博士論文

ソフトウェア品質マップによる
高品質ソフトウェア開発のための
定量的品質マネジメント技術に関する研究

2017 年　1 月

佐　藤　孝　司

目次

第 1 章　序論

1.1　本研究の背景

近年，IoT (Internet of Things) などのセンサー機器などから得た情報のネットワーク化やビッグデータによる情報分析に加え，情報セキュリティが一連のシステムとして開発・構築されるなど，社会情報基盤を支える IT システムは，ますます複雑化・大規模化の傾向にある．このシステムに障害が発生したときの社会的な影響を考慮すると，IT システムを提供する IT ベンダーは，従来にも増して，さらなる高度な信頼性が要求される．

一方，IT システムの開発の側面においても，ソフトウェア開発が，OSS (Open Source Software) をはじめとするソフトウェア部品等の流用と既存部の改造の組み合わせによる開発が増加する傾向にあり，成果物の構成要素の複雑化とブラックボックス化による改造の品質リスクや，オフショア開発の増加などの分散開発の拡大により現場が遠くに離れてよく見えなくなる管理上のリスクが増大している．

複雑化・大規模化する IT システム開発では，出荷後の 1 件の品質問題（バグ）が社会に大きな影響を与えることから，このようにますます品質確保が厳しくなる開発環境においても，高品質のソフトウェア開発が必須である．

しかし，独立行政法人情報処理推進機構(IPA)による重要インフラ等のシステム障害対策の報告[1-1]では，報道された IT サービス障害の発生件数が 2009 年から 2015 年にかけて増加傾向にあり，特に 2015 年度は，マイナンバー関連等のシステム稼働直後の初期障害が多く発生しているなど，重要な社会インフラシステムにおける障害が増加しているのが現状である(図 1-1).

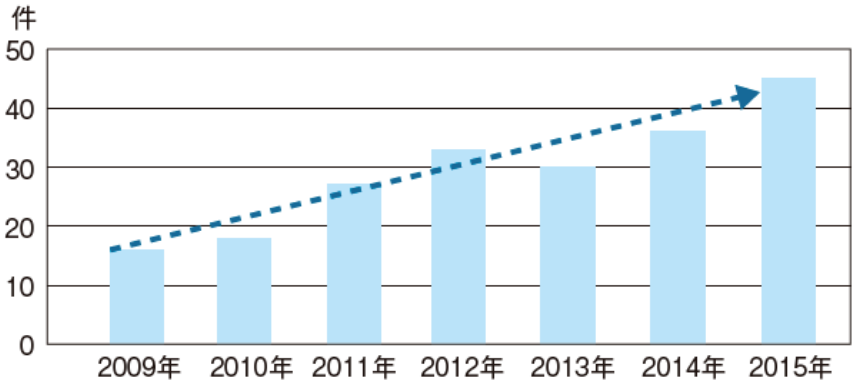


図 1-1　報道された IT サービス障害の発生件数の推移

また，システムがダウンするなどの現象の場合は，直ちに不具合が検知される．しかし，システムは一見正常に運用されているが，実は重大な問題を抱えたままで誰も気がつかな

い中で誤りが生じているようなケースはやっかいである．このようなシステムの開発，保守，運用に携わる組織の責任は極めて重い[1-1]ため，このようにますます複雑化・大規模化したシステムの高品質ソフトウェア開発は，極めて重要である．

なお，本研究において，「バグ」という用語は，「要求された機能を遂行する機能単位の能力の，縮退または喪失を引き起こす，異常な状態」（JIS X 0014:1999 で定義する「障害（Fault）」，「不具合」，および「欠陥(Defect)」と同等）という意味で使用する．

1.2 本研究の目的

「測定できない事柄を，コントロールするわけにはいかない」．アメリカ合衆国のソフトウェア工学者であるトム・デマルコの言葉である[1-2]．現状や推移を知りたければ，何か「基準」や「ものさし」が必要である．このとき，測定した結果をどのようにマネジメントとして制御するのかが，さらに重要であるといえる．一般的に，ソフトウェア開発組織として安定的にソフトウェアを供給するために，また，開発中に検出される品質問題の後戻り作業の低減や，出荷後に品質問題の発生によりお客さまにご迷惑をかけることを極限に抑えるために，ソフトウェア開発組織は品質マネジメントシステムを構築し，ソフトウェア品質管理のための開発プロセスの見える化や，組織的に品質を向上させるための定量的品質マネジメントや組織的なプロセス改善活動などが提唱され，各企業や組織において実践されている[1-3][1-4][1-5][1-6][1-7].

特に， ソフトウェアの中に潜在する「バグ」は目に見えないため，定量的な品質マネジメントにより，「測る」ことと，測った結果から科学的に「判断」することが重要である．ともすれば，定量的品質マネジメントの重要さをはき違えて「測る」ことのみに重点を置きすぎて，測った結果を科学的に「判断」しないままソフトウェア開発が進むことが往々にして散見するが，その場合は，「測る」目的や意味を再確認しなければならない．

本研究の目的は，このようにソフトウェア品質を向上させための定量的な品質マネジメントをさらに効果的に推進するには，どのような工夫や新たな取り組みをすべきかについて，実践研究の紹介とその有効性の評価結果をもとに論じることである．定量的品質マネジメントにおける「測る」と「判断」の観点では，筆者の組織は定量的品質マネジメントの推進において数多くの成功や失敗を繰り返してきた長年の経験から，ソフトウェア品質マップという品質分析手法を確立して，測った結果をどのように科学的に「判断」するかについてまず論じ，この手法の効果を検証する．また，「測る」という観点では，品質を評価するための，より効果のあるメトリクスを抽出し検証する．特に，品質マネジメントの成熟した組織においては，開発プロセスが成熟しているため，プロセスメトリクスを「測る」だけでは科学的な「判断」が難しくなる傾向にある．そこで，本研究では，プロダクトメトリクスに焦点をあてて，多方面から検証を行い，効果的なプロダクトメトリクスについて考察する．

これらにより，定量的品質マネジメントの高度化による高品質ソフトウェア開発のあるべき姿を明らかにしていく．

1.3　本研究の対象領域

本研究では，ソフトウェア開発において品質を向上させるための品質マネジメントの領域の中でも，ソフトウェア品質管理手法の新しい提案とその品質分析手法および定量的品質管理で使用するメトリクスの改善について論じている．

したがって，本研究の対象領域は，ソフトウェア品質知識体系ガイド（SQuBOK GuideV2）[1-8]の体系で示すと，「第２章ソフトウェア品質マネジメント」の「2.1 KA:ソフトウェア品質マネジメントシステムの構築と運用」，「2.19 KA:品質分析・評価のマネジメント」，および「第 3 章ソフトウェア品質技術」の「3.1 KA:メトリクス」に相当する．また，プロジェクトマネジメント知識体系（PMBOK®ガイド第 5 版）[1-9]の体系で示すと，「第 8 章プロジェクト品質マネジメント」の「8.2 品質保証」，および「8.3 品質コントロール」に相当する．

1.4　本論文の構成

本論文は，本章を含めて全 10 章で構成されている．

第 2 章は高品質ソフトウェア開発の定量的品質マネジメント技術とは何かについて述べる．第 3 章はプロセス成熟度の高い組織における定量的品質マネジメントの実際と課題を論じる．

この課題への解決策と多方面から検証を行った実践研究を第 4 章から第 8 章で論じる．第 4 章では高品質ソフトウェア開発の定量的品質マネジメント技術を具現化したソフトウェア品質マップを構築し，その概略，手順，実施例，効果分析について論じる．第 5 章から第 8 章までは，定量的品質マネジメントを維持・改善するためのプロダクトメトリクスについて論じる．第 5 章から第 7 章までは，ソースコードの複雑性に関するプロダクトメトリクスの有効性について，多方面の観点における検証結果を論じる．また，第 8 章は，上流工程での有効な品質管理技術である設計レビューに焦点を当て，従来とは異なる新たなプロダクトメトリクスに関する研究結果とその有効性を論じる．

