マルチベンダー情報システム開発における障害修正工数の要因分析

松村知子[†] 門田暁人[†]
森崎修司[†] 松本健一[†]

ソフトウェア開発プロジェクトにおいて，重大な（修正工数の大きな）障害の発生は，開発コストの超過や納期の遅延を招くため大きな問題である．本論文では，障害の修正工数に寄与する要因を明らかにするために，日本のある典型的なマルチベンダー中規模情報システム開発を対象とし，設計から総合試験までの 6ヶ月間に生じた障害の特性値を数多く収集し，修正工数に寄与する要因を統計的に分析した．分析の結果，総合試験で発見された障害が単体試験/コーディング工程で発見されたものより 4.88 倍の工数がかかることがわかった．これは，「障害の発見が遅れるほど修正工数が増大する」という法則を裏付けている．また，障害が混入してすぐに発見された場合と比較して，2 工程以上遅れて発見された場合に工数が 4.44 倍になった．このことから，「滞留する時間が長ければ長いほど，障害の修正コストが高くなる」という法則も裏付けられた．さらに，障害の再現度や重要度などが修正工数に影響することも明らかになった．

Analyzing Defect Correction Effort Using Defect Attributes in a Multi-Vendor Information System Development

Tomoko Matsumura,[†] Akito Monden,[†] Shuji Morisaki[†] and Ken-ichi Matsumoto[†]

This paper describes an empirical study to reveal factors influencing defect-correction effort in software development. In the study we collected various attributes (metrics) of defects found in a typical medium-scale, multi-vendor information system development project in Japan over a six-month period. We then statistically analyzed the relationship between the defects' attributes and the correction effort. The analysis confirmed the well-known principal "defects are the more expensive the later they are detected" by revealing that defects detected in the "system test" were 4.88 times more expensive than those detected in the "coding/unit test". Another principal "defects are more expensive the longer they survive in software" was also confirmed by revealing that defects, which survived two or more development phases, were 4.44 times more expensive than those detected immediately. We also identified other factors, such as defect repeatability and severity, that had a significant influence on the correction effort.

1. はじめに

ソフトウェア開発における障害 (バグ，仕様変更，設計変更など) の発生は，コスト (工数) 超過や納期遅延の主原因の一つである．そこで，工程や機能ごとに障害発生件数を調べ，その発生要因との関係を明らかにする研究や分析法が提案されている．例えば，ODC(Orthogonal Defect Classification: 直交欠陥分類法)[2)3)8)] では，障害管理票に基づいて各障害をいくつかの種類に分類し，工程間や作業間で各種類の障害発生件数の偏りの有無を調べることで，改善すべき工程や作業を特定する．また，DCA(Defect Causal Analysis：障害原因分析)[7)14)15)] では，各障害のより詳細な情報 (自然言語による記述やインタビュー結果) を収集し，特性要因図 (Fishbone diagram) などを用いて障害の原因を階層的に整理し，障害発生件数との関係を分析する．

ただし，開発のコスト超過や納期遅延を防ぐ観点からは，障害の件数のみならず，障害の修正工数に着目した要因分析が必要である．一般に，障害の修正工数のばらつきは大きく，たとえ 1 件の発生であっても，深刻な工数超過や納期遅延につながる場合があるためである．

† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

そこで，本論文では，ある中規模システム情報システム開発を対象とし，障害を特徴付ける特性値を収集し，修正工数との関係を分析する．収集する特性値としては，障害の「原因」や「発生した機能」といった障害自体の情報に加え，障害の「混入工程」,「発見工程」,「修正工程」などの工程に関する情報を含む．分析では，各特性値と修正工数との関係の有無，および，関係の大きさを箱ひげ図や t 検定を用いて統計的に調べる．また，分析結果の解釈には，各障害の修正実施記録やプロジェクト関係者へのインタビューなどを通して得た情報を用いる．

本論文で分析対象とするのは，今日の日本における情報システム開発の典型とも言える「マルチベンダー開発」である．具体的には，経済産業省の支援の受けて進められた約 330K ステップ規模の情報システム開発を対象とした．このプロジェクトでは，日本の複数の中核ソフトウェア企業が開発を担当し，奈良先端科学技術大学院大学 EASE プロジェクト☆1と情報処理推進機構（IPA）下に設置されたソフトウェアエンジニアリングセンター（以降，SEC と呼ぶ）☆2が連携してデータ収集体制の構築とデータ分析を行った．開発はウォーターフォールプロセスにより進められ，ユーザ的立場の企業が要求を出し，プロジェクト管理担当企業の指揮のもとで開発担当各社がサブシステムの開発をそれぞれ行い，社内単体試験，社内結合試験を経て，社間結合試験，社間総合試験が行われた．

本論文における分析と ODC や DCA との違いは，統計分析を行う点である．障害の発生件数はプロジェクトごとの属性であるため，単一のプロジェクトで行うことのできる統計分析は著しく限定される．一方，障害の修正工数は，障害ごとの属性であるため，1 つのプロジェクトの事例においても（多数の障害が発生していれば）統計的な分析が可能である．従来，障害の修正工数に関して「障害の発見が遅れるほど修正工数が増大する」という法則が知られている[10]．ただし，修正工数がどの程度増大するかについての具体的な事例報告は，1970 年代の事例が存在するのみである[4)5)9)12)]（5．参照）．当時からは，プログラミング言語や開発環境，テスト環境が著しく進歩しているため，今日でもこれらの法則が適用できるかは，定かではない．また，上記の分析では，混入工程や障害原因など他の特性値との関係については言及されていない．そのため，今日の典型的なプロジェクトを対象として，より詳細で定量的な分析を行い，従来の法則の裏付けや，特性値と修正工数間の関係について明らかにすることが必要である．

