ソフトウェア開発プロセスの並列作業に基づくプロセスの複雑さの提案

尾花 将輝　花川 典子　飯田 元

ソフトウェア開発プロジェクトでは想定していた開発計画とプロジェクト終了後の開発実績には大きな隔たりがある．例えば，顧客の要望で機能要件が変更され，それに伴い変更作業のプロセスが当初の計画に追加される等である．予期せず追加されたプロセス (フラグメントプロセス) は開発プロセス全体を複雑化する．複雑になったプロセスはプロダクト品質に影響すると予想されるが，従来の研究ではプロセスの複雑さとプロダクト品質の関係が明確に示されていない．そこで，本稿ではソフトウェア開発プロジェクトの並行作業に着目したプロセスの複雑さを提案し，プロダクト品質との関係を明確にする．プロセスの複雑さはフラグメントプロセスによって開発プロセスがどのように複雑化するかを定量的数値で示す．本提案を産業界における 8 つのプロジェクトを用いてプロセスの複雑さとプロダクトの品質の関係を調査した．結果，PC 値と致命的障害には相関係数 0.786 を確認した．更に，プロジェクトのプロセスの複雑さを用いてリリース後のプロダクト品質を予測した結果，実測値に近い品質を予測することができた．

In software development projects, large gaps between planned development process and actual development exist. For example, the process of the requirement repeated definition by the demand of a sudden customer in the design process. We call the added process a fragment process. Although complicated development processes may influence product quality, conventional researches did not clarify relationships between process quality and product quality. Therefore, we propose a metric of process complexity focusing on concurrent processes in order to clarify relationships between process quality and product quality. Process complexity is a value that can quantitatively measure modifications of an original development process. In 8 industrial projects, we investigated relationships between values of the process complexities and product qualities. As a result, correlation coefficient between process complexities and important failure is 0.786. In addition, we show a case study of process complexity in order to predict post-release product quality.

1　はじめに

ソフトウェア開発プロジェクトにおいて，想定していた開発計画と実際の開発実績には大きな隔たりがある [1]．例えば，顧客の急な要望による計画にない作業の追加や，インクリメンタル開発プロセスで計画されたサブシステムの機能が全く異なる機能へと変更される等が挙げられる．このような開発計画に新たな作業が追加される事は開発現場では問題視されている．急な追加作業が発生することで作業が円滑に進まなくなり，結果としてプロダクト品質に影響する場合もある．しかし，急な顧客要望による追加作業を割愛すると顧客にとって有益なシステムにならない．追加作業や計画変更を少なくするため，要件定義を確実にする研究も行われているが [2] [3] [4]，顧客の満足度を向上させるために開発現場では細かな計画変更や追加作業が頻繁に発生する．追加作業や計画変更は避けられない事象だとしても，無節操に変更するのではなく，最終プロダクトに与える影響を考慮して慎重に計画を変更して新しい開発プロセスを構築する必要がある．

そこで，本研究では追加作業やプロジェクト途中での計画の変更等がどのようにプロセスに影響を与え

Process Complexity Based on Concurrent Tasks for Software Development.
Masaki Obana, 奈良先端科学技術大学院大学, Nara Institute of Science and Technology.
Noriko Hanakawa, 阪南大学, Hannan University.
Hajimu Iida, 奈良先端科学技術大学院大学, Nara Institute of Science and Technology.
コンピュータソフトウェア, Vol.29, No.4 (2012), pp.278–292.
[研究論文] 2011 年 12 月 28 日受付.

るかを示す新たな尺度，プロセスの複雑さ (Process Complexity) を提案する．プロセスの複雑さは同時実行プロセス数，各プロセスに関わる開発者数，及びプロセスの実行期間の値の 3 つのパラメータから求める．プロセスの複雑さは計画されたプロセスと予定外の追加プロセスの間に発生する相互作用が開発プロセスに影響を与えるというコンセプトを基に提案する．相互作用とは設計書変更等によって発生するプロセス間の調整等のコミュニケーションを意味する．相互作用がプロセスを複雑にすると仮定し，開発プロセスの複雑さを定量的に計測する尺度を本稿では提案する．また，プロセスの複雑さの 3 つのパラメータは，産業界のプロジェクトで一般的に利用されている工程管理表[5] から抽出する．構成管理された工程管理表からプロジェクトの進捗に伴い追加された作業や変更された作業を追跡し，プロセスの複雑さの変化を逐次計測する．

2 章ではプロセスメトリクスとプロジェクトリスク管理の関連研究を示し，3 章ではプロセスの複雑さを提案する．4 章では 8 つの実プロジェクトへの適用事例を示し，5 章でプロセスの複雑さの利用例について考察し，6 章でまとめと今後の課題を述べる．

2 関連研究

ソフトウェア開発プロセスのメトリクスを計測する研究は多く実施されている．CMM はハンフリーが示すプロセス成熟度モデルであり，組織におけるプロセスの成熟に関する 5 レベルを提案している[6]．レベルを決定する際に様々なメトリクス (総合試験欠陥率，試験密度，レビュー密度等) 値が収集される．坂本らは企業内でウォータフォール開発のプロセス改善を実施したが，その際にプロセス改善度合いを計測するメトリクスを用いた[7]．これらのメトリクス値の計測目的はプロセスの成熟度を計測しプロセスを組織的に改善することである．レビュー実施回数や指摘事項数がメトリクスとして取り上げられている．また，条件分岐や制御構造等のプロセスコントロールフローのわかりやすさからプロセスの複雑さを提案している[8]．プロセスの複雑さを計測するという目的は同じであるが，我々の手法では条件分岐等がない工程管理表から作業期間，並列実行プロセス数，特に他の並列実行プロセスとの間の関連を考慮してプロセスの複雑さを計測する点が異なる．

プロセスモデリングについての研究も多く提案されている．Cugola らは容易にタスクを追加することができるプロセスモデル言語を提案している[9]．Fuggetta らはソフトウェア開発環境における問題点や様々な開発プロセスにおける問題を解決するためのツールを提案している[10]．さらに，Garcia らはプロセスモデルの保守性や変更容易性などを，GQM ベースのメトリクスで評価している[11]．これらにおいて，修正作業の追加を変更容易性として着目している点は我々の提案するフラグメントプロセスと似ているが，これらのプロセスモデリングはプロジェクトをシミュレートすることでプロセスの評価や改善をすることが目的である．我々の研究は，1 つのプロジェクト内でプロセスの複雑さの変化を計測し，プロセスとプロダクト品質との関係を明確にすることを目的としており，これらのモデリング言語を用いたプロセス評価や改善のための研究とは目的が異なる．

一方，ベストプラクティスに基づく，複数のプロセスを統合して成長した実践的なラショナル統一プロセスが提案されている[12]．反復型開発，ユースケース駆動，リスク駆動等の考え方を取り入れ，アジャイルや.NET 等の最新技術テーマにも対応した実践的な統合プロセスである．管理や規模の問題もあるが，体系的なプロセスの実践には有益である[13]．1 つのプロジェクトに複数のプロセスが混在するという考え方は我々のプロセスの複雑さの考えに似ている．しかし，プロセス体系としてラショナル統一プロセスと，プロジェクト実行中に細かなプロセスが発生するフラグメントプロセスは根本的に異なる発想である．さらに本提案ではプロセスとプロダクト品質の関係まで言及する点が異なる．

プロセスとプロダクト品質の関係に言及した研究としては，青木らがアジャイル開発の実証データにてプロセスとプロダクトの関係を導いた例があり[14]，レビュー回数やテストケース数等のプロセスメトリクス値とシステムテスト工程のフォールト数の関係を重回帰式にて求めている．本研究はプロセスとプロダクト

の関係を明確にするという点で目的を同じくするが，青木らがプロジェクトが終了した時点で確定するメトリクス値を用いるのに対し，我々は日々変化する工程管理表を用いてプロセスの複雑さを計測するため，プロジェクト前半や中盤でプロセスの複雑さからリリース後のプロダクト品質を予測できる．

