Vol.2010-SE-170 No.2
2010/11/11

COSMIC 法で求めた 開発生産性バラつき要因の分析

藤井拓[†]　　木村めぐみ[†]

本論文では，機能規模 COSMIC 法に基づいてプロジェクトの開発生産性を測定し，生産性のバラつきの要因を分析するための方法を提案している．本手法では，生産性のバラつきのもたらす要因を分析するために，開発労力を開発部分と作業種別の組み合わせで分類し，記録する．さらに本手法を 6 プロジェクトの測定結果に適用し，生産性のバラつきに大きく影響した要因を求めた結果を報告する

Analysis on Variation of Project Productivity Using COSMIC Method

Taku Fujii[†]　and Megumi Kimura[†]

A technique for analyzing major factors to affect project productivity is proposed in this paper. Development effort is categorized and recorded with the combination of development parts and activity kinds to analyze the factors influenced project productivity. Application results of the technique to six projects are reported

1. はじめに

現在請負開発のビジネスでは，開発依頼者の方針によって開発プロセス，実装言語，アーキテクチャなどがプロジェクト毎に異なるということも珍しくない．さらに各企業が独自のビジネスニーズに基づいて開発するビジネス系のソフトウェアの開発ではオンラインとバッチなどの処理形式が異なるソフトウェアを 1 つのプロジェクト内で開発する場合も多い．そのため，開発生産性に影響を及ぼす潜在的な要因が多く，それらの要因のどれが大きく影響しているかを特定することが困難になっている．開発生産性に大きく影響する要因が特定できないと，開発能力をさらに高めるために改善すべき部分，プラクティスの有効性，改善の効果などが明確にならない．

また，ソフトウェア開発全体を考えると新たなソフトウェアを一から作る新規開発だけではなく，既存のソフトウェアを修正したり，拡張したりする改修の占める割合は大きい．そのため，新規開発だけではなく，改修の開発生産性に影響する要因を明らかにすることも求められている．

本研究では，ビジネス用ソフトウェアの開発プロジェクトの規模，労力などを測定し，そこから求まる開発生産性に影響する要因を特定する方法を提案する．本研究の方法は，規模の測定に機能規模測定手法 COSMIC 法を用いることと，プロジェクトや処理の種別等に基づいてプロジェクトやソフトウェアを分類すること，さらに開発作業の労力を作業部分と作業種別の組み合わせで分類して記録することに基づいている．本論文の 2 節では COSMIC 法と開発生産性との関係について説明する．さらに，3 節では開発労力の分類と開発生産性に影響する要因の分析方法について説明する．4 節では，本研究のアプローチを 6 つのビジネス用ソフトウェア開発プロジェクトに適用し，オンラインやバッチ処理のソフトウェアの開発生産性に大きな影響を及ぼした要因の特定を試みた結果を報告する．最後に 5 節ではまとめと今後の課題を述べる．

2. COSMIC法と開発生産性

多様なプロジェクトの生産性を測定するためには実装言語や実装構造に依存しない機能規模の測定方法が必要になる．本研究では，そのような機能規模を測定方法としてCOSMIC法1)，2)を用いている．COSMIC法は，図 1に示すようにシステムの境界をまたがったデータの移動の数と永続ストレージに読み書きするデータ移動の数により機能規模を求める方法である．

[†]株式会社オージス総研　技術部ソフトウェア工学センター
Software Engineering Center, IT R&D Dept., OGIS-RI Co., Ltd.