以降，2．では分析に用いた収集データの概要説明し，3．で分析手順を示す．4．で各ステップでの分析結果を示し，5．では関連研究，6．でまとめと今後の課題について述べる．

2. 収集データ

2.1 対象プロジェクトの概要

対象プロジェクトは，経済産業省の支援を受けて，COSE☆3参加企業によって実施されたプローブ情報システムの開発である．開発の実施は，プロジェクト管理担当の 1 社と開発担当の 5 社によって行われた．5 社のうち 2 社はそれぞれ 2 箇所の拠点で開発を行ったため，結果として物理的に離れた 7 拠点で開発が進められた．プロジェクト発足当初から EASE プロジェクトと SEC の各研究員がプロジェクトの定例会議に参加し，データ収集体制の構築とデータ分析を担当した．また，分析結果を開発者へ報告するための個別企業との定期的ミーティングの場が設けられ，EASE および SEC の研究員が分析結果のフィードバックとインタビューを行った．

開発システムのサイズは，約 330K ステップ (KLOC) で，ほとんどのプログラムは C/C++で書かれているが，一部はスクリプト言語や Python で書かれている．38 個のサブシステムで構成され，ファイル数は約 1400 ファイルである．

2.2 データ収集方法と収集項目

データ収集にあたり，本プロジェクト参加企業 6 社で運用されている障害票と IEEE 障害標準項目[1)]を参考に，開発の事前の協議により収集すべき障害の特性値を決定した．

データ収集ツールとして，1 拠点を除いて共通の障害管理ツール GNATS☆4 を導入し，残りの 1 拠点は自社の標準ツールをカスタマイズすることで対処した．収集した障害データは，SEC 内のデータベースに蓄積され，EPM (Empirical Project Monitor)[16)] を使って分析用に加工した．

データ収集は 7 拠点全てで行ったが，データ収集経験の浅い 1 つの拠点は分析対象から除外した．開発関

☆1 EASE：Empirical Approach to Software Enginnering http://www.empirical.jp/top.html

☆2 ソフトウェアエンジニアリングセンター（SEC）：Software Engineering Center） http://sec.ipa.go.jp/index.php

☆3 ソフトウェアエンジニアリング技術研究組合（COSE）：COnsortium for Software Engineering

☆4 GNATS：GNU Bug Tracking System http://www.gnu.org/software/gnats/

係者へのインタビューや得られたデータの偏りから，収集データ自体の信頼性が疑われたためである．

データの収集工程，すなわち障害を発見する工程は，「コーディング／単体試験」から「総合試験」までを対象とした（本プロジェクトでは，コーディングと単体試験を同一の工程とみなしている）．本プロジェクトでの試験工程は，各拠点内での単体試験・結合試験後，全拠点のプロダクトを結合した統合環境での社間結合試験・総合試験というステップで実施された．また，一部の拠点では，コードレビューが行われたが，これは「コーディング／単体試験」工程に含まれる．

障害の混入工程は，基本設計から総合試験までのものを分析対象とした．要求分析工程で混入した障害，すなわち，要求仕様の誤りと変更については，本プロジェクトでは異なる管理票を用いたため，分析対象外とした．

表 1 は収集した欠陥の特性値のうち，本論文に用いたものの一覧である．これ以外にも，自由記述項目として，問題内容・問題原因詳細・修正方法・確認方法などがあるが，統計分析には用いないので表には示していない．

本論文では，統計分析を行うにあたって，修正工数を目的変数とし，表 1 の各項目を説明変数とする．また，各説明変数のとりうる値（選択項目）をカテゴリと呼ぶ．たとえば，変数「重要度」に対しては,「重大」「中度」「軽微」の 3 つがカテゴリとなる．

2.3 収集データの概要

障害修正工数の統計値を**表 2** に示す.「度数」は障害件数を示し，978 件について修正工数が記録されていた．残りの 39 件は欠損またはゼロが記録されていたため，分析対象外とした．修正工数の平均値が約 2 人時であったのに対して，中央値が 1 人時と小さいことから，値の分布に偏りがあることが伺える．修正工数の最大値は 92 人時であり，これは約 3.8 人日に相当する．

各変数のカテゴリごとの障害件数を，表 1 の障害件数の欄に示す．表より，障害の全体的な特徴として，新規に開発されたコンポーネントから発見された障害が大部分を占めることや，障害の混入工程，発見工程ともに「コーディング／単体試験」が多数を占めることなどが分かる．

3. 分 析 方 法

分析は，(1) データ前処理，(2) 各変数と修正工数の関係の分析，(3) 変数間の関係の分析，という手順で行った．分析結果の解釈には，各障害の修正実施記録やプロジェクト関係者へのインタビューなどを通して得た情報を用いる．

表 2 修正工数（人時）の統計情報
Table 2 Statistics of Defect Correction Effort (human-hour)

度数	有効	978
	欠損値	39
平均値		2.13
中央値		1
標準偏差		4.58
分散		20.96
最小値		0.08
最大値		92
パーセンタイル	25	0.75
	50	1
	75	2

