察
- 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
  - 各因子が生産性や品質に及ぼす影響を分析するのが困難

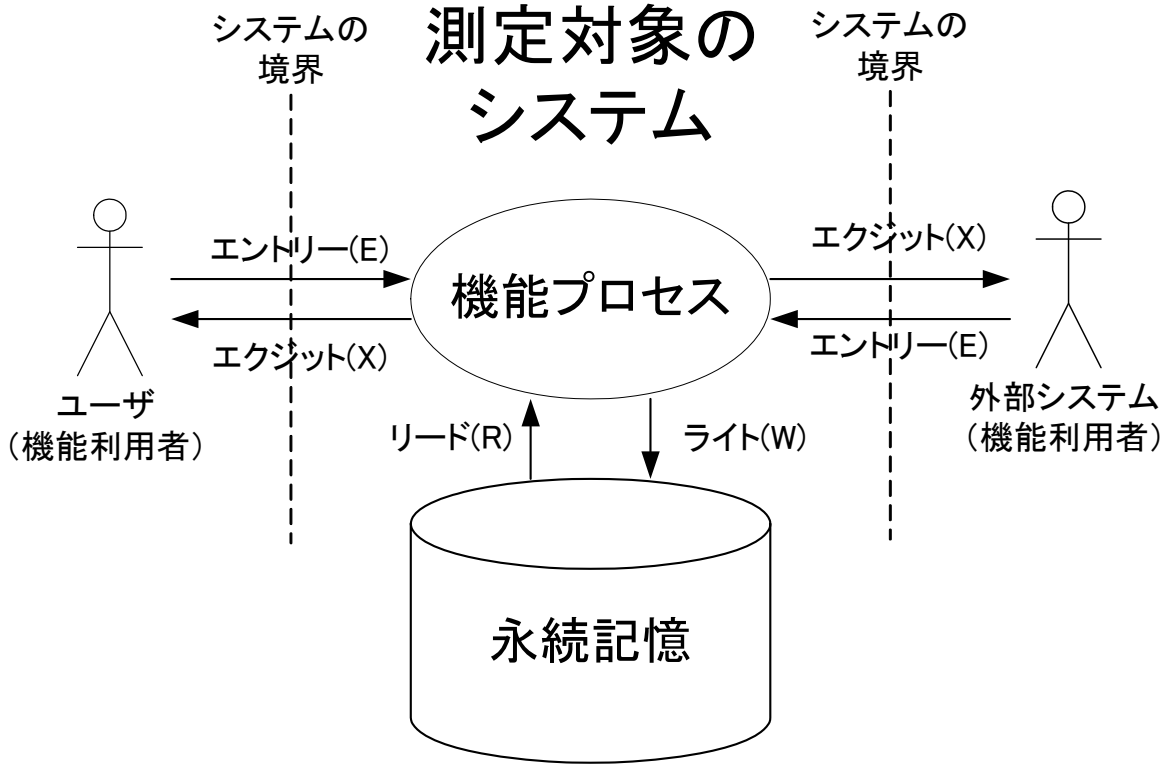
特に受託開発で開発の形態が多様化している

研究の狙いとアプローチ

- 様々な開発プロジェクトを測定することで開発上で有効な技術や開発プラクティスを見出す
  - 場合により、有効なものは異なるかもしれない
- そのために、以下の2つの手段が有効だろうと考えている
  - 機能規模測定手法COSMIC法
  - プロジェクト分析システムIDeAn

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



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…

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

開発生産性に影響する因子

- 開発生産性は、以下のように様々な因子の影響を受ける
  - 開発能力
    - プロセス（作業とその順序、役割、成果物、ガイドライン）
  - 経験（ドメイン、技術）
  - 技術的難易度
  - 開発条件（期間、チーム規模）
  - 技術的な工夫
    - 共通機能を括りだしたり、再利用や自動化により開発を効率化する
  - （仕様などの）変更や追加

