NAIST-IS-DD1061005

博士論文

作業並列度に基づくプロセスの複雑さ尺度

尾花　将輝

2013年2月20日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

尾花　将輝

審査委員：

飯田 元 教授	（主指導教員）
松本 健一 教授	（副指導教員）
藤川 和利 教授	（副指導教員）
花川 典子 教授	（阪南大学）
吉田 則裕 助教	（副指導教員）

作業並列度に基づくプロセスの複雑さ尺度*

尾花　将輝

内容梗概

近年，ソフトウェアが果たす社会的役割が重要となってきており，高品質なソフトウェアを開発する必要性が高くなっている．ソフトウェア開発プロジェクトではプロジェクト発足時に開発計画を立案し，計画に沿ってソフトウェア開発を行う．様々な作業をすべて含む正確な開発計画を立案することが，開発をスムーズに進行する重要な要素の１つとなる．様々な要因を十分考慮した質の高い開発計画が立案されたならば，ソフトウェア開発途中における計画の変更がなく，予定された人員配置と作業にてスムーズなソフトウェア開発が実施でき，結果として高い品質のソフトウェアを生産できる．しかし，実際のソフトウェア開発プロジェクトでは開発当初とプロジェクト終了時の計画には大きな隔たりが存在する．

プロジェクト当初に立案した開発計画に想定しない追加作業が発生することで計画が変更されプロジェクト全体に影響を与える．例えば，顧客の要望で機能要件が変更され，それに伴い仕様書，設計書，ソースコード等の変更作業が開発計画に後から追加される．このように当初の開発計画にない作業をプロジェクト実施中に追加される，つまり予期せぬ作業が追加されることによりプロセス全体が複雑化し，複雑となったプロセスによって開発されたソフトウェア品質（プロダクト品質）が低下する．

そこで本研究では，産業界におけるソフトウェア開発プロジェクトで収集したデータを基にしたソフトウェア開発におけるプロセスの複雑さのメトリクス（Process Complexity）を提案する．提案するプロセスの複雑さとは，ソフトウェア開発における計画に無い追加作業がプロセス全体へ与える影響を定量的に計測するものであ

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文，NAIST-IS-DD1061005, 2013年2月20日.

る．計測には産業界で一般的に利用される工程管理表を用いる．工程管理表には計画当初の作業（工程）から，何らかの要因によって後から追加された作業まで記載されている．工程管理表より抽出する値は開発者数，１つの作業の期間，並列に実行されている作業数の３つの値である．３つの値により，計画には無い作業が発生した際の発生したプロセス以外の作業依存関係を計測することが提案するプロセスの複雑さとなる．提案するプロセスの複雑さを複数のプロジェクトに適用した結果，プロセスの複雑さとシステムが利用できない致命的な障害件数には強い相関があることを確認した．

また，プロセスの複雑さの有用性を検証するために産学連携で開発した p-HInT システムの開発に参加した．p-HInT システムは大人数講義を対象とした授業支援システムである．学生が利用する端末には無線 LAN とブラウザが搭載された低機能な携帯情報端末を対象としたシステムである．主な機能は着席順学生一覧，小テスト，出席確認，座席指定，みんなの声である．p-HInT システムの開発の一員となることで実際にプロジェクト実施中に起きた問題点を把握し，プロダクトの品質に影響を与えるイベントや事象を調査を行った．結果，プロセスの複雑さの値が増加する際には何らかの障害が発生する可能性があることを確認した．

キーワード

ソフトウェアプロセス，プロセスメトリクス，工程管理表，追加プロセス，並列プロセス , システム開発

Process complexity metrics based on concurrent processes*

Masaki Obana

Abstract

Recently, high quality software is urgently required because social importance of software becomes steadily. Development plans also become more important as same as software quality. Smooth development based on a correct development plan leads high quality software. If a correct and high quality development plan is established at planning phase, planned resources such as human and cost, and time would smoothly consume as planned. As a result software quality will be high because of the smoothly development. However, in industrial projects, almost projects change an original development plan.

Unexpected works not including an original plan suddenly occur on the way of a project. The unexpected works are added to the original plan. Therefore the original plan (original development process) becomes complicated steadily. Software quality will be low because of the complicated process.

A new metric named "process complexity" is proposed with industrial software development projects' data. A purpose of process complexity is quantitative measurement in order to clarify complicated development process. The new metric requires only workflow management tables that are popular in industrial software development projects. At first, an original workflow management table includes an original plan as a simple process like a waterfall development process. However, on the way of a project, several unexpected works are added to the

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1061005, February 20, 2013.

original plan. The workflow management table is revised by the unexpected works. Extracted values of the unexpected work are the number of developments and the number of concurrent executing works. As a result of adapting an industrial project, a value of correlation coefficient between values of process complexity and values of product quality is high.

We joined an industrial software development project in order to collect empirical data. The project has developed a lecture support system named "p-HInT" . A feature of the system is usage of low performance mobile terminals such as mobile game and smart phone. Functions are a student list with seat location, test, attendance, student voice. By joining a member of the project, empirical problematic events during the project were clarified. In addition, the usefulness of the "process complexity" metric was confirmed in the project.

Keywords:

Software Process, Process Metrics , Workflow Management Table , Additional Process , Concurrent Process , System Development

目次

図目次

表目次

関連発表論文

学術論文誌

1. 尾花将輝, 花川典子. 携帯情報端末を用いた大人数授業改善のための p-HInT システムの構築. 日本ソフトウェア科学会　コンピュータソフトウェア, Vol. 27, No. 4, pp. 114–132, 2010. （第 4 章に関連する）
2. 尾花将輝, 花川典子, 飯田元. 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. 日本ソフトウェア科学会　コンピュータソフトウェア　Vol. 29, No. 4, pp. 278–292, 2012. （第 3 章，第 5 章に関連する）

査読付き国際会議・ワークショップ

1. Masaki Obana, Noriko Hanakawa, Norihiro Yoshida and Hajimu Iida. Process Fragment Based Process Complexity with Workflow Management Tables. In *Proceedings of the International Workshop on Empirical Software Engineering in Practice (IWESEP2010)*, pp. 7–12, 2010. （第 3 章に関連する）
2. Masaki Obana, Noriko Hanakawa, and Hajimu Iida. A process complexity-product quality (PCPQ) model based on process fragment with workflow management tables. In *Proceedings of the International Conference on Product Focused Software Process Improvement (PROFES 2011)* , pp. 171–185, 2011. （第 5 章に関連する）

査読付き国内会議

1. 尾花将輝, 花川典子. ブレンディッド開発プロセスにおける複雑さのメトリクスの提案. 中島震, 鷲崎弘宜（編）, ソフトウェア工学の基礎 XVI , 第 16 回ソフトウェア工学の基礎ワークショップ (FOSE'09), pp. 51–58, October 2006. （第 3 章に関連する）
2. 尾花将輝, 花川典子, 飯田元. 工程管理表のフラグメントプロセスに基づくソフトウェア開発プロセスの複雑さのメトリクス. 松下誠, 紫合 治（編）, ソフトウェアエンジニアリング最前線 2010, ソフトウェアエンジニアリングシンポジウム 2010（SES2010）, pp. 89–96, 2010. 学生奨励賞受賞（第 3 章, 第 5 章に関連する）
3. 尾花将輝, 花川典子, 飯田元. ソフトウェア開発プロセスの複雑さに着目した PCPQ モデルの構築. ソフトウェアエンジニアリングシンポジウム 2011（SES2011）, 2011. 学生奨励賞受賞（第 5 章に関連する）

その他の発表論文

査読付き国際会議・ワークショップ

1. Noriko Hanakawa, Masaki Obana. A Plagiarism Detection System For Reoirts Based On A Large-scale Distribution Environment Using Idle Computers. In *Proceedings of the International Conference on Product Software Development and Process Improvement (CATE2012)* , 2012.
2. Noriko Hanakawa, Masaki Obana. A Case Study: a metrics for meeting quality on a software requirement acquisition phase. In *Proceedings of the International Conference on Product Software Development and Process Improvement (Profes2012)* , 2012.
3. Noriko Hanakawa, Masaki Obana. Lecture Improvement based on Twitter Logs and Lecture Video using p-HInT. In *Proceedings of the International Conference on Computers in Education* (ICCE2010), pp328-335, 2010.
4. Noriko Hanakawa, Masaki Obana. Mobile Game Terminal Based Interactive Education Environment For Large-Scale Lectures. In *Proceedings of the International Conference on Web-based Education (WBE2010)*, 2010.

査読付き国内会議・研究会

1. 尾花将輝，妹背武志，花川典子，飯田元．教室コンピュータを利用した大規模不正コピーレポート検出システムの開発 高田眞吾，福田浩章（編），ソフトウェア工学の基礎 XVII，第 17 回ソフトウェア工学の基礎ワークショップ

(FOSE'10), pp. 191–192, 2010.

2. 花川典子，尾花将輝．システム仕様定義工程におけるミーティングの質を計測するメトリクスの提案．鵜林　尚靖　岸　知二（編）, ソフトウェアエンジニアリング最前線 2009，ソフトウェアエンジニアリングシンポジウム 2009（SES2009）, pp. 186, 2009.
3. 尾花将輝, 花川典子．使い捨てプログラムを組み込んだインクリメンタル開発プロセスの提案と実践．鵜林　尚靖　岸　知二（編）, ソフトウェアエンジニアリング最前線 2009，ソフトウェアエンジニアリングシンポジウム 2009（SES2009）, pp. 186, 2009.
4. 尾花将輝, 花川典子．組込み機器を用いた 3 層 IT アーキテクチャの提案と実装．松下誠, 川口真司（編）, ソフトウェア工学の基礎 XV，第 15 回ソフトウェア工学の基礎ワークショップ (FOSE'08), pp. 81–86, 2008.　貢献賞受賞
5. 尾花将輝, 花川典子．Web アプリケーションにて家電操作を可能とする組込み機器システムの開発．松下誠, 川口真司（編）, ソフトウェア工学の基礎 XV，第 15 回ソフトウェア工学の基礎ワークショップ (FOSE'08), pp. 157–158, 2008.

査読無し国内研究会

1. 尾花将輝, 花川典子．マイコンを利用した 3 層 IT アーキテクチャ設計の提案 -DollHouse とマイコンによるホームネットワークの実装-．第 52 回システム制御情報学会研究発表講演会，pp.213-214，2008.

第 1 章

序論

1.1. 研究の背景と目的

近年，社会におけるソフトウェアの役割が重要視されている．一方でソフトウェア開発の規模が大きくなる反面，短期間で開発することが社会的に求められている．しかし，実際のソフトウェア開発において短期間にて，顧客の全ての要望を満たすシステムを開発することは困難である．そのため，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が計画に追加されるが挙げられる．このような予期しない追加作業はプロセス全体を複雑化し，結果としてリリース後に大きな障害が発生することに繋がる．しかし，その反対に顧客からの急な要望を割愛しては顧客満足度の高いシステムはできない．

ソフトウェアを高品質に開発するには急な追加作業を極力減らし，開発を円滑に行う必要がある．そのために顧客からの要求を確実にする提案や [47]，プロセスの改善を行う提案が行われている [58]．しかし，実践的な開発では図 1.1 のようにプロセス全体が複雑化する．図 1.1 の計画時にはシンプルなウォータフォールプロセスにて開発する予定であった．しかし，要件定義時には顧客からシステムのプロトタイプの作成要望や，設計時には追加機能要望により，再設計，再開発というプロセスが追加される．このようにして，プロセスが複雑化し，結果としてリリース後のプロダクトの品質に影響を与えることが問題点である．

そこで，本研究ではこのようにプロセス全体が複雑になる点に着目したプロセスの

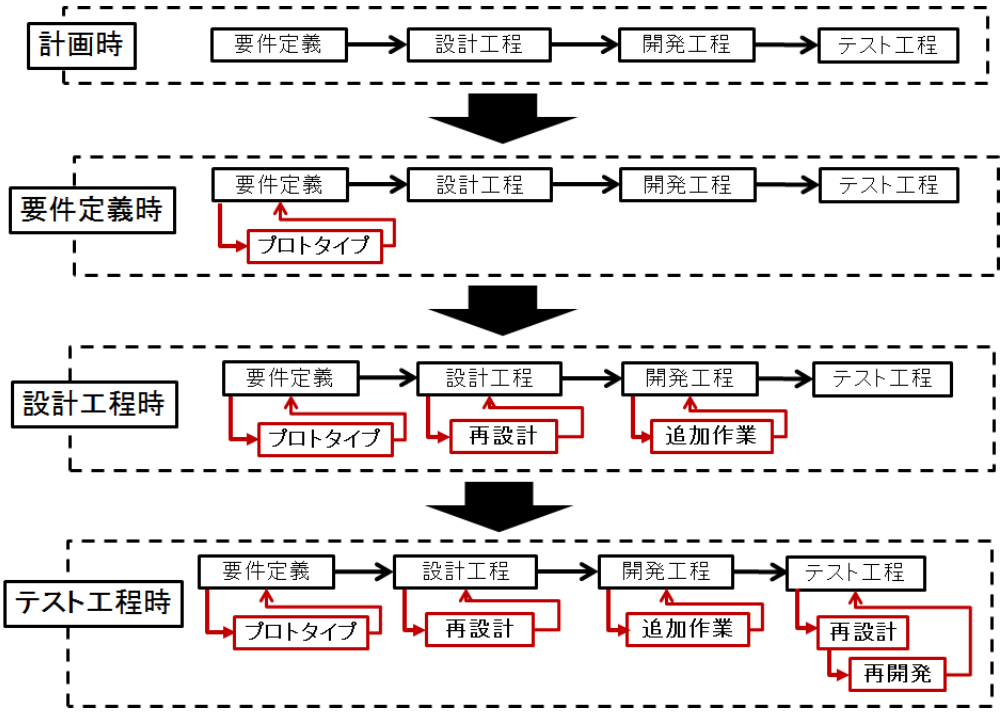


図 1.1　ソフトウェア開発プロセスの複雑化の例

複雑さの定量的計測に取り組み，プロジェクト中に発生する追加要望がプロダクトに与える影響を明確にすることを目的とする．本研究を達成するためのアプローチとして，実践的プロジェクトのデータを利用したプロセス実行時の複雑さの計測である．実践的プロジェクトのデータとして p-HInT システムの開発データを利用する．

（問題点）追加要望のよるプロセス全体の複雑化とプロダクト品質への影響

ソフトウェア開発における開発プロセスがプロダクトへ影響を与えることは過去の研究からも提言され続けている．そのために開発プロセスの改善に関する研究は多々存在する．これらは，ソフトウェア開発プロセスの未成熟さがプロダクト品質を低下させるという着想点から提案されている．しかし，実際の開発プロセスがプロダクトにどの程度，またはどのように影響を与えるかまでは明確にされていない．特に開発中の発生する予期せぬプロセス（タスク）がプロダクト品質に与える影響は大きいと考えられるが，この関係を明確にする研究は少ない．

（目的）開発プロセスの定量化とプロダクト品質への影響の明確化

（問題点）を解決するために本研究では開発中に発生する予期せぬプロセスに着目した新たな尺度，プロセスの複雑さ（Process Complexity）を提案する．予期せぬプロセスを定量的に計測することで開発プロセスとプロダクト品質との関係を明確にする事を本研究の目的とする．提案するプロセスの複雑さは一般的に利用される工程管理表から抽出できる，人，時間，並列作業量から計測する．３つのパラメーターは CMM レベルを取得していない組織の工程管理表でも記述されており，且つプロセスへ影響する要素に着目した．これにより，提案するメトリクス値とリリース後のプロダクトの関係があるかを明確にする．

（アプローチ）実践的データを利用した開発プロセスの複雑さの計測

本研究のプロセスの複雑さのメトリクスは納期が定められているプロジェクトを前提としている．そこで，本研究では授業支援システムである p-HInT システムを産学連携にて開発し，具体的なデータ収集を行った．p-HInT システムは 1 つの授業にて 400 台の携帯情報端末を利用するシステムであり，2008 年 4 月から運用を開始した．2009 年 11 月の総利用人数が 3000 人以上のシステムであり，2012 年 11 月現在でも利用されているシステムである．p-HInT システムの開発に実際に関わる事で開発に関するメール等のデータ収集や，具体的な障害発生した時期とその要因を把握した．これにより，提案するプロセスの複雑さとプロダクトの関係を分析する．