以降，各手順の詳細を述べる．

3.1 データ前処理

分析に先立って，以下の通りデータの前処理を行った．

- 未入力データの扱い：説明変数のいくつかには「未入力」カテゴリがあるが，これはすべて欠損値として扱った．
- 入力ミスと思われるデータの削除：明らかな入力ミスと思われるデータを欠損値として扱った．例えば,「混入工程」の方が「発見工程」よりも後工程となっていたケースなどである．
- 工程の統一：拠点内での試験の実施は各社の基準に沿って行われたため，試験の実施工程にはばらつきがあった．ほとんどの拠点では，単体・結合試験の 2 工程を行ったが，1 拠点では，単体・結合・総合試験の 3 工程を行っていた．分析に際して，後者については「結合試験」「総合試験」の２工程を合わせて１つの「結合試験」とみなした．また，社間結合試験・総合試験は，全体の件数が少ないため「総合試験」に統合した．
- 「発見遅延」の追加：「混入工程」と「発見工程」の工程差を「発見遅延」という新たな説明変数として設けた．この変数によって，プロダクトにバグが残る時間が長いほどバグ除去のためのコストが高くなる[10] という法則を定量的に検証することが可能となる．

3.2 各変数と修正工数の関係の分析

まず，表 1 の各変数と修正工数との関係の有無，および，関係の大きさを分析する．前者（関係の有無）については，各変数の 1 つのカテゴリに属する障害と，それ以外のカテゴリに属する障害について，修正工数の平均値の差の検定 (t 検定) を行う．有意水準は 1%($p < 0.01$) とする．この検定を，全ての変数の全

表 1 障害項目一覧表
Table 1 A List of Failure Report Items

項目名	選択項目（カテゴリ）	障害件数	項目説明	項目名	選択項目（カテゴリ）	障害件数	項目説明
発見箇所	新規	963	障害を発見した	優先度	高	320	障害解決の
	改造	12	モジュールの再		中	611	優先度合い
	再利用 (未改造)	2	利用の状況		低	46	
発見工程	未入力	13	障害を発見した	障害原因	未入力	47	障害混入原因の
	コーディング/単体試験	650	工程		論理エラー	288	種類
	結合試験	279			演算エラー	52	
	総合試験	33			インタフェース・		
発見作業	分析・解析中	108	障害を発見した		タイミング問題	56	
	システム動作中	827	作業		データ処理問題	361	
	（テスト運用）				データ範囲不正	30	
	レビュー	42			データ問題	17	
機能	入力データチェック機能	105	障害を発見した		仕様書 (設計書) 不正	27	
	演算機能	111	システムの機能		仕様書 (設計書) 品質問題	2	
	データ編集機能	254			機能強化・拡張	20	
	ファイル更新機能	15			性能問題	8	
	データ出力機能	176			相互操作性問題	3	
	連動（組み合わせ）処理	17			標準一致問題	1	
	限界処理	4			ハードウェア障害	0	
	外囲条件異常検知機能	2			操作ミス	24	
	その他	293			指摘ミス (仕様どおり)	7	
再現度	再現頻度大	704	故障発生の		原因不明	3	
	再現度小	94	再現頻度		その他	31	
	再現なし	35		発見遅延の	未入力	331	障害を混入工程も
	不明 (未確認)	144		原因	レビュー未実施	3	しくは本来発見す
重要度	重大	340	障害の重要さ		レビュー指摘もれ	35	べき工程で発見で
	中度	503	の度合い		再レビュー及び修正		きなかった理由
	軽微	132			確認もれ	12	
混入工程	未入力	41	障害混入の原因		工程間引継ぎ		
	基本設計	1	となった工程		コミュニケーション不足	1	
	詳細設計	86			試験項目抽出もれ	26	
	コーディング/単体試験	789			テストそのもののもれ	6	
	結合試験	57			環境上の問題で		
	総合試験	3			後工程にもっていった	11	
					結果確認ミス	5	
					その他	547	

てのカテゴリについて行うことで，修正工数と関係がある（有意差あり）と判定される変数とカテゴリを特定する．後者（関係の大きさ）については，修正工数の平均値の差を用いる．検定で有意差があった変数・カテゴリであっても，差が小さいもの (1.0 人時未満) は分析から除外する．

なお，t 検定では目的変数が正規分布に従っていることが前提となる．修正工数の分布を正規 P-P プロットにより調べた結果，対数正規分布に従うことが分かったため，t 検定では修正工数の対数値を用いることとする．

次に，工程に関する変数（障害の混入工程，発見工程，発見遅延）については，より詳細に分析する．具体的には，工程別の箱ひげ図を描いて修正工数の分布を比較することで，工程の違いが修正工数に与える影響を分析する．

3.3 変数間の関係の分析

各変数が修正工数へ与える影響を明らかとするためには，各変数と修正工数との (1 対 1 の) 関係を分析するのみならず，変数間の関係の分析も必要である．本論文では，紙面の制限上，障害の混入工程と発見工程の関係に限定して分析を行う．具体的には，ODC と同様，混入工程と発見工程の全ての組み合わせについて障害を分類する[8]．そして，各分類における修正工数の平均値を統計的に比較することで，混入工程，発見工程，及び，修正工数の 3 者の関係を分析する．