第 9 章は，第 4 章から第 8 章までの実践研究や検証結果を踏まえて，高品質ソフトウェア開発のための定量的品質マネジメント技術の成功要因について考察する．第 10 章は，結論として本研究を総括するとともに残された課題について述べる．

本論文の章構成を図 1-2 に示す．

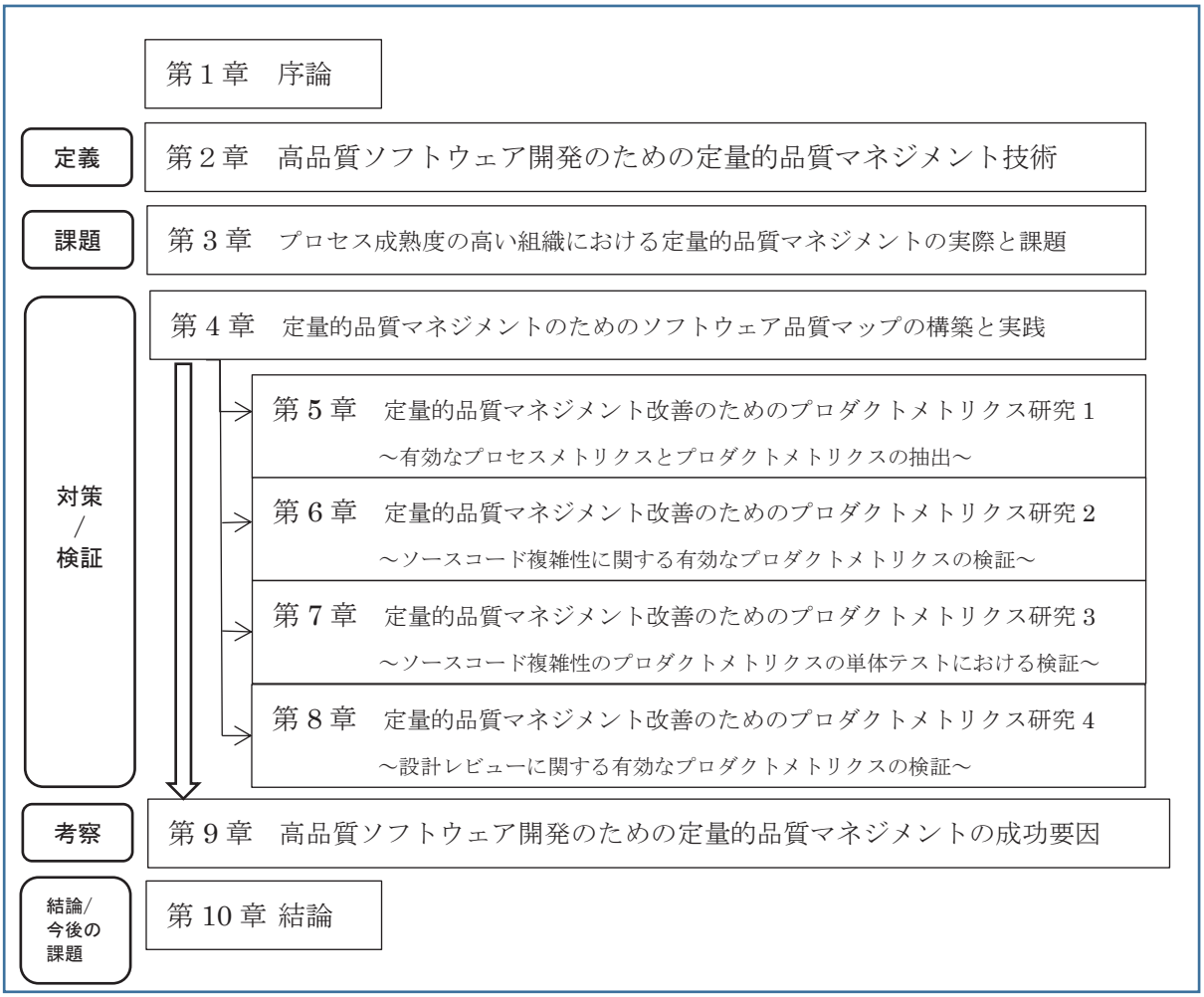

第 1 章　序論
定義
第 2 章　高品質ソフトウェア開発のための定量的品質マネジメント技術
課題
第 3 章　プロセス成熟度の高い組織における定量的品質マネジメントの実際と課題
対策
／
検証
第 4 章　定量的品質マネジメントのためのソフトウェア品質マップの構築と実践
第 5 章　定量的品質マネジメント改善のためのプロダクトメトリクス研究 1
～有効なプロセスメトリクスとプロダクトメトリクスの抽出～
第 6 章　定量的品質マネジメント改善のためのプロダクトメトリクス研究 2
～ソースコード複雑性に関する有効なプロダクトメトリクスの検証～
第 7 章　定量的品質マネジメント改善のためのプロダクトメトリクス研究 3
～ソースコード複雑性のプロダクトメトリクスの単体テストにおける検証～
第 8 章　定量的品質マネジメント改善のためのプロダクトメトリクス研究 4
～設計レビューに関する有効なプロダクトメトリクスの検証～
考察
第 9 章　高品質ソフトウェア開発のための定量的品質マネジメントの成功要因
結論/
今後の
課題
第 10 章 結論

図 1-2　本論文の章構成

第 2 章　高品質ソフトウェア開発のための定量的品質マネジメント技術

2.1　はじめに

本章では，初めに，ソフトウェア開発組織におけるソフトウェア開発の難しさについて，最近のソフトウェア開発組織の事業環境の特徴を踏まえて述べる．次に，ソフトウェア開発を成功させるためのマネジメント技術である品質マネジメントと定量的品質マネジメントについて述べる．次に，日本のソフトウェア開発組織における定量的品質マネジメントに関する実態調査の分析結果から，定量的品質マネジメントの有効性について論じる．最後に，高品質ソフトウェア開発を実現するための定量的品質マネジメントのあるべき姿について論じる．

2.2　組織におけるソフトウェア開発の難しさ

ソフトウェアの開発においては，開発を担当する一人ひとりの開発スキルが直接的に成果物の品質に影響を与える性質をもつ．例えば，コーディングを担当する人は，使用するソフトウェア開発の言語の知識がなければ開発が一歩も進まないことは当たり前である．しかし，品質の良いソフトウェアを開発するために，どれほどの深い開発知識や開発経験のスキルが必要であるかを定義することは難しい．

ソフトウェア開発言語の一通りの文法知識があれば，一見すると要件を満たしているソフトウェアを開発することができるかもしれない．しかし，使用した開発言語や開発対象となるソフトウェア製品の開発経験が浅ければ，本来必要とされる異常系処理を漏らしたり，製品の外部要因となるイベントの発生の有無を判断できずに必要な処理を誤ったりする．さらには，開発者本人以外には解読し難いような癖のあるソースコードや，省略が多くて記述の不十分な仕様書や，将来に改造するときに影響範囲が理解し難い設計書類などが開発の成果物としてそのまま組織内で引き継がれた場合には，引き継がれた新しい開発者は，成果物を理解するのに非常に苦労する．さらに，必要以上に開発工数を費やし，品質劣化のリスクも大きくなる．これは，ソフトウェア開発組織が出荷したソフトウェア製品について，10 年や 20 年の長期間の改造や保守を続けるには大きな問題となる．

また，仮に開発者全員が最高レベルの開発スキルを備えてソフトウェアを開発したとしても，開発対象物の規模や難易度と開発納期等との不整合や，開発における利害関係者と開発者との要件等の認識のずれなどの多くの要因が複雑に絡み合うことも，直接的あるいは間接的に成果物の品質に大きな影響を与える．このように開発する上で多くの複雑な制約がかかるソフトウェア開発組織の中でも，品質や生産性を向上していくためには，ソフトウェア開発が開発者のスキルのみに依存しないように，組織的に開発プロセスを改善することが重要である．

今やソフトウェアの大規模化や複雑化にともない，ソフトウェア開発における品質や生産性の向上の実現がソフトウェア工学における研究の主要な目標に位置付けられている．ソフトウェアの品質や生産性を向上させるためには，開発されたソフトウェアプロダクトだけでなく，その開発プロセスを対象として作業の改善を実施することが必要であることから，ソフトウェアを開発する組織において品質マネジメントは当たり前の概念になっている[2-1].