3 プロセスの複雑さの提案

プロセスの複雑さはプロセスがプロダクトにどのように影響を与えるかを明確にすることで，将来的にはプロセスから最終的なプロダクト品質を予測することを目的としたメトリクスである．本章では，プロセスの複雑さはプロジェクト当初に計画されたプロセスに対して予期せず追加されたプロセス (以下フラグメントプロセス) を抽出することで計測される．本章ではまず，フラグメントプロセスについて述べ，プロセスの複雑さについて説明する．

3.1 フラグメントプロセス

ソフトウェア開発において，開発全体を 1 つのプロセスとして計画して実行することが多い．しかし，実際のプロジェクトにおいて計画工程で予定した開発プロセスがそのままプロジェクト終了まで継続することは稀である．例えばウォータフォール開発プロセスで開発計画を作成したとしても，顧客の要望によるプロトタイプの作成や，仕様変更，またテスト工程中のプログラム変更等により追加作業が随時発生し，プロジェクト全体としての開発プロセスは複雑化する．また，インクリメンタル開発プロセスのように複数リリースがある場合，前バージョンの障害発生による改修プロセスと予定されていた新機能開発プロセスを並行実行する場合がある．本稿ではプロジェクト実行途中で発生した計画外のプロセスをフラグメントプロセスと呼ぶ．フラグメントとは「断片」という意味があり，小さなプロセスが突発的に発生し，断片的に追加され並行実行される様からフラグメントプロセスと命名した．

過去の事例の一部を用いたフラグメントプロセスの概念を図 1 に示す．計画フェーズでは分析，設計，実装，テストの単純なプロセスであった．多くの場合，

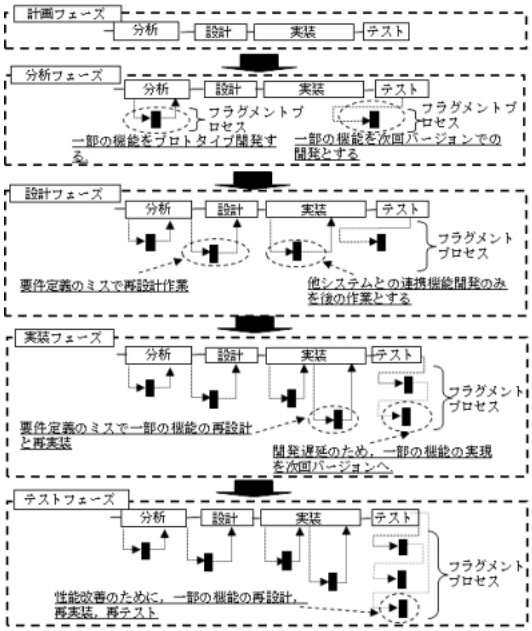

図 1 フラグメントプロセスの概念図

計画時にはシステムに関する情報が少ない (分析されていない) ため，相対的に大まかなプロセス (マクロプロセス) [15] で計画することが多い．しかし，プロジェクトが進行すると情報量が多くなり，プロセスも複雑に変化する．図 1 の場合，分析時に，当初予定していなかったプロトタイプ開発が顧客の要望により追加された．また，分析結果から期間とコスト制約により全ての機能の実装が難しいことが分かり，一部の機能を次バージョンへ移行した．さらに，設計フェーズでは要件定義ミスの発見による要件定義のやり直しプロセスの追加や，他システムの連携設計では他システム開発進捗の遅れにより連携部分の実装を後にする，といったように計画が変更された．実装フェーズでは要件定義ミスや設計ミスの発見によるやり直しプロセスが発生し，テストフェーズでは性能テスト結果に基づき設計のやり直しプロセスが発生した例が示されている．このように顧客の要望変更，他システム関連などの外部要因，テスト結果により新たに発生したプロセスがフラグメントプロセスである．

最後に本稿で述べるプロセスの定義について述べる．本稿では計画されたプロセス，フラグメントプロ

セスの 2 つが存在する．計画されたプロセスには計画段階に存在した多数のプロセスを 1 つのプロセスとして考える．これによってフラグメントプロセスによる開発プロセスへの影響の大きさを明確化する．一方でフラグメントプロセスには簡単な仕様書の修正作業から，1 つの機能に対する設計・実装・テストなど一連の作業を含むものなど様々な粒度が存在するが，ここでは計画されたプロセスに無い全ての一連の作業をフラグメントプロセスとする．また本稿で扱う「プロセス」は SLCP(Software Life Cycle Process) の定義に準じている．つまり，アクティビティやタスクと言った意味を持つ事も可能であるが，本稿ではプロセスの複雑さの計測時にプロセス単位での計測を行うために一連の作業群をプロセスと呼ぶ．

3.2 プロセスの複雑さの概念と特徴

まず，本稿で提案するプロセスの複雑さの概念を簡単に示す．本稿におけるプロセスの複雑さとはフラグメントプロセスの追加に起因する開発プロセスへの影響を定量化したものである．

システム開発における細かなプロセスの多くは互いに依存関係を持つ．例えば，データベース構築，アプリケーション開発，インフラ構築等のプロセスが存在した場合，これらは一見，単独で実施できる独立したプロセスであるが，実際には複数プロセスに影響を受けるプロダクトに関する打ち合わせや，結合テスト実施方法の相談等の調整作業が発生する．ここでは，このようにプロセス間で発生する打合せや相談，設計書変更作業等，複数プロセス間の影響を調整する活動全般をコミュニケーションと呼ぶ．

開発計画の作成時には，個々のプロセスに要する労力だけではなくコミュニケーションの負荷を考慮する必要がある．さらに注意すべきは，計画当初のスケジュールは予め計画にあるプロセス間のコミュニケーションを考慮した上で作成されるが，後からプロセスを追加，変更する際には，その結果必要となるコミュニケーションの負荷が見過ごされがちなことである．

本稿で提案するプロセスの複雑さとは，プロジェクト当初に計画されたプロセス対して追加されたプロセス，すなわちフラグメントプロセスが計画に与える影響に着目し，それを定量化するものである．図 2 の例を用いて説明する．まず，プロジェクト当初に計画されたすべてのプロセスを 1 つの「計画されたプロセス」として扱う．図 2 では要件定義プロセス，設計プロセス，実装プロセス，テストプロセスの黒矢印を四角で囲っているのがこれに相当する．次に,「計画されたプロセス」にフラグメントプロセス 1，2(図中黒色で表記) を追加することを考える．このとき，プロジェクト全体では，追加されたフラグメントプロセスの工数のみならず，フラグメントプロセス間，および,「計画されたプロセス」とフラグメントプロセスの間で発生するコミュニケーション分の工数が余計に必要となる (図中，模式的に斜線で表記)．このように，後から追加されるプロセスが現在実行中もしくはすでに計画されたプロセスに与える影響を定式化し，計測可能としたものが本稿におけるプロセスの複雑さである．

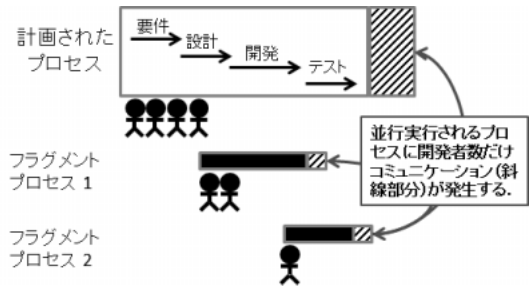

図 2 プロセスの複雑さの概念図

次に，上記のようなプロセスの複雑さの概念を具体的に構成するための方針を述べる．提案するプロセス複雑さの概念は，その適用可能範囲を可能な限り広めるために，産業界で標準的に用いられている工程管理表を対象に容易に抽出可能な基本要素を基に求められる．すなわち，従来の定量的プロセス管理手法等 [16] [17] [18] の多くの提案が前提としている CMM レベル 4 の成熟度を達成していない組織でも本概念を活用することができることを目指した．具体的には，プロセスの複雑さに影響する基本要素として「プロセス実行期間」,「開発者数」,「同時実行プロセス数」の 3 要素に焦点を絞った.「プロセス実行期間」はプロセスの規模を意味し，プロセスの規模はコミュニケーション