4. 分析結果

4.1 各変数と修正工数の関係

表 3 は，t 検定で有意差が認められ，他カテゴリとの平均値の差が 1.0 人時以上の変数カテゴリの一覧である．表の左カラムから順に，説明変数，カテゴリ，

表 3 t 検定で有意差の認められたカテゴリ
Table 3 Categories that had siginificance in t-test

説明変数	カテゴリ	障害数	平均修正工数
発見工程	コーディング/単体試験	650	1.70
	総合試験	33	8.29
修正工程	総合試験	33	8.29
混入工程	詳細設計	86	3.89
発見遅延	0 工程	639	1.70
	2 工程	54	7.54
	3 工程	5	4.80
発見遅延の原因	試験項目抽出漏れ	26	5.84
	環境上の問題で後工程に持っていった	11	7.55
	結果確認ミス	5	4.00
障害原因	指摘ミス（仕様どおり）	7	0.75
	その他	31	1.15
再現度	小	94	3.70
	再現なし	35	0.44
重要度	軽微	132	0.88
発見作業	レビュー	42	0.99
（参考）全障害		978	2.13

各カテゴリに属する障害数と平均修正工数を示している．以降，表中の各変数について結果の詳細を述べる．

[**発見工程**] 発見工程が「コーディング/単体試験」のとき平均修正工数が小さく (1.7 人時),「総合試験」のとき大きかった (8.29 人時)．各工程の修正工数の箱ひげ図を**図 1** に示す.「コーディング/単体試験」と「結合試験」の差はさほど大きくないものの,「総合試験」で急激に修正工数が増大する傾向にあった．

[**混入工程**] 混入工程が「詳細設計」のとき平均修正工数が大きかった (3.89 人時)．各工程の修正工数の箱ひげ図を**図 2** に示す．混入工程は，設計工程で混入するものにより多くの工数がかかる傾向にあった．これは，設計レベルの変更を行わなければならないため，いわゆる「手戻り」の作業量が多く，修正範囲も大きいため，修正作業・確認作業に時間がかかったと考えられる．一方，混入工程が結合試験以降の工程でも，修正工数は増大する．これは，障害そのものの修正の困難さよりは，関連するプログラムへの影響範囲などにより，プログラム変更や回帰試験などで慎重な対応を求められることが多いためと思われる．特にデグレードによる障害は，第三者によるレビューが重点的に行われることが多いことにより，修正により多くの時間を要することが多い．

[**発見遅延とその原因**] 発見遅延が 0 工程（すなわち，混入工程＝発見工程）の場合に平均修正工数が小さく (1.70 人時)，発見遅延が 2 工程になると 7.54 人時，3 工程では 4.80 人時となった (ただし，発見遅延＝ 3 工程の障害は 5 件しかない点に注意)．発見遅延と修正工数の関係を表す箱ひげ図を**図 3** に示す．図より，発見遅延が 0 工程と 1 工程とでは修正工数にほとんど差がないが，2 工程になると急激に増大していることが分かる．

次に,「発見遅延の原因」に着目する．この変数は，発見の遅れた障害（発見遅延が 1 工程以上のもの）のみに用意されたものである．発見遅延の原因別の修正工数，及び，t 検定の結果を**表 4** に示す．件数が少な

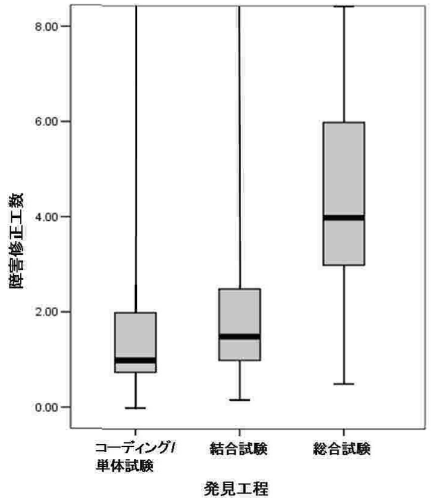


図 1 発見工程と修正工数の関係
Fig. 1 Defect Correction Effort in Each Phase Where Defects Were Detected

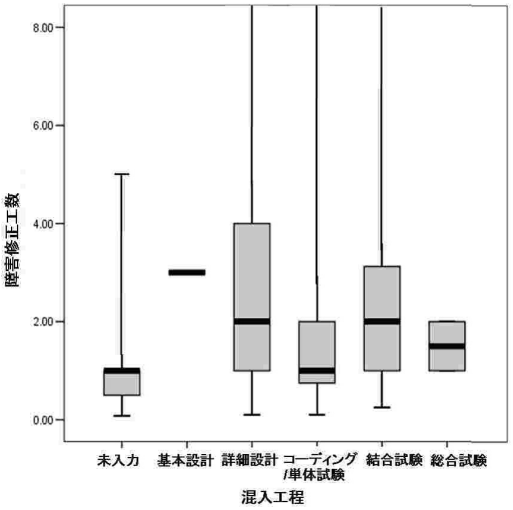


図 2 混入工程と修正工数の関係
Fig. 2 Defect Correction Effort in Each Phase Where Defects Were Introduced

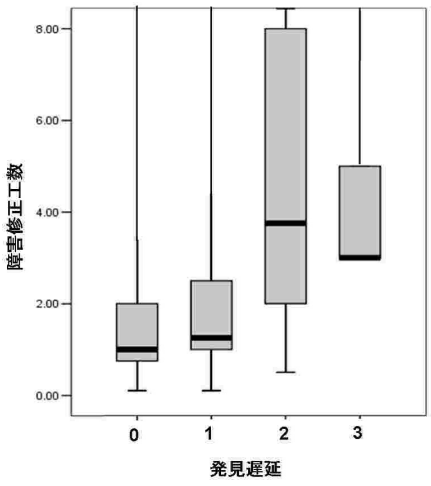


図 3 発見遅延と修正工数の関係
Fig. 3 Defect Correction Effort in Each Detection Delay

表 4 発見遅延の原因と修正工数の関係
Table 4 Relation between Defect Correction Effort and Cause of Detection Delay