2.3 品質マネジメント

ISO9000 では，品質マネジメントを「品質に関して組織を指揮し，管理するための調整された活動」と定義している(ISO9000:2005)[2-2]．また，品質マネジメントを有効に実施するためには，プロセスアプローチを採用することを奨励している(ISO9000:2008)[2-2]．このプロセスアプローチとは，組織内で用いられる開発プロセス，および特にそのプロセス間の相互作用を体系的に明確にし，運営管理することである．品質マネジメントシステムにプロセスという考え方を持ち込むことによって，インプット→変換活動→アウトプットという流れでシステム構成要素を整理できるとともに，構成要素間の相互作用を明らかにできる．また，プロセスアプローチにより，組織的に品質マネジメントシステムを改善するために，その有効性を損なう原因となる開発プロセスを特定してプロセスを改善するというサイクルを容易に実行できるようになる[1-8].

ソフトウェア開発における成果物の品質を確保するための品質マネジメントには，ISO9001 に代表されるように一定の開発プロセスを効率よく，かつ，有効に運用することが求められる．QCD(Q:Quality, C:Cost, D:Delivery)は品質マネジメントの基本的な要素であり，ソフトウェア開発における QCD を向上させるためには，開発されたソフトウェア成果物（プロダクト）だけでなく，その成果物を開発するプロセスを対象として，そのプロセスの作業を改善していくことが重要である[1-9].

2.4 定量的品質マネジメント

品質マネジメントは，開発プロセスを定義し運用し改善するサイクルを組織的に確立していくことを目指すが，開発プロセスを確実に運用し，さらには，開発プロセスを経て生産された成果物（プロダクト）の品質を確かなものにするためには，これらのプロセスおよびプロダクトを定量的に測定し判断することが必要になる．定量的品質マネジメントは，科学的な開発方法の代表的なものの一つであり，ソフトウェア開発の多方面で適用可能な方法である．もちろん，定量的品質マネジメントは，ソフトウェア工学における他のさまざまな手法と同様に，銀の弾丸いわゆる特効薬ではない．しかし，程度の差こそあれ確実にプロジ

ェクト管理や組織の成熟度の向上，生産性や信頼性の向上に寄与する最も基本的な手法の一つであるといえる[2-3][2-4].

また，ソフトウェア開発では，定量的品質マネジメントによる定量化が必須である．特に，定量化することによって効果があると思われるのは，プロジェクト計画時のさまざまな見積もり，プロジェクト遂行中の QCD 管理（中間製品の品質管理，コスト管理，進捗管理など），および組織内のソフトウェア開発プロセスの改善などである．これらを定量的に管理することにより，より精度の高いプロジェクト管理が行えるようになり，よりレベルの高いソフトウェア開発プロセスを構築できるようになる．

定量化は，まず対象を測定することによりその姿を正確に把握することから始めなければならない．どのようなメトリクス（測定量と測定方法）を用いて対象を測定すべきかについては，目的によって異なる．目的に応じたメトリクスの選定方法については，すでに GQM[2-5]などさまざまな方法が提案されている．選定によって得られた結果は異なる数値あるいは分類なので，そこから意味のある情報を得るにはさまざまな統計的処理が必要になり，さらに統計的処理を施した結果に対して正しい意味づけを行う必要がある[2-6].

2.5　定量的品質マネジメントのデータ分析結果からみえるソフトウェア開発の現状

独立行政法人情報処理推進機構(IPA)ソフトウェア高信頼化センター(SEC)が毎年調査しているデータ白書[2-7]，2015 年から始まった組み込みソフトウェア系のデータ白書[2-8]，データ白書関連の SEC セミナーのアンケート結果[2-9]，およびメトリクス分析に関するさまざまな検討報告書[2-10][2-11]などから，日本におけるソフトウェア開発の定量的品質マネジメントの現状がみえてくる．

ソフトウェア品質管理の組織内での実施状況[2-9]では，アンケート対象者の約 4 割が定量的管理を実施していない．また，定量的管理の能力については受講者の 6 割以上が不足や懸念を感じている(図 2-1)という結果である．ソフトウェア開発における品質の重要性と，継続的に品質を確保する開発の方法としての定量的品質マネジメントの重要性が定説化されて久しいにも関わらず，寂しい結果と言わざるを得ない．

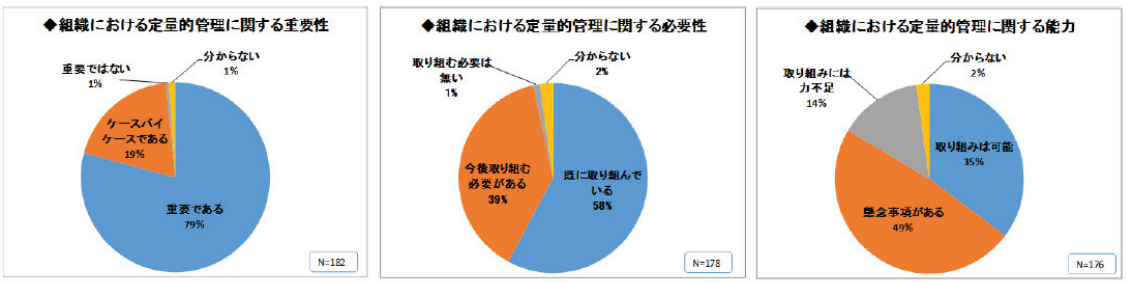


図 2-1 ソフトウェア品質管理に関するアンケート結果

さらに，ソフトウェアの定量的管理の効果について，定量的管理の実施頻度が高く，品質保証プロセスの収集データ項目が多い方が，相対的に信頼性が高い傾向が見られる（中央値で約 1.6 倍の差）．さらに，定量的管理を実施している場合は，実施していない場合と比較して，プロジェクト成功率が約 2 倍高い．また，計画どおりにプロジェクト完了する割合が高い(図 2-2,図 2-3,および図 2-4 を参照[2-12])という結果であり，定量的品質マネジメントは品質向上やプロジェクト成功に有効であるといえる．なお，本章で引用している図の「不具合」とは，「バグ」と同意である．また，図 2-2 の KFP とは，1,000FP（ファンクションポイント）である．