1.2. 本論文の構成

本論文の主要部分は大きく 3 つの章から構成される．

第 3 章では，**問題点**に対し，**目的**を達成するために開発中に発生する追加プロセスを定量的に計測する**プロセスの複雑さを提案する**．プロセスの複雑さは開発時に利用される工程管理表に記載される人，時間，並列作業数から計測する．

第 4 章では，プロセスの複雑さの有用性の検証を行うために開発に参加した p-HInT システムの開発について述べる．p-HInT システムの開発に参加することで，参加しなければわからない開発時の詳細なデータを収集することが本章の目的であ

る．これにより，プロジェクト中に発生する事象と提案するプロセスの複雑さの関係を調査し，プロセスとプロダクトの因果関係を分析する．また，プロセスの複雑さの着想に至った経緯を明確にする．

第 5 章では，本論文の**目的**であるプロセスとプロダクトの関係を明確にする．**第 3 章で提案する**プロセスの複雑さを**第 4 章で述べた** p-HInT プロジェクトを含めた**8 件**のプロジェクトに適用し，検証を行う．検証にはリリース後に発生する致命的な障害との相関を求める．また，各種のプロジェクトの特性とプロセスの複雑さの関係について述べる他に，第 4 章で述べる p-HInT システムの困難な要求とプロセスの複雑さの関係についての分析を行う．

第 2 章

関連研究

2.1. 品質の高いソフトウェアを開発するための研究

ソフトウェア工学は大規模なソフトウェアを高品質に開発することができるかを追及する研究分野である．高品質なソフトウェアを開発するためにはソフトウェアやプロセスを定量的に計測する事は必然である．本章ではソフトウェアを定量的に計測するメトリクスとメトリクスに関する研究について述べる．

2.2. プロダクトメトリクスとプロセスメトリクス

ソフトウェア開発におけるプロダクトの品質やプロセスを定量的に計測する研究は多く存在する．特にプロダクトやプロセスを定量的に計測した値をソフトウェアメトリクス（メトリクス）と呼ばれ，代表的なメトリクスとしてプロダクトメトリクスとプロセスメトリクスが存在する．プロダクトメトリクスは単純にソースコードの行数を計測した LOC（Line Of Code）やサイクロマティック数 (Cyclmatic metric) 等のソースコード複雑さを指す事が多く [43]，プロダクトを定量的に計測したメトリクスである．

一方，プロセスメトリクスはソフトウェア開発プロセスを定量的に計測したものであり，開発工数や，開発期間（時間），イベント（レビュー）等を計測したものであり，プロセス改善に利用することが多い．大規模なシステム開発におけるプロセスメトリクスの計測は重要な要素であり，過去に開発した類似システムから開発す

る工数の予測を行う際等に利用されることが多い [1, 64]．そこで，表 2.1 に代表的なメトリクスをまとめた．

計測されたメトリクスを用いることでプロダクト品質低下要因を調査し，次期計画のソフトウェア開発に取り入れる事が一般的である．本研究で提案するプロセスの複雑さはプロセスメトリクスに分類され，従来のメトリクスでは存在しない，同時刻の並行作業数に着目している．

プロダクトメトリクス	プロセスメトリクス
ユースケース数	工数
FP(Function point)	工期
LOC(Lines Of Code)	生産性
Halstead[57]	顧客との対話時間
サイクロマティック数 [44]	インタビュー実施回数
パブリックインスタンスメソッド数	開発開始後の仕様変更回数・変更率
インスタンス変数の数	ソースコードの変更回数・変更率
特殊化指数	プロトタイピング実施の有無
CK メトリクス (CBO，RFC，DIT，NOC)[13]	静的解析ツールの仕様・不使用
LCOM[10]	コードレビュー実施の有無
メソッドの呼び出し数	レビュー効率
クラス間結合度 [38]	欠落除去率
モジュールのファンイン・ファンアウト [32]	開発者数
情報フローの複雑さ	
コードクローン数 [3]	
コメント量（密度）	
テストケース数・網羅率	
欠落密度	
設計書・仕様書のページ数	

表 2.1　ソフトウェアメトリクス一覧

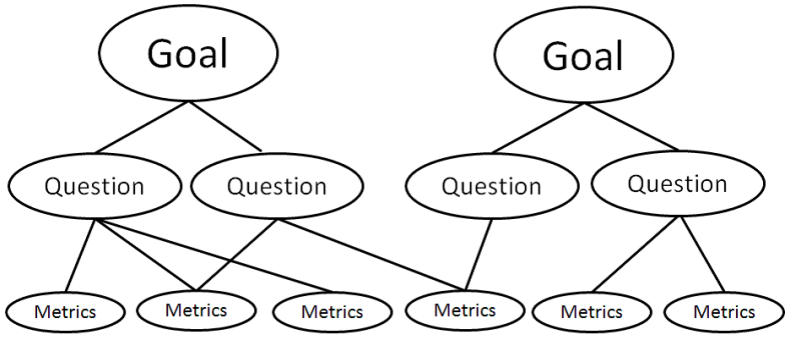


図 2.1　GQM 概念図

2.3. メトリクスの利用法

2.3.1. GQM パラダイム

ソフトウェアメトリクスはただ計測するだけではソフトウェア開発の評価や改善を成すことはできない．そこで，ソフトウェアメトリクスを汎用的に利用するためのフレームワーク（モデル）として，GQM（Goal - Question - Metrics）パラダイムが提案された．

GQM パラダイム（以下 GQM）は Basili 等によって提案されたソフトウェアメトリクスを利用した総合的な計測の枠組である [6]．GQM はデータの計測や分析は (1) トップダウンに行うべき，(2) データ分析の目的（仮説）の明確化するべきであるという 2 つの着想から成り立つ．図 2.1 に示すように計測の目標を明確にし（図 2.1 中の Goal），目標を達成するためには仮説を立て（図 2.1 中の Question），そのためにソフトウェアメトリクスを用いて計測・分析（図 2.1 中の Metrics）を行う必要がある．

GQM を用いた研究では，提案者である Basili 等がソフトウェアプロセスの改善を目的としたソフトウェアエンジニアリングプロセスモデルを提案している [5]．また，GQM を実際のプロジェクトに適用し，GQM の問題点を提唱した研究も存在する [56]．同様にプロセス改善度合を計測する際に GQM を利用する研究や [58]，プロセスモデルの保守性や変更容易性を GQM を用いて計測されている [22]．

2.3.2. メトリクスを利用した数理モデルと見積もり

ソフトウェアメトリクスの計測は対象属性の数値化が基本的な役割である．計測されたメトリクスはソフトウェアプロセスの評価，改善や工数見積もり等に利用され今も様々な提案が行われている [40]．次項では，ソフトウェアメトリクスを利用する際に一般的によく利用される数理モデルとその関連研究について述べる．

COCOMO モデル

COCOMO モデルは Boehm によって提案された工数見積モデルである [8]．メトリクスである LOC 値を FP(Functoin point 法等) により予測し，説明変数とすることで工数を予測するモデルである．COCOMO には 3 つの階層が存在し，初級，中級，上級と別れており，初級 COCOMO では単純に LOC 値 (KLOC) を基に開発工数を予測する単一変数モデルである（式 2.2，2.2 参照）．中級以上では初級 COCOMO に加え，15 項目のシステム開発特性（コスト要因）を含め，上級では中級に更に開発フェーズごとの変動要因を組み込んだモデルである．また，システムの開発特性を 22 項目となり，説明変数を LOC に限定しない COCOMOⅡモデルも存在する [9]．なお，COCOMO モデル，COCOMO モデルⅡは要求仕様の決定後は仕様変更が無く，見積り範囲はシステム設計からシステム結合テストまでを対象とする制約事項がある．

$$E = \alpha(KLOC)^{\beta} \qquad (\alpha > 0, \beta > 1,\ E : \text{開発工数（人月）}) \tag{2.1}$$

$$D = cE^{d} \qquad (c > 0,\ 0 < d < 1,\ D : \text{開発期間（月）}) \tag{2.2}$$

信頼度成長モデル

ソフトウェア信頼度成長モデルはソフトウェアの信頼性に関わる影響要因を考慮し，定量的に信頼性評価を行うための数理モデルである [69]．一般的にフォールト検出数等を時間の経過から観測し，その変化を曲線とすることで，今後発生するフォールト数の予測や平均故障発生時間間隔を計測する事に利用される [70]．

モデルとしては一般的にフォールト等の検出が時間の経過と共に変化する非同次

ポアソン過程（NHPP）が利用される事が多い．時刻 t の時点で検出されるフォールト数（目的変数）を H(t), 総フォールト数を a とした場合の指数形ソフトウェア信頼度成長モデルは以下の式となる．

$$H(t) = \alpha * \beta * e^{-\beta t} \tag{2.3}$$

$H(t)$：時刻 t で検出されるフォールト数

α：総フォールト数

β：フォールト 1 個当りの発見率

また，指数形ソフトウェア信頼度成長モデルに時間の経過と共にフォールトの検出率が変化する概念を導入した習熟 S 字形ソフトウェア信頼度成長モデルが以下の式となる．

$$H(t) = \frac{\alpha * \beta(1+c)e^{-\beta t}}{1 + c * e^{-\beta t}} \tag{2.4}$$

c：テスト能力の熟練度傾向

他にも，修正指数形，遅延 S 字形テスト労力依存型等の様々な信頼度成長モデルがある [69]．

信頼度成長モデルは現在までに多数の研究が行われており，オープンソースを対象として各コンポーネント間の依存関係を信頼度成長モデルで信頼性を評価している [63]．また，様々なソフトウェア信頼度成長モデルを統合した提案 [60] や，ガンマソフトウェア信頼性モデル等，様々な研究が提案されている [52, 68]．

重回帰モデル

重回帰モデルは特定の属性である値（目的変数）を関係のある複数の値（説明変数）から予測値を求める数理モデルである．例えば，プロジェクト途中で計測された

複数のメトリクスからプロジェクト終了までの工数やソースコード規模を予測する等が挙げられる [11, 62]．同時にツールの普及により，工数予測等で広く実施されてきた [12, 67].

重回帰モデルは複数の説明変数から 1 つの目的変数を求める．重回帰モデルの計測式は以下となる．

$$y = \sum_{i=1}^{N} \beta + \alpha_i * x_i \tag{2.5}$$

上記の式の y を目的変数とし，x を説明変数とすることで値を予測するものである．係数 α 及び β は与えられるデータを重回帰分析にて求める．また，重回帰モデルの他に回帰モデルが存在するが，下記の式のように 1 つの説明変数から目的変数を求める数理モデルである．

$$y = \sum_{i=1}^{N} \beta + x_i \tag{2.6}$$

ロジスティック回帰モデル

ロジスティック回帰モデルは重回帰モデルと同様に複数の説明変数から目的変数を予測するものである．重回帰モデルでは目的変数を連続変数とすることで工数等を予測することに長けている．しかし，フォールト等が存在するかを予測する際には重回帰モデルでは予測することは困難である．そこで，目的変数を 0 から 1 とすることで，発生確率を予測する．以下がロジスティック回帰モデルの式となる．

$$log\frac{p}{1-p} = \sum_{i=1}^{N} \beta + \alpha_i * x_i \tag{2.7}$$

ロジスティック回帰モデルを利用した研究では Fault-prone モジュールの判別が有名である．目的変数にソフトウェアのフォールト潜在の疑わしさとし，Fault-prone モジュールの予測に利用される [3, 4, 31]．また Fault-prone モジュールの予測の研究ではソフトウェアメトリクスを利用した線形判別分析，ニューラルネットワーク，分類木等による Foult-prone 判別モデルが提案されている [23, 36, 46].

2.4. ソフトウェアプロセスに関する研究

本研究は提案するプロセスの複雑さからプロダクトの関係を明確にすることが目的である．明確化されたプロセスの複雑さは今後プロセス改善の研究に利用できると考えられる．そこでメトリクスを利用したプロセス改善に関する研究について述べる．

2.4.1. ソフトウェアプロセスのモデル化

ソフトウェアプロセスではプロセスを何らかの規定に基づき形式化，抽象化し，規約に従い記述する必要がある．そのため，ソフトウェアプロセスモデルではPMBOK（Project Management Body of Knowledge）やSLCP（Software Life Cycle Process）等のようなソフトウェアプロセスのメタモデルが提案されている．PMBOKはプロダクトやサービス等を生み出だすアクティビティの組み合わせによってプロセスが成形されると定義されている．また，プロセスをWBS（Work Breakdown Structure）にて階層化し，線表へと表記する事も行われている．この線表を産業界では工程管理表やマスタースケジュール等と呼ばれ，現在も広く利用されている．

2.4.2. CMMI

CMMIはソフトウェアプロセスを改善するためのモデルである．組織のプロセスの成熟度を5段階で評価することで，その組織のプロセス能力を評価し，改善方針に役立てるために利用される．初期，管理，定義，定量化，最適化と5段階の定義され，各段階で細かく達成すべきである項目が定められている．これらの項目を調べる際には総合試験欠落率，試験密度やレビュー効率等の様々なメトリクス値を収集し，指標として利用される [33].

2.4.3. プロセスデータ収集ツールと可視化ツール

ソフトウェアプロセスを分析するためのプロダクトメトリクスやプロセスメトリクスのデータ収集は煩雑である．本節ではソフトウェア開発に関わるデータ収集を自動収集する取り組みについて述べる．Java や PHP 等様々な言語に対応する統合開発環境 Eclipse では Eclipse Metrics Plugin が提供されている [54]．開発中のコードから LOC やサイクロマティック数等の様々プロダクトメトリクスを自動的に計測し，計測値から適正範囲の観点から可視化を行なっている．同様に C 言語や C++ を対象としたプロダクトメトリクスを計測するアプリケーション CCCC 等が存在する [39].

EASE プロジェクトにて開発された EPM(Emprical Project Monitor) ではソフトウェア開発プロジェクトを対象とした自動収集ツールが提供されている [17]．版管理システム CVS 等を利用し LOC や障害件数，メールログ等のデータ収集，及び分析を自動で行い，開発者に負担を掛ける事なくプロジェクトの状況を把握することができる．また，過去のプロジェクトの開発プロセスを可視化分析するためのツール Project Replayer が提案されている [50].

2.4.4. ソフトウェアプロセスに関する研究

ソフトウェアプロセスに関する研究は様々提案されている．そこで本研究と関係のある研究について述べる．まず，プロセスモデリングについての研究について述べる．Cugola らはソフトウェア開発における追加タスクに着目し，容易にタスクを追加することのできるプロセスモデル言語を提案している [14]．また，Garcia らは仕様変更等によって変更されるプロセスモデルの保守性や変更の容易性などを，GQM ベースのメトリクスで評価している [22]．更に，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 [21]．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本研究にて提案する後から追加されるプロセスの概念と類似するが，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．本研究は，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．

次にプロセス改善に関する研究について述べる．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 [58]．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されている [34]．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，本研究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．

一方，オブジェクト指向に基づいた開発の経験や専門知識を取り入れ，複数のプロセスを統合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 [37]．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 [41]．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提案する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本研究と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．

また，本研究はプロセスからプロダクト品質の関係を明確化する研究である．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，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 [2]．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，本研究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．

第 3 章

プロセスの複雑さの提案

3.1. 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 [55]．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 [15, 35, 47]，開発現場では顧客の満足度を向上させるために細かな計画変更や追加作業に対応している．開発現場での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プロセス全体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，ソフトウェア開発プロジェクトにおけるプロセス全体の複雑さを定量的に計測する尺度を本研究では提案する．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 [59] 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．

3.2. フラグメントプロセスの概念とプロセスの定義

3.2.1. 本研究のプロセスの定義

本研究では計画されたプロセス，フラグメントプロセスの 2 つを提案することでプロセスの複雑さの計測を行う．本節ではプロセスの定義を行う．

プロセス

本研究で提案するプロセスの複雑さとは，プロセスの実行時の複雑さを計測して得られる値の尺度である．本研究で扱う「プロセス」は SLCP(Software Life Cycle Process) の定義に準ずる．プロセスには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，本研究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．本定義の「プロセス」は「プロセスインスタンス（作業実行系列)」の意味を含むが，本稿では全て「プロセス」と統一する．また，ソフトウェア開発プロジェクトにて本定義である「プロセス」を全て集合したものを「全体のプロセス（プロセス全体)」と定義する．

プロセスモデル

ウォータフォールプロセスやインクリメンタルプロセス等を本研究ではプロセスモデルとして扱う．本研究のプロセスモデルの複雑さを計測するものではなく，プロセスの並行実行時における複雑さを計測する事が本研究の提案である．