障害遅延の原因	障害数	p 値	平均修正工数
レビュー未実施	2	0.773	1.50
レビュー指摘漏れ	14	0.220	2.27
工程間引継ぎ コミュニケーション不足	5	0.376	3.65
再レビュー漏れおよび 修正確認漏れ	1	0.613	3.00
試験項目抽出漏れ	25	0.063	6.02
テストそのものの漏れ	3	0.972	2.83
環境上の問題で後工程に 持っていった	7	0.001	10.86
結果確認ミス	3	0.176	4.00
その他	159	0.236	3.05
（参考）全障害	978	—	2.13

いためカテゴリ間の有意差はほとんど見られなかったが,「環境上の問題で後工程に持っていった」場合に,平均修正工数が大きくなった (10.86 人時). この原因は,「試験すべき機能として認識してはいたが, 結合試験や総合試験まで試験環境が整わないために試験ができず, 結果的に障害の発見が遅れた」というケースを示している. この結果からも, 試験の一部を後回しにすることは, 開発工数増大のリスクとなりうることが伺える. 次に平均修正工数が大きかったのは「試験項目抽出漏れ」であり, 全障害の平均修正工数の約 3 倍を要していた.

[障害原因] 件数は 7 件と少ないものの, 障害が指摘ミス（仕様通り）のとき平均修正工数が小さかった (0.75 人時).「指摘ミス」とは, 障害として報告されたが, 報告そのものが誤りで, 実際には障害でなかったケースである. 他の原因（論理エラー, 演算エラー, データ処理問題など多数）は, 修正工数との有意な関係が認められなかった. また「その他」カテゴリの障害は, 修正工数が小さい傾向にあったが, その内訳は不明である.

[再現度] 再現度が「小」のときに平均修正工数が大きかった (3.70 人時). 再現性の低い障害は（バグ位置が把握しづらく）デバッグに時間がかかったのではないかと考えられる. また,「再現なし」では修正工数が小さかったが, 大部分はレビューで発見されたバグであった. レビューで発見されたバグは, テスト（実行）による発見ではないため「再現」という概念がなく, また, 通常回帰テストを行わないため, 修正確認の工数が低かったと思われる.

[重要度, 優先度] 重要度が「軽微」のときに平均修正工数がやや小さかった (0.88 人時). 重要度が「重大」と「中度」の差はみられなかった. インタビューの結果から, 発見者が, 修正箇所や修正方法が分かりやすいと判断した障害に対して, 重要度「軽微」を選択する場合があることがわかった. 一方, 優先度と障害工数の間に関係は見られなかった.

[発見作業] 発見作業がレビュー（コードレビュー）であった場合に平均修正工数がやや小さかった (0.99 人時). [再現度] の項に述べたとおり, レビューで発見されたバグは回帰テストを必要としないためと思われる.

[その他の変数] 残りの 2 つの変数「発見箇所」と「機能」は修正工数との関係が認められなかった.

「発見箇所」については, ほとんどが「新規」開発部分からの障害で,「改造」「再利用」部分からの障害がほとんどなかったため有意差が出なかったと考えられる. システムとしては, 改造や再利用部分がかなり含まれているが, 大部分は別の信頼性の高いシステムからの流用であったため障害の発生が少なかった.

「機能」については,「データ編集機能」で最も数多くの障害が発見され, 続いて「データ出力機能」,「演算機能」,「入力データチェック機能」の順に障害が多かったが, 各カテゴリの平均修正工数には有意差がなかった.

4.2 変数間の関係

混入工程と発見工程の全ての組み合わせについて障害の平均修正工数を算出し, 比較を行った. **表 5** は, 3 つの発見工程「コーディング／単体試験」「結合試験」「総合試験」のそれぞれについて, 混入工程ごとの障害件数, 他の混入工程との平均値の差の検定 (t 検定)

表 5 発見工程別混入工程の t 検定の結果
Table 5 Table of Result of t-Examination of Phases Where Defects Were Introduced in Each Phase Where Defects Were Detected

説明変数	カテゴリ	障害件数	p 値	平均修正工数
発見工程：コーディング/単体試験				
混入工程	詳細設計	47	0.307	2.74
	コーディング/単体テスト工程	587	0.911	1.64
発見工程：結合試験				
混入工程	詳細設計	33	0.000	5.50
	コーディング/単体テスト工程	171	0.016	2.02
	結合テスト工程	50	0.160	2.53
発見工程：総合試験				
混入工程	詳細設計	5	0.797	4.80
	コーディング/単体テスト工程	21	0.143	10.74
	結合テスト工程	6	0.388	3.67

を行った結果 (p 値)，及び，平均修正工数を示す．

前節同様，有意水準を 1%とすると，表 5 より，発見工程が「コーディング／単体試験」または「総合試験」の場合は，混入工程の全てのカテゴリで $p > 0.01$ であり，混入工程の違いは修正工数に有意な影響を与えているとはいえない．一方，発見工程が「結合試験」のとき，混入工程が「詳細設計」の場合に平均修正工数が有意に大きかった (5.50 人時)．