プロジェクト毎にこれらの因子の大きさが異なるので、どの因子の影響度が大きいかを特定するのは難しい

プロジェクト内の開発生産性と画面生産性

- 同一プロジェクト内であれば、開発生産性に影響する因子は限定できる
  - 例えば、「開発プロセス」の影響は同じだと考えられる

- 例えば、画面生産性

$$画面生産性_{画面} = \frac{機能規模_{画面}}{実績労力_{画面}}$$

同一プロジェクト内での生産性のバラつきの主な因子

- 「開発能力」、「経験」、「開発条件」が一定だとすれば、残る因子は以下の3つ
  - 技術的難易度
  - 技術的な工夫
  - 変更や追加

分析の視点

- 画面生産性
  - 生産性のよい画面と悪い画面があるか
- 労力配分
  - 生産性のバラつきは労力配分の違いに起因するか
    - 作業種別＝「作業部分」X「作業」
    - 作業部分として「UI」系の作業、「非UI」系の作業に注目
- コードの内訳
  - 画面固有と画面共通の配分を調べることで、技術的な工夫による改善の余地があるか

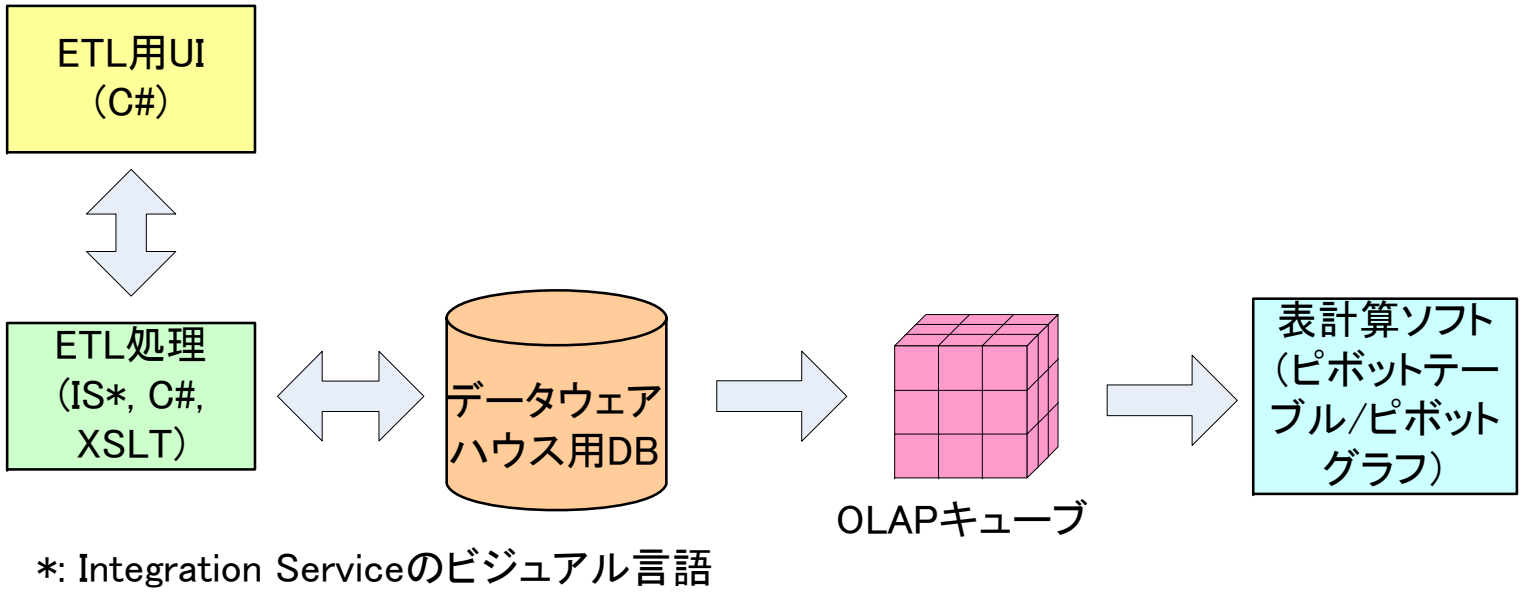
プロジェクトのメンバーの意見（実感）も大事！

作業種別の定義

		中分類		
		UI	非UI	どちらでもない
大分類	画面	● 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

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

今回の報告のポイント

- 画面生産性のバラつきはあるか？
  - どの程度の範囲でばらつくか
- 画面生産性のバラつきの要因を特定できるか？

測定対象のプロジェクト

- 対象：ビジネス系のWebアプリケーション
- 開発言語：Java、JSP（一部JavaScript)
- アーキテクチャ：J2EE
  - OJF(Seaser2)を適用
- 体制：社員＋協力会社
- 開発プロセス：インクリメンタルな開発プロセス（複数ウォーターフォール）