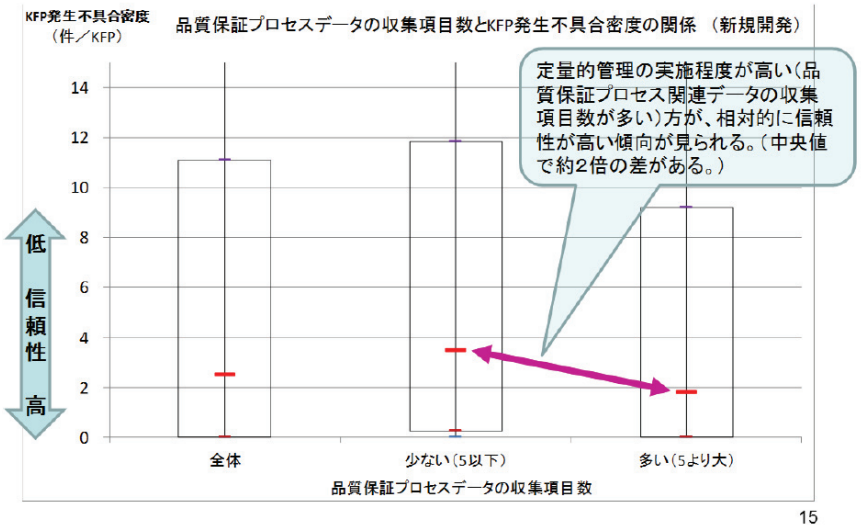


図 2-2 品質保証プロセスデータの収集項目数と KFP 発生不具合密度の関係

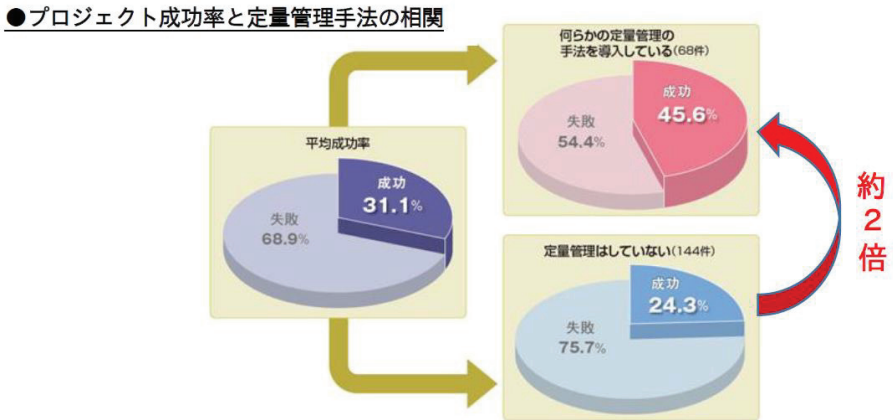


図 2-3 プロジェクト成功率と定量管理手法の相関

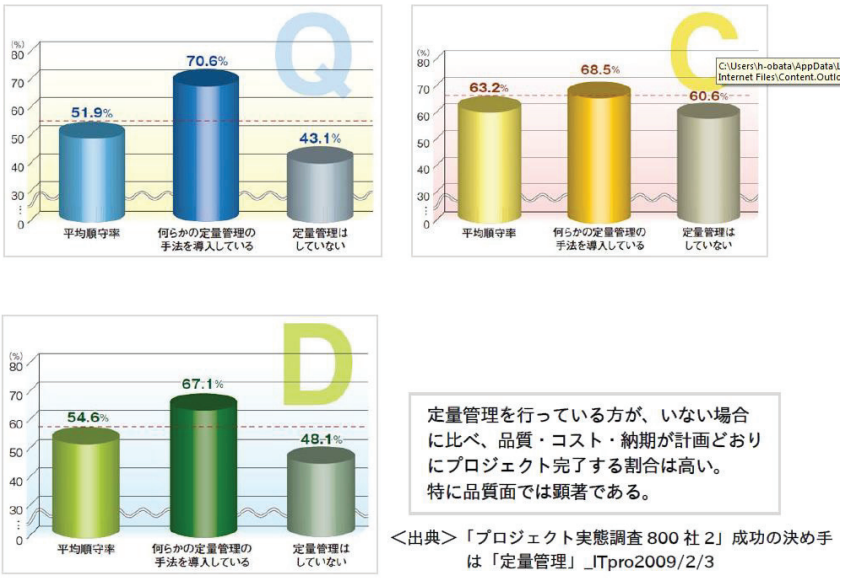


図 2-4 定量管理の実施有無と品質・コスト・納期の遵守率

では，定量的管理といっても具体的にはどのようなメトリクスを活用して，どのような効果を得られているのかについて，前述の IPA の調査結果を考察する．

まず，定量的管理のもととなるメトリクスについて，どのような種類があるのか，候補の一覧を表 2-1[2-10]に示す．基本的には，上流工程からテスト工程まで一貫して，費やした工数と対象となる成果物規模を正規化のベースとした摘出されたバグ数（上流工程におけるレビューで摘出されたバグを含む）と，バグを摘出するために費やした量（レビュー工数，テスト工数，およびテスト項目数）を比較することで，当該工程における品質の程度を測っていることがわかる．

表 2-1 メトリクスの候補一覧表

分類	メトリクス （括弧内は本章で使用する別の表記）	意味	目的の例
上流工程	レビュー摘出バグ数/レビュー工数	レビュー工数当たりの摘出バグ数	レビューの十分性，効率性を評価
	レビュー摘出バグ数/成果物規模	成果物規模当たりの摘出バグ数	レビューの十分性，有効性を評価
	レビュー工数/成果物規模	成果物規模当たりのレビュー工数	レビュー工数の十分性を評価
	レビュー工数/開発工数 （設計レビュー工数比率）	開発工数に対するレビュー工数の比率	レビュー工数の十分性を評価
	上流工程のバグ摘出率 （上流工程の不具合摘出比率）	全工程における摘出バグ数に対する，上流工程におけるレビュー摘出バグ数	レビューの有効性を評価
	成果物規模/レビュー時間	レビュー時間当たりの成果物規模	レビューの効率性，適切性を評価
	開発工数/成果物規模	成果物規模当たりの開発工数	開発作業への投入工数の十分性，生産性を評価
	レビュー工程別のバグ除去率	摘出バグ数＋見逃しバグ数に対する，見逃しバグ数の比率	レビューの有効性を評価
テスト工程	テスト摘出バグ数/テスト工数	テスト工数当たりの摘出バグ数	テストの十分性，効率性を評価
	テスト摘出バグ数/テスト項目数 （テスト検出能率）	テスト項目数当たりの摘出バグ数	テストの十分性，効率性を評価
	テスト摘出バグ数/成果物規模 （テスト検出不具合密度）	成果物規模当たりの摘出バグ数	テストの十分性を評価
	テスト工数/成果物規模	成果物規模当たりのテスト工数	テストの十分性を評価
	テスト項目数/成果物規模	成果物規模当たりのテスト項目数	テスト項目の十分性を評価
出荷後	出荷後バグ密度 （発生不具合密度）	成果物規模当たりの出荷後に摘出されたバグ数	成果物の品質を評価

次に，メトリクスと品質との関係について述べる．

図 2-5 および図 2-6 では，レビュー工数と品質の関係を示している．レビュー工数のメトリクスには，設計レビュー工数比率（表 2-1 の開発工数当たりのレビュー工数と同意）を使用している．品質の程度を表すメトリクスには，不具合発生密度（出荷後に発生した開発規模当たりのバグ数（表 2-1 の出荷後バグ密度と同意））を使用している．なお，図の KSLOC は，1,000LOC(コード行数)と同意である．

設計レビュー工数比率が低くなるにつれて，不具合発生密度が 1,000LOC(コード行数)当たり 0.05 件以上と高い（信頼性が相対的に低い）ものが増えている．特に，設計レビュー工数比率が 2%未満の領域においてその傾向が顕著である．一方，設計レビュー工数比率が 7%以上の領域では，不具合発生密度が 1,000LOC(コード行数)当たり 0.05 件以上のものは少数である．具体的には，新規開発の場合，不具合発生密度が 1,000LOC(コード行数)当たり 0.05 件以上のものは少数である．上流工程においてレビューをある程度十分に実施することで，品質を確保できるということが検証されている．