Vol.2010-SE-170 No.2
2010/11/11

COSMIC法で測定する 4 種類のデータ移動を表 1に示す．これらのデータ移動はデータグループと呼ばれる単位で測定される．データグループは正規化された論理データモデルのエンティティのサブセットに概ね対応する3)．

COSMIC法では 1 つのデータグループの 1 つのデータ移動を 1 CFP という機能規模の単位として測定する．また，1 つのエントリーをきっかけとする一連のデータ移動を機能プロセスと呼ぶ．さらにユーザの視点で捉えたソフトウェアの機能のまとまりを利用者機能要件 (FUR: Functional User Requirement) と呼ぶ．機能規模は最初に機能プロセス単位で求め，次いで各機能プロセスが属するFUR毎の機能規模を求める．さらに，ソフトウェアを構成する，すべてのFURの機能規模を合計したものがソフトウェアの機能規模になる．このように求めた機能規模を本論文では「正味機能規模」と呼ぶ．

COSMIC 法では，ソフトウェアが変更された際の変更の規模の測定方法も規定されている．それは，FUR 単位でなんらかの変更が加えられたデータ移動の総和を求めるというものである．変更が加えられたデータ移動としては以下の 3 種類がある．

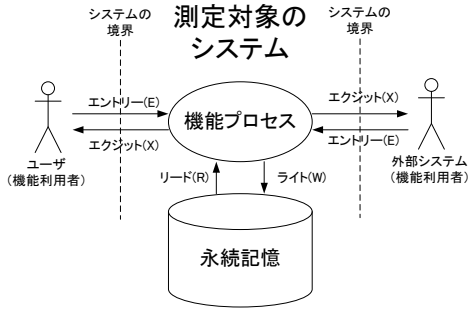

図 1　COSMIC 法の概念図

表 1　COSMIC 法で測定するデータ移動

データ移動の種類	説明
エントリー (E)	システムの境界からユーザインタフェース (UI) や通信を通じて入力されることに対応するデータ移動
エクジット (X)	UI や通信によりシステムの境界に出力されることに対応するデータ移動
リード(R)	永続ストレージから読みだされるデータ
ライト（W）	永続ストレージに対して書き込まれるデータ

- 追加されたデータ移動
- 変更されたデータ移動
- 削除されたデータ移動

ここで「変更されたデータ移動」とは，移動するデータグループの属性やデータ移動を生ずる演算が変更されたということを意味する．これらの 3 種類のデータ移動の総和として求めた機能規模を本論文では「変更機能規模」と呼ぶ．

世の中のソフトウェア開発プロジェクトは大別すると以下の 2 種類に分類できる．

- 新規開発プロジェクト：新たなソフトウェアを一から開発するプロジェクト
- 改修プロジェクト：過去に開発されたソフトウェアを修正/変更/拡張するプロジェクト

新規開発の場合には，「追加されたデータ移動」しか存在しえない．そのため，新規開発では「変更機能規模」と「正味機能規模」の値が等しくなる．それに対して，改修では「変更されたデータ移動」の数として求める「変更機能規模」と「ソフトウェア全体のデータ移動」として求める「正味機能規模」の値が異なることが多い．改修プロジェクトの実装作業は「変更されたデータ移動」に相当する部分に対して行われるため，そのようなプロジェクトの開発労力と相関するのは「変更機能規模」だと考えられる．

つまり,変更機能規模を用いることでこれらの 2 種類のプロジェクトで開発しなければならないソフトウェアの規模を定量化にすることができる．以降，本論文では特に断りがない場合には「変更機能規模」を「機能規模」と呼ぶことにする．

新規開発プロジェクトであれ，改修プロジェクトであれ，開発に要した労力と開発対象のソフトウェアの変更機能規模から以下のように開発生産性を求めることができる．

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

このような式で求めた開発生産性を本論文では「機能規模生産性」と呼ぶ．

前述したように，新規開発と改修の場合とでは変更機能規模の算出に用いるデータ移動の種類が異なる．規模測定に用いるデータ移動の種類が異なれば，機能規模生産性にも影響を及ぼす可能性がある．そのため，新規開発と改修の機能規模生産性を比較する際には算出に用いているデータ移動の種類の違いが影響しているという可能性も考慮した方がよいと考えられる．