目的：アプリケーションの基本機能を実装し、評価する
（結合、システムテストは簡略に行われている）

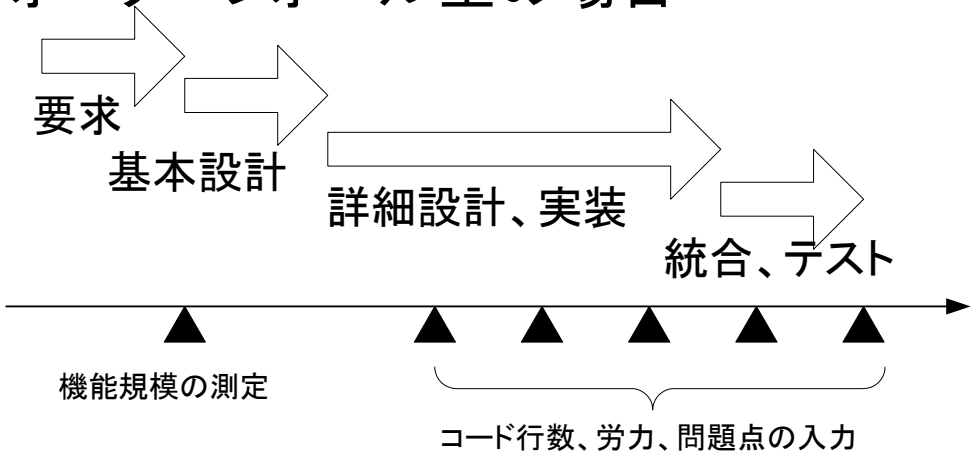
測定の概要

主な測定データ	測定タイミング
機能規模	開発当初
Javaコード行数	2週間周期
労力	2週間周期(*)
コードカバレッジ	2週間周期

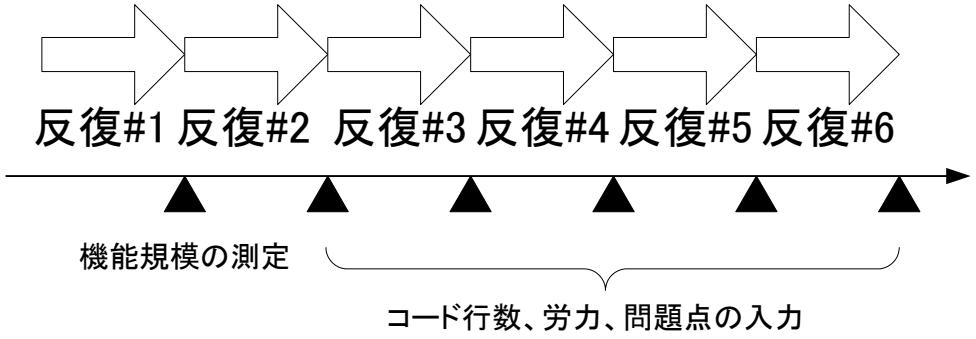
*: 労力は、作業者x作業部分x作業種別x日付で記録

COSMIC+IDeAnによる測定

ウォーターフォール型の場合



反復型の場合



機能規模の測定

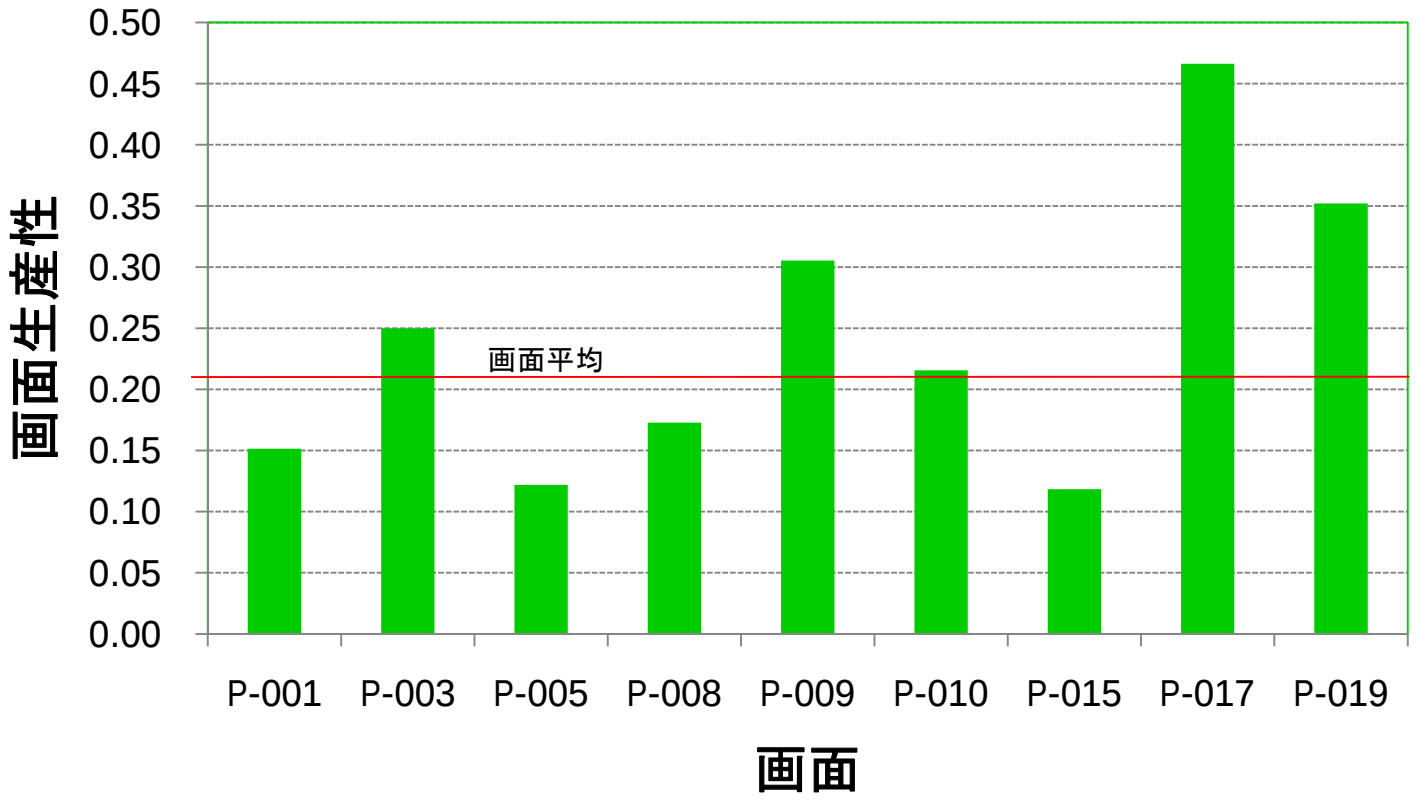
- 情報源
  - ユースケース定義
  - 画面定義
  - 不明点は質疑（30分程度）
- 測定の時期と所要時間
  - 開発の提案段階
  - 半日程度
- 備考
  - 実際の開発範囲は、当初の提案の半分に削減された

機能ユニット毎の機能規模 、画面難易度、労力、コード行数

機能ユニットID	機能規模	難易度	全コード量	本体コード量	テストコード量	作業時間
P-001	10	高	304	132	172	66
P-002	0	低	27	27	0	15.7
P-003	15	高	635	257	378	60
P-005	11	中	559	264	295	90.25
P-008	28	高	3134	1208	1926	162
P-009	20	高	1425	684	741	65.5
P-010	9	高	944	556	388	41.75
P-015	5	中	436	208	228	42.25
P-017	19	中	1174	364	810	40.75
P-019	11	中	932	480	452	31.25
プロジェクト共通	10	中				676.5
画面共通	10	中	10893	6328	4565	530.25