前節の結果からは，上流工程 (詳細設計) で混入した障害は，より下流のコーディング／単体試験で混入したものより修正に多くの工数を要していた (平均 3.89 人時) が，表 5 より，詳細設計で混入した障害であっても，すぐ次の工程であるコーディング/単体試験で発見された場合には，修正工数は大きくない (平均 2.74 人時)．しかし，さらに次の工程である結合試験まで発見が持ち越されると，修正工数が大きくなった (平均 5.50 人時)．一方，結合試験で見つかった障害であっても，その一つ前の工程であるコーディング／単体試験で混入したものは，修正工数は小さかった (平均 2.02 人時)．これは，発見遅延が 2 工程以上になると修正工数が増大するという前節の結果にも一致する．

また，前節より，総合試験で発見された障害は，より上流の工程で発見された障害よりも多くの修正工数を要していた (平均 8.29 人時) が，表 5 より，この現象は，障害の混入工程に関わらず確認された．このことから，いずれの工程で混入した障害も，総合試験まで発見が持ち越されると，修正工数が平均的に増大するといえる．

以上の結果より，本プロジェクトに関して，次のことが示唆される．

- 設計時に混入した障害 (設計バグ) は，遅くとも単体試験で発見することが望ましい．結合試験まで発見が持ち越されると修正工数が増大する．
- コーディングで混入した障害は，コードレビューや単体試験で発見することが望ましいが，結合試験まで発見を持ち越したとしても致命的ではない（それほど修正工数は大きくならない）．しかし，総合試験では修正工数が著しく増大する．ソースコードが複雑であるほどコーディング上の誤りが混入しやすいことを考慮すると，ソースコードの複雑さに基づくテスト計画[13] が有用となる可能性がある．
- 結合試験で障害を見逃すことは開発工数増大の原因となる．いずれの工程で混入した障害も，総合試験まで発見が持ち越されると，修正工数が著しく増大する．

さらに，総合試験で発見された障害の修正工数が大きい原因について，開発関係者へのインタビューなどから次のことが分かった．

- 総合試験で障害が発見された場合には，その原因がどの拠点のプログラムであるかをまず調査する必要があり，その調査に時間を要する場合があった．
- 発見された障害が複数拠点のプログラムにまたがるものであった場合，プログラムの変更に関して拠点間で調整が必要となり，時間を要した．
- 総合試験で発見・修正された障害は，再び総合試験を行って障害が修正されていることを確認する必要があるが，この回帰試験により多くの工数を要した．総合試験は，単体試験や結合試験と比べて試験環境構築のためのオーバーヘッドが大きくなるためである．

これらの事情は，マルチベンダー開発に典型的なものであることもインタビュー結果から分かった．

5. 関連研究

障害の修正工数を分析した研究として，IBM 社における 1976 年の Fagan の分析[11] GTE 社における 1977 年の Daly の分析[9]，および，TRW における 1974 年の Boehm の分析[4] がある．**図 4** は，これらの事例をまとめたもの[6] に，本論文の結果を追記したものである．図の横軸は，障害の発見工程であり，縦軸は，コーディング工程における修正工数を 10 としたときの，相対修正工数であり，対数表現となっている．