3. プロジェクトと作業の分類

3.1 プロジェクトの分類と機能ユニット

2.で説明したように，世の中のビジネス用ソフトウェア開発プロジェクトは新規と改修に大別される．さらに，プロジェクトが開発するソフトウェアを処理形式で分類すると以下のような種類がある．

- オンライン
- バッチ
- サービス

これらの処理形式の違いにより，開発に用いられる実装言語やフレームワーク（ライブラリ）が異なることも多い．また外界との相互作用に関するデータ移動の量も異なると考えられる．そのため，これらの処理形式の違いにより機能規模生産性が異なる可能性がある．

実際の商業的な開発プロジェクトを考えた場合には，１つのプロジェクトの中でオンラインとバッチなどの異なる処理形式のソフトウェアを併せて開発することも多い．本研究では後述する「機能ユニット」という単位で測定を行うことで処理形式が異なるソフトウェア部分の機能規模生産性を分離して測定することを可能にしている．

機能ユニットは，プロジェクトで開発するソフトウェアをエンドユーザの視点で画面，バッチ処理，サービスなどの機能単位に分解したものである．また，機能ユニットは COSMIC 法の FUR に対応する．そのため，機能ユニット単位で機能規模を求めることができる．さらに機能ユニット毎の開発労力を記録すれば，機能ユニット単位の機能規模生産性を求めることができる．求まった機能ユニットの機能規模生産性を「オンライン」と「バッチ」などの処理種別毎に集約すれば処理種別毎の機能規模生産性を求めることができる．

また，開発プロジェクトによっては開発作業の一部を複数の協力会社に委託する場合もある．近年，委託先を国内に限らず海外に求めることも増えている．そのような場合も各協力会社に委託する範囲は対応する機能ユニット単位で分割できることが多い．そのため，協力会社の分担範囲の機能ユニットについて機能規模と開発労力を測定すれば協力会社毎の機能規模生産性を求めることができる．プロジェクトによっては，協力会社に特定の作業を委託する場合もある．例えば，プロジェクト管理，要求の定義，テスト（システムテストや結合テスト）を自社員で行い，詳細設計や実装・単体テストを協力会社に作業委託するような場合である．このような場合にも，協力会社に機能ユニットと作業種別の両方の組み合わせで労力を記録することで協力会社毎の委託作業範囲における単位機能規模あたり開発作業種別毎の作業時間を求めて，機能規模生産性への影響を把握することができる．

機能ユニット単位の機能規模生産性を調べることで，生産性が異常に高かったり，低かったりするような特異なケースを見つけることができる．そのような特異なケースは，特定の状況で機能規模と労力の相関が全体的な傾向から著しくずれた結果生じるものである．そのずれを生んだ要因を明らかにすることで機能規模の測定値による見積もりの精度を向上させることに役立つことが期待できる．

3.2 開発生産性のバラつきを生む要因と労力

先の節で述べた「新規」と「改修」の開発種別では機能規模の測定方法が異なったり，「オンライン」と「バッチ」などの処理種別の違いで求められる実装技術やソフトウェアアーキテクチャが異なる．それらは，開発生産性に大きな影響を及ぼす可能性がある．そのため，本研究では開発生産性のデータは基本的に開発種別と処理種別で層分けし，そのグループ内で機能規模生産性のバラつきを究明していくという方針を採用することにした．

さらに，開発種別と処理種別で層分けをした各層の開発生産性に大きな影響を与える要因として経験的に以下のようなものを想定した．

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

筆者らは，これらの要因の中で A)，B)，C)，E)，F)，G)については表 2 に示すような方法で定量化することができると考えている．そのため，要因の影響を調べるためには機能規模や労力以外にこの表の「元データ」の列にある測定値を可能な限り測定することが望ましいといえる．