プロセスの複雑さ

提案するプロセスの複雑さはソフトウェア開発プロジェクトにおけるプロセスの実行時の複雑さである．つまり，特定時刻に並行実行されるプロセスの複雑さを定量的数値とする．本研究ではプロセスの複雑さは後から追加される作業（後の述べるフラグメントプロセス）に着目し計測される．また，プロジェクト開始時から終了時のプロセスの複雑さの値が増加する様子を「プロセスの複雑化」となる

3.2.2. フラグメントプロセス

ソフトウェア開発プロジェクトにおいて計画工程で予定した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プロセスは複雑化する．また，インクリメンタル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というプロセスを並列実行する場合がある．

本研究では，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列実行される様からフラグメントプロセスと命名した．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，フラグメントプロセスの発生要因は顧客の要望，仕様変更だけでは無く，他システムとの連携による外部要因や開発の遅延等の開発者，及び顧客が予期せず発生したプロセス全てを指す．

3.2.3. 計画されたプロセス

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実行することが多い．本研究では，計画段階にプロセスモデルを基に作成されたプロセス群を

「計画されたプロセス」と定義する．計画されたプロセスは多数のプロセスによって構成されるが，本研究では計画されたプロセスで構成されている個々のプロセスを考慮せず，1 つのプロセスとして定義する．これにより，フラグメントプロセスがプロセス全体に与える影響を明確にする．

また，計画されたプロセスを 1 つのプロセスとすることで，プロセスモデルに依存しない尺度となる．つまり，ウォータフォールプロセスやインクリメンタルプロセス等のプロセスモデルの種類に影響しないため，将来に新たなプロセスモデルが提案されても，提案するプロセスの複雑さを利用することができる．

3.2.4. フラグメントプロセスの事例

過去の事例の一部を用いたフラグメントプロセスの例を図 3.1 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，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）[49] 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 3.1 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．

3.3. プロセスの複雑さの概念と特徴

プロセスの複雑さはフラグメント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，プロジェクト実施中の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尺度である．本研究では，プロジェクト当初に計

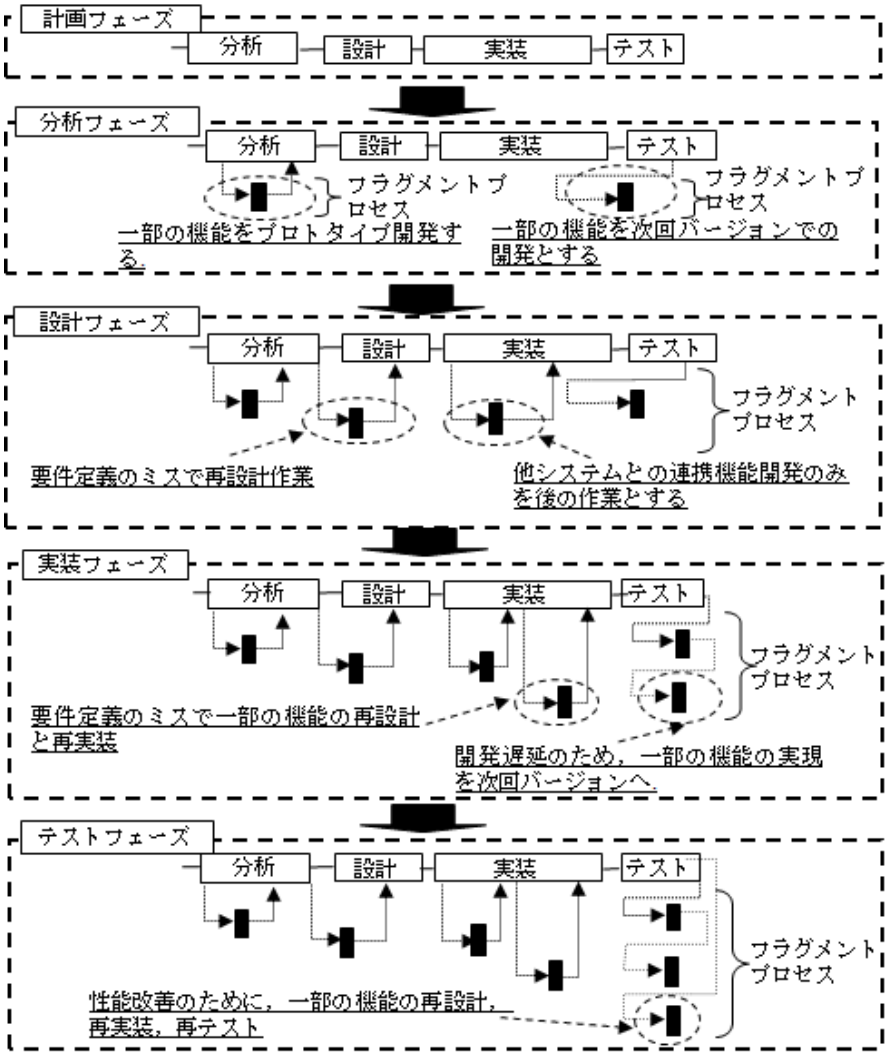


図 3.1　フラグメントプロセスの例

画されたプロセスに対してフラグメントプロセスを抽出し，計測することでプロセスの複雑さとしている．以降，プロセスの複雑の概念と特徴について述べる．

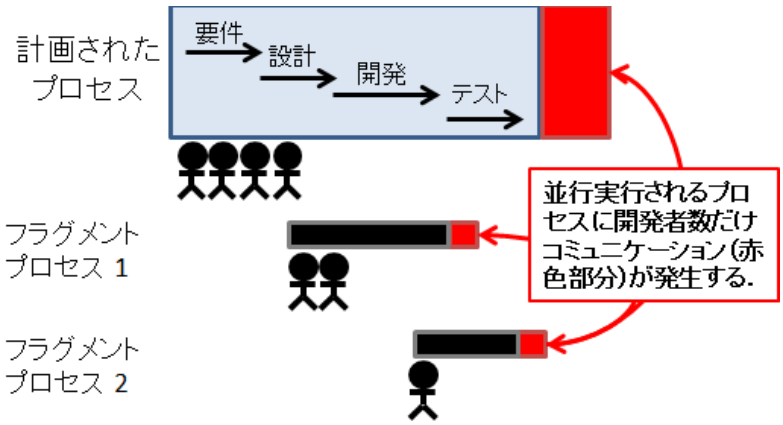


図 3.2　プロセスの複雑さの概念図

3.3.1.　プロセスの複雑さの概念

まず，本研究で提案するプロセスの複雑さの概念を簡単に示す．本研究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がプロセス全体に与える影響の大きさ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．しかし，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に要する労力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考慮不足なことが多い．

本研究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

スに対し，追加されたプロセス，すなわちフラグメントプロセスがプロセス全体に与える影響に着目し，それを定量化するものである．図 3.2 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 3.2 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青色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，模式的に赤色で表記）．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研究におけるプロセスの複雑さである．

3.3.2. 計画されたプロセスとプロセスモデルの関係

本提案は計画されたプロセスを 1 つのプロセスとすることで，計画されたプロセス自身の複雑さに依存しない尺度とする．この理由として，計画されたプロセスはウォータフォールモデルなどの様々なプロセスモデルを利用する可能性があるため，プロセスモデルとその実行構成である個別のプロセス群を本研究では議論の対象外とするためである．

計画されたプロセスを計測の対象外とすることで，フラグメントプロセスがプロセス全体に与える影響だけを計測することに焦点を絞ることができる．つまり，計画されたプロセスで想定外の事象を全てフラグメントプロセスとする．例えば，ウォータフォールモデルで構成した計画されたプロセスでは，テスト段階での仕様変更は想定外であり，プロセスモデルには含まれていない．しかし，仕様変更があった場合，その想定外の作業をフラグメントプロセスとすることで，フラグメントプロセスとプロセス全体との関係を明確にする尺度である．

さらに，仕様変更などのフラグメントプロセスが許容されるプロセスモデルにも適用可能である．つまり，フラグメントプロセスを許容するプロセスモデルが想定していない事象をフラグメントプロセスとして抽出することで提案するプロセスの複雑さを適用することが可能である．例えば，急な開発メンバの交代などで発生す

る引き継ぎ作業がフラグメントプロセスとなる．

このように，プロセスモデルを基に構成された計画されたプロセスを 1 つのプロセスとすることで，プロセスモデルが想定していない全ての事象をフラグメントプロセスとすることができる．したがって，本提案はプロセスモデルやフラグメントプロセスの内容に影響されない汎用性の高い尺度である

3.3.3. プロセスの複雑さのパラメータ抽出の方針

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 [7, 16, 66] の多くの提案が前提としている CMM レベル 1 の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られる．「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．

3.3.4. プロセスの複雑さの汎用性について

本研究の目的はフラグメントプロセスによるプロダクト品質への影響を明確にすることである．そのためにプロセスを示す尺度の 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，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．

3.3.5. 本研究の貢献

本研究は産業界における実際のソフトウェア開発に貢献するものである．近年の産業界におけるソフトウェア開発はコスト制約に厳しく，多大な労力を費やし工数の削減を実施している．計画に無い作業の実施は十分な作業実施時間を確保できず，ソフトウェア品質の低下を招く要因となる可能性が高い．そのため，追加要望や仕様変更等の開発は避けたい．しかし，顧客からの追加要望や仕様変更を容認しなければ顧客満足度の高いシステムが開発できない．

追加要望を容認したとしても，限られた工数内で開発を実施することになる．結果，時間的に無理な開発によって品質の低いソフトウェアとなる可能性が高くなり，顧客満足度が下がる．この解決策として，計画されたプロセスを実行せず，追加要望を先に実行する事も考えられる．しかし，産業界のプロジェクトは納期が厳密に指定されているため，計画されたプロセスの実行期間が予定より短くなってしまい，結果的に開発遅延による新たな追加プロセスが発生する．

現在の産業界における追加要望等の予期せぬ作業の問題は予期せぬ作業に対し，その実行判断基準が無い事にある．本提案はこの問題点に着目し，プロセスの複雑さを提案している．つまり，予期せぬ作業がプロセス全体に与える影響を定量化することでプロダクト品質を予測し，予期せぬ作業の実行判断基準の助けとなるための研究である．

提案する尺度とプロダクト品質に関係があるならば，先に述べた問題点の解決方法の１つとなる．例えば，ソフトウェア開発中に追加要望が発生した時点でプロセスの複雑を計測し，過去のプロジェクトのプロセスの複雑さからソフトウェア品質の比較を行う．比較することで，進行しているプロジェクトがどの程度のソフトウェア品質となるかを過去のデータより予測する事が早期に可能である．

プロセスの複雑さを利用することで，追加要望の実施時期の再検討や，計画されたプロセス自身の変更等，プロセスの複雑さの制御によるプロセス改善を早期に実施することが可能となる．または，ソフトウェア開発遅延によるプロセスの増加の場

合は早期に人員増加する根拠の 1 つとなる．このような例は実際の開発現場で開発者の経験あるいは顧客の経験と勘で実施されている．この点についての詳細な議論は 5.8 節で述べる．

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.

$$PC_{(t)} = \sum_{i=1}^{N_{(t)}} ((Num\_dev_{(t)_i} * term_{(t)_i}) * Concurrent_{(t)_i}) \tag{3.1}$$

$$Concurrent_{(t)_i} = 1 + L_{(t)_i} * m_{(t)_i} \tag{3.2}$$

$PC_{(t)}$: 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で表記される．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計測する事が提案する PC 値である．また，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 [20]．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 [24, 61]．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は困難なため，本研究では $m_{(t)_i}$=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. プロセスの複雑さ計測式と妥当性

提案するプロセスの複雑さは工数（開発者数×開発期間）に並行プロセス数を重みとして乗算している．つまり，従来のプロセスメトリクスである工数では考慮不足である並行プロセスを考慮した新たな工数を示す尺度である．本研究では，この「新

たな工数」をプロセス実行時の複雑さとしている．フラグメントプロセスの並行実行による工数見積もりをより正確に計測した値が提案するプロセスの複雑さとなる．

工数はソフトウェア開発プロジェクトにおいての作業量を計測する際のもっとも一般的な指標である．工数にフラグメントプロセスの並行実行プロセス数を重みとして乗算することで，工数にプロセス間のコミュニケーションの量を追加し，より正確な工数を求めている．

フラグメントプロセスの並行プロセス数の重みとは，フラグメントプロセス自身の工数と同時実行プロセス数 $L_{(t)_i}$，同時実行プロセス間の関連の重み $m_{(t)_i}$ から求める．同時実行プロセス数が増えるとコミュニケーション量の増加が予想でき，さらにプロセス間の関連が強ければコミュニケーション量はさらに増加する．

また，並行プロセス数の重みを計測する際に利用される $m_{(t)_i}$ は他のプロセス間の依存関係によるコミュニケーション量を示しており，関係が強ければ $m_{(t)_i}$ の値が大きくなり，依存関係が弱ければ値が小さくなる．例えば，システム運用の説明というフラグメントプロセスの場合，仕様書の変更や，設計書の変更が無いため，他のプロセスへの影響力が小さいため，$m_{(t)_i}$ の値が小さくなる．逆に，仕様変更等の様々なプロダクト（仕様書や設計書）に影響を与えるフラグメントプロセスの場合は他のプロセス間の依存関係が大きいため，$m_{(t)_i}$ の値が大きくなる．しかし，このようなコミュニケーション量の定量的数値の抽出は困難であるため，本研究では最大値の 1 としている．

フラグメントプロセスの並行プロセス数の重みを工数に乗算して正確な工数を求めるので，提案する式が妥当と考える．ただし，提案する式は 3.3.3 項でも述べているように汎用性を高めるために厳密な値は計測できない．しかし，プロセス全体の複雑さを定量的に計測し，さらにプロジェクト間で比較することによって産業界での有効性を示すことができる．

3.6. 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

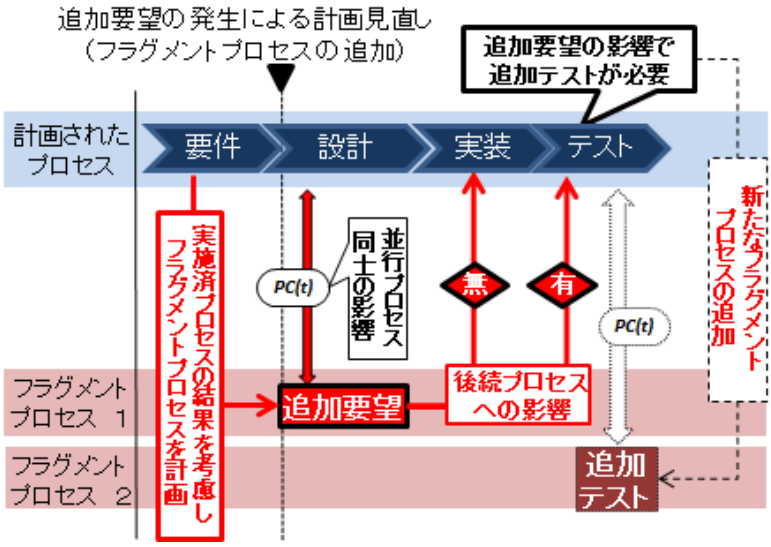


図 3.3　フラグメントプロセスと他のプロセスとの相互作用

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 3.3 の例を用いて述べる．

図 3.3 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」の影響をうけて作成されたものである．すなわち，プロジェクト管理者が無意味に「追加要望」の実施期間や開発者数等を決定するのではなく，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．

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．

3.7. 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 値の測定に関して，汎用性を高めるための制約と PC 値で計測できる限界について述べる．

3.7.1. 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，管理項目をプロセス，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．

3.7.2. 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 [24, 26]．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研究の 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，本提案はフラグメント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．

3.8. 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 3.4 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

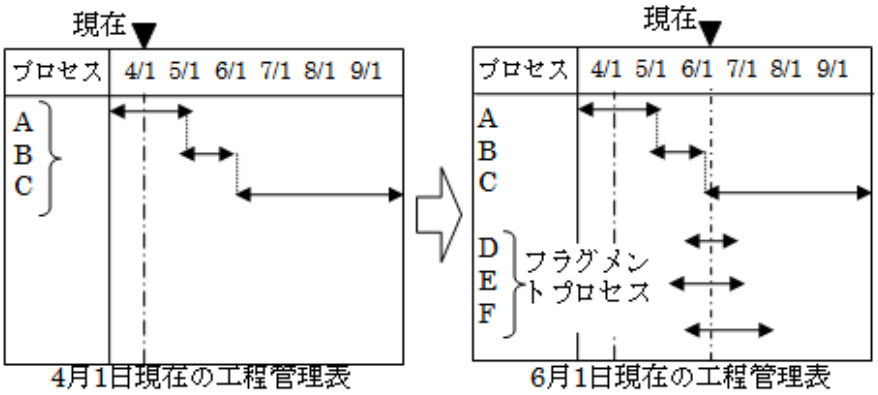


図 3.4　工程管理表からのフラグメントプロセス抽出例

示す．表の横軸は日付，縦軸はプロセスを示す．図 3.4 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 3.4 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（システムエンジニア），PG（プログラマ），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 3.4 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 3.4 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 3.4 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 3.4 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 3.4 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 3.4 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 3.4 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，計画されたプロセスと 3 つのフラグメントプロセスによるプロセス複雑さは，$PC_{(60)}$=((3 × 1) × (1 + 3 × 1)) + ((1 × 0.166) × (1 + 3 × 1)) + ((1 × 0.25) × (1 + 3 × 1)) + ((1 × 0.333) × (1 + 3 × 1)) = 12.000+0.664+1.000+1.332=14.996 となる.