ただし，この結果は，ソフトウェアの信頼性が相対的に高いか低いかの検証としては有効であるといえるが，ソフトウェアの信頼性を絶対値で考えた場合に，ソフトウェアを利用しているお客様にとっては，必ずしも有効であるとはいえない．なぜなら，お客様の基幹システムにおいて，たとえ 1 件のバグであっても，お客様の業務に致命的な影響を与える可能性があるからである．このことを，ソフトウェアの開発者は肝に銘じておく必要がある．この検証では，不具合発生密度の閾値を 1,000LOC(コード行数)当たり 0.05 件としているので，20,000LOC(コード行数)当たりに 1 件の不具合（バグ）があるかどうかを相対的に検証していることになる．このお客様にとっての信頼性ついては，次節で論じる．

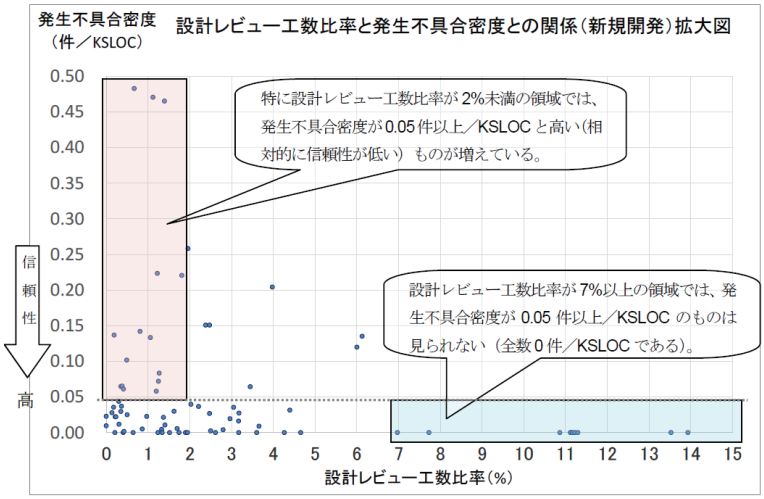


図 2-5 新規開発における設計レビュー工数比率と発生不具合密度との関係

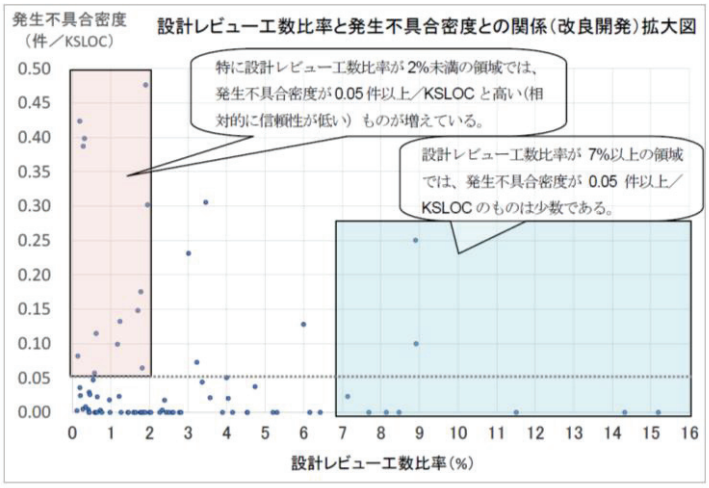


図 2-6 改良開発における設計レビュー工数比率と発生不具合密度との関係

次に，図 2-7 は，上流工程での不具合摘出比率（表 2-1 の上流工程バグ摘出率と同意）と発生不具合密度（表 2-1 の出荷後バグ密度と同意）との関係を示している．ソフトウェアのバグは上流工程（設計工程やコーディング工程）で作り込まれるため，テスト工程でバグを摘出するよりも，上流工程で作り込まれたバグを可能な限り当該工程内で摘出するほうが効率的である．また，バグの修正による開発の後戻り工数の削減や，バグの修正により新たに別のバグを作り込む機会を減少させる点でも重要である．そこで，上流工程において，どれほどのバグを摘出できたかを表す不具合摘出比率（上流工程バグ摘出率）のメトリクスにより，上流工程での品質確保の程度を推測することができる．

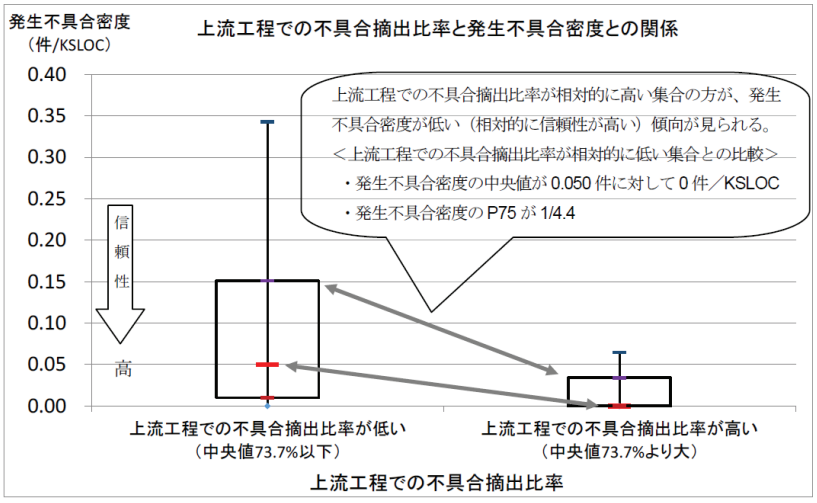


図 2-7 上流工程での不具合摘出比率と発生不具合密度との関係

その結果，新規開発の場合は，不具合摘出比率（上流工程バグ摘出率）が中央値 73.7%以下の相対的に低い集合と，中央値より大きく相対的に高い集合とでは，後者の方に発生不具合密度（出荷後バグ密度）が低い（相対的に信頼性が高い）傾向が見られる．具体的には，新規開発の場合は，前者の発生不具合密度（出荷後バグ密度）の中央値が 1,000LOC（コード行数）当たり約 0.05 件であるのに対して，後者の発生不具合密度（出荷後バグ密度）の中央値は 0 件である．また，発生不具合密度（出荷後バグ密度）の 75 パーセンタイル値については，約 4.4 倍の開きが見られる．対数化後のデータを用いたウェルチ(Welch)の t 検定では，10%有意で差が認められる．なお，改良開発の場合には，このような顕著な傾向は見られなかった．

図 2-8,図 2-9,図 2-10, および図 2-11 は, 発生不具合密度(出荷後バグ密度)が 1,000LOC(コード行数)当たり 0.02 件未満のプロジェクトを良群，1,000LOC(コード行数)当たり 0.02 件以上のプロジェクトを否群に分けて，両者にどのような開発プロセスの差異があるかを分析して，信頼性の変動要因と思われる各種のメトリクスの値にどの程度の差があるかを調べた結果である．この結果，良群には，以下の傾向がみられた．

・レビュー工数密度が高い（1%有意）.
・上流工程不具合摘出比率（上流工程バグ摘出率）が高い(5%有意).
・テスト検出不具合密度（テスト摘出バグ密度）が低い(5%有意).
・テスト検出能率（テスト項目当りの摘出バグ数）が低い(5%有意).