本研究ではこれらの要因の中で A)と B)の影響を分離して測定するために開発作業を作業部分と作業種別に分類して労力を記録することにした．

労力の記録で用いる作業部分は以下の 2 つのレベルに分類している．

- 大分類：機能ユニット，画面/バッチ共通，プロジェクト共通
- 中分類：UI，非 UI，どちらでもない

表 2 開発生産性に影響する要因とそれらを定量化する方法

開発生産性に影響する要因	定量化の方法	元データ
共通機能の括りだしの度合い	アプリ固有とアプリ共通部分の開発労力やコード行数の割合	コード行数, 労力データ
UI の難易度	UI の部分とそれ以外の機能部分の開発労力の量	労力データ
開発プロセスの重さ（ドキュメント量）	各作業で作成する成果物の数, または設計作業などの労力の量	成果物数, 労力データ
指摘事項/欠陥の検出密度	コードレビュー, 結合テストやシステムテストで検出した単位コードあるいは単位機能規模あたりの指摘事項や欠陥の数	問題点管理データ, コード行数, 機能規模
開発途上での開発依頼者からのフィードバックの量	仕様変更や仕様追加の数, 変更規模	問題点管理データ, 機能規模
単体テストが自動化されている割合	単体テスト自動実行時に測定されたテストカバレッジ	テストカバレッジ

ここで「画面/バッチ共通」とはアプリケーションの共通機能部分を表し,「プロジェクト共通」とはプロジェクト管理など機能と直接結びつかない作業を表す. また「UI」は UI 機能の開発作業,「非 UI」は UI 以外の機能の開発作業,「どちらでもない」は UI 機能とも UI 以外の機能とも分類しにくい開発作業を表す.

労力の記録で用いる作業種別として，要求，分析/設計，実装・単体テスト，結合テスト，システムテスト，プロジェクト管理など一般的なソフトウェア開発の作業種別を用いている.

表 3 は，1 つのプロジェクトにおいて設定した作業部分（大分類）や作業部分（中分類）と作業種別の組み合わせの例を示したものである．この表の各マスに示された作業種別は，労力の記録を簡素化するために「作業部分（中分類)」と「一般的な開発作業種別名」を連結したものになっている。

3.3 開発生産性のバラつきの分析方法

現在のところ，開発生産性のバラつきを分析するために以下のようなデータを主として用いている.

表 3 作業労力の分類

		中分類		
		UI	非 UI	どちらでもない
大分類	画面固有	● 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)	● 設計
	画面/バッチ共通		● 詳細設計(非 UI) ● 実装・単体テスト(非 UI)	● 設計
	プロジェクト共通			● 設計 ● プロジェクト管理

- 機能規模生産性
- 10CFP 当たりのアプリ固有/アプリ共通/プロジェクト共通の作業労力
- 10CFP 当たりのアプリ固有の作業種別毎の作業労力
- 指摘（欠陥）密度

これらのデータを用いた分析のステップは, 以下のとおりである.

A) 機能規模生産性によりプロジェクトの平均的な生産性から外れているプロジェクトを特定する
B) 10CFP あたりのアプリ固有/アプリ共通/プロジェクト共通の作業時間より, 生産性を低下/向上させている作業を大きなレベルで特定する
C) 10CFP あたりのアプリ固有の作業種別毎の作業時間で UI の難易度や特定の開発作業の作業時間が生産性に影響しているかを確認する
D) 指摘（欠陥）密度と対比することで C)で特定した開発作業の作業時間が指摘（欠陥）密度の多さに起因しているかを確認する

これらの分析ステップで，データを処理分類毎や協力会社毎に集約することで処理分類や協力会社毎を分離して分析を行うことができる.

なお,「単体テストの自動化の割合」と「開発途上での開発依頼者からのフィードバックの量」については，これらが当てはまるプロジェクトが現時点では少数に留まっている．そのため，これらのデータを分析段階でまだ活用できていない.