量に影響する.「開発者数」はコミュニケーションが成立するルート数に影響し，人数が多いと会話数が多くなると考える.「同時実行プロセス数」は同時実行するプロセス間のコミュニケーションルート数に影響し，同時実行プロセスが多いと同時に配慮すべき事柄が増えることを示す．これら 3 要素に基づいて個々のプロセスの持つ複雑さを定め，それらの総和をプロセスの複雑さとした．より正確な定義は 3.4 節に示す．

最後に，本稿で提案するプロセスの複雑さの概念の汎用性について述べる．我々の基本目的はプロセス品質とプロダクト品質との関係を明確にすることである．そのために，まず，プロセス品質を示す指標の 1 つとしてプロセスの複雑さに着目し，開発現場で幅広く用いられる工程管理表を用いて計測する汎用的な手法を提案している．例えば，アプリケーション開発プロジェクトとインフラ整備のプロジェクトでは性質の全く異なるプロダクトを開発するため，プロダクト品質自体を直接管理するためには，異なるメトリクスや評価尺度の適用が必要となる．一方，我々の提案するプロセスの複雑さは，どのような性質のプロダクトを開発するプロジェクトであっても工程管理表から同じ方法で計測することができる．汎用性の高い計測手段を提供することで，プロダクトの構造や性質に依存せずにプロダクト品質を議論することが可能になる．

3.3 プロセスの複雑さ概念の妥当性

フラグメントプロセスが実行済みのプロセスから影響を受けることや，今後実行される予定のプロセスに影響を与えることは十分考えられる．しかし，提案するプロセスの複雑さは，追加されたフラグメントプロセス群の同時実行数，つまり，並列性のみを用いて定義されており，因果関係 (あるいは依存関係) を直接表した要素は含まれていない．この理由について図 3 の例を用いて述べる．

図 3 には「計画されたプロセス」として,「要件定義」,「設計」,「実装」,「テスト」等が存在し，設計工程で「追加要望」というフラグメントプロセスが追加された状態が示されている．ここでフラグメントプロセス「追加要望」はすでに完了した「要件定義プロセス」等の影響をうけて作成されたものである．すなわち,「追加要望」の実施期間や開発者数等の値には，過去のプロセスから及ぶ因果関係をプロジェクト管理者が考慮した結果が反映されていると捉えるべきである．同様に，将来実施予定のフラグメントプロセスを追加する場合，図 3 において，未実施の「実装」と「テスト」プロセスに関しては,「追加要望」フラグメントプロセスが追加された影響に基づいて,「追加テスト」のフラグメントプロセスが工程管理表に追加され，現在実施中，あるいは完了したプロセスの実施内容に基づいた値が設定される．このように，完了プロセスからの情報伝達に関するコストはフラグメントプロセスの追加時点で見積もられており，因果関係に由来する複雑さはここに間接的に反映されている．

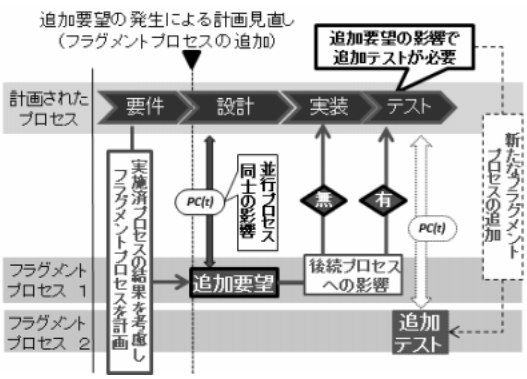

図 3 フラグメントプロセスと他のプロセスとの相互作用

一方で，同時実行中であるプロセス間のコミュニケーションはその時点での実行プロセス数により変動するため，事前の評価が困難である (より正確にはフラグメントプロセスを追加する毎に並列するプロセスの見積りをすべてやり直すことは現実的に行なわれない)．このため，各時点の同時実行プロセスに基づいて近似的計算を行なっているが，この部分が複雑さの定義式内で直接表現している内容に相当する．

3.4 プロセスの複雑さの計測式

プロセスの複雑さの計測式 (以下 PC 値) を以下に示す.

$$PC_{(t)} = \sum_{i=1}^{N_{(t)}}((Num\_dev_{(t)_i} * term_{(t)_i}) * Concurrent_{(t)_i})$$

$$Concurrent_{(t)_i} = 1 + L_{(t)_i} * m_{(t)_i} \quad (1)$$

$PC_{(t)}$: 時刻 t のプロセスの複雑さ (時刻 0〜t までの終了プロセスも含む)

$N_{(t)}$: 時刻 t の全プロセス総数 (時刻 0〜t までの終了プロセスも含む)

$Num\_dev_{(t)_i}$: 時刻 t の i 番目のプロセスに関わる開発者数

$term_{(t)_i}$: 時刻 t の i 番目のプロセスの実行期間の全体開発期間に対する割合. 1.0 に近い場合はプロセスの実行期間がほぼ開発期間全体にわたる

$L_{(t)_i}$: 時刻 t の i 番目のプロセスの同時実行プロセス数 (ただし自身は含まない)

$m_{(t)_i}$: 時刻 t の i 番目のプロセスと他の同時実行プロセスとの関連の重み

提案する PC 値は，計画されたプロセスとフラグメントプロセスの複雑さの総和となる．但し，各フラグメントプロセスの複雑さに対して同時実行される他のプロセスからの影響を考慮する．この時，計画されたプロセス内には複数のプロセスが存在したとしても，1 つのプロセスとする．各プロセスの複雑さは，$Num\_dev_{(t)i}$ と $term_{(t)i}$ と $Concurrent_{(t)i}$ の値で決まる．$Num\_dev_{(t)i}$ は時刻 t の i 番目のプロセスにかかわる開発者数であり，$term_{(t)i}$ は時刻 t の i 番目のプロセスのプロジェクト全体の開発期間に対する実行期間の割合を示す．まず，$Num\_dev_{(t)i} \times term_{(t)i}$ の計算によって 1 つのプロセスの複雑さを求める．この計算は人×時間で工数を意味するが，多くの人が関わり，作業時間が長い作業の方が一般的に複雑であるという考え方に基づく [19]．また，$Concurrent_{(t)i}$ は同時実行するプロセス間の関係を示す値である．同時実行によるプロセスはお互いに作業の同期やプロセス間の関連が発生する [20] [21]．その同期や関連の重みづけをする変数が $Concurrent_{(t)i}$ である．

$Concurrent_{(t)i}$ は時刻 t の i 番目のプロセスが同時実行する $L_{(t)i}$ 個のプロセスと同期をとる必要があることを示し，その同期の重みが $m_{(t)i}$ であり，0 から 1 の範囲で表す．例えば，あるプロセスが全く他の同時実行するプロセスと関係のない単独で実施できるプロセスは $m_{(t)i}$=0 となる．反対に他のプロセスと非常に関連が大きく頻繁に同期をとる必要があるプロセスは $m_{(t)i}$=1 となる．同期をとる必要のあるプロセス数とその重みにて $Concurrent_{(t)i}$ の値が決まる．但し，$m_{(t)i}$ の値は組織毎に異なる可能性が高いため厳密な抽出方法は今後の課題とし，本稿では $m_{(t)i}$=1 とする．これらによって，時刻 t の i 番目のプロセスの工数 ($Num\_dev_{(t)i} \times term_{(t)i}$) に $Concurrent_{(t)}$ を乗算し，当プロセスの本来の工数に並列プロセス間の関連に関する作業工数の増分 ($L_{(t)i} \times m_{(t)i}$) を加えた工数を求める．$Concurrent_{(t)}$ の ($1+L_{(t)i} \times m_{(t)i}$) の式の “1” は時刻 t の i 番目のプロセスの本来の工数を意味する．これらによって，1 つのプロセスの工数である ($Num\_dev_{(t)i} \times term_{(t)i}$) に同時実行の他プロセスとの同期の負荷を $Concurrent_{(t)i}$ で重みづけした値を 1 つのプロセスの複雑さとした．