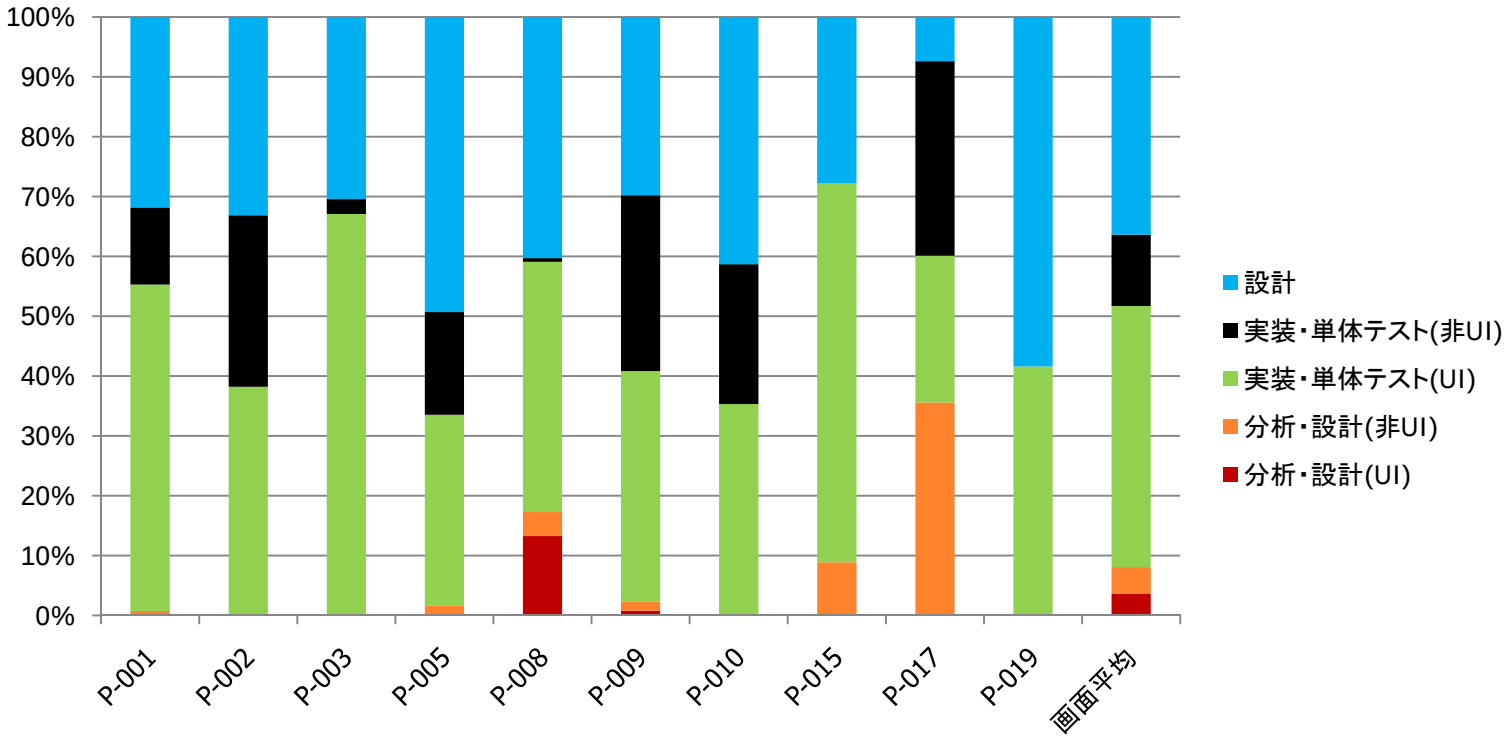
画面難易度と機能規模は提案段階9月1日時点の測定値

画面生産性（機能規模）



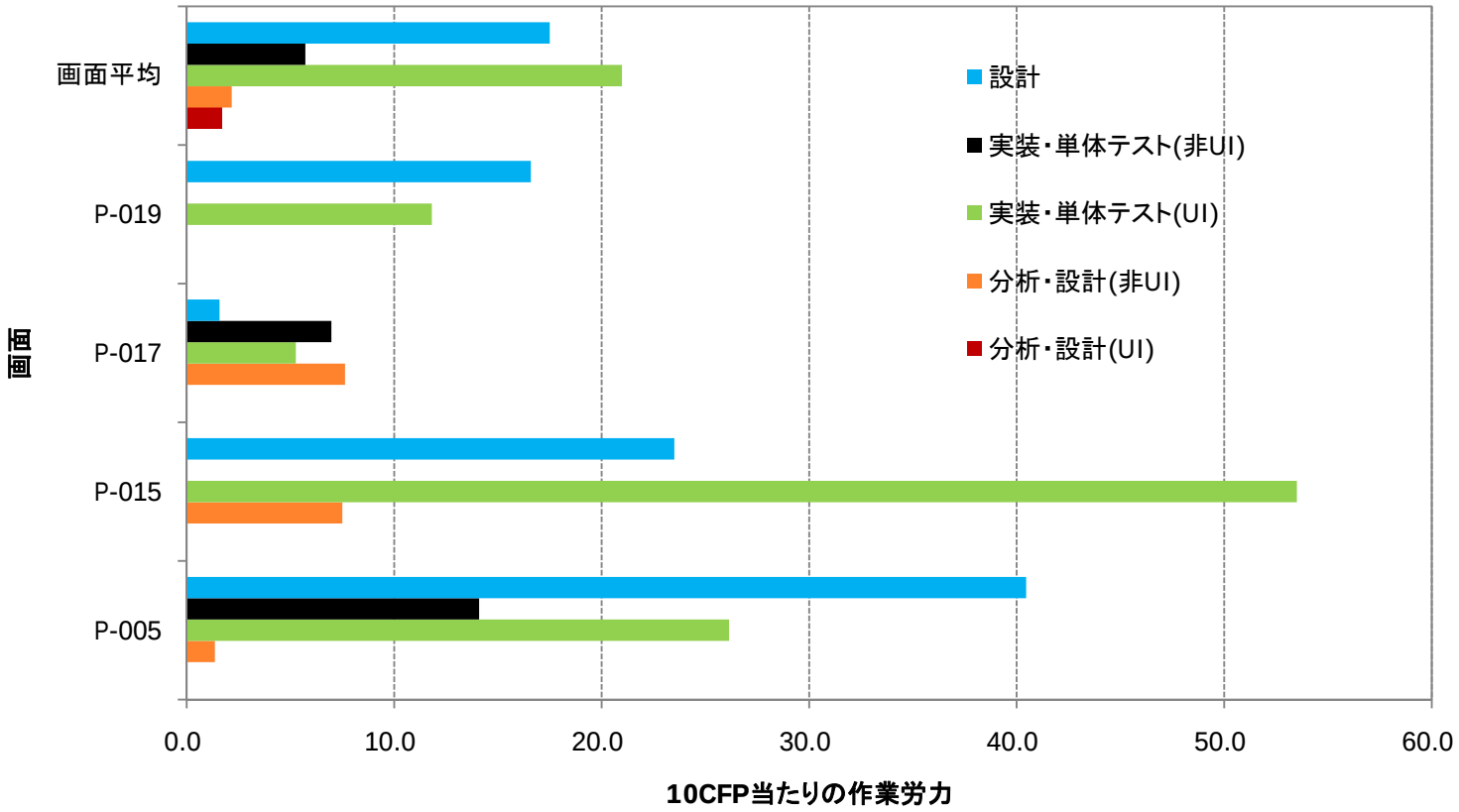
全画面の平均画面生産性は0.21CFP/人時間

画面毎の労力配分



全体的な傾向として、「設計」と「実装・単体テスト(UI)」の割合が70%以上、「実装・単体テスト(非UI)」が10-30%！

10CFP当たりの作業労力



プロジェクトメンバーのフィードバック

- 画面生産性が高い画面
  - マスター系で簡単な画面だったためではないか
- 画面生産性が低い画面
  - 見栄えがよくないので、再作業を行ったためではないか