図中，実線は，TRW 社における複数プロジェクトの事例を示し，破線は，Boehm による 2 つの小規模

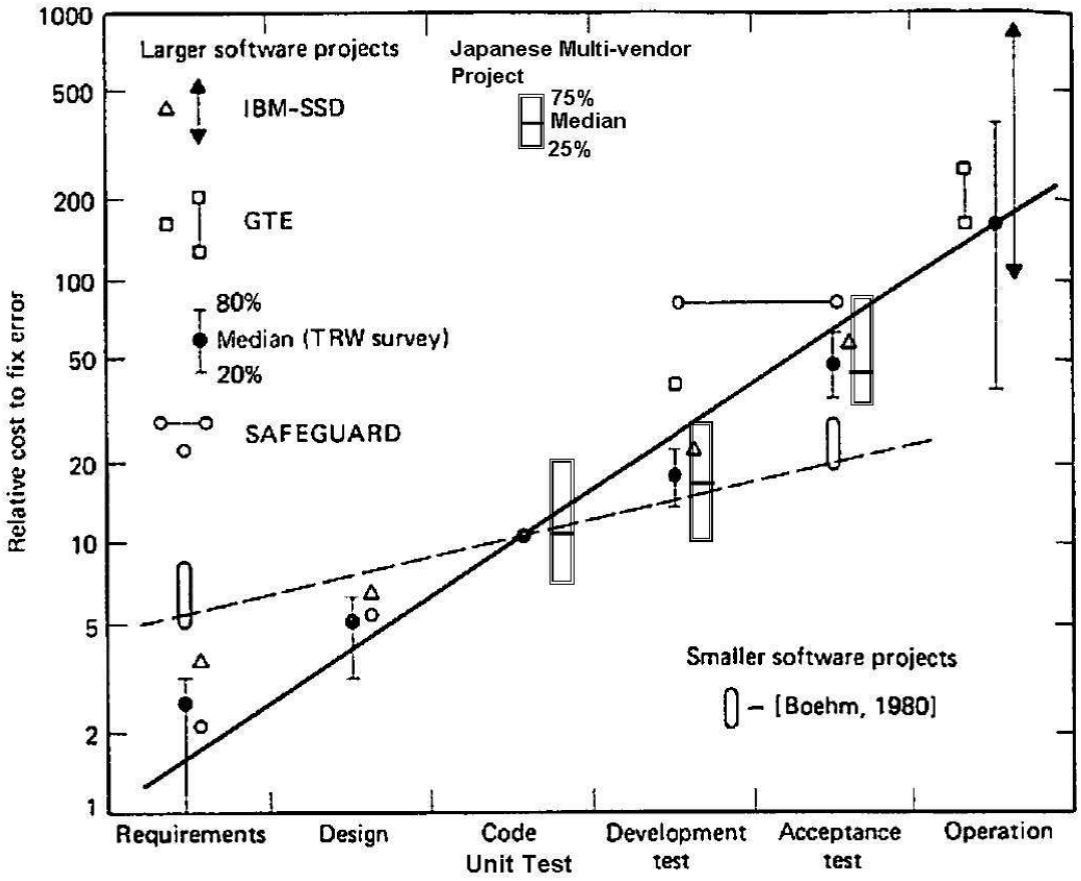


図 4 Boehm のグラフと日本のマルチベンダプロジェクトの比較
Fig. 4 Bohem's Result Vs. Janapnese Multi-Vendor Project

アプリケーション開発の事例を示している[5]．図中の Japanese Multi-vendor project は，本論文の結果である．

この図より，Boehm は次のことを述べている．

- 大規模開発 (実線) では，要求仕様のバグが製品出荷後に発見されると，要求分析工程で発見された場合と比べて，修正工数が 100 倍にも達する．
- 小規模開発 (破線) では，その傾向は弱まるが，結合テスト工程で発見されたバグの相対工数は，要求分析工程で発見された場合と比べて４倍になる．

本論文の結果は，これらの事例に反したものではなく，大規模開発の事例と小規模開発の事例の中間の値をとることがわかった．このことから，障害発見工程と修正工数の（相対的な）関係は，開発環境が進歩した今日の事例においても，1970 年代と同様の傾向を示すことがわかった．ただし，これら過去の事例では，いずれも工程間の相対的な修正工数，もしくはプロジェクト全体の工数に対する相対的な修正工数で述べており，具体的な修正工数を明らかにしていない．修正工数の計測方法やその計測結果についても，具体的に記述されていないものも多い[9)12)]．Boehm の計測は学生実験で各自が報告する週次の作業工数（設計，コーディング，テスト，バグ修正など）から概算した値に基づいており[5]，本論文のように障害単位での修正工数に基づいた分析ではない．Daly[9] と Heimann[12] は，発見工程別のバグ発見工数（Finding Effort）について言及しているが，いずれも混入工程との関係や再現度，重要度，発見遅延の原因などについて言及されていない．

6. まとめと今後の課題

本論文では，障害の修正工数に寄与する要因を明らかにするために，日本のあるマルチベンダー中規模情報システム開発を対象とし，設計から総合試験までの 6ヶ月間に生じた障害の特性値を収集し，修正工数に寄与する要因を分析した．

分析の結果，得られた主な知見は下記の通りである．

- 「障害の発見が遅れるほど修正工数が増大する」という従来の法則が定量的に裏付けられた．コーディング／単体試験で発見された障害の修正工数が平均 1.70 人時であったのに対して，総合試験で発見された場合に平均 8.29 人時と約 4.88 倍になった．
- 「障害が滞留する時間が長ければ長いほど，障害の修正コストが高くなる」という法則も裏付けられた．発見遅延（すなわち，混入工程と発見工程の差）が 0 工程と 1 工程とでは平均修正工数にほとんど差がないが，2 工程になると急激に増大した (7.54 人時)．一方,「発見遅延の原因」に着目すると,「環境上の問題で後工程に持っていった」場合に，平均修正工数が大きくなった (10.86 人時)．
- 混入工程と発見工程の関係については，混入工程にかかわらず，発見工程がコーディング／単体試験工程の場合に修正工数が小さく (1.70 人時)，総合試験工程の場合に大きい (8.29 人時) ことがわかった．一方，結合試験工程で発見された障害は，詳細設計工程で混入した障害について修正工数が著しく大きい (5.50 人時) ことがわかった．これらの結果から，詳細設計工程で混入した障害は単体試験工程までに発見されることが望ましく，コーディング工程で混入した障害は結合試験工程までに発見されることが望ましいといえる．
- 障害の再現度や重要度も修正工数に影響する要因であることがわかった．再現度が「小」のときに平均修正工数が大きく (3.70 人時)，重要度が「軽微」のときに平均修正工数がやや小さかった (0.88 人時)．

これらの結果は，より効率的な，すなわちより投資対効果 (ROI) の高い開発プロセスの改善方法を示唆している．例えば，設計工程で混入した欠陥を結合試験前に検出することによって修正コストを低減できるため，設計レビューをきちんとやることの重要性が確認できた．一方，もしテスト環境の問題で実施を後工程に回すテストケースがある場合，あらかじめマネージャは障害発生時のリスクを多めに見積もっておくべきである．このような実データの基づく分析結果を開発組織に示すことで，プロセス改善の大きな動機付けになると期待される．

本論文では，設計工程以降混入した障害を対象とした分析を行ったが，要件定義工程で混入した障害を収集・分析できなかった．しかし，要件定義に起因する障害はより修正工数の増大が見込まれることから，これについては今後の重要な課題である．また，他のプロジェクトについても同様のデータ収集・分析を行い，本論文の結果の一般性を確認していくことも，今後の重要な課題である．