第 4 章

適用プロジェクト p-HInT システムの開発

4.1. はじめに

本章ではプロセスの複雑さを適用するプロジェクトで開発した p-HInT（portable Hannan Internet Community Tool）システムについて述べる．実際に開発，運用に参加することで，開発時の詳細なデータを収集した. これによって，プロセスの複雑さの提案の基となる基礎データが明らかになった．本章では，基礎データを収集したプロジェクトとそのシステム内容を詳細に紹介することで，プロセスの複雑さの着想に至った経緯を明確にする．同時に次章で紹介する検証の理解を助けることを本章の目的とする．p-HInT システムは大人数講義（200 人以上の講義）の授業改善を目的とした授業支援システムである [25, 27, 28, 29, 30, 53].

p-HInT システムは学生がニンテンドー DS$^{(R)}$（以下 DS）や Play Station Portable$^{(R)}$（以下 PSP）等の携帯情報端末を授業に持参し，小テストや出席等に利用するシステムである．2007 年 10 月から開発を開始し，2008 年 4 月から第 1 バージョンの運用を始めた．主な機能を以下に示す.

- 着席順学生一覧（学生名特定）
- 小テスト・ドリル機能（瞬時集計機能付）
- 出席確認機能（CSV 保存）

- 席替え機能（座席指定）
- みんなのこえ機能（学生から教員へのメッセージ）
- レッドカード機能（教員から特定学生へのメッセージ送信）

p-HInT システムは汎用性の高い Web アプリケーションであり，通信手法として無線 LAN（IEEE802.11）を採用している．従来の技術で開発されたシステムであるが，DS や PSP 等のゲーム端末で 200 人から 400 人同時アクセス可能なシステムとした．また学生端末を特定機種に限定にしないことにより，システムの汎用性を高める狙いがある．普及率を考慮すると携帯電話を学生端末とすることが最良だが，携帯電話はパケット料金の発生や通信量の制限等の問題がある．そのため DS や PSP 等のゲーム機をクライアントの主体とした．

本システムは 2008 年 4 月から運用を開始し，2009 年 11 月時点で，延べ 28 科目，34 クラス，受講者数 3000 人以上の運用実績がある．計画当初の開発形態は 2 年間のインクリメンタル開発プロセスで行う予定であった．しかし，授業改善に必要とする機能を明確にできなかったため，4 回のウォータフォール開発プロセスへと変更された．これにより，リリース後にシステムの問題点を分析し，次期開発へと組み込む事が可能となった．結果，同一科目同一教員の授業で，p-HInT システムを利用したクラスでの成績向上に繋がるシステムを開発することができた．

4.2. p-HInT システムの概要

4.2.1. シンプルなシステム設計と汎用性の高いシステム

学生端末を限定せずに汎用性の高いクライアントを装備することが本システムの特徴であり，本システムを広く普及させることが狙いである．そのため，ブラウザ機能と無線 LAN 技術の基本だけを利用したシンプルな Web アプリケーションとしてシステム設計を行った．シンプルなシステム設計とすることで，DS や PSP 等の低性能端末から iPod touch，スマートフォン，ノートパソコンなどの高機能機種でも動作可能な汎用性の高いシステムとした．ただし，本システムは無線 LAN で自動相互通信を行っているため，無線 LAN 機能が搭載されていない携帯電話ではシステムの一部の機能が動作しない．

4.2.2. 無線 LAN による 400 台以上の安定接続

２つめの設計の特徴は，一般的なブラウザ機能や無線 LAN 技術を用いてゲーム機等の低性能な端末機器で 1 教室 200 台以上，同時接続 400 台以上が安定接続することである．つまり，通信やブラウザの最低限の機能しか組み込まれていない低性能携帯情報端末でも安定した通信ができるシステムを設計した点である．この特徴は，コンピュータ教室で有線にて接続されたパソコンのブラウザベースによる授業支援システムと最も異なる点である．有線接続されたコンピュータや，数十台程度の無線接続された携帯情報端末を対象としたシステム設計ではなく，設計当初から 1 教室で 200 台以上の安定接続を目指したシステム設計である．

4.2.3. ゲーム機を学生端末として設計

本システムは DS や PSP 等のゲーム機を学生端末の主体とした．これは DS や PSP にはブラウザと無線 LAN が搭載 (または販売) されているためである．従来の授業支援システムの多くはパソコン等の高性能機器を対象に設計されている．本来ならば p-HInT システムもノートパソコンや PDA 等の高機能端末を採用し，多彩な機能を導入すべきであった．しかし，高機能なクライアントにするのではなく，敢えてゲーム機でも動作可能な Web アプリケーションとした理由は２つある．

１つめは現在 DS や PSP が第 2 の携帯情報端末として所有している人が多い点である．開発当時（2007 年）は現在のようなスマートフォンはまだ普及しておらず，ブラウザが稼働する端末で最も所持率が高かったのが DS や PSP 等のゲーム機であった．もちろん携帯電話の所有率が最も高いが，パケット料金発生や，機種によって異なる通信量制限によって表示できる画面の固定化や，教員機から学生の携帯電話への自動相互通信ができない等の問題がある．そのため大人数の授業支援システムとして使用するには不向きであると判断した．同様に高機能である PDA やノートパソコンも所有率が低いので不向きと判断した．

２つめはバッテリーの持続性である．近年のノートパソコンのバッテリーは優れているが，それでも 3〜5 時間である．またノートパソコンのバッテリーの持続時間は機種によって大きく異なるため，学生のノートパソコンを特定の機種に限定する

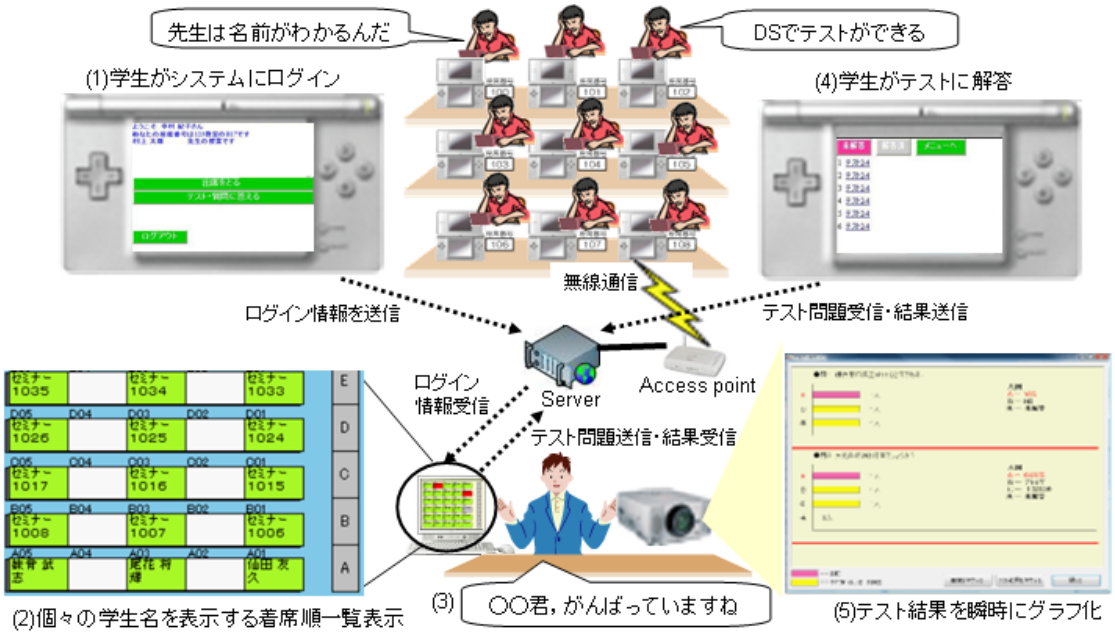


図 4.1　p-HInT システム利用イメージ図

必要があった．それに対して DS は 5〜19 時間（実測で 7 時間程度）のバッテリー持続が可能である．また，乾電池等を用いた簡易充電器が多く存在する点もノートパソコンとは大きく異なる点である．１授業が 90 分で 1 日に複数の授業を受講する学生にとって，バッテリー持続時間の長いゲーム機が最適と判断した．

さらに，前述したように低性能なゲーム機にシステム設計を合わせることによって利用できる携帯情報端末の範囲が広くなり，システムの汎用性が向上する点もゲーム機を主要クライアントとして選択した理由の 1 つである．

4.2.4.　p-HInT システム利用イメージ

本システムの利用イメージを図 4.1 に示す．学生は一人一台 DS 等の携帯情報端末を授業に持参する．学生は携帯情報端末にユーザ ID とパスワードと座席番号を入力し，p-HInT システムにログインする (図 4.1 の (1))．同時に教員機の着席順学生一覧上に着席位置にログインした学生の氏名が表示される (図 4.1 の (2))．これにより，匿名性の低い授業となり，授業中「○○君よくがんばっていますね」等の名前を

特定した授業進行ができる (図 4.1 の (3))．逆に「○○君静かにしなさい」等の特定学生を注意する事ができ，学生の授業に対する緊張感を持続させることができる．本システムの利用イメージを図 4.1 に示す．学生は一人一台 DS 等の携帯情報端末を授業に持参する．学生は携帯情報端末にユーザ ID とパスワードと座席番号を入力し，p-HInT システムにログインする (図 4.1 の (1))．同時に教員機の着席順学生一覧上に着席位置にログインした学生の氏名が表示される (図 4.1 の (2))．これにより，匿名性の低い授業となり，授業中「○○君よくがんばっていますね」等の名前を特定した授業進行ができる (図 4.1 の (3))．逆に「○○君静かにしなさい」等の特定学生を注意する事ができ，学生の授業に対する緊張感を持続させることができる．

また，教員は現在ログイン中の学生を出席管理情報として保存することができる．授業中に出席カード等を学生に配布し，出席確認を行う手間を省くことができる．更に，教員が授業中に小テストを実行すると，学生は携帯情報端末上からテストに解答する (図 4.1 の (4))．解答結果が瞬時に集計され，教員画面やプロジェクタに投影して，学生理解度を把握することができる (図 4.1 の (5))．また，他の学生がどの程度理解しているかを知ることで学生間の競争的教育が可能となる [51].

4.2.5. p-HInT システム全体像

p-HInT システムにおけるシステム構成図（ネットワーク図）を図 4.2 に示す．メインとなる Web サーバには Microsoft 社の IIS(Internet Information Service) を用いており（図 4.2 の左上点線部分），データベースサーバには Microsoft SQL Server 2005 を用いている．

本システムのソフトウェアは，Web サーバとデータベースサーバを含むサーバソフトウェアと，教員が操作する教員機クライアントソフトウェア（アプリケーション），学生が操作する学生機クライアントソフトウェア（Web アプリケーション）から構成される．教員機ソフトウェアは VB.NET で実装されており，小テストの問題等のデータをデータベースサーバへアップロード，学生の解答データをデータベースからダウンロードする．つまり，p-HInT システムのサーバへアクセスし，データベース上のデータを教員機上に GUI 表示・更新するソフトウェアである．学生機ソフトウェアは ASP.NET と VB.NET を組み合わせた Web アプリケーションであり，携帯情報端末のブラウザに小テスト問題等を表示し解答データをサーバへ送信

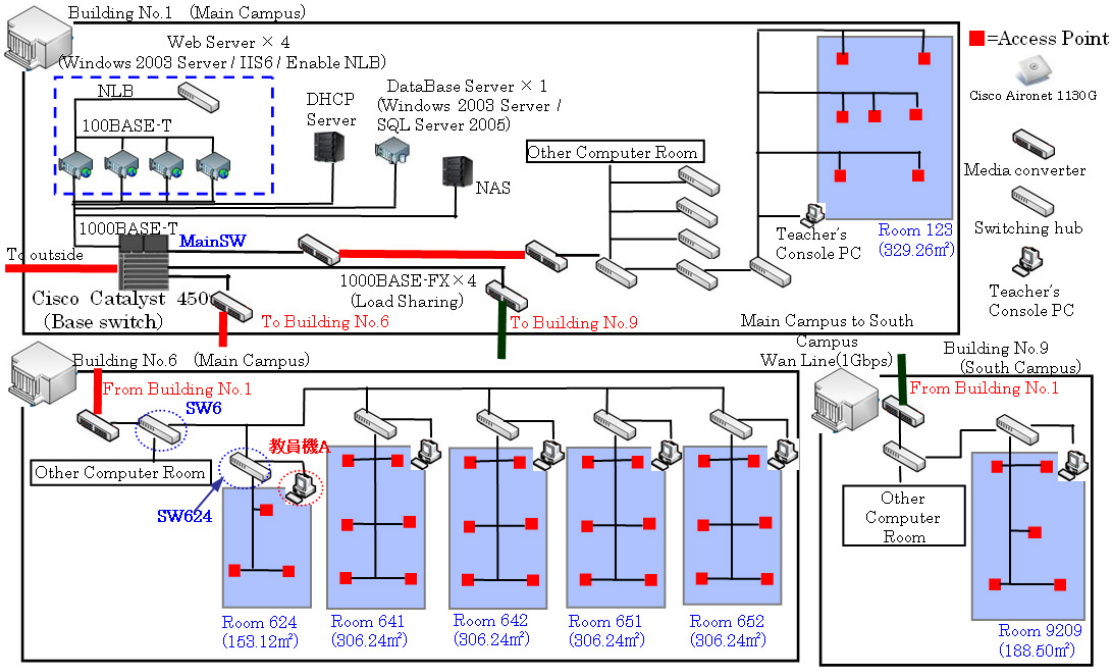


図 4.2　p-HInT の基本システム設計

する．さらに出席状況や座席番号，みんなのこえの履歴をサーバよりブラウザ上に表示する．

4.2.6.　p-HInT システム機能詳細

p-HInT システムには授業改善を目的とした大きく 6 つの機能が搭載されている．教員機上に表示される p-HInT の各機能を図 4.3 に示す．

着席順学生一覧機能

学生が DS で座席番号と ID，パスワードを入力し p-HInT システムにログインすると，図 4.3 のように教員機の着席順学生一覧の着席位置に学生名が表示される．座席を左クリックする事で学籍番号やレッドカード機能の警告回数を参照できる．また，p-HInT にログインして正常に受講している学生は着席順学生一覧の座席が緑色で表示，ゲーム等で遊んでいる学生は赤色で表示され，一目で学生の行動が把握で