生産性が高い画面の特徴

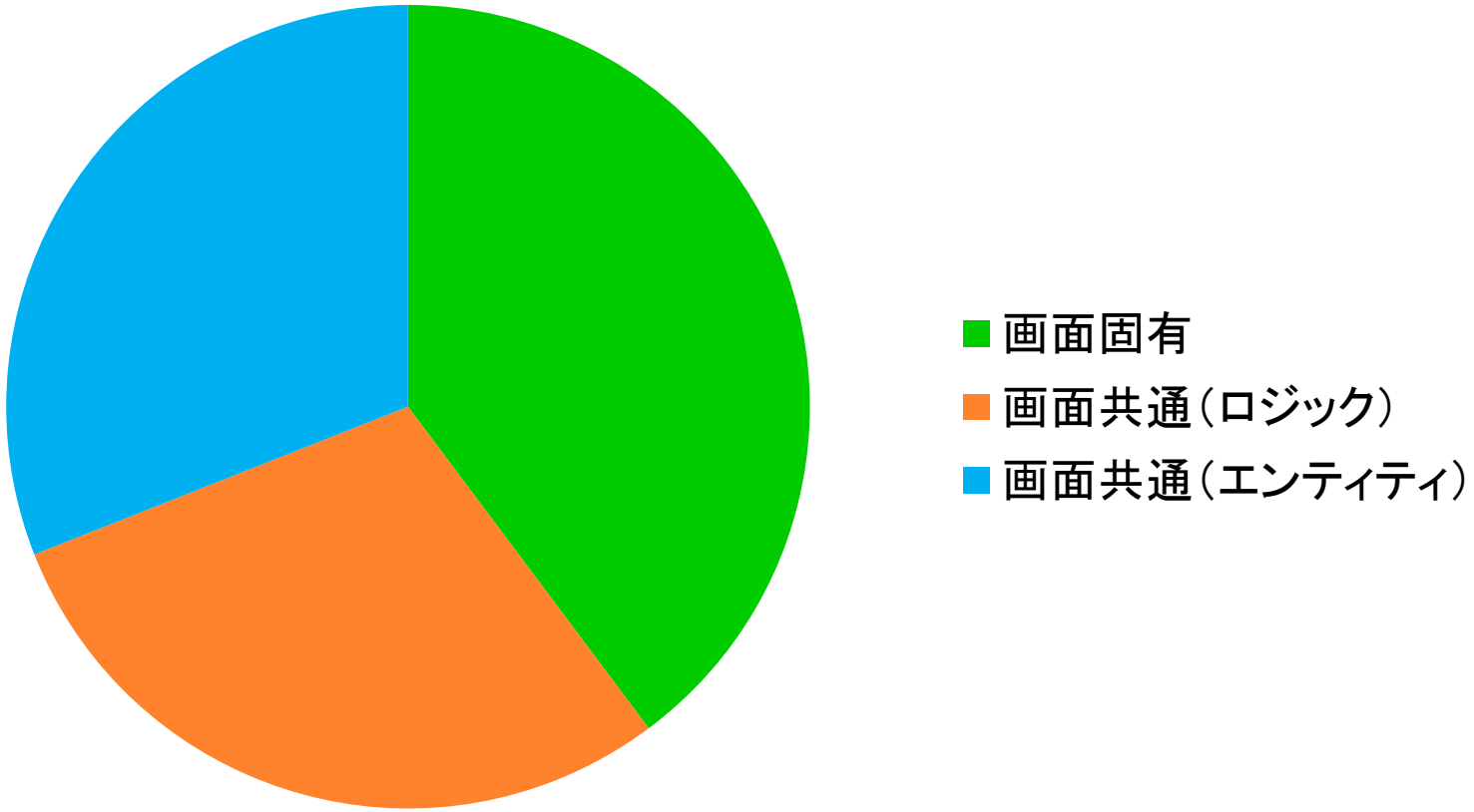
- 非UI部分(Java)の割合が極端に高い画面！
  - 「設計」と「実装・単体テスト（UI）」の合計が30%程度で、「詳細設計（非UI)」と「実装・単体テスト（非UI）」の合計が70%弱
- UI部分(JSP)の割合が極端に高い画面！
  - 「設計」と「実装・単体テスト（UI）」の合計がほぼ100%

画面生産性に影響している2つの要因

労力配分のパターン	再作業	画面生産性
UIと非UIの各々に、ある程度労力が分散している	あり	低
	なし	平均的
UIと非UIのどちらかに、労力が集中している	なし	高

JSPでUIを作っている場合に、「UIと非UIの各々にある程度労力が分散している」と生産性が低い可能性がある

非UI(Java)コードの内訳



非UIコードの内訳と労力配分から分かること

- 共通機能の括り出し
  - そこそこ共通機能が括り出されている
  - 画面毎の「実装・単体テスト(非UI)」の労力配分が全体的に小さいので、画面生産性のバラつきの主要因ではない
- 共通機能の内訳
  - 半分程度がエンティティ系のコードであり、データベース設計から機械的に決まる

開発生産性を高めるためには（画面）

- 労力配分
  - 「設計」と「実装・単体テスト(UI)」割合高
- 生産性の向上
  - 「実装・単体テスト(UI)」の効率化
    - UIと非UIにロジックが分散している場合にも効率的に開発できる言語や開発環境を採用すべき
  - 「設計」の効率化
    - ドキュメントの量を減らす？

開発生産性を高めるためには（共通機能）

- 労力配分
  - 「設計」と「実装・単体テスト(非UI)」割合高
- 生産性の向上
  - 「実装・単体テスト(非UI)」をさらに向上させるのは難しく、効果も限定的
  - 「設計」の効率化が望まれる
    - ドキュメントの量を減らす？

今回の分析結果のまとめ

- 画面生産性はバラつくのか？
  - 平均の1/2から2倍のバラつきがある
- 生産性が低い原因は？
  - 「UI」と「非UI」でロジックが分散している
  - 再作業
- 改善した方が良い点は
  - UIの開発効率を上げる工夫
  - 設計作業の効率を上げる工夫