3.5 PC 値の制約と限界

PC 値の導出に利用するパラメータのうち，$m_{(t)i}$ 以外は工程管理表の構成管理から容易に抽出できる値である．本節では提案する PC 値の測定に際して考慮すべき制約や限界について述べる．

3.5.1 PC 値の制約

本提案は汎用性を高めるため，産業界で一般的に利用されている工程管理表から機械的に抽出できる 3 つのパラメータのみを利用している．しかし，複数プロジェクトで PC 値を比較する際，工程管理表の管理項目の粒度を統一する必要がある．CMM のレベル 3 以上の企業や組織には標準の工程管理表テンプレートや管理標準が存在し工程管理方法が規定されている．したがって，統一された粒度の工程管理が行われるので PC 値を求める際も統一された粒度の値が抽出できる．CMM レベル 2 以下の組織では，統一された工程管理が行われていない可能性が高い．そこで，工程管理表の管理単位において，管理項目がプロセス，アクティビティ，タスクと分類し，時間管理を日毎，週毎，月毎と分類，人管理を個人毎，作業グループ毎，

課毎，部毎に分類して，時間管理と人管理のそれぞれの粒度ごとに工程管理表を分類し，統一された粒度の工程管理表を使ったプロジェクト間で PC 値の比較ができると考える．

また，PC 値はフラグメントプロセスによるコミュニケーションの量を平均的に求めてプロセスの複雑さを表現する．もちろん，個々の具体的な作業内容やその難易度，開発者のスキルや経験，もしくはプロジェクトの特徴等によってもフラグメントプロセスによるコミュニケーションの内容や量も異なる．しかし，本提案では工程管理表から機械的に取得できない具体的な作業内容や難易度等を含めずに，機械的に取得できる 3 要素のみでコミュニケーション量を決定する．これによって，PC 値は多くのプロジェクトで利用できる汎用性の高いプロセスの複雑さの尺度が実現できる．

3.5.2 PC 値の限界

PC 値を計測する際の限界について述べる．まず，$m_{(t)i}$ は同時実行するプロセス間の関連や同期の強さを示す重みであるが，この値は工程管理表の記載事項から単純に抽出することができない．そのため，計測対象であるプロジェクトの厳密なプロセスの複雑さを求める事が困難である．過去の研究では，同時実行される作業のコミュニケーション量を求められており，各作業量の 1 割から 3 割が他作業からの影響があると提示されている [20] [22]．しかし，このような値は組織やプロジェクト毎によって値が異なるため安易に決められない．ただし，複数プロジェクト間で PC 値を比較する際は，統一された $m_{(t)i}$ の値が使われるならば，PC 値の比較においてほとんど影響はない．つまり，プロジェクト A の PC 値がプロジェクト B の PC 値より高いか低いかに焦点を絞って議論するならば，$m_{(t)i}$ の値が統一される限りプロジェクト A とプロジェクト B の PC 値の大小関係に影響を及ぼさない．したがって，過去のプロジェクトの PC 値と比較して現在の PC 値の大小を議論する本稿の 4 章では $m_{(t)i}$ の値を 1 として PC 値を計算する．正確なプロセスの複雑さの値としての PC 値ではないが，複数プロジェクト間でのプロダクト品質と PC 値の関係の比較において，設定された $m_{(t)i}$ の値による影響はないと考える．

続いて計測における限界について述べる．PC 値は工程管理表から機械的に抽出できる 3 つのパラメータを用いて計測される．そのため，PC 値からプロセスが複雑になった具体的要因を知ることはできない．例えば，新規開発や，既存システムのバージョンアップや更にウォータフォールプロセスでの開発のようなプロジェクトの特徴と PC 値の関係を述べる事は困難である．しかし，本提案は並列プロセスがプロダクトへどのように悪影響を与えるかを明確にするものである．つまり，計測対象のプロジェクトに致命的な障害発生が潜在しているかを示す尺度として PC 値が十分活用可能と考える．

3.6 フラグメントプロセスの抽出方法と PC 値の計測例

PC 値は工程管理表にて記載された計画されたプロセスとフラグメントプロセスに基づき計測する．図 4 に工程管理表の例からフラグメントプロセスの抽出方法を示す．表の横軸は日付，縦軸はプロセスを示す．図 4 の左の表は計画されたプロセスだけを示し，右の表は 5 月 1 日に予期せぬ要件が追加され，3 つのフラグメントプロセスが追加された工程管理表を示す．つまり，図 4 の A，B，C は計画されたプロセスであり，D，E，F はフラグメントプロセスを示している．この時に関わっている開発者の粒度はグループ単位とし SE，PG，顧客の 3 グループであり，フラグメントプロセスは PG のみが関わっていたとする．図 4 の工程管理表から 6 月 1 日には計画されたプロセスと 3 つのフラグメントプロセスが同時実行することとなる．つまり，プロジェクト開始時と現段階の工程管理表の差分からフラグメントプロセスの抽出や PC 値のパラメータの値を抽出することができる．本提案は工程管理表さえ存在すればアジャイル開発プロセス等の様々な開発プロジェクトでも利用可能である．

次に PC 値の計測例を示す．プロジェクトの進行に伴い図 4 のように，当初予定されていなかった 3 つのフラグメントプロセスが追加される．図 4 の例では，180 日のウォータフォールプロセスがあったとする．このプロセスに追加プロセスとして，フラグメント

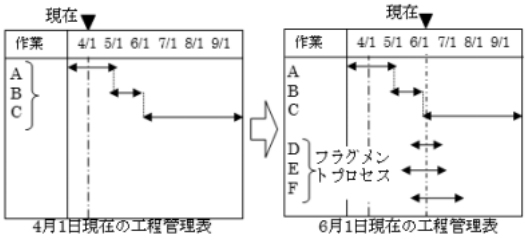

図 4 工程管理表からのフラグメントプロセス抽出例

プロセス D が 30 日，フラグメントプロセス E が 45 日，フラグメントプロセス F が 60 日が発生したと仮定する．計画されたプロセスが 1 つ，フラグメントプロセスが 3 つ発生するため，$N_{(t)}$=4 とする．計画されたプロセスを i=1，追加されたフラグメントプロセスを i=2, 3, 4 とすると，時刻 t が 6 月 1 日 (60 日目) の PC 値は以下の計算となる．ただし，$m_{(t)i}$=1 と統一して計算する．

(1) i=1(図 4 中の計画されたプロセス A, B, C)

1. $Num\_dev_{(60)1}$=3
2. $L_{(60)1}$=3
3. $term_{(60)1}$=180/180=1
4. $Concurrent_{(60)1}$=1+3 × 1=4.0

(2) i=2(図 4 中の D のフラグメントプロセス)

1. $Num\_dev_{(60)2}$=1
2. $L_{(60)2}$=3
3. $term_{(60)2}$=30/180=0.166
4. $Concurrent_{(60)2}$=1+3 × 1=4.0

(3) i=3(図 4 中の E のフラグメントプロセス)

1. $Num\_dev_{(60)3}$=1
2. $L_{(60)3}$=3
3. $term_{(60)3}$=45/180=0.25
4. $Concurrent_{(60)3}$=1+3 × 1=4.0