表 4　測定/分析したプロジェクトのプロフィール

プロジェクトID	開発種別	処理種別	開発ライフサイクル	協力会社	プロセスの重さ	単体テストの自動化
A1	新規	オンライン	進化型	国内	通常	部分的に実装
B	新規	オンライン+バッチ	インクリメンタル	国内	軽量	フルに実装
C	新規	オンライン	ウォーターフォール	国内	通常	未実装
A2	改修	オンライン	進化型	国内	軽量	部分的に実装
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	通常	未実装
E	新規	バッチ	ウォーターフォール	なし	通常	未実装

4.　プロジェクトの開発生産性の分析例

前節までで説明したCOSMIC法と作業種別の分類を適用して，これまで 6 つのプロジェクトの測定を行った．測定対象の 6 つのプロジェクトのプロフィールを表 4に示す．

表 5　開発期間と機能規模の測定結果

プロジェクトID	開発期間（ケ月）	正味機能規模(CFP)	変更機能規模(CFP)
A1	6	111	111
B	4	330	330
C	5	42	42
A2	6	274	274
D	6.4	384	384
E	1.2	18	18

これらのプロジェクトのうちでA1 と A2 は同じシステムを対象としたものである. A1 は，新規開発でシステムの評価版リリースを開発するプロジェクトである．それに対して，A2 は A1 のリリースに基づく開発依頼者からフィードバックに基づいて機能拡張（改修）を行ったプロジェクトである．A2 では設計作業を大幅に削減して開発プロセスも軽量化している．B プロジェクトでは 1 ケ月毎にその時点までに完成しているソフトウェアを顧客に対して示し，顧客からの仕様の追加や変更などのフィードバックを受けて開発を行った．B でも開発期間の短さと仕様の追加や変更に対応するため設計作業を削減している．B 以外のオンライン系はシンクライアントであるのに対して B の UI はリッチクライアントであった．さらに D ではオンライン機能の一部の実装・単体テスト作業を実験的にオフショアに委託している．D 以外のバッチ処理は Java で実装されているのに対して，D のバッチ処理は C 言語で実装されている．

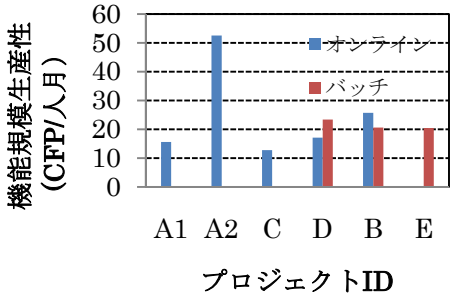

図 2　機能規模生産性
（オンライ/バッチ別）

測定した 6 つのプロジェクトの開発期間，正味及び変更機能規模，労力を表 5に示す．また，これらのプロジェクトのオンライン/バッチ別の機能規模生産性を図 2に示す．

機能規模生産性のデータより，「バッチ処理」と比べて「オンライン処理」の機能規模生産性が大きくバラついていることが分かる．さらに，オンラインの中では A1 と C の生産性が低く， A2 の生産性が高いことが分かる．

次に, 10CFP当たりのアプリ固有/アプリ共通/プロジェクト共通の作業時間を求めたものが図 3である．このグラフでは，プロジェクト共通からプロジェクト管理に関する作業時間を分けて示している．