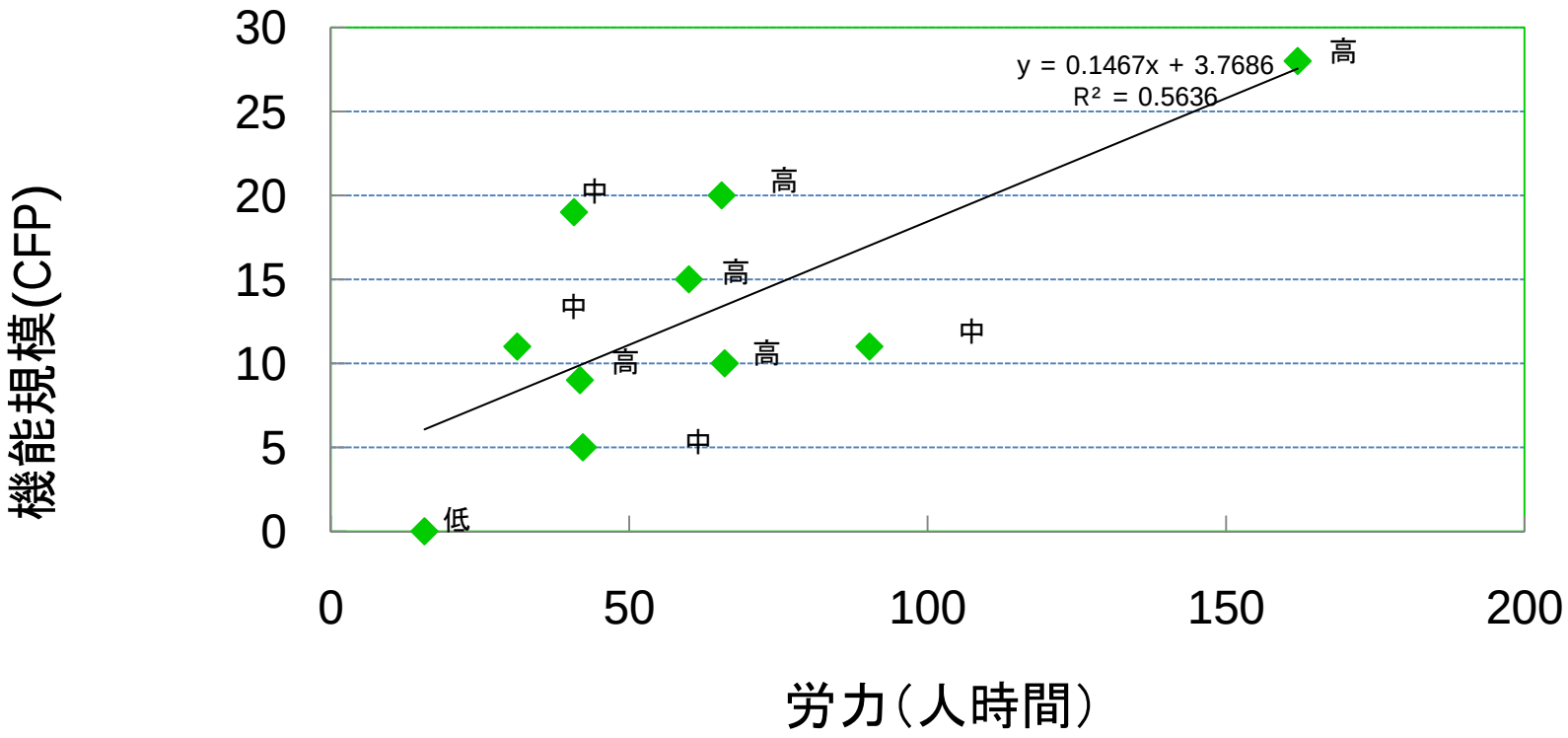
今後の課題

- 他のプロジェクトの測定結果との比較
  - 開発生産性のバラつき
  - 労力配分
  - コード行数の内訳
- 開発作業の効率化案を考える

参考文献

- 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
  - JFPUGのサイト（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
  - JFPUGのサイト（http://www.jfpug.gr.jp/cosmic/CFFP-index.html）から入手可能

機能規模と労力との相関



機能規模と労力は、それなりに相関がある
→機能規模の測定で労力を見積もれる可能性がある