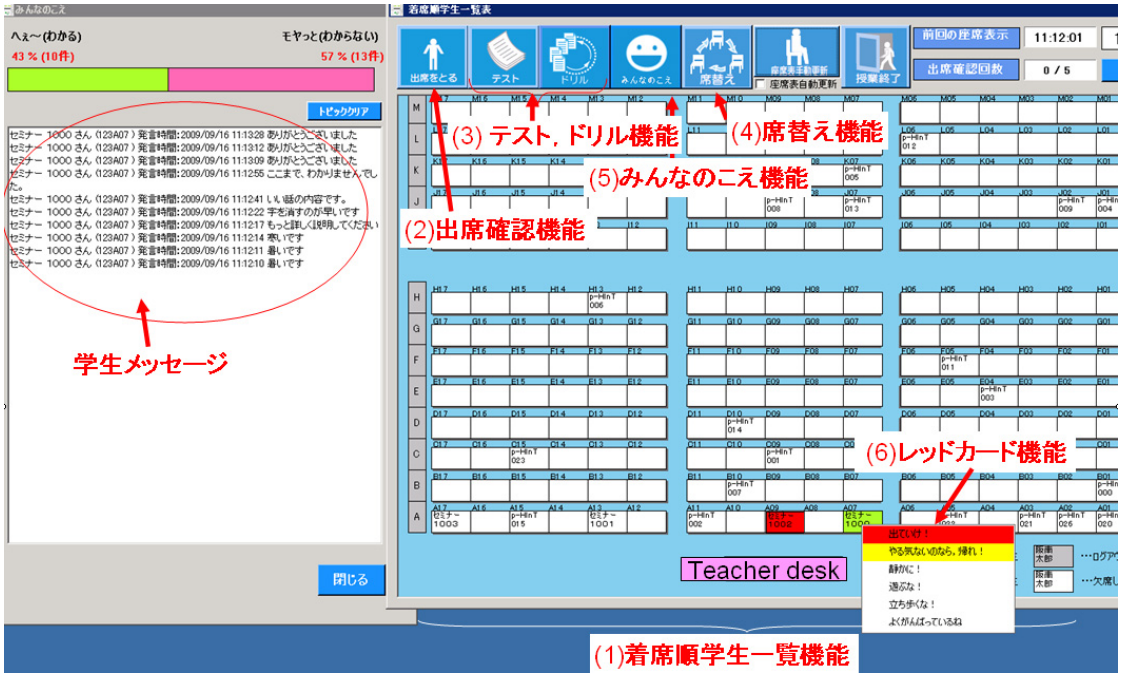


図 4.3　教員機の p-HInT メイン画面イメージ

きる．

出席管理機能

出席カード等の紙ベースで実施していた出席管理を DS で瞬時に出席を集計する．２つの出席確認方法があり，１つめが教員が授業中に口頭で指示した合言葉を入力する出席確認である．２つめが着席順学生一覧で出席確認を行う方法である．これは着席順学生一覧で緑色の座席（通常出席）で表示されている学生だけが出席扱いになる．

テスト機能

大人数の学生に問題を出題，解答し，瞬時に集計する機能である．教員機のテスト出題画面イメージは図 4.4 の左部分に示し，学生 DS 上のテスト回答画面を図 4.4 の右部分に示す．テストは選択式，複数選択，テキスト解答，自由記述等の問題が出題可能である．

席替え機能

授業中の私語対策として教員から多くの要望があり，実装された機能である．教員機から座席を指定すると，学生画面に「あなたの座席番号は○○です」と表示され，学生は指定された座席に着席する．座席指定は学籍番号順，ランダム順，一列空け，前詰めなどが細かく指定できる．学生の座席位置を指定して私語相手の友達と離すことで私語を削減する目的の機能である．

みんなのこえ機能

学生から教員に自由なタイミングで発言する機能であり，現在の Twitter のような機能でもある．大人数の授業において学生が教員にリアルタイムで発言することは難しい．例えば「エアコンが寒い」「先生の声が小さい」等を 200 人もの学生の前で発言することは授業の進行を妨げとなり学生としては気が進まない．このような発言を DS から教員へ発言し，教員はチャットルームのような画面で学生の発言を参照する機能である（図 4.3 の学生メッセージ参照）．更に授業のどこに学生は興味を持っているかを把握するための「へぇ〜」ボタン，「モヤっと」ボタン機能も追加した．これにより，学生は教員が授業進行している最中に「へぇ〜」ボタンや「モヤっと」ボタンを自由なタイミングで押すと教員画面にそれぞれのボタンの押下回数累計グラフがリアルタイムで表示される．

レッドカード機能

着席順学生一覧の学生名を右クリックする事で個々の学生にメッセージを送信する機能である（図 4.3 のレッドカード機能参照）．これは授業中に特定の学生が騒ぎ，それを注意することで授業全体の雰囲気を悪くしてしまうことを避けるため，特定の学生だけを注意する機能である．学生の DS 上に図 4.5 のような警告メッセージが表示される．また授業をまじめに聞いている学生等には「よくがんばっているね」等の学生のやる気が向上するメッセージも送信できる．

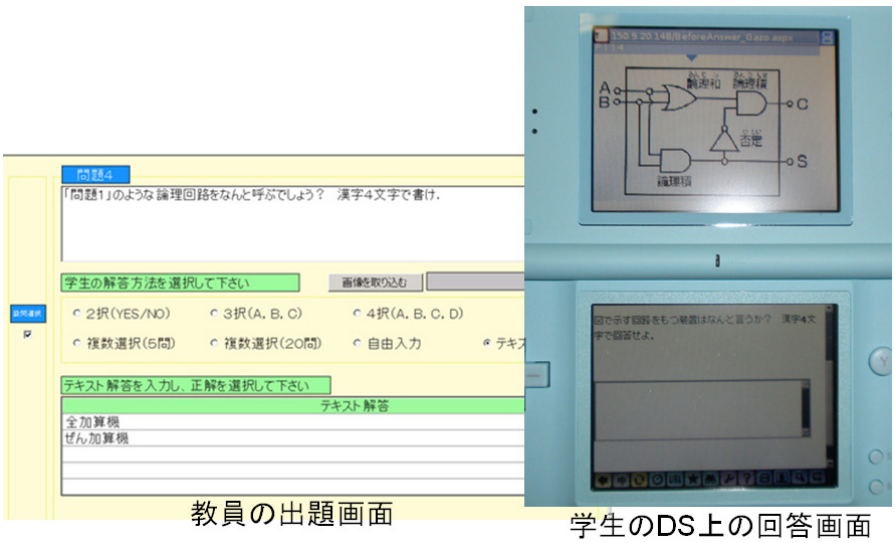


図 4.4　教員機の出題画面と学生 DS 上のテスト解答画面

4.3.　p-HInT システム開発における試行錯誤

4.3.1.　信頼性向上の実績

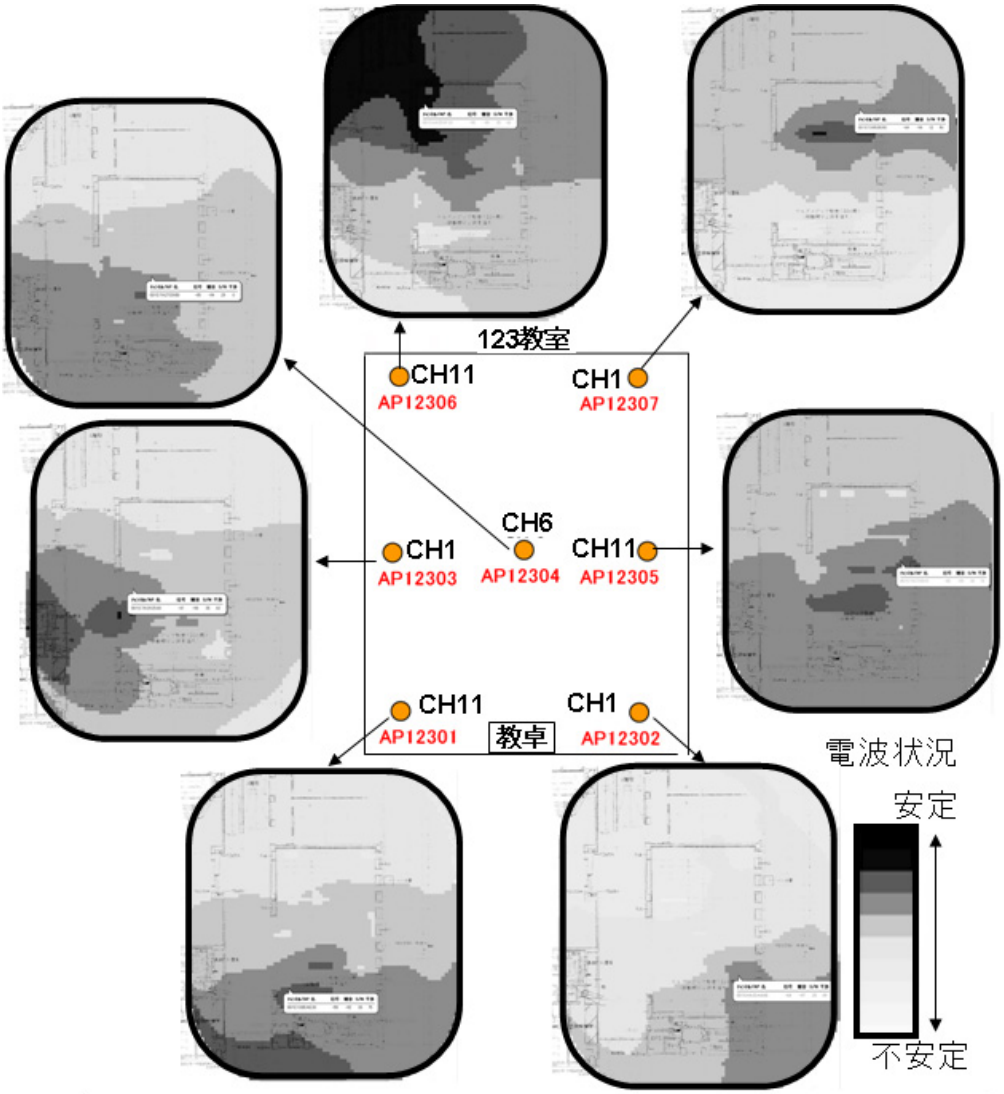
p-HInT システムは 400 台以上の低性能端末をクライアントと想定したシステムである．通常のコンピュータより低性能・低機能な端末にて安定接続・稼働を実現することは難しい．2008 年度前期の第 1 バージョンでは障害件数をカウントする体制が整っておらず正確な件数は不明だが，授業を中断する障害が頻発した．特に 2008 年度前期学期末の成績を判定するテストの実施中に障害が発生してテスト自身が無効となる大きなトラブルも発生した．2008 年度後期の第 2 バージョンでは 1 授業あたり 5.1 回の障害（全てのエラーを含む）が発生した．その後，リファクタリング等の改善を進めた結果，2009 年度後期（第 4 バージョン稼働時）には 1 授業あたりの障害発生が 0.36 件となり，システムの信頼性を向上することができた．p-HInT システムの安定接続・稼働するためのシステム設計と運用の工夫を以降で紹介する．

図 4.5　レッドカード機能による DS 上の警告

4.3.2.　教室全体をカバーするための AP 配置

ホテルや駅などの広いエリアを少ないアクセスポイント (以下 AP) で効率的に配置する方法の研究は盛んに行われている [18, 65]．しかし，本システムは教室の狭いエリアに複数の AP を配置する必要がある．本システムで採用した Cisco Aironet 1130G は最大接続台数は 255 であるが，推奨台数は 50 である．したがって，本学の最大収容人数 306 人の教室では最低でも 7 台の AP を配置する必要があった．しかし，狭いエリアに多くの AP を配置する研究やアルゴリズムが存在せず，また実践報告もなかった．そこで，性能テストを繰り返しながらトライアンドエラーで AP 配置を行った．

まず，複数の AP が狭いエリアに存在するので電波干渉の問題がある．利用できるチャンネルは 1，6，11 の 3 つのチャネルであり，4 台以上を設置する場合は電波干渉の問題が発生する．そこで AP の電波出力を最弱にし，同じチャネルの AP 間の距離を最大になるよう教室に配置した．図 4.6 に最大収容人数 306 人，教室面積が 329.26m^2，7 つの AP を配置した時の電波状況を示す．図 4.6 では 1 つの AP が教室全体にどのぐらい電波を網羅しているかを示している．このように教室全体



AP1 台あたりの最大接続数			最大接続台数
CH1	CH6	CH11	
40	40	50	
40		50	310
40		50	

図 4.6　123 教室における AP 配置

に電波が行きわたる状況を,「同じチャネルの AP 間距離を最大」と「電波出力を最弱」の方針を基に各教室で電波状況を計測しながらトライアンドエラーで配置した.

4.3.3. 低性能機器を考慮したシステム設計

p-HInT システムは学生の所持率が高い DS や PSP をクライント機器として推奨している．また，無線 LAN 機能，ブラウザが搭載された様々な端末も利用可能である．しかし，機器によって性能が大きく異なる．例えば DS は詳細不明の IEEE802.11 に類似した通信規格に対し，ニンテンドー DSi は IEEE802.11g である．p-HInT システムはあらゆる Web ブラウザ搭載機器をクライアントにできる汎用性の高いシステムを目指しているので，最も低性能である DS（ブラウザも含む）をターゲットにシステム設計をおこなった．DS の性能制約を以下に示す.

- 通信が遅い
- JavaScript 等に機能の制約がある
- ローミング機能が不安定
- クッキーが不安定

低速通信に対する工夫

DS は通信速度は実測で約 1Mbps 以下であった．また，DS のブラウザは Flash に対応しておらず，JavaScript の機能にも様々な制限があった．例えば，アラート表示は可能であるが，ページの自動更新機能は動作しない等である．さらに，HTML のタグも「b」タグや「h2」タグ等は利用できない．実際にプログラムを作成し，DS 上で調査することで得られた情報である.

しかし，最も困難な点は極端な通信速度の遅さのため発生する操作性の悪さであった．例えば，ラジオボタンを選択後にスクロールして送信ボタンを押したつもりが，キャンセルボタンを押したことになるケースである．これは ASP.NET のポストバックと呼ばれる機能が原因であり，ラジオボタン選択操作だけで IIS サーバへ問合せが発生する．通信が遅いため，問合せ結果のページをサーバから受け取る前にユーザのスクロール操作が発生する．その後，サーバからページを受け取りブラウザ上に表示するので，スクロール以前の画面でクリックしたことになり，ユーザが押

したボタンとは異なるボタンを押下したこととなる．これらの現象はノートパソコン等の高性能の端末ではほとんど発生しない．

そこで，ポストバック処理の細分化を行った．ポストバックではセッション管理に必要な情報をサーバへ渡す処理も含まれるため，全てのポストバックを無効にする事はできない．そこで，サーバへ問合せる必要のないポストバック機能をコントロールごとに細分化して無効に設定した．p-HInT システムで無効にしたコントロールは小テストのラジオボタン，チェックボックス，みんなのこえ機能のプルダウンリストである．ポストバック機能はサーバとの通信が頻繁に発生するが，ASP.NET のプログラミングの容易性のために装備された機能であり，主要コントロールにはデフォルトで有効とされている．本件は DS のような低性能端末でなければ起こらない低性能端末独自の問題であった．

低性能機器を考慮した AP チューニング

p-HInT システムは 1 つの教室に電波干渉を考慮して複数台の AP 配置を行っている．これによりノートパソコン等の IEEE802.11g 等の高性能無線 LAN が搭載されている機器は安定通信した．しかし DS 等の低性能無線 LAN では安定接続が困難であった．DS は通信に関する詳しいリファレンス等が提供されておらず, IEEE802.11 に類似しているが，正確には不明の通信規格であった．特徴を調査した結果，DS は通信速度が遅く（実測 1Mbps 以下），かつ，ローミング機能が不安定であった．そのため電波が届く AP のうち，チャンネルの値が小さい AP へ優先的に接続した．本来電波が最も強い AP へ接続するはずだが，電波が弱いのにも関わらず値の小さいチャンネルへと接続した．この結果として電波が弱い上に値の小さいチャネルの AP へ負荷が集中する相乗作用にて通信エラーが続出した．

そこで AP のチューニングを実施した．p-HInT システムで利用している AP は最大接続台数が 255 台の業務用の Cisco Aironet 1130G である．開発当初は AP へのアクセス台数制限を掛けなかった．そのため DS のローミング機能が動作せず，値の小さいチャネルの AP へアクセスが集中してしまい，結果 AP の過負荷となり通信エラーが発生した．この時の過負荷となった AP の接続台数はおよそ 90 であった．

チューニング内容としては，AP への最大接続台数に制限をかけることで AP への過負荷を回避した．設定例を図 4.6 に示す．図 4.6 のアクセスポイント設定ではチャ

ンネル (以下 CH)1 の AP には最大接続を 40 台とし，CH6 と CH11 の AP には 50 の最大接続台数とした．つまり，CH1 の AP に接続しようとしても上限の 40 に到達している場合，その次の CH6 へ接続を試みる．CH6 を教室の中央に配置したので教室のどの位置からも比較的に接続しやすい．しかし，CH6 も上限を 50 台としていると，結果としてチャネルの最も大きい CH11 へ接続する．このように AP への接続を分散させ負荷の均等を配慮した．ただし，この方法では遠い位置の AP に接続してしまい，電波が弱い状況になる可能性がある．しかし，電波が弱い問題よりも AP への過負荷を避けることを優先する必要があると考えたチューニング方法である．