(4) i=4(図 4 中の F のフラグメントプロセス)

1. $Num\_dev_{(60)4}$=1
2. $L_{(60)4}$=3
3. $term_{(60)4}$=60/180=0.333
4. $Concurrent_{(60)4}$=1+3 × 1=4.0

従って，3 つのフラグメントプロセスが追加されたプロセス複雑さは，

$PC_{(60)}$=12.000+0.664+1.000+1.332=14.996 となる．

4 プロセスの複雑さの適用

産業界における 8 プロジェクトにてプロセスの複雑さを適用した．適用結果を本章で述べる．

4.1 プロジェクトへ適用する際の前提

本節ではプロジェクトへ適用する際の前提について述べる．まず，8 つのプロジェクトからフラグメントプロセスが発生することにより，リリース後の障害に関係があるかを調査する．また，複数のプロジェクト間で PC 値とプロダクト品質を比較するため，3 章の PC 値の制約と限界で述べた $m_{(t)i}$ の値を今回の適用するプロジェクトでは $m_{(t)i}$=1 とする．また，プロダクト品質は障害管理表 (リリース後の障害管理) にて記載される障害レベルとその件数を用いる．障害レベルは最も高いレベルの障害を SS，逆に運用に支障が無いレベルの障害を C とする 5 段階のレベルからなる．

本稿のようにプロセスの複雑さとプロダクト品質の関係を比較する際にはプロジェクト間の工程管理表と障害管理表の管理粒度を合わせる必要がある．そこで，本稿で適用する 8 プロジェクトでは，工程管理表のプロセス実行期間が週単位だったものを全て日単位へと変更し統一している．同時に開発者数はグループ単位とし，管理項目の粒度はプロセス単位となっている．また障害管理表では障害レベルを SS から C ランクの 5 段階で管理しており，各障害は発生した全てのエラーが記載されている．

4.2 8 プロジェクトの適用結果

図 5 に 8 プロジェクトの PC 値の推移を示す．グラフ中の数値は最終 PC 値である．最も高い PC 値はプロジェクト D の 49.5 であり，最も低い値はプロジェクト F の 11.6 である．したがって，プロジェクト D は多くのフラグメントプロセスを含む複雑なプロセスであることがわかり，プロジェクト F はフラグメントプロセスが少ないシンプルなプロセスであることがわかる．また，各プロジェクトの障害レベルの件数を表 1 に示す．表 1 から PC 値が高いプロジェ

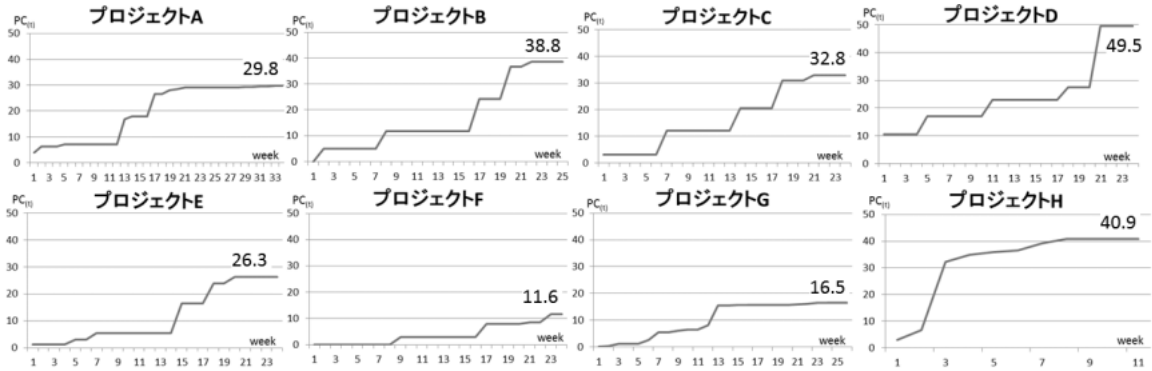

図 5　8 プロジェクトの PC 値の推移

表 1　8 プロジェクトの PC 値と障害件数の詳細

プロジェクト名	最終的なPC値	障害管理表による障害レベル					全障害件数
		SS	S	A	B	C	
A	29.8	4	4	6	9	11	34
B	38.8	2	1	6	10	13	32
C	32.8	2	3	19	1	1	26
D	49.5	10	6	4	1	0	21
E	26.3	3	3	2	1	2	11
F	11.6	0	0	7	3	1	11
G	16.5	0	0	8	9	3	20
H	40.9	12	0	2	0	4	18

表 2　8 プロジェクトの PC 値と障害件数の相関係数

SS障害件数	S障害件数	A障害件数	B障害件数	C障害件数	全障害件数
0.786	0.554	−0.171	−0.258	0.152	0.402

クトは障害レベル SS の件数も多い傾向となった．そこで，各プロジェクトの PC 値と障害件数の関係について分析を行った．まず，PC 値と障害発生件数をピアソンの積率相関係数で分析したところ，0.402 と弱い相関であり，PC 値と障害件数にはあまり関係が無いと言える．そこで，PC 値と各障害件数を積率相関係数で分析したものを表 2 にまとめた．表 2 より，PC 値と致命的な障害である SS レベルの障害には相関係数 0.786 となり，PC 値と SS レベルの障害には相関が認められた．つまり，PC 値はシステム開発における致命的障害をプロセスの観点から発見できる可能性が確認できた．

4.3　PC 値の上昇と障害の関係

本節では PC 値が上昇すると致命的な障害が発生するかの調査を行った点について述べる．調査対象はプロジェクト A, D, E であり，障害管理表と工程管理表から，それぞれの障害の原因となったプロセスを実施した工程を特定した．調査結果を図 6 に示す．

図 6 では SS レベルと S レベルの障害の原因となったプロセスの実行期間を斜線の四角で示した．たとえば，プロジェクト A では 18 週目から 23 週目に致命的な障害である SS レベルの障害 4 件の原因がプロセスに埋め込まれた．実際の SS レベルの 4 件の障害はリリース後に発生したが，原因は 18 週目から 23 週目のプロセスで埋め込まれたことを意味する．同プロセスでは S レベルの 2 件の障害の原因も埋め込んだことを示す．ただし，特定されたプロセスには実施期間が数週間あり，数週間のどの時点で埋め込んだかは明確にできなかった．

図 6 において全てのプロジェクトに共通の点として，PC 値が著しく増加した後のプロセスに SS レベルの障害の原因が多く含まれている．最もわかりやすい例はプロジェクト D であり，19 週目に著しく PC 値が増加した．このあとのプロセスにおいてプロジェクト D で発生した SS レベルの障害件数の半数以上である 6 件の障害の原因を埋め込んでいる．同様に，プロジェクト E でも 17 週目の追加されたプロセスによって，全ての SS レベルの障害の原因がそのプロセス実行中に埋め込まれた．

このように，フラグメントプロセスの発生により PC 値が増加し，致命的な障害を引き起こす原因をプロセスに埋め込む可能性が高いことがわかった．つまり，PC 値が上昇すると致命的な障害を引き起こす可能性が高くなる事を顧客に示し，プロジェクト全体の