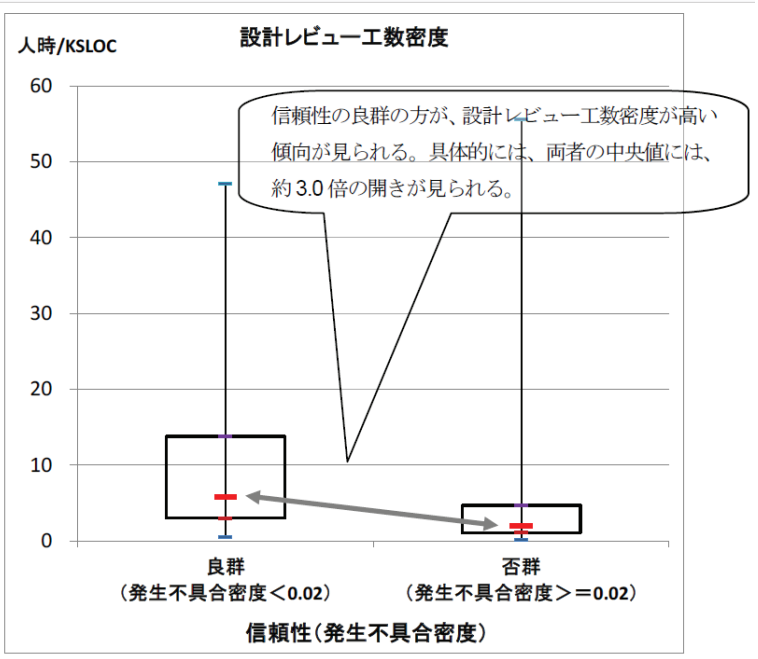


図 2-8 設計レビュー工数密度と発生不具合密度の関係

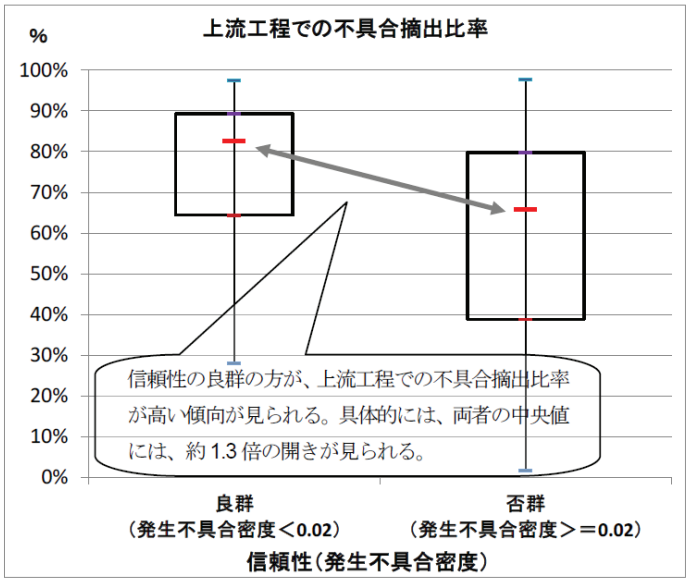


図 2-9 上流工程での不具合摘出比率と発生不具合密度の関係

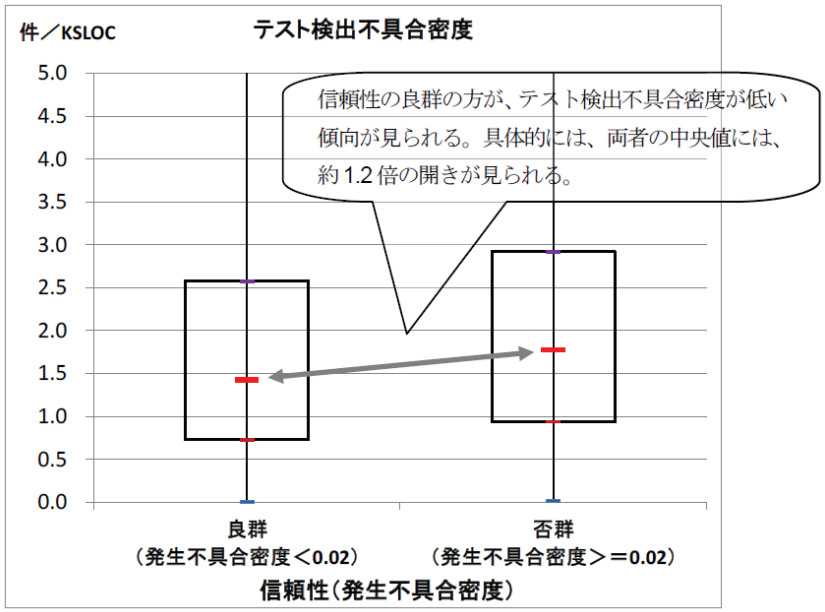


図 2-10 テスト検出不具合密度と発生不具合密度の関係

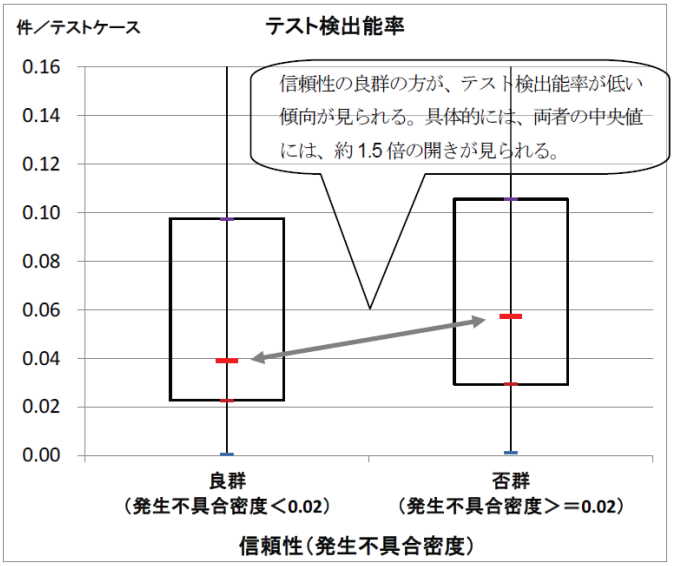


図 2-11 テスト検出能率と発生不具合密度の関係

これらの結果を定量的品質マネジメントの視点でまとめると，ソフトウェア開発の中間地点において，特に上流工程において，レビュー工数とレビューにより摘出したバグ数を測定し，開発規模等により正規化したメトリクス基準値を設定して運用することにより，定量的管理が効果的に推進できるといえる．

2.6 高品質ソフトウェア開発のための定量的品質マネジメント

前述のとおり，上流工程での有効な品質管理技術は設計レビューであり，設計レビューのプロセスを測る主なメトリクスは，レビュー工数密度と上流工程におけるレビューで摘出したバグ密度である．レビューにどれほどの時間をかけて，どれだけのバグを摘出したかについて，測定することが重要である．ソフトウェア開発組織のプロセス成熟度が低い場合には，レビュー工数を掛けた分だけ，レビューにより摘出できるバグ数が増加するので，設計レビューは上流工程での有効な品質管理技術といえる．

しかし，ソフトウェア開発組織のプロセス成熟度が高い場合には，定量的品質マネジメントは確実に実施され安定的なプロセス運用ができるように組織のプロセス能力が成熟しているため，プロセス成熟度が低い組織に比べて，レビュー工数やレビューによるバグの摘出数は，飽和状態になると推測できる．つまり，レビュー工数を限りなく増加させれば，上流工程以降にはバグがゼロになるという単純な論理が開発現場にあてはまるわけではない．開発工数の限度と，バグ検出におけるレビュー技術面での限界があると推測している．特に複雑化・大規模化するシステムでは，既存の機能をすべて理解した上で，外部要因などの全ての起こりうるタイミングなどを机上のレビューで見つけ出すには限界がある．このよう

にソフトウェア開発組織のプロセス成熟度が高い場合においては，さらなる高品質ソフトウェア開発を実現するためには，従来から提唱されているような，「測らなければマネジメントはできない」,「開発プロセスを定量的に測り見える化する」,「上流工程で作りこまれたバグは早い段階で検出する」「レビュー工数をメトリクスとして管理する」などの，一連の考え方は理解され，実践され，効果を出してきたものの，さらなる工夫やレビュー工数以外の有力なメトリクスが必要になる．