機能制限の中でのセッション管理

Web アプリケーションでセッション管理をする場合，セッション情報をクライアントのクッキーに保持させる方法が一般的である．しかし，DS はクッキーを安定して保持することができず，DS 側でのセッション管理が困難であった．そこでセッション情報をサーバ側に全て保持して，DS とのセッション管理を行う必要があった．

クライアント側のクッキーを使用せずにセッション管理する方法は多く提案されているが，p-HInT システムでは IP アドレスを使ったセッション管理を採用した．セッション ID を IP アドレスと関連付けた情報をサーバで保持し，接続が切断されたとしてもサーバが IP アドレスからセッション情報を取得して，セッションが継続できる手法である．この手法による弊害としては同一機器から異なるログイン ID でログインした場合，どちらかのログイン ID が優先されてしまい，両者を識別できないことである．しかし，DS 等の多くの携帯情報端末は複数のブラウザを開くことができない点や，１人１台の端末を持参することが前提であり，また，出席等の代返を防止するために本手法を採用した．

さらに本手法を IIS 上で実施する弊害として，サーバのメモリの逼迫することである．これは，IIS の上位の ASP.NET フレームワーク中で生成されるワーカプロセスにログイン情報，操作ログ，プログラム中の変数の値，ページ遷移情報等すべての情報を蓄積するためである．結果，1 日の 200 人規模の授業が連続で 5 コマ続く場合，最後の授業で学生がログインできない障害が発生した．メモリが不足して新しいログイン情報を保持する領域が確保できなかったためである．

そこで授業時間帯外でワーカプロセスを自動再起動させた．つまり，IIS サーバを

再起動せずワーカプロセスの再起動だけでメモリをクリアする運用である．ワーカプロセスの再起動が発生するとログイン情報も消去されるので再ログインが必要になるが，授業中ではないので学生のログイン数も少ないと判断した．また，IIS サーバは稼働しているので p-HInT 全体が停止するわけではなく，授業準備中の教員への影響が少ない方式である．これによってサーバ機のメモリを逼迫する問題を解決した．

4.3.4. 400 人同時アクセスのためのデータベースチューニング

データベースは Microsoft SQL Server 2005（以下 SQL Server）を利用し，ストアドプロシージャでデータベースにアクセスする方法を採用した．システム設計時には複数データベースサーバによる負荷分散の必要性を検討した．400 人同時アクセスをスムーズに行うためには複数データベースサーバで負荷分散した方が好ましい場合がある．しかし，実際の授業中に小テストを実施する時には学生は同じ小テストテーブルの同じ問題データを参照する場合が多い．この場合，SQL Server の SELECT 文のキャッシュ機能を使うことで，一人目の問い合わせ結果をキャッシュに保持し，他の学生の同じ問い合わせはキャッシュデータを利用することができる．これによってデータベースの検索アクセス回数を大幅に削減できる．またネットワーク負荷に関しても，DS の通信速度が実測で 1Mbps 以下であるため，400 人が同時に同じ小テストを実施してもアクセスが分散すると考えた．さらに，更新系データベースを複数個利用すると 2 つのデータベースの同期を取るための処理が必要となりデータベースの負荷が増える．これらを考慮した上で，1 台のデータベースサーバを利用するシステム設計とした．

さらに，一連のまとまったデータベースアクセス処理をデータベースへ登録するストアドプロシージャ方式を採用した．クライアントプログラム側は 1 つの関数をコールするだけで一連のまとまった処理がデータベースサーバ側で実行され，結果だけをクライアントへ送信する方法である．DS の通信速度が遅いことを考慮して，ストアドプロシージャを最大限に利用したデータベース設計とした．

上記のデータベース設計にて実際に 400 台の DS で性能テストを実施した．その結果，小テスト実施時にデータベースのデッドロックが発生した．ただし，100 台程度の DS での性能テストでは発生しなかった．原因の 1 つはストアドプロシージャ

の開始時にデータベースにロックをかけ，終了時にアンロックをしたことである．通信量を削減するためにプロシージャを大きな処理のかたまりとしたため，一回のロック時間が長くなったからである．

そこで，1 つのプロシージャの中で必要に応じてロック，アンロックを細かく設定する方式へ変更した．また，SELECT 等の参照系命令専用のスナップショットを作成し，更新系命令と分離して参照系命令を実行する方式を採用した．さらに，限りなくデッドロック発生を避けるために，大きなかたまりのプロシージャをいくつかに分割した．分割する際には通信量が増加しないよう，慎重に分割するプロシージャを選択し，分割する処理を検討された．これらの対策の結果，以降の 400 台同時接続の性能テストでも本番の授業でもデッドロックは発生していない．

4.3.5. 教員機ソフトウェアと DS の相互通信

本システムはシンプルな Web アプリケーションソフトウェアアーキテクチャである．ただし，教員機は Web ベースではない単独実行するクライアントソフトウェアである．教員自身のノートパソコン等から教員機ソフトウェアを操作したいという要望のために，教員機を Web ベースのソフトウェアとすることも検討した．しかし，敢えて単独実行するクライアントソフトウェアとした．その理由は，教員機と DS の相互通信の安定稼働のためである．

図 4.3 に示す着席順学生一覧の学生の緑表示（通常出席），赤表示（ゲームで遊ぶなどで通信切断）は，教員機ソフトウェアから DS へ定期的に通信確立のチェックを行い通信状況から判断した結果である．さらに，DS からの返答がない場合，再度確認の通信を複数回送信する．このように 1 教室で 200 台以上の DS に対して頻繁な通信チェックを安定的に行うために教員機ソフトウェアを単独実行のクライアントソフトウェアとした．もし，教員機ソフトウェアを Web ベースとすると，Web サーバからすべての DS への通信チェックを定期的に行う必要があり，ネットワーク全体の負荷とサーバの負荷が増大する．したがって，p-HInT 全体を完全な Web アプリケーションにせずに，教員機ソフトウェアを単独実行のクライアントソフトウェアとすることで安定通信を実現した．

4.4. まとめ

p-HInT システムのような 400 台の低機能・低性能な携帯情報端末をクライアントとした Web アプリケーションの開発は試行錯誤によって実現されている．その試行錯誤をまとめると以下の 4 点となる．

(1) 低性能端末ではクッキーを使わずに，サーバ側で IP アドレスとセッション ID にてセッション管理をする．ただし，クライアントが一人一台の専用端末であることが前提である．同じ端末から 2 人のユーザがログインすることができないという弊害があるが，出席確認時に代返ができないというメリットもある．また，Web サーバを IIS で利用する場合はワーカプロセスの再起動で安定したセッション管理が実現できる．

(2) ローミング機能が不安定な低性能端末を教室のような狭いエリアで数百台レベルの同時接続をする場合，以下のように AP を設定，設置する．

- **(i)** 同時接続 255 台が設定可能な AP でも接続上限は 50〜60 とすること．
- **(ii)** チャネル 1，6，11 の最大接続台数をたとえば，40，50，60 のように，値の小さいチャネルを少ない値で設定すること．
- **(iii)** AP 4 台以上設置する場合，同じチャネルの AP をなるべく距離を離すこと．
- **(iv)** AP の電波範囲を極小にして，1 つの AP で接続できるエリアを小さくすること．

(3) 低性能端末は Flash や JavaScript に制限があり，HTML タグにも制限がある．更に通信速度を考慮したシステム設計を必要とし，些細なサーバアクセスでも削減する必要がある．したがって，ASP.NET を用いた場合はポストバック機能をコントロールごとに有効無効を設定する必要がある．

(4) Microsoft SQL Server をデータベースとして利用し，400 以上の同時アクセスをスムーズに処理する場合，ストアドプロシージャ中のロック・アンロックを細かく設定する．また，クライアントからの参照系アクセスのために SELECT 専用のスナップショットを利用する．

第 5 章

プロセスの複雑さの適用

本章では 4 章で述べた p-HInT プロジェクトとその他の 4 つのプロジェクトにプロセスの複雑さを適用した．ただし，p-HInT プロジェクトは各バージョンを 1 つのプロジェクトとして扱い合計 8 つのプロジェクトとした．適用結果を本章で述べる．

5.1. 適用する 8 つのプロジェクトについて

本提案であるプロセスの複雑さはプロジェクトの詳細が不明でも工程管理表に全てのプロセスとその実行期間，担当する開発者数が記載されていれば計測可能である．そこで，4 章で紹介した p-HInT プロジェクト以外の新たなプロジェクトの 4 つに提案するプロセスの複雑さを適用する．

4 つのプロジェクトは産業界におけるソフトウェア開発組織が実施したプロジェクトである．これら 4 つのプロジェクトは教育系のアプリケーションを専門に開発する組織で開発された．p-HInT プロジェクトと 4 つのプロジェクトの開発期間や，工数を表 5.1 にまとめる．Project A と Project B は大学の全学規模アプリケーションの開発である．プロジェクト A は完全な新規開発であるが，Project B は母体システムのバージョンアップである．また，Project C と Project D は開発期間と工数だけを明記しているが，4 つのプロジェクトの詳細は守秘義務により述べることができない．

本章では，プロジェクトの詳細が不明のプロジェクトにも適用することができ，プロセスの複雑さからプロダクト品質の関係を定量的に計測する．これにより，開発

者のスキル，プロセスモデルの依存関係等の特徴を考慮せずプロセスの複雑さからプロダクトの関係を明確にすることが本章の目的である．

5.2. プロジェクトへ適用する際の前提

本節では提案するプロセスの複雑さを 8 つのプロジェクトに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．障害件数の他にも仕様書の変更回数やプロセスの変更回数を利用することも考えられるが，本提案である PC 値は開発者のスキルの影響によるフラグメントプロセスの発生も考えられる．そのため，今回はプロダクト品質を示すリリース後の障害件数を利用した．

障害レベルはシステムが停止する障害等の致命的な障害をレベル SS，逆にほとんど発生せず，運用に支障が無い障害をレベル C とする 5 段階のレベルからなる．本研究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．

本研究で適用する 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

表 5.1　8 プロジェクトの詳細

プロジェクト名	開発期間	工数 (人日)	システム概要
p-HInT V1	6 ヶ月	1660	クライアント・サーバの授業支援システム
p-HInT V2	6 ヶ月	1895	クライアント・サーバの授業支援システム
p-HInT V3	6 ヶ月	1175	クライアント・サーバの授業支援システム
p-HInT V4	6 ヶ月	610	クライアント・サーバの授業支援システム
Project A	9 ヶ月	1160	教育系 Web アプリケーション
Project B	7 ヶ月	1215	教育系 Web アプリケーション
Project C	7 ヶ月	680	詳細不明
Project D	3 ヶ月	720	詳細不明

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．

5.3. 8 プロジェクトの適用結果

図 5.1 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 p-HInT V2 の 49.5 であり，最も低い値は p-HInT V4 の 11.6 である．したがって，p-HInT V2 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，p-HInT V4 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 5.2 に示す．表 5.2 から PC 値が高いプロジェクトは障害レベル 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ピアソンの積率相関係数で分析を行った [45].

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

表 5.2　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	全障害件数	障害管理表による障害レベル				
			SS	S	A	B	C
p-HInT V1	32.8	26	2	3	19	1	1
p-HInT V2	49.5	21	10	6	4	1	0
p-HInT V3	26.3	11	3	3	2	1	2
p-HInT V4	11.6	11	0	0	7	3	1
Project A	29.8	34	4	4	6	9	11
Project B	38.8	32	2	1	6	10	13
Project C	16.5	20	0	0	8	9	3
Project D	40.9	18	12	0	2	0	4

表 5.3　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

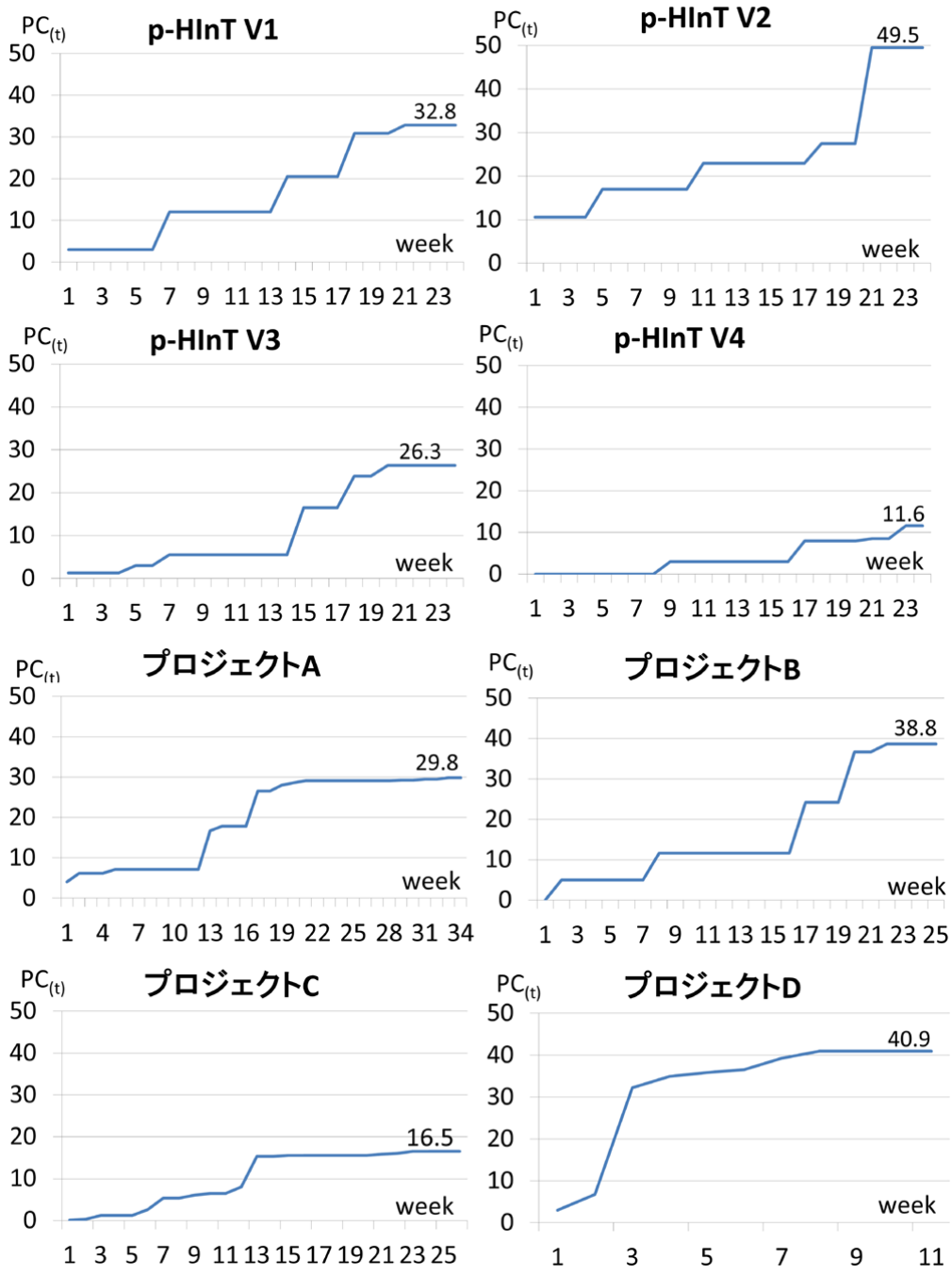


図 5.1　8 プロジェクトの PC 値の推移

障害件数を積率相関係数で分析したものを表 5.3 にまとめた．表 5.3 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．

5.4. 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 と p-HInT プロジェクト V2,V3 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 5.2 に示す．

図 5.2 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赤色の四角で示した．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 5.2 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 p-HInT V2 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 p-HInT V2 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，p-HInT V3 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．