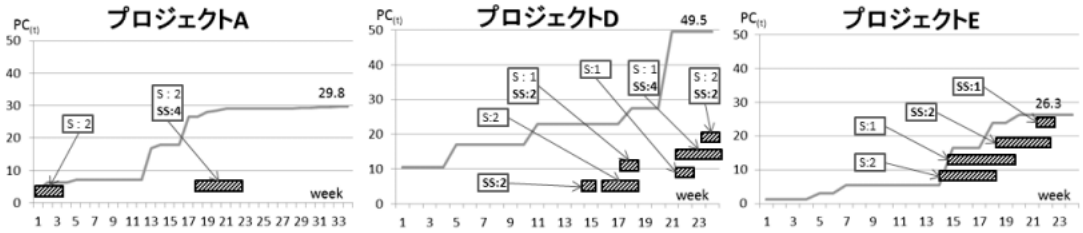

図 6 3 プロジェクトの障害発生プロセスと障害件数の関係

工程見直しの必要性を顧客に示唆する上で有益であると考える．

4.4 PC 値とリリース後の障害の関係

8 つのプロジェクトのうち 6 プロジェクトに関して開発者のインタビューを行い，その結果から PC 値とプロジェクトの特徴について考察を述べる．まず，プロジェクト A は新規システム開発であり，比較的 PC 値が高いプロジェクトである．開発者へのインタビューによると，顧客が新しいシステムを早期にイメージすることが難しく，顧客の要求が頻繁に変更され，追加要望が多数発生した．つまり，要求変更により追加要望が多くなり，フラグメントプロセスの増加につながったと考えられる．同様にプロジェクト C も新規システム開発であり，顧客はシンプルな GUI を要望したが，開発者がそれをイメージできず，結果として，プロトタイプを作成するフラグメントプロセスが発生した．また，テスト工程ではネットワークエラーによるネットワークチューニングのためのフラグメントプロセスが発生した等により PC 値が増加した．それに対して，プロジェクト B はすでに稼働している母体システムのバージョンアップであった．顧客も母体システムを熟知し，追加する機能のイメージも把握しているため，要求変更が頻繁にされることはなかった．しかしプロジェクト後半のテスト工程で仕様ミスが発覚し，再設計，再実装，再テストと 15 週目以降に PC 値が増加した．また，プロジェクト D は母体システムのバージョンアップであったが，PC 値は開発期間を通して常に高かった．開発者へのインタビューによれば，プロジェクト D は前バージョンのシステムに多くの障害が発生し，その障害対応とプロジェクト D で本来実施すべき開発作業の同時進行が行われた．つまり，計画された新規機能の開発を行いながら，多くの障害対応のためのフラグメントプロセスが発生した結果として，プロジェクト D の PC 値は常に高い値を示した．プロジェクト E は母体システムのリファクタリングが主な開発内容であった．したがって，新たな機能開発を行わずリファクタリングに集中したため，PC 値が比較的低くなったと考えられる．その後のプロジェクト F はプロジェクト E で未実装機能の開発のため，開発者が機能内容や顧客の意思を熟知しており，また母体システムも安定していたため全体を通して PC 値は低い値を維持した．

インタビュー結果から，PC 値が比較的高いプロジェクトはプロジェクト自身に何らかの問題を抱えるためフラグメントプロセスが発生し，結果リリース後の障害も多くなる傾向にあった．つまり，PC 値にはプロジェクトの特徴を考慮せず計測したにも関わらず，プロジェクトの特徴が並行プロセス数に表れたと考えることができ，並行プロセス数を計測する PC 値とリリース後の障害には密接な関係があることがわかった．

4.5 PC 値とシステム開発規模の関係

8 つのプロジェクトの PC 値と開発規模の関係について述べる．本稿の提案する PC 値は工程管理表に記載される全ての作業を含む．そのため，ソフトウェア開発の他にもネットワーク設定やサーバ設定等のインフラ構築の作業も含まれている．インフラ等の作業が含まれるためシステム開発の規模をソースコード量等では計測できない．そこで，本稿では全作業の工数を開発規模として計測した．ソフトウェア開発以外の工

程も含んだ PC 値を計測する理由として，ネットワークの設定ミス等のインフラ構築によるシステム障害の比率が高くなっている [23]．つまり，近年のシステム開発はソフトウェア開発だけではなく，ソフトウェアを利用するためのネットワーク環境のインフラ構築もシステムを開発する上で重要な品質要素となるため，本稿ではインフラ等の作業を含んだ工程管理表を用いている．同時に PC 値はインフラ構築プロジェクトでも利用可能であるため，クラウド化が進む近年のシステム開発プロジェクトへの適用も可能である．

8 つのプロジェクトの開発規模を表 3 に示す．8 つのプロジェクトの内，最も工数が多いプロジェクトはプロジェクト D であり，最も工数が少ないプロジェクトはプロジェクト F であった．プロジェクト D は前節でも述べたとおり，新規機能の開発と母体システムの障害対応のため大きな工数となり，かつ，致命的障害も多く発生した．同時に最も工数が少ないプロジェクト F はフラグメントプロセスの発生しない安定したプロジェクトだった．この 2 つのプロジェクトを見ると PC 値と工数には密接な関係があると考えられる．そこで，8 つのプロジェクトの工数と SS レベルの障害をピアソンの積率相関係数を分析した結果 0.267 となった．同時に工数と障害発生件数を積率相関係数にて分析した結果 0.368 となり，共に本提案の PC 値よりも小さい相関となった．この理由として，工数では計測できない並行プロセス数という点を PC 値は計測できる点にある．例えば，プロジェクト H は工数が 720 と比較的小さいが，SS レベルの障害件数は 12 件であり，致命的障害が多く発生した．この理由として納期が非常に短く，かつ，並行プロセス数が多いプロジェクトであったことが考えられる．1 つのプロセス自体は比較的に短期間で終わるプロセスが多いが，並行プロセス数が多いため設定ミスが発生し，結果多くの致命的障害が発生した．つまり，提案する PC 値では工数等の開発規模では測れない短期間で無理に遂行するプロジェクトも正確に計測できることがわかった．

表 3　8 プロジェクトの工数と障害件数

プロジェクト名	最終的なPC値	工数(人日)	全障害件数	SS障害レベル件数
A	29.8	1160	34	4
B	38.8	1215	32	2
C	32.8	1660	26	2
D	49.5	1895	21	10
E	26.3	1175	11	3
F	11.5	610	11	0
G	16.5	680	20	0
H	40.9	720	18	12

4.6　PC 値の利用方法

本節では PC 値の利用方法について述べる．提案する PC 値は工程管理表へ追加される並行プロセス数 ($L_{(t)_i}$ 値) を制御することによりリリース後のプロダクト品質を高くする試みである．図 7 の左がフラグメントプロセスによって複雑化した工程管理表 (図中点線矢印がフラグメントプロセス) を示し，右側が並行プロセス数を制御した工程管理表の例である．

図 7 左の工程ではプロジェクト開始時に顧客から前システムの改善と，改善に伴う新たな機能の追加要望が発生したとする．顧客は容易な修正程度の気持ちで要求したが，開発者の工程は前システムの改善や追加機能による再設計や，新たなテストケースの作成等，様々な工程が追加される．結果として，並行プロセス数が増加し，プロセスが複雑化することで致命的な障害の原因を含むプロセスとなる可能性がある．そこで，PC 値を利用し，顧客と共に複雑化したプロセスを改善し，プロセスに致命的な障害の原因を混入することを避ける．

さらに，顧客と開発者で実現機能の優先順位等を決めて，次バージョンへの移行など，並行プロセス数を減らすプロセス改善を行う．図 7 の場合は，顧客との話し合いによって，フラグメントプロセスである前システムの改善，機能 D の開発，インフラ再構築，追加テストが従来の計画である機能 A, C の開発よりも優先順位が高いという結論となり，10 月 1 日迄に実現する要望を整理した結果，図 7 の右の新しい計画へと変更された．さらに PC 値は 30 に抑えることができ，プロダクト品質の安定も期待できるプロセス改善が実現できた例である．