図 3より，A1 はBやDと比較してプロジェクト共通と画面共通の作業時間が大きいこと，Cでもプロジェクト管理や画面共通の作業時間が大きいことが分かる．

「オンライン」処理

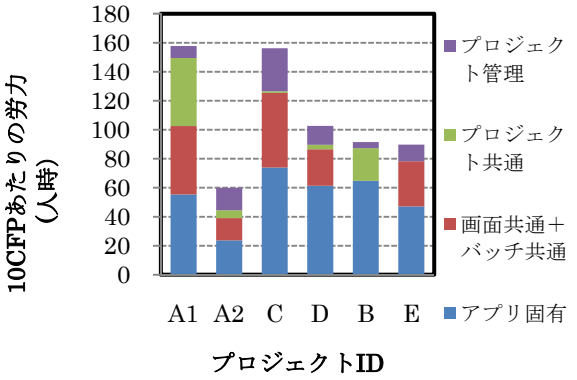

図 3　10CFP あたりの作業労力
（アプリ固有/共通，プロジェクト共通別）

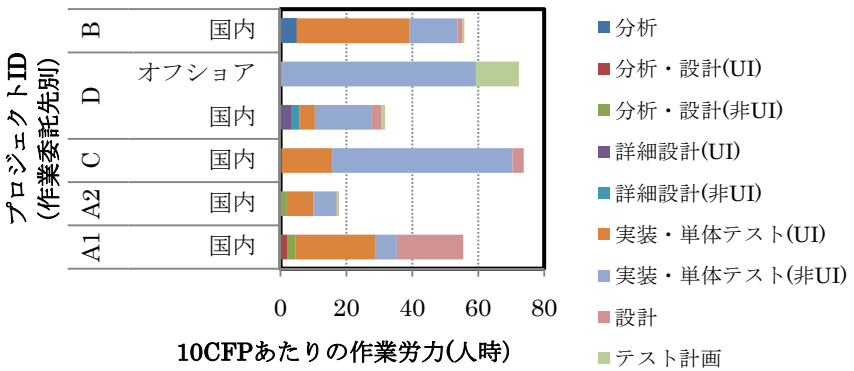

図 4　10CFP あたりの作業労力（アプリ固有）

についてアプリ固有の 10CFPあたりの作業時間をより詳細な作業種別で分解したものを図 4に示す．Dのオフショア委託分については「実装・単体テスト」の作業をUI，非UIに区別して記録できなかった．そのため，Dのオフショア委託分の「実装・単体テスト」作業は便宜的にすべて「実装・単体テスト（非UI)」として分類している．

図 4から，A1 とBは実装・単体テスト(UI)の作業労力を大きいのに対して，CとDのオフショア委託分は実装・単体テスト（非UI）の労力が大きいことが分かる．また，A2 はA1 を除く他のプロジェクトと比べて実装・単体テスト（非UI）の労力が小さいこ

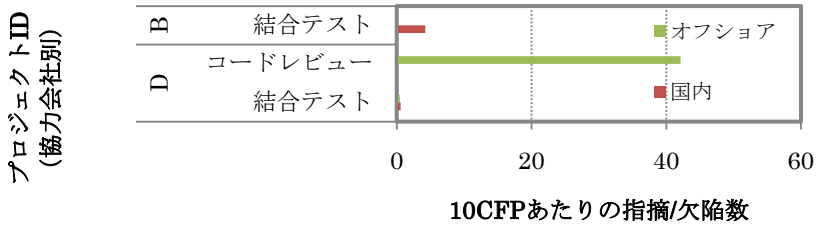

図 5　10CFP あたりの指摘/欠陥数

表 6　プロジェクトの測定データから分かったこと

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

とが分かる．

B及びDの「オンライン処理」について 10CFP当たりの指摘/欠陥数を示したものが図 5である． この図の指摘数はプログラムコードレビュー段階の指摘数であり，欠陥数は結合テストで検出された欠陥数である．

図 5からDのオフショア委託分についてコードレビューの指摘数が非常に多いことが分かる．その一方で，Dの国内協力会社とオフショア委託分の両方について結合テストで検出された欠陥密度はほぼ同じレベルであることが分かる．また，Bプロジェクトの欠陥数がDよりも多いことが分かる．なお，国内協力会社への委託分については細かいレベルまでのコーディング規約を開発者が十分に把握しているなどの理由で，A1，A2，B，C，Dではコードレビューは行っていない．

これらの測定結果から分かることをまとめたものが表 6である．この表で，機能規模生産性が低いものは労力の列において「大」となっている作業の効率が低かったり，指摘密度が高いことが影響していると考えられる．その逆に，機能規模生産性が高いものは労力の列において「小」となっている作業の効率が高かったことが影響していると考えられる．