したがって，高品質ソフトウェア開発の実現においては，定量的品質マネジメントをさらに高度化する必要がある．定量的品質マネジメントの高度化とは，定量的品質管理で使用されるメトリクスの改善による高度化と，定量的品質管理における品質データ分析の高度化の，2 種類の高度化を推進することであると考えている．本研究の目的では，定量的品質マネジメントは，「測る」ことと，測った結果から科学的に「判断」することと述べたが，定量的品質マネジメントを高度化するということは，メトリクスの改善により「測る」ことを高度化し，品質データ分析手法の改良により測った結果から科学的に「判断」することを高度化すると言い換えることができる．第 4 章から第 8 章において，高度化に向けた定量的品質マネジメント改善の実践研究と検証結果を述べる．

2.7 まとめ

本章では，主にソフトウェア開発組織の実態調査から定量的品質マネジメントの有効性を論じた．独立行政法人情報処理推進機構(SEC)の調査データから日本のソフトウェア開発組織における定量的品質マネジメントの現状をまとめると，以下になる．

1. 定量的管理の実施頻度が高い方が，相対的に信頼性が高い傾向が見られ，プロジェクト成功率が約 2 倍高いという結果から，定量的品質マネジメントが品質向上に有効である．

2. 定量的品質マネジメントのメトリクスと出荷後の製品品質の関係は，レビュー工数密度が高い，上流工程のバグ摘出率が高い，テスト工程の摘出バグ密度が低い，およびテスト検出能率が低いほうが，相対的に出荷後のバグ密度が低く，品質が良い．

最後に，この結果をもとに，高品質ソフトウェア開発を実現するための定量的品質マネジメントはどうあるべきかについて論じた．

第 3 章　プロセス成熟度の高い組織における定量的品質マネジメントの実際と課題

3.1　はじめに

前章では，ソフトウェア開発における製品品質の向上には定量的品質マネジメントが重要であることを述べた．また，日本のソフトウェア開発組織の調査データから品質向上に有効なメトリクスの傾向としては，レビュー工数密度が高い，上流工程における摘出バグ率が高い，テスト摘出バグ密度が低い，およびテスト検出能率が低いことが挙げられた．一方，さらなる高品質ソフトウェアを実現するための定量的品質マネジメントとは，「測る」ことと，測った結果から科学的に「判断」することをさらに高度化することが必要であると課題提起した．

本章においては，筆者の組織における定量的品質マネジメントの実践研究を通じて前章の結果を検証してみる．

初めに，筆者の組織における定量的品質マネジメントを実施するための開発プロセスやメトリクスの仕組みについて取り上げる．また，定量的品質マネジメントの有効な方法として高度化された開発環境，すなわちソフトウェアファクトリについて取り上げる．次に，さまざまなプロセス改善施策の結果，定量的品質マネジメントにおけるプロセスメトリクスやプロダクトメトリクスの推移と品質との関係について論じる．最後に，筆者の組織における定量的品質マネジメントの効果と，さらなる品質向上を目指すための課題について論じる．

3.2　プロセス成熟度の高い組織における定量的品質マネジメントの実際

3.2.1　開発プロセス

筆者の組織が開発するソフトウェアは，企業の基幹システムや社会インフラシステムなどに組み込まれるため，常に高い品質が要求される．この組織では，全社的な品質改善活動[3-1]や，全社標準的な品質管理手法である品質会計[3-2]を約 30 年推進した経験の蓄積により，2002 年にはソフトウェア開発プロセス成熟度モデル統合 CMMI(Capability Maturity Model　Integration)[3-3]レベル 5 を達成した．したがって，この組織のソフトウェア開発プロセスの成熟度は高く，出荷後バグ数を 1/20 に削減するなどの品質改善の成果を出してきた[3-4][3-5]．特に，筆者の組織が独自に開発した品質会計制度[3-2]により，定量的品質マネジメントは早期から開発部門に定着している．このような組織的な活動による定量的品質マネジメントの浸透により，開発プロセスの早い段階から品質を作り込むことで，品質問題に起因する手戻り作業を削減すると同時に，出荷後のお客様システムにて発生する品

質問題も削減してきた.

なお，筆者の組織の開発プロセスを図 3-1 に示す．開発プロセスは，ウォーターフォール開発モデルを適用している．開発プロセスは，基本設計からコーディングまでの上流工程を上工程（かみこうてい）と呼び，単体テストからシステムテストまでをテスト工程と呼ぶ．上工程は，設計書やソースコードのドラフト作成工程（BDD=Basic Design Draft, FDD=Function Design Draft, DDD=Detail Design Draft, CDC=Coding）と設計書やソースコードのレビュー工程（BDR=Basic Design Review, FDR=Function Design Review, DDR=Detail Design Review, CDR=Code Review）で構成されている．

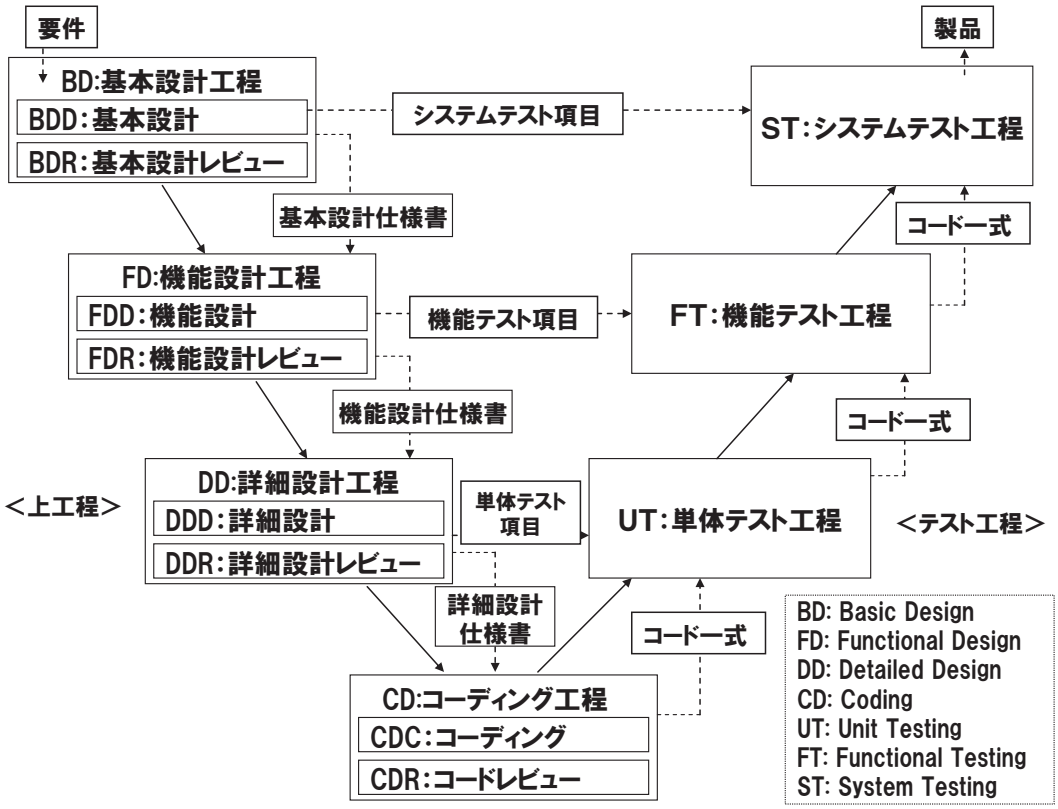


図 3-1 開発プロセス

3.2.2 主なプロセスメトリクスとプロダクトメトリクス

筆者の組織が適用している主なプロセスメトリクスとプロダクトメトリクスを表 3-1 に示す．主なプロセスメトリクスはバグ数とレビュー工数とテスト項目数に関するものである．表 2-1 で示した日本の主なプロセスメトリクスとほぼ同じである．また，主なプロダクトメトリクスは，ソースコードの静的解析ツールにより集計や解析したものである．