類似のプロセス改善が前述のプロジェクト E，F にて実際に実施された．プロジェクト E は母体システムのバージョンアップであり，プロジェクト F はプ

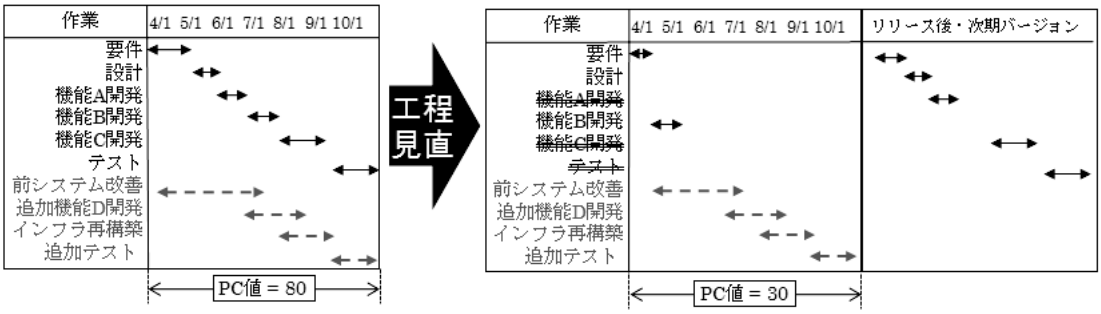

図 7 PC 値を利用したプロセス改善方法例

ロジェクト E の機能追加バージョンアップのシステム開発である．しかし，母体システムの品質が低いためにプロジェクト E では母体のリファクタリングと新規機能の実装を行う必要があった．顧客と開発者の話し合いにより，プロジェクト E ではリファクタリングを中心に行い，新規機能をプロジェクト F にて実装する事で並行プロセス数を減らした．同時にプロジェクト E，F で実装する機能を再度見直すことで，必要性の低い機能の実装を見送った．結果，プロジェクト E，F 共に他のプロジェクトと比較しても少ない障害件数となった．ただし，このプロセス改善は開発者，顧客の経験に基づいて実施された．そこで，経験の代わりに定量的な PC 値を用いることで，プロダクト品質と機能の実現をトレードオフしながら，顧客の要望を反映したシステムを提供するためのプロセス改善を実現できる．

また，PC 値を定量的に計測することで開発者が追加要望を渋ることも考えられる．しかし，PC 値を利用したプロセス改善は，追加要望を単に容認するのではなく，並列プロセスの多発による開発者の混乱を避けるために，計画されているプロセスの削除も含むプロセス改善である．つまり，開発者にとっても開発がスムーズに進むプロセスへ改善される可能性が高くなる．顧客にとっても最終的なプロダクトの品質の保証され，両者にとってメリットがあるプロセス改善が実現できる．

5 考察

本提案の最終目的はプロセスの複雑さが将来的なプロダクト品質に及ぼす影響を予測することである．本章ではプロダクト品質への影響を予測するための PC 値活用方法について概略を述べる．

5.1 PC 値活用のための方針

はじめに，本稿で提案する PC 値を活用するためにはプロダクト品質を示す必要がある．プロダクト品質を表すメトリクスには ISO/IEC9126 をはじめとし，これまでに様々な提案が行なわれているが，本節ではプロダクト品質に及ぼすリスクの概念として，リリース後の障害レベルとその件数に基づいた「障害比重」と呼ぶ指標を仮定する．障害比重を利用する理由として，過去に提案されるプロダクト品質を示す提案は組織によって利用できないものや，別途データ収集を必要とする可能性がある．そこで，今回はどの組織でも発生した障害を管理した障害管理表から指標を求める障害比重とした．

障害管理表には致命的な障害である障害レベル SS から軽微な障害である障害レベル C までが記載されているものとする．このとき，障害比重は，障害管理表に記載された障害の加重和を求めたもの，すなわち，各障害レベルの障害件数と障害レベル別に定めた重みの積の総和である．本章の例では，障害レベル SS の 5 ポイントから障害レベル C の 1 ポイントまで各レベルに 1 ポイントずつ重みの差をつけている．

障害レベルは順序尺度に相当する概念であるため，各障害レベルの重みに 1 ポイントから 5 ポイントのような比例値を設定し，加重和とする妥当性については慎重に検討すべきだが，障害レベル別に所要コスト

の平均等の実績値を求め，それを基に各レベルの重みを定めた場合には，障害比重はコスト増大等のリスクを表現する指標として有望である．そこで，障害レベル別の重み付けについては今後の課題としつつ，以下では，単純な比例値 (ポイント) に基づいた障害比重を用いて，PC 値活用の一案を示す．

5.2 PC 値を活用したプロダクト予測モデルの一例

前節で述べた障害比重を利用した PC 値の利用例について述べる．例えば，過去のプロジェクトにおける PC 値 (プロセスの複雑さ) と障害比重 (リリース後の障害発生コスト指標) との間に相関の高い予測モデルが得られた場合には，新規のプロジェクト進行途中に，定期的に PC 値を求める，あるいは，計画された将来のプロセスからの予測値として PC 値を求めることで，リリース後の障害の発生リスクの増大を指標として示すことができる．

表 4 は 8 プロジェクトに対して求めた PC 値 (プロジェクト完了時の PC 値) と障害比重である．PC 値と障害比重の間ではピアソンの積率相関係数は 0.801 と高い値を示している．このように PC 値と障害比重に何らかの因果関係があると前提できた場合に PC 値から障害比重への予測式を求める．図 8 は 8 プロジェクトのデータを基に近似線 (線形近似) で求めた例である．具体的な予測式の求め方は今後の課題とするが，この例では R 値を参考に単純に予測式を求めており，これを用いて新たなプロジェクト I についての予測を試みた．

プロジェクト I は母体のあるシステムの改修プロジェクトである．プロジェクト I の PC 値の推移を図 9 に示す．プロジェクト I は開発期間が 20 週のプロジェクトであったが，プロジェクト I は前半から PC 値が上昇した．作業内容を確認するとデータ移行やデータ連携のフラグメントプロセスが追加され，プロジェクト当初からの本来計画されるはずのプロセスがプロジェクト途中で追加されていた．プロジェクト I の障害比重値は 60(障害件数は SS レベル 7 件，A レベル 7 件，C レベル 4 件) となった．そこでプロジェクト I の 9 週目の PC 値を計測したところ PC 値は 34.9 であり，図 8 にて求めた式より障害比重は 68.2 と予測された．事後における障害比重の実測値は 60(図 8 に＊のマーク) であり，予測値との間に 14% 程度差異を含むものの，比較的精度の高い予測ができた．

構築した予測モデルの利用例を示す．プロジェクト I の例では 9 週目のフラグメントプロセス発生により，PC 値が 23.6 から 34.9 に増大した．この結果，障害比重の予測値は 51.8 から 68.2 に増大し，品質の低下が懸念されるため，追加人員の要請や，顧客や関係者

表 4　8 プロジェクトの PC 値と障害比重

プロジェクト名	最終的なPC値	障害比重	障害管理表による障害レベル				
			SS	S	A	B	C
A	29.8	83	4	4	6	9	11
B	38.8	65	2	1	6	10	13
C	32.8	82	2	3	19	1	1
D	49.5	88	10	6	4	1	0
E	26.3	37	3	3	2	1	2
F	11.6	28	0	0	7	3	1
G	16.5	45	0	0	8	9	3
H	40.9	70	12	0	2	0	4