この測定結果をプロジェクトメンバーに示したところ，生産性が低下した要因として表 7に示すようなものが挙がった．また，A2 の生産性が高かった要因としては次の 2 点が挙げられた．

表 7　生産性低下の要因

プロジェクト ID	要因として挙がった点
A1	● 設計作業に不慣れなメンバーで作業効率が悪かった ● 特殊な UI 仕様の実現で試行錯誤(再作業)があった
C	● DI(Dependency Injection)コンテナを使った開発が初めてであった
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

- 設計作業を大幅に削減した
- 特殊な UI 仕様の実現方法に試行錯誤がなくなった

今回の測定でプロジェクトの生産性の低下としてはっきりと現れなかったが，B プロジェクトは以下の点で他のプロジェクトよりも厳しい条件の下で開発したと考えられる．

- 顧客からの仕様の追加や変更を積極的に受け入れた
- 機能規模で同等の D プロジェクトと比べて相対的に短期に開発を行った
- リッチクライアントで UI の難易度が高かった

このような厳しい条件で機能規模生産性が他のプロジェクトと比べて見劣りしなかったのはプロセスの軽量化やテストの自動化などの効果によるものである可能性がある．

また，A2 の高い生産性にもプロセスの軽量化の効果が表れている可能性がある．それらの効果を影響の有無をはっきりさせるためには，改修プロジェクトでの機能規模生産性の値が新規開発での値と比較できるかどうかという点を今後検証していく必要がある．

今回測定した範囲のバッチ処理においてプロジェクトの生産性のバラつきはオンラインに比べて相対的に小さかった．また，結合テスト段階で検出された欠陥密度も低かった．それらの点は，以下の 2 点に起因していると考えている．

- バッチ処理の実装に使った言語やフレームワークへの習熟度が高かった
- バッチ処理はオンライン処理に比べてテストケース数が相対的に少なく，テストの網羅性を上げやすい

今回の測定/分析を通じて単体テストなどの「テスト段階」の定義がプロジェクト毎に違うことが分かった．今回測定した範囲では B を除くプロジェクトで「単体テスト」に画面やバッチ処理単位のテストが含まれていた．そのため，結合テスト以降で検出される欠陥密度が図 5 に示されるように低くなっている．そのような「テスト段階」の定義の影響も若干あると思うが測定した全プロジェクトにおいて結合テスト以降の作業労力の割合が全労力の 20%以下に留まっていた．これらのデータは，世の中で一般的に言われているような結合テスト以降の作業労力が全労力の 40%程度という値と大きく異なる．この結合テスト以降の労力が少ないのは「単体テスト」段階で欠陥密度を低減できているためだと考えられる．

5. まとめ

本論文では機能規模 COSMIC 法に基づいてプロジェクトの機能規模の測定結果，及び開発部分と作業種別の組み合わせによる労力の測定結果より，機能規模生産性を求め，生産性のバラつきの要因を分析するための方法を提案した．さらに，本手法を 6 プロジェクトの測定結果に適用した結果，プロジェクトの生産性のバラつきに大きく影響する要因の特定に有効であることが示された．

今後，改修プロジェクトについてさらに生産性のデータを蓄積し，変更機能規模の測定値と実績労力の関係を明らかにしていく予定である．また，本手法で特定された生産性に影響する要因と生産性との間の因果関係を定量的に示すモデルを構築することを試みる予定である．

謝辞
本論文で紹介したプロジェクトの測定結果を得るのに協力して頂いた（株）オージス総研の技術部ソフトウェア工学センターのメンバーとプロジェクトメンバーにこの場を借りて感謝の意を表します．

参考文献

1) COSMIC法ver3.0 測定マニュアル, http://www.jfpug.gr.jp/cosmic/CFFP-index.html
2) Abran A., Desharnais J.M., Maya M., St-Pierre D.,Bourque P., Design of a functional size measurement for Real-Time Software, UQAM, Research report No 13-23,1998.

3) 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,
http://www.jfpug.gr.jp/cosmic/CFFP-index.html