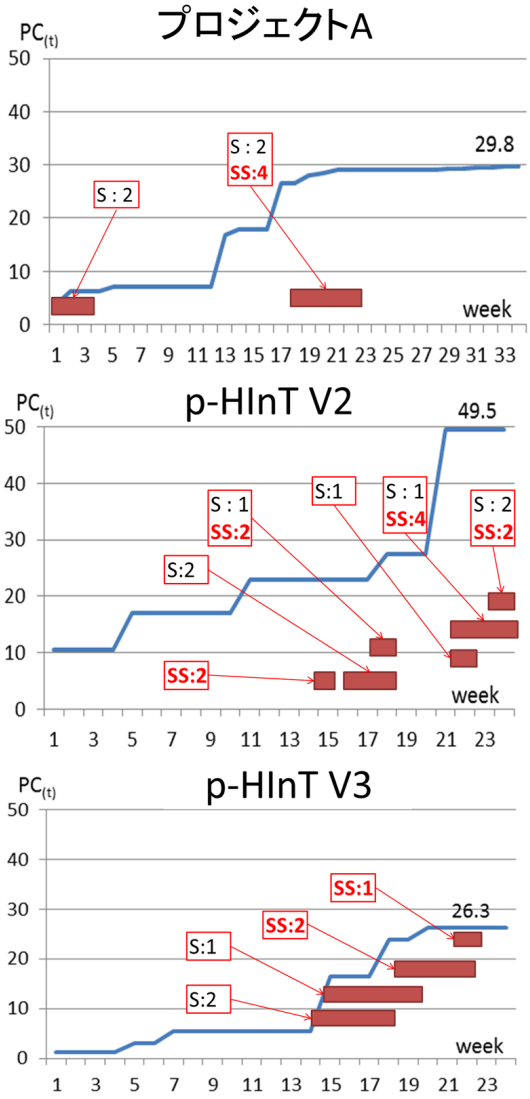


図 5.2　3 プロジェクトの障害発生プロセスと障害件数の関係

5.5.　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へ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 p-HInT V1 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，p-HInT V2 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，p-HInT V2 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 p-HInT V2 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，p-HInT V2 の PC 値は常に高い値を示した．p-HInT V3 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 p-HInT V4 は p-HInT V3 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列プロセス数に表れたと考えることができ，並列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．

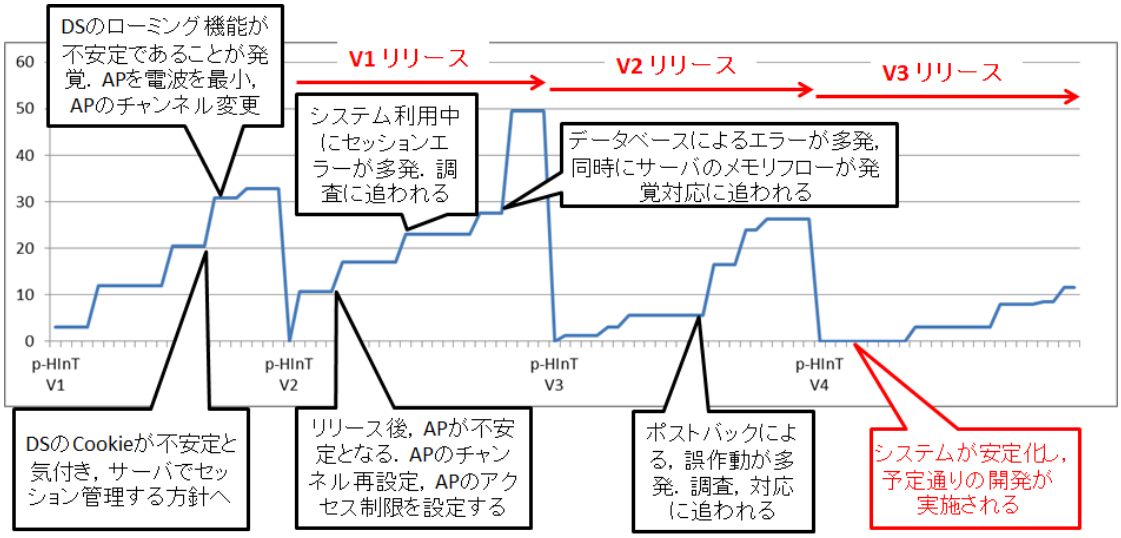


図 5.3　p-HInT プロジェクトの PC 値の推移と発生エラー

5.6.　p-HInT システムの要求と PC 値の関係

4 章で述べた p-HInT システム開発では，顧客からの困難な要求と PC 値には関係があると考えられる．4 章は p-HInT システム開発で発生した問題とその対策方法を試行錯誤で実現した様子をまとめた．そこで，それぞれの発生した問題とその対策を実施したイベントと PC 値の時系列な関係を図 5.3 に示す．

図 5.3 は，p-HInT プロジェクト全体の PC 値の推移の図に発生した問題とそれに対する対策を行ったイベントを追加したものである．PC 値が急激に上昇する際には 4 章で述べた問題に対する対策イベントが発生していることがわかる．以下に最も重大な問題発生とそれに対する対策実行のイベントについて説明する．

本システムを開発する上で最も大きな問題がアクセスポイントの設定であった．アクセスポイント設定は，運用テスト段階にならないと正確な動作が確認できない．特に低機能で通信仕様が不明確な携帯情報端末を大量に利用した場合は運用テストで問題が発覚する場合が多い．したがって，開発の後半の運用テスト工程に入ってから対応に追われている事がわかる．また，数台では動作したセッション管理も 200 台以上の携帯情報端末の DS で動作させると不安定になる現象も発生した．この点

図 5.4　p-HInT システムのテスト時の写真 1

図 5.5　p-HInT システムのテスト時の写真 2

について調査を行ったが，詳細な仕様が非公開であり，手探りで法則性が見つけるしか方法がなく，大変困難であった．したがって，テスト工程は多くの工数を費やした

実際のテスト時の写真を図 5.4 図 5.5 に示す．これらの図のように，作業者一人辺り 10 から 20 台の DS を割り当て，一斉に送信ボタン等を押すなどのテストを繰り返した．つまり，台数が多い時のリクエスト送信による同時アクセスの負荷テストである．本テストを準備するためには人員の確保，400 台の DS の充電などがあり，運用テストや負荷テストの実施回数が必然的に限られていた．

その結果，前節でも述べたように，テスト実施回数が制限されていたために十分な運用や負荷テストが実現せずに多くのバグを含んだリリースがバージョン 1 であった．バージョン 2 の開発ではバグ修正を行いながら，従来予定の新規システム開発を並列で行ったため PC 値が大きくなった．また，バージョン 2 の開発中に発生したセッションエラーについても，DS の独自の問題である．つまり，DS のクライアントでは Cookie が保持できないために，サーバでセッションを管理したため，メモリリークが発生しログインできない現象が発生した．p-HInT システムは Windows の IIS を利用しており，29 時間に一度ワーカプロセスを再起動する仕組みとなっている．この方法はアクセスが少ないシステムならば対応できるが，1 時間で 700 人以上のアクセスあるシステムでは，サーバ側でセッション管理することに対応できなかった．

このように，p-HInT システムの開発で問題となった時期と PC 値の上昇を照らし合わせたところ，多くの場合一致する結果となった．つまり，実現方式を模索しつつ

開発するシステムは PC 値を参考に開発をすすめる事でプロダクト品質を予測する可能性があることがわかった.

5.7. 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．本研究の提案する 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，本研究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ことが挙げられる [42]. 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，本研究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 5.4 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 p-HInT V2 であり，最も工数が少ないプロジェクトは p-HInT V4 であった．p-HInT V2 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.

表 5.4　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
p-HInT V1	32.8	1660	26	2
p-HInT V2	49.5	1895	21	10
p-HInT V3	26.3	1175	11	3
p-HInT V4	11.6	610	11	0
Project A	29.8	1160	34	4
Project B	38.8	1215	32	2
Project C	16.5	680	20	0
Project D	40.9	720	18	12

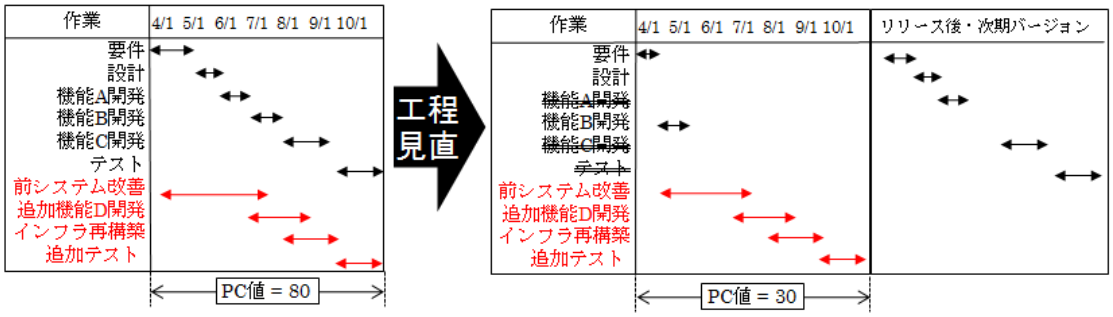


図 5.6　PC 値を利用したプロセス改善方法の例

同時に最も工数が少ない p-HInT V4 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列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，並列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，並列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．

5.8.　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列プロセス数（$L_{(t)_i}$ 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 5.6 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（図中赤矢印がフラグメントプロセス）を示し，右側が並列プロセス数を

制御した工程管理表の例である．

図 5.6 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．顧客は容易な修正を考え要求したが，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，並列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，並列プロセス数を減らすプロセス改善を行う．図 5.6 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 5.6 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 p-HInT V3 にて実際に実施された．p-HInT V3 は母体システムのバージョンアップであり，p-HInT V4 は p-HInT V3 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 p-HInT V3 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，p-HInT V3 ではリファクタリングを中心に行い，新規機能を p-HInT V4 にて実装する事で並列プロセス数を減らした．同時に p-HInT V3 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．具体的なリファクタリングとしてデータベースのパフォーマンス改善を行う事により，次期リリースのシステムの性能が向上したと考えられる．結果，p-HInT V3，p-HInT V4 共に他のプロジェクトと比較しても障害件数となった．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

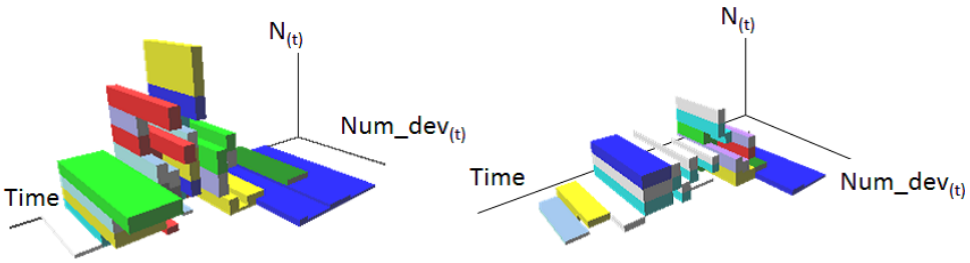


図 5.7 プロセスの複雑さ可視化ツール

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．この時，図 5.7 のようなプロセスを可視化するツールを利用し，開発者と顧客の両者でプロセス改善を実施する [48]．ことで，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．

5.9. 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．本節ではプロダクト品質への影響を予測することを目的としたの PC 値の活用方法について述べる．

5.9.1. PC 値活用のための方針

はじめに，本研究で提案する 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[19] 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

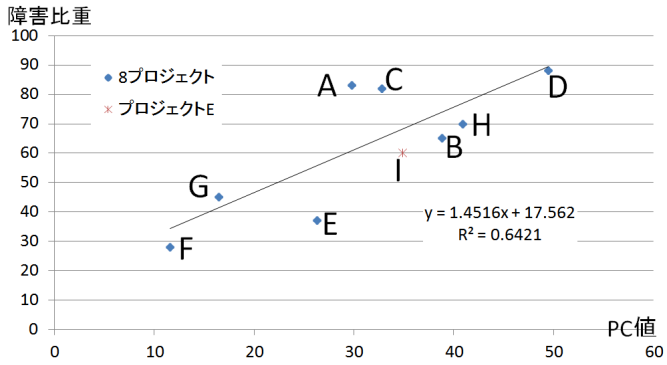


図 5.8　8 プロジェクトの PC 値と障害比重の関係

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．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

5.9.2.　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

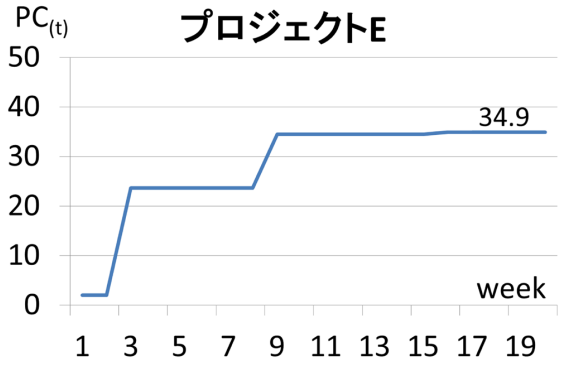


図 5.9　プロジェクト E の PC 値の推移

大を指標として示すことができる．

表 5.5 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 値と障害比重に相関を確認できた場合，PC 値から障害比重の予測式を求める．図 5.8 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．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 E についての予測を試みた．

プロジェクト E 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 E の PC 値の推移を図 5.9 に示す．プロジェクト E は開発期間が 20 週のプロジェ

表 5.5　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
p-HInT V1	32.8	82	2	3	19	1	1
p-HInT V2	49.5	88	10	6	4	1	0
p-HInT V3	26.3	37	3	3	2	1	2
p-HInT V4	11.6	28	0	0	7	3	1
Project A	29.8	83	4	4	6	9	11
Project B	38.8	65	2	1	6	10	13
Project C	16.5	45	0	0	8	9	3
Project D	40.9	70	12	0	2	0	4

クトであったが，プロジェクト E 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 E 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 E の 9 週目の PC 値を計測したところ PC 値は 34.9 であり，図 5.8 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 5.8 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 E 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．

第 6 章

結論

本研究では，プロセスとプロダクトの関係を明確にするために (1) 産学連携で開発した p-HInT システムの開発データを基に (2) プロジェクト実施中に発生した追加プロセスが他のプロセスに与える影響を計測したプロセスの複雑さという尺度を提案した．

p-HInT システムの開発には困難な要求が多数存在した．その困難な要求のうち，特に低機能端末への対応についての工夫点をまとめたのが以下となる．

(1) セッション管理
(2) アクセスポイントの設定
(3) ブラウザの機能の利用制約と通信量の削減
(4) 低性能無線 LAN を考慮したデータベースのチューニング

これらの工夫点は計画時に無かった要望（対策）のため，本来の開発と並列に改善プロセスとして実施されたプロセスである．

そこで，本研究では急な対策や要望がソフトウェア開発プロジェクトにどのような影響を与えるかを定量的に示したプロセスの複雑さ（PC 値）を提案した．プロセスの複雑さは産業界のソフトウェア開発プロジェクトでも利用される工程管理表から計測する．工程管理表に記載される並列作業数，開発者数，開発期間割合から計測する．計測方法は，工程管理表に後から追加されるフラグメントプロセス（計画されないプロセス）の並列作業数，開発数，開発期間割合を積算し，時刻 t までの n 個の

プロセスの総和から成り立つ.

提案したプロセスの複雑さを開発した p-HInT システムと 4 つの別のプロジェクトへ適用した．結果，提案するプロセスの複雑さとリリース後の致命的な障害には相関があることを確認した．同時に工数とリリース後の障害予測についての比較も行った．結果，プロセスの複雑さでは工数よりもリリース後の致命的な障害との相関が高いことがわかった．また，一部のプロジェクトから PC 値が上昇する際には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．これにより，プロセスの複雑さを利用することで，致命的な障害を予測することのできる可能性を示唆した.

また，適用したプロジェクトと PC の関係についての複数のプロジェクトに対しインタビューを実施した．各プロジェクトのうち，2 つが新規システム，6 つがシステムのバージョンアップとなり，新規システム開発は開発するシステムの利用イメージを明確にできなかったため，追加プロセスが発生した．また，バージョンアップのシステムでも全バージョンの改修作業が発生したため，プロセス全体が複雑なった．つまり，プロセスの複雑さはこのようなプロジェクトの開発形態を考慮せずに計測するが，考慮せずとも開発形態による依存関係は工程管理表から計測できる事がわかった．また，開発に参加した p-HInT システムの困難な要求と PC 値を分析すると，PC 値が上昇する際には 4 章で述べた工夫点が発生した期間が一致することがわかった．このように，プロセスの複雑さは工程管理表から容易に抽出できる 3 つのパラメータから計測するが，様々な開発に関する要因と関係があることがわかった.

プロセスの複雑さのコンセプトを利用したプロセス改善が実施していたプロジェクトが存在した．プロセスの複雑さは並列作業数が増加するとプロダクトに影響を与えるコンセプトである．p-HInT プロジェクト V3 では改修作業と新規機能開発の計画が存在したが，システムを安定化させるために改修作業だけの計画へと変更された．これは，プロセスの複雑という値を減少させるためのプロセス改善であり，実際に p-HInT プロジェクト V3 では致命的な障害が p-HInT プロジェクト V2 よりも減少していた．つまり，プロセスの複雑さを制御することにより，将来リリースされるプロダクト品質を向上させる可能性がある.