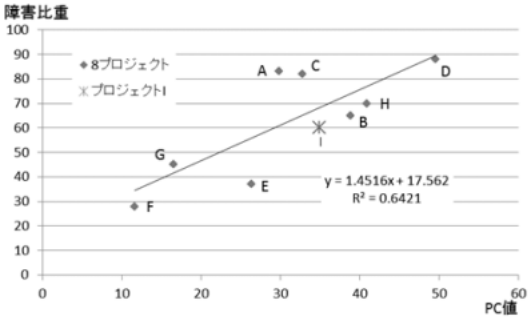

図 8　8 プロジェクトの PC 値と障害比重の関係

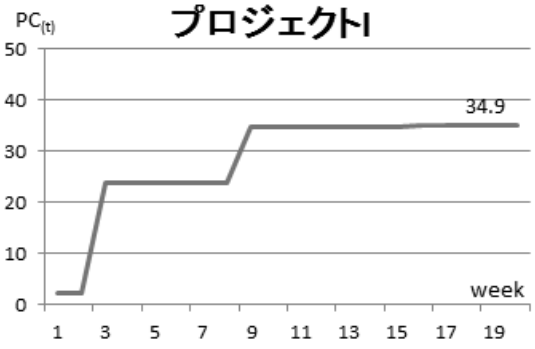

図 9　プロジェクト I の PC 値の推移

等との調整を早期に実施することでプロダクト品質の低下を未然に回避できたと考えられる．

6 おわりに

本稿では，プロセスとプロダクト品質の関係を明確にするためのプロセスの複雑さを提案した．プロセスの複雑さはフラグメントプロセスによるコミュニケーションの発生に着目し，3 つのパラメータを工程管理表から抽出することで計測される．3 つのパラメータは開発者数，同時実行プロセス数，プロセス実行期間であり，これらを個々のプロセスに計測した総和でプロセスの複雑さとなる．作業内容やその難易度，開発者のスキルのパラメータを利用しないので，具体的なプロセスの複雑さや，複雑になった要因を調査することは困難となるが，工程管理表が存在するならばプロダクトの種類の異なるプロジェクトにも適用することができる汎用的な尺度である．提案したプロセスの複雑さとリリース後の障害との関係を産業界の 8 つのプロジェクトで検証した結果，PC 値と致命的な障害件数には積率相関係数で相関が確認された．これにより，プロセスの複雑さの値が大きくなった場合，リリース後の障害に何らかの影響を与える可能性があることを確認できた．またプロセスの複雑さを用いたプロダクト品質予測モデルを考察した．プロダクト品質予測モデルは過去の複数プロジェクトの最終的な PC 値と障害比重の値を用意し，それらを X 軸と Y 軸上にプロットする事でモデルを構築する．プロセスの複雑さとプロダクト品質の関係が確認された 8 つのプロジェクトを用いてプロダクト品質予測モデルを構築し，新たなプロジェクトを予測した結果，実測値に近い予測を行う事ができた．今後の課題として，より具体的なプロセスの複雑さを計測することができる手法を提案し，精度の高いプロダクト品質予測モデルを構築する予定である．

謝辞

本研究の一部は文部科学省科学研究費基盤研究 (C)22500027，および文部科学省科学研究費基盤研究 (C)21500045 の補助を受けた．

参考文献

[1] Royce, W.: *Software Project Management: A unified Framework*, Addison-Wesley Professional, USA,1998.

[2] Damian, D. and Chisan, J.: An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management, *IEEE Transactions of Software Engineering*, Vol. 32, No. 7 (2006), pp. 433–453.

[3] 海谷治彦，北澤直幸，長田晃，海尻賢二：類似既存システムの情報を利用した要求獲得支援システムの開発と評価, 電子情報通信学会論文誌, Vol. J93-D, No. 10(2010), pp. 1836–1850.

[4] 中谷多哉子，藤野晃延：ロールに着目したビジネス領域における要求獲得手法 RODAN の提案, 情報処理学会論文誌, Vol. 48, No. 8(2007), pp. 2534–2550.

[5] Guide to the Software Engineering Body of Knowledge (SWEBOK), http://www.computer.org/portal/web/swebok.

[6] Humphrey, W. S.: *Managing the software process*, Addison-Wesley Professional, USA, 1989.

[7] 坂本啓司，田中敏文，楠本真二，松本健一，菊野亨：利益予測に基づくソフトウェアプロセス改善の試み，電子情報通信学会論文誌，Vol. J83-D1，No. 7(2000), pp. 740–748.

[8] Cardoso, J.: Complexity analysis of BPEL Web processes, *Software Process : Improvement and Practice Journal*, Vol. 12, Issue 1(2006), pp. 35–49.

[9] Cugola, G.: Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models, *IEEE Transaction of Software Engineering*, Vol. 24, No. 11(1998), pp. 982–1001.

[10] Fuggetta, A. and Ghezzi, C.: State of the art and open issues in process-centered software engineering environments, *Journal of Systems and Software*, Vol. 26, No. 1(1994) pp. 53–60.

[11] Garcia, F., Ruiz, F. and Piattini, M.: Definition and empirical validation of metrics for software process models, in *Proceedings of the 5th International Conference Product Focused Software Process Improvement*, 2004, pp. 146–158.

[12] Kruchten, R.: *The Rational Unified Process*, Addison-Wesley Professional, USA, 2000.

[13] Manzoni, L.V. and Price, R.T.: Identifying extensions required by RUP (rational unified process) to comply with CMM (capability maturity model) levels 2 and 3, *IEEE Transactions of Software Engineering*, Vol. 29, No. 2(2003), pp. 181–192.

[14] 青木俊樹，山田茂 :プロセスデータに基づくソフトウェア開発プロジェクトの品質指向型定量的評価法に関する考察 (不確実な状況における意思決定の理論と応用), 数理解析研究所講究録, 1589, 2008, pp. 215–221.

[15] Obana, M., Hanakawa, N., Yoshida N. and Iida, H.: Process Fragment Based Process Complexity with Workflow Management Tables, in *International Workshop on Empirical Software Engineering in Practice*, 2010, pp. 7–12.

[16] Curtis, B., Seshagiri, G.V., Reifer, D., Hirmanpour, I. and Keeni, G.: The Case for Quantitative Process Management, *IEEE Software*, Vol. 25, No. 3(2008), pp. 24–28.
[17] Rao, U. S., Kestur, S. and Pradhan, C.: Stochastic Optimization Modeling and Quantitative Project Management, *IEEE Software*, Vol. 25, Issue 3(2008), pp. 29–36.
[18] Card, D. N., Domzalski, K. and Davies, G.: Making Statistics Part of Decision Making in an Engineering Organization, *IEEE Software*, Vol. 25, Issue 3(2008), pp. 37–47.
[19] Brooks, F. P. Jr.: *The Mythical Man-Month*, Addision Wesley, MA, 1975.
[20] Hanakawa, N.: A project reliability growth model based on communication for software development, *International Journal of Knowledge Engineering and Software Engineering*, Vol. 20, Issue 5(2010), pp. 665–677.
[21] 竹田昌弘 :オープンソースソフトウェアの開発プロセスに関する考察, 立命館ビジネスジャーナル, Vol. 43, No. 4(2004), pp. 49–63.
[22] Hanakawa, N., Matsumoto, K. and Torii, K.: A communication workload estimation model based on relationships among shared works software development projects, in *Proceedings of the 9th Asia-Pacific Software Engineering Conference*, IEEE Computer Society Press, 2002, pp. 571–580.
[23] 松田晃一，金沢成恭 : 情報システムの障害状況 2011 年後半データ, SEC Journal, Vol. 8, No. 1 (2012), pp. 6–8.

尾花将輝

2007 年阪南大学経営情報学部卒．2009 年阪南大学大学院企業情報研究科修了後，阪南大学情報システム課嘱託職員．現在奈良先端科学技術大学院大学博士課程後期に在籍．実践的ソフトウェア開発，開発プロセスに興味を持つ．

花川典子

1984 年大阪教育大学卒．2000 年奈良先端科学技術大学院大学博士後期課程修了．2002 年阪南大学・経営情報，同大学大学院企業情報研究科講師，2007 年同大学院教授．博士 (工学)．ソフトウェア開発プロセスに興味を持つ．電子情報処理学会，情報処理学会，IEEE 各会員．

飯田　元

1988 年大阪大学基礎工学部情報工学科卒業．1991 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．1995 年奈良先端科学技術大学院大学情報科学センター助教授．2005 年同大学情報科学研究科教授．博士 (工学)．ソフトウェアプロセスを中心としたソフトウェア工学の研究に従事．情報処理学会，電子情報通信学会，IEEE，ACM 各会員．