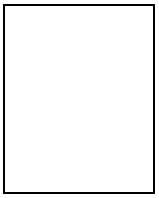
謝辞 本研究の一部は文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた．本研究にあたり多くのご協力を頂いたソフトウェアエンジニアリング技術研究組合参加企業，ソフトウェアエンジニアリングセンター，EASE プロジェクトの関係諸氏に感謝します．

参 考 文 献

1) *IEEE Standard 1044-1993 IEEE Standard Classification for Software Anomalies* (1993).
2) Bassin, K.A., Kratschmer, T. and Santhanam, P.: Objectively evaluating software development, *IEEE Software*, Vol.15, No.6, pp.66–74 (1998).
3) Bhandari, I., Halliday, M., Traver, E., Brown, D., Chaar, J. and Chillarege, R.: A Case Study of Software Process improvement During Development, *IEEE Trans. on Software Engineering*, Vol.19, No.12, pp.1157–1170 (1993).
4) Boehm, B. W.: Software engineering, *IEEE Trans. on Computers*, Vol.25, No.12, pp.1226–1241 (1976).
5) Boehm, B. W.: Developing small-scale application software product: some experimental results, *IFIP(International Federation for Information Processing) Congress*, pp. 321–326 (1980).
6) Boehm, B. W.: *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice-Hall (1981).
7) Card, D.: Learning from our Mistakes with Defect Causal Analysis, *IEEE Software*, Vol.15, No.1, pp.56–63 (1998).
8) Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B. and Wong, M.: Orthogonal Defect Classification-A Concept for in-Process, *IEEE Trans. on Software Engineering*, Vol.18, No.11, pp.943–956 (1992).
9) Daly, E.: Management of software developmentRelationships, *IEEE Trans. on Software Engineering*, Vol.3, No.3, pp.229–242 (1977).
10) Endres, A. and Rombach, D.: *A Handbook of Software and Systems Engineering, Empirical Observations, Laws and Theories*, Pearson Education Limited, UK (2003).
11) Fagan, M.E.: Design and code inspections to reduce errors on program developmenet, *IBM Systems Journal*, Vol.15, No.3 (1976).

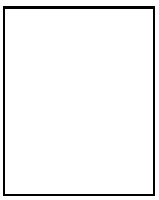
12) Hiemann, P.: A new look at the program development process, *In Programming Methodology, Lecture Notes in Computer Science*, No.23, pp.11–37 (1974).
13) Li, P.L., Herbsleb, J., Shaw, M. and Robinson, B.: Experiences and Results from Initiating Field Defect Prediction and Product Test Prioritization Efforts at ABB Inc., *ICSE '06: Proceedings of 28th International Conference on Software Engineering*, pp.413–422 (2006).
14) Mays, R., Jones, C., Holloway, G. and Studinski, D.: Experience with defect prevention, *IBM Systems Journal*, Vol.29, No.1 (1990).
15) Nakajo, T. and Kume, H.: A Case History Analysis of Software Error Cause-Effect Relationships, *IEEE Trans. on Software Engineering*, Vol.17, No.8, pp.830–838 (1991).
16) 大平雅雄，横森励士，阪井 誠，岩村 聡，小野英治，新海 平，横川智教：ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム，電子情報通信学会論文誌 D-I， Vol.J88-D-I, No.2, pp.228–239 (2005).

(平成 17 年 11 月 18 日受付)
(平成 18 年 2 月 4 日採録)



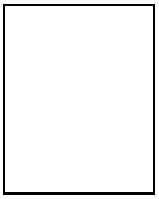
松村 知子

平成 16 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年より同大情報科学研究科/EASE プロジェクト研究員．博士（工学）．エンピリカルソフトウェア工学研究，主に文部科学省委託研究プロジェクトにおいて産学官連携による定量データ計測に基づくソフトウェア開発プロジェクト管理支援の研究に従事．電子情報通信学会，IEEE 各会員．



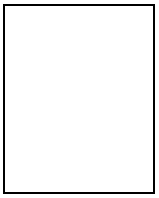
門田 暁人（正会員）

平成 6 年名古屋大学工学部電気学科卒業．平成 10 年奈良先端科学技術大学院大学博士後期課程修了．同年同大情報科学研究科助手．平成 16 年同大情報科学研究科助教授．平成 15〜16 年 Auckland 大客員研究員．博士（工学）．定量的ソフトウェア開発支援，ソフトウェアプロテクションなどの研究に従事．電子情報通信学会，日本ソフトウェア科学会，教育システム情報学会，IEEE，ACM 各会員．



森崎 修司（正会員）

平成 13 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年株式会社インターネットイニシアティブ入社．オンラインストレージサービスの立ち上げ/企画/開発，EPCglobal で RFID ソフトウェアの国際標準策定活動に従事．平成 17 年 EASE プロジェクト/奈良先端科学技術大学院大学情報科学研究科研究員．博士(工学)．エンピリカルソフトウェア工学，ネットワークを通じた知識共有の研究に従事．



松本 健一（正会員）

昭和 60 年大阪大学基礎工学部情報工学科卒業．平成元年同大大学院博士課程中退．同年同大基礎工学部情報工学科助手．平成 5 年奈良先端科学技術大学院大学情報科学研究科助教授．平成 13 年同大情報科学研究科教授．博士 (工学)．ソフトウェア品質保証，ユーザインタフェース，ソフトウェアプロセス等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．