そこで，プロセスの複雑さを活用するための利用例を示した．プロセスの複雑さからリリース後の障害を回帰モデルで予測する．リリース後の障害の指標として障

害比重という尺度とし，過去のプロジェクト障害比重を目的変数，プロセスの複雑さを説明変数としたプロダクト品質予測の利用例である．この利用例に過去の 8 つのプロジェクトを適用し，新たなプロジェクトのプロダクト品質を予測したところ，比較的精度の高い予測ができた．

これらの本研究の成果により，ソフトウェア開発プロセスの観点からリ定量的にプロダクト品質を向上することが可能となる．具体的には複雑化したプロセスを定量的に計測することで，提案したプロセスの複雑さを制御するプロセス改善を実施することで，プロダクト品質を向上することが可能となる．これにより，無作為に後から追加されるプロセスがプロダクトにどの程度影響を与えるかを考慮し，慎重に追加プロセスの実行を思案できるソフトウェア開発が実施できる．

謝辞

本研究を進めるにあたり，多くの方々に御指導，御協力，御支援頂きました．ここに謝意を添えて御名前を記させて頂きます．本当にありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 飯田 元教授には，本研究の全てにおき熱心な御指導を賜りました．研究内容だけではなく，研究に対する考え方，研究者としての心構え，論文執筆，発表についても多くの御助言を頂きました．また，御人柄についても大変学ばさせて頂きました．3 年間大学院生活を達成することができたのは先生の御指導と御人柄によるものと確信しております．心より厚く御礼を申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には，要所要所で本研究に対し貴重な御指導，御助言を賜りました．また，企業との共同研究に際しても，御助言を頂きました．先生の御指導のおかげで，研究者としての考え方を身につけることができたと確信しております．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 藤川 和利教授には，本研究を進めるにあたり，貴重な御指導を賜りました．学内の発表において多数の御質問と御指導を頂きました．先生から頂いた御指摘は，研究を客観的に見つめ直す上で重要なものとなりました．心より感謝申し上げます．

阪南大学 経営情報学部 花川 典子教授には，阪南大学在籍時代から様々な御指導を賜りました．また，本学入学後も研究や，思考，また人生についても数えきれない程の助言を頂きました．今の自分が存在するのは先生のおかげであることは確実であると確信できます．厳しくも優しい御言葉，御厚意を多々賜りました．心から感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 吉田 則裕助教には，本論文を執筆

するに対し貴重な御助言を賜りました．また，研究活動や学生生活に関しても，常日頃から様々な御助言，御厚意を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人准教授には，学会参加時には様々な御助言や，を頂きました．心より感謝申し上げます

奈良先端科学技術大学院大学 情報科学研究科 市川 昊平 准教授，ならびに CAMARGO CRUZ Ana Erika 助教には，本論文に対し貴重な御助言を賜りました．また，常日頃から様々な御助言を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 田中 康 特任准教授，ならびに高井 利憲 助教には，本論文に対し貴重な御助言を賜りました．また，JAXA 演習時には楽しい時間を満喫させて頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 伏田 享平 特任助教（現 NTT データ），には本論文に対し多くの貴重な御助言を賜りました．また，研究の楽しさや厳しさ，論文投稿時や発表，学生生活に関しても貴重な御助言を頂きました．心より感謝申し上げます．

阪南大学 阪本 泰彦 様，森 章 様，濱田 佐知子様には，p-HInT システム開発に関するデータの提供をして頂きました．また，開発に関しても数えきれない程の御協力をして頂きました．ここに感謝申し上げます．

株式会社 富士通マーケティング 田代 克彦 様，山本 剛 様には，貴重な企業資料を御提供頂きました．また，論文投稿の際には，実務の観点から重要且つ貴重な御意見を数多く頂きました．心より感謝申し上げます．

株式会社 富士通マーケティングの開発に参加して頂きました開発者の皆様には，開発に関して様々な御協力をして頂いた他にも貴重な企業資料を御提供頂きました．また，論文投稿の際には，実務の観点から重要且つ貴重な御意見を数多く頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 ソフトウェア設計学研究室　藤原 賢二 様には，日頃より様々な御協力と御助言を頂くと共に，常日頃より楽しい時間を過ごさせて頂きました．ありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア設計学研究室の皆様には，日頃より多大な御協力と御助言を頂くと共に常日頃より楽しい時間を過ごさせて頂きました．ありがとうございました．

奈良先端科学技術大学院大学 川本 理恵 様，呂 悠妃 様には，研究の遂行に必要な事務処理など，多岐にわたり御助力頂きました．心より感謝申し上げます．

本研究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 の補助を受けさせて頂きました．心より感謝致します．

最後に，日頃より私を励まし，応援して下さった両親，兄姉に心より深く感謝します．

参考文献

[1] A. Albrecht and J. Gaffney. Software function, source lines of code and debelopment efort prediction. *IEEE Transactions of Software Engineering*, Vol. 9, No. 6, pp. 83–92, 1979.

[2] 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, 第 1589 巻, pp. 215–221, 2008.

[3] 馬場慎太郎, 吉田則裕, 楠本真二, 井上克郎. Fault-prone モジュール予測へのコードクローン情報の適用. 電子情報通信学会論文誌, Vol. J91-D, No. 10, pp. 2559–2561, 2008.

[4] V. R. Basili, L.C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions of Software Engineering*, Vol. 22, No. 10, pp. 751–761, 1996.

[5] V. R. Basili and H. D. Rombach. The tame project : towards improvement-oriented software environments. *IEEE Transactions of Software Engineering*, Vol. 14, No. 6, pp. 758–773, 1988.

[6] V. R. Basili and D. M. Weiss. A methodorogy for collecting valid software engineering data. *IEEE Transactions of Software Engineering*, Vol. 8 SE-10, No. 6, pp. 728–738, 1984.

[7] C. Bill, V. S. Girish, R. Donald, H. Iraj, and K. Gargi. The case for quantitative process management. *IEEE Software*, Vol. 25, No. 3, pp. 24–28, 2008.

[8] B. W. Boehm. *Software Engineering Economics.* rentice-Hall Advances in

Computing Science & Technology Series, 1981.

[9] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation with COCOMOII*. Prentice Hall PTR Upper Saddle River, NJ,, 2000.

[10] L.C. Briand, J.W. Daly, and J. Wust. A unified framework for cohesion measurement in object-oriented system. *Empirical Software Engneering*, Vol. 3, No. 1, pp. 65–117, 1998.

[11] L.C. Briand, K.E. Eman, D. Surmann, I. Wieczorek, and K.D. Maxwell. An assessment and comparison of common software cost estimation modeling techniques. In *Proceeding of the 21th International Conference on Software Engineering(ICSE1999)*, pp. 313–322, 1999.

[12] L.C. Briand, T. Langley, and I. Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. In *Proceeding of the 22th International Conference on Software Engineering(ICSE2000)*, pp. 377–386, 2000.

[13] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions of Software Engineering*, Vol. 20, No. 6, pp. 476–498, 1994.

[14] G. Cugola. Tolerating deviations in process support systems via flexible enactment of process models. *IEEE Transactions of Software Engineering*, Vol. 24, No. 11, pp. 982–1001, 1998.

[15] D. Damian and J. Chisan. 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, pp. 433–453, 2006.

[16] N. C. David, D. Kevin, and Giyn D. Making statistics part of decision making in an engineering organization. *IEEE Software*, Vol. 25, No. 3, pp. 37–47, 2008.

[17] EASE プロジェクト. EASE プロジェクト. http://www.empirical.jp/.

[18] T. Farag, N. Funabiki, and T. Nakanishi. An access point allocation algo-

rithm for indoor environments in wireless mesh networks. *IEICE Transactions on Communications*, Vol. E92-B, No. 3, pp. 784–793, 2009.

[19] International Organization for Standardization. *ISO/IEC 9126:1991, Software enginnering – Product quality.* International Organization for Standardization, 1991.

[20] Jr. Frederick P. Brooks. *The Mythical Man-Month.* Addision-Wesley, 1975.

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

[22] F. Garcia, F. Ruiz, and M. Piattini. Definition and empirical validation of metrics for software process models. In *Proceeding of the 5th International Conference Product Focused Software Process Improvement(PROFES 2004)*, pp. 146–158, 2004.

[23] R. A. Gray and G. S. MacDonell. Software metrics data analysis - explorning the relative performance of some commonly used modeling techniques. *Empirical Software Engneering*, Vol. 4, No. 4, pp. 297–316, 1999.

[24] N. Hanakawa. A project reliability growth model based on communication for software development. *International Journal of Knowledge Engineering and Software Engineering*, Vol. 20, No. 5, pp. 665–677, 2010.

[25] N. Hanakawa, Yamamoto G., K. Tashiro, H. Tagami, and S. Hamada. p-hint: Interactive educational environment for improving large-scale lecture with mobile game terminals. In *Proceeding of the 16th International Conference on Computers in Education(ICCE2008)*, pp. 629–634, 2008.

[26] N. Hanakawa, K. Matsumoto, and K. Torii. A communication workload estimation model based on relationships among shared works software development projects. In *Proceeding of the 9th Asia-Pacific Software Engineering Conference(APSEC2002)*, pp. 571–580, 2002.

[27] N. Hanakawa and M. Obana. Lecture improvement based on twitter logs and lecture video using p-hint. In *Proceeding of the 18th International Conference on Computers in Education (ICCE2010*, pp. 328–335, 2010.

[28] N. Hanakawa and M. Obana. Mobile game terminal based interactive education environment for large-scale lectures. In *Proceeding of the Eighth IASTED International Conference on Web-based Education (WBE2010)*, 2010.

[29] N. Hanakawa and M. Obana. A plagiarism detection system for reports based on a large-scale distribution environment using idle computers. In *Proceeding of the 15th IASTED International Conference on Computers and Advanced Technology in Education(CATE2012)*, 2012.

[30] 花川典子. 大人数講義の双方向教育を実現した授業支援システム. ICT 活用教育方法研究論文誌, Vol. 13, No. 1, pp. 36–40, 2010.

[31] H. Hata, O. Mizuno, and T. Kikuno. Fault-prone module detection using large-scale text features based on spam filtering. *Empirical Software Engneering*, Vol. 15, No. 2, pp. 147–165, 2010.

[32] S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions of Software Engineering*, Vol. SE-7, No. 5, pp. 510–518, 1981.

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

[34] C. Jorge. Complexity analysis of BPEL web processes , Software Process:. *Improvement and Practice*, Vol. 12, No. 1, pp. 35–49, 2006.

[35] 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, pp. 1836–1850, 2010.

[36] 亀井靖高, まつ本真佑, 柿本健, 門田暁人, 松本健一. Fault-prone モジュール判別におけるサンプリング法適用の結果. 情報処理学会論文誌, Vol. 48, No. 8, pp. 2651–2662, 2007.

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

[38] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, Vol. 23, No. 2, pp. 111–122, 1993.

[39] T. Littlefair. Cccc - c and c++ code counter. http://cccc.sourceforge.net/.

[40] 野中誠, 水野修. fault-prone モジュール予測技法の基礎と研究動向. ソフトウェアエンジニアリングシンポジウム 2010 チュートリアル資料（SES2010), 2010.

[41] L.V. Manzoni and R.T. Price. 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, pp. 181–192, 2003.

[42] 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. 30, No. 28, pp. 6–8, 2011.

[43] 松本健一（訳）, 片山卓也, 土居範久, 鳥居宏次（監）. メトリクス，ソフトウェア工学大辞典. 朝日書店, 1998.

[44] T. McCabe. A software complexity measure. *IEEE Transactions of Software Engineering*, Vol. SE-2, No. 4, pp. 308–320, 1986.

[45] 森棟公夫, 照井伸彦, 中川満, 西埜晴久, 黒住英司. 統計学. 有斐閣, 2008.

[46] C. J. Munson and M. T. Khnshgoftaar. The detection of fault-prone programs. *IEEE Transactions of Software Engineering*, Vol. 18, No. 5, pp. 423–433, 1992.

[47] 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, pp. 2534–2550, 2007.

[48] M. Obana, N. Hanakawa, and H. Iida. A process complexity- product quality (pcpq) model based on process fragment with workflow management tables. In *Proceeding of the 11th International Conference on Product Focused Software Process Improvement (PROFES 2011)*, pp. 171–185, 2011.

[49] M. Obana, N. Hanakawa, N. Yoshida, and H. Iida. Process fragment based process complexity with workflow management tables. In *Proceeding of the International Workshop on Empirical Software Engineering in Practice(IWESEP2010)*, pp. 7–12, 2010.

[50] K. Ohkura, K. Goto, N. Hanakawa, S. Kawaguchi, and H. Iida. Project replayer with email analysis - revealing contexts in software development. In *Proceeding of the 13th Asia-Pacific Software Engineering Con-*

ference(APSEC2006), pp. 453–460, 2006.

[51] 大即洋子, 坂東宏和, 加藤直樹, 中川正樹. 対話型電子白板を用いたグループ間の競争による学習を支援する教育ソフトウェアの一例とその効果. 情報処理学会論文誌, Vol. 44, No. 6, pp. 1635–1644, 2003.

[52] 岡村寛之, 安藤光昭, 土肥正. 一般化ガンマソフトウェア信頼性モデル. 電子情報通信学会論文誌, Vol. J87-D-I, No. 8, pp. 805–814, 2004.

[53] p-HinT. ICT を活用した双方向教育システムの構築. http://www2.hannan-u.ac.jp/p-hint/index.html.

[54] Eclipse Metrics Plugin. Eclipse metrics plugin. http://eclipse-metrics.sourceforge.net/.

[55] Walker R. *Software Project Management: A Unified Framework*. The Addison-Wesley Object Technology, 1998.

[56] H. D. Rombach. Design measurement : Some lessons learned. *IEEE Software*, Vol. 7, No. 2, pp. 17–25, 1990.

[57] 高橋良英. C 言語ソフトウェア保守工程における halstead のソフトウェアサイエンス計測と障害密度との関係の分析. 電子情報通信学会論文誌, Vol. J82-D-I, No. 8, pp. 1017–1034, 1999.

[58] 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エ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, pp. 740–748, 2000.

[59] SWEBOK. Guide to the software engineering body of knowledge (swebok). http://www.computer.org/portal/web/swebok.

[60] Furuyama. T. and Y. Nakagawa. A manifold growth model that unifies software reliability growth models. *International Journal of Reliability, Quality and Safety Engineering*, Vol. 1, No. 2, pp. 161–184, 1994.

[61] 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, 第 43 巻, pp. 49–63, 2004.

[62] 田村晃一, 柿本健, 戸田航史, 角田雅照, 門田暁人, 松本健一, 大杉直樹. 工数予測における類似性に基づく欠損値補完法の実験的評価. コンピュータソフトウェア, Vol. 26, No. 3, pp. 45–55, 2007.

[63] 田村慶信, 山田茂, 木村光宏. オープンソース共同開発環境に対するソフトウェア信頼性評価法に関する考察. 電子情報通信学会論文誌, Vol. J88-A, No. 7, pp. 840–847, 2005.

[64] 角田雅照, 大杉直樹, 門田暁人, 松本健一, 佐藤慎一. 協調フィルタリングを用いたソフトウェア開発工数予測方法. 情報処理学会論文誌, Vol. 46, No. 5, p. 5, 2005.

[65] 上村香菜子, 舩曳信生, 中西透. スケーラブル無線メッシュネットワークのための通信路構成最適化アルゴリズム. 電子情報通信学会論文誌, Vol. J92-B, No. 9, pp. 1526–1537, 2009.

[66] S. R. Uma, K Srikanth, and P. Chinmay. Stochastic optimization modeling and quantitative project management. *IEEE Software*, Vol. 25, No. 3, pp. 29–36, 2008.

[67] C Walston and C. Felix. A method of programming measurement and estimation. *IBM Systems Journal*, Vol. 16, No. 1, pp. 54–73, 1977.

[68] S. Yamada, J. Hishitani, and S. Osaki. Software-reliability growth with a weibull test-effort: a model and application. *IEEE Transactions on Reliability*, Vol. 42, No. 1, pp. 100–106, 1993.

[69] 山田茂. ソフトウェア信頼性モデル. 日科技連, 1994.

[70] 山田茂, 得能貢一, 井上圭. コスト評価基準に基づくソフトウェア信頼性/安全性を考慮した最適リリース問題. 電子情報通信学会論文誌, Vol. J82-A, No. 1, pp. 64–72, 1999.