表 3-1 主なプロセスメトリクスとプロダクトメトリクス

カテゴリ	No	メトリクス	意味
プロセスメトリクス	1	上工程摘出バグ密度	(上工程（設計，コーディング）のレビューで摘出したバグ数) / 開発規模(KLOC)
	2	レビュー工数密度	(上工程（設計，コーディング）のレビュー（設計レビュー，コードレビュー）工数（人・時間）) / 開発規模（KLOC)
	3	上工程工数密度	(上工程（設計，コーディング，レビュー）の工数（人・時間）) / 開発規模（KLOC)
	4	上工程バグ摘出率	上工程摘出バグ数 / (上工程摘出バグ数+テスト工程摘出バグ数)
	5	テスト工程摘出バグ密度	(テスト工程で摘出したバグ数) / 開発規模（KLOC)
	6	テスト項目数密度	テスト項目数 / 開発規模（KLOC)
	7	テスト工数密度	(テスト工程の工数（人・時間）) / 開発規模（KLOC)
	8	開発時総摘出バグ密度	(上工程摘出バグ数+テスト工程摘出バグ数) / 開発規模（KLOC)
プロダクトメトリクス	9	有効行数	総行数からコメント行数と空白行を除いた値(クラス単位で集計)(LOC)
	10	コメント率（%)	総行数の中でコメント行数の割合 ((コメント行数 / 総行数) * 100)
	11	コード解析アラーム数密度	(コード解析ツールが導出するコード脆弱性や欠陥の可能性のある処理のアラーム数) / 開発規模(KLOC)
	12	サイクロマチック数密度	(分岐命令による経路複雑度を表す数値（クラス単位で集計）) / 開発規模（LOC)
	13	分岐条件数密度	（分岐命令の条件式の数（クラス単位で集計)) / 会開発規模（LOC)
	14	最大ネスティング平均値	各メソッドの最大ネスティング数のクラス平均値

3.2.3 ソフトウェアファクトリによる定量的品質マネジメント

筆者の組織では，ソフトウェアの開発環境を各開発プロジェクトで構築するのではなく，組織内の全ての開発プロジェクトをソフトウェアファクトリと呼ぶ統一したクラウド基盤環境下に集約している．ソフトウェアファクトリの概要を図 3-2 に示す．全ての開発者は，ソフトウェアファクトリが提供する構成管理，アクティビティ管理，プロダクトメトリクス管理，およびプロセスメトリクス管理の機能を一様に利用できる．全ての開発プロジェクトは，ソフトウェアファクトリを利用することにより，プロジェクト個別の開発環境の構築や運用保守のコストが不要になる．また，ソフトウェアファクトリの利用者は，ソフトウェアファクトリが提供する機能を利用することにより，誰もが一定レベル以上の均一したサービスを享受できる．

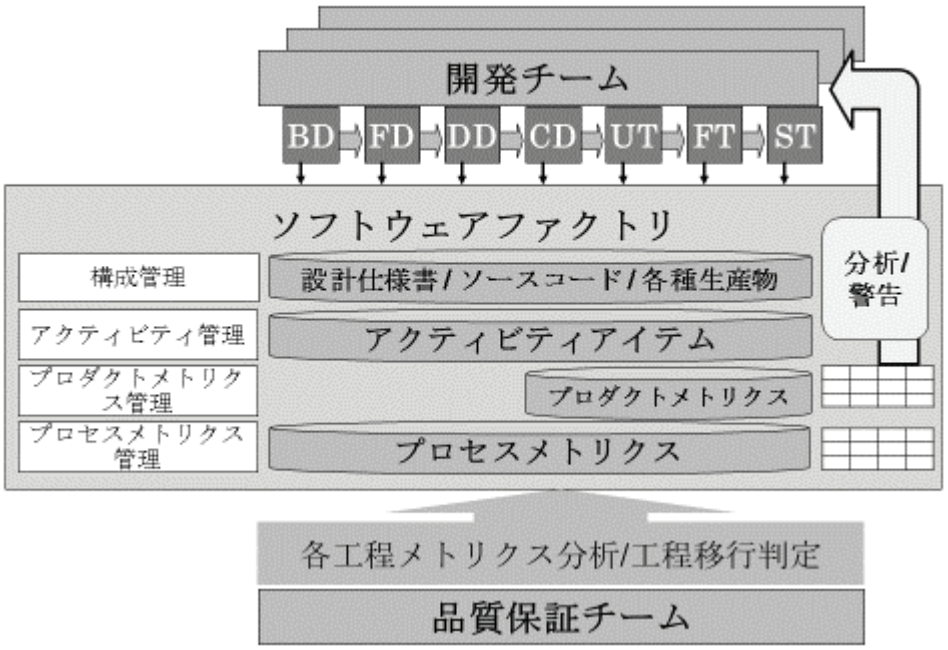


図 3-2 ソフトウェアファクトリ概要

ソフトウェアファクトリは，開発プロセスの計画時から開発終了時までのソフトウェア開発サイクル全期間で利用できる．主なソフトウェアファクトリの機能は以下のとおりである（表 3-2 を参照）．

- 構成管理は，開発者の開発した成果物（ソースコードやドキュメント）を一元的に集約して管理する．成果物の版管理や複数ファイルの構成管理ができる．構成管理に関するツールを統一して使用している．
- アクティビティ管理は，開発プロジェクトの日々の活動項目を一元的に集約し，現在のプロジェクトの活動状況を把握し管理する．管理する対象は，問題点（インシデント），バグ，アクションアイテムなどである．構成管理で使用しているツールと連携することで，構成管理上の成果物と変更作業との紐付けを可能にする．
- プロダクトメトリクス管理は，主にコーディング工程からテスト工程終盤までの期間で使用している．ソースコードを入力として，開発規模や複雑性を表すメトリクスや脆弱性を検出する解析ツールなどを使って多角的な視点で診断した結果を出力し，これらの出力結果をもとにソースコードを適切に是正することにより品質を向上させる．従来の品質管理プロセスに加えて，このプロダクトメトリクス管理の機能を併用することにより，より強化された品質管理プロセスを適用することができる．
- プロセスメトリクス管理は，ソフトウェア開発ライフサイクル全般で使用している．各工程におけるレビュー工数などのプロセスメトリクスの基準値と収集されたメト

リクスの実績値をチェックし，工程移行時などにプロセスが確実に実施されていることを審査する．

表 3-2 主なソフトウェアファクトリの機能

機能	概要
構成管理	ソースコードや各種仕様書などの開発中の成果物を一括して集約し，関係する開発者全員が参照できる版管理等の構成管理を行う．
アクティビティ管理	開発プロジェクトに関連する日々のアクティビティを集約し，アクティビティのステータスを管理する．インシデント等の問題点や課題の管理やアクションアイテム管理も行う．構成管理機能と連携して作業管理と成果物管理の整合をとる．
プロダクトメトリクス管理	ソースコードを入力として多角的な視点のソースコード解析ツールの診断結果を出力する．ソースコードの規模や複雑性の日々のメトリクスの測定結果や，コード解析診断による脆弱性の指摘結果を管理する．これらのデータをもとに，ソースコードは適切に修正されるように管理される．
プロセスメトリクス管理	各工程におけるメトリクスの基準値と収集されたメトリクスの実績値をチェックし，工程移行時にはプロセスの実施を確実に行っていることを確認し，各工程の成果物の品質が確実に作り込まれていることを管理する．

3.2.4 ソフトウェアファクトリによるメトリクス管理

ソフトウェアファクトリを使用することにより，すべての開発者はプロセスメトリクスのデータ登録とソースコードのチェックアウトを毎日実施する．ソフトウェアファクトリは夜間にプロセスメトリクスの自動集計とソースコードの解析を行い，翌朝までに結果を出す．その結果，ソフトウェアファクトリにより集計・解析されたデータをもとに，開発プロジェクトのリーダや品質保証チームは，開発者と早期にプロセスやプロダクトの状況について弱点分析などの議論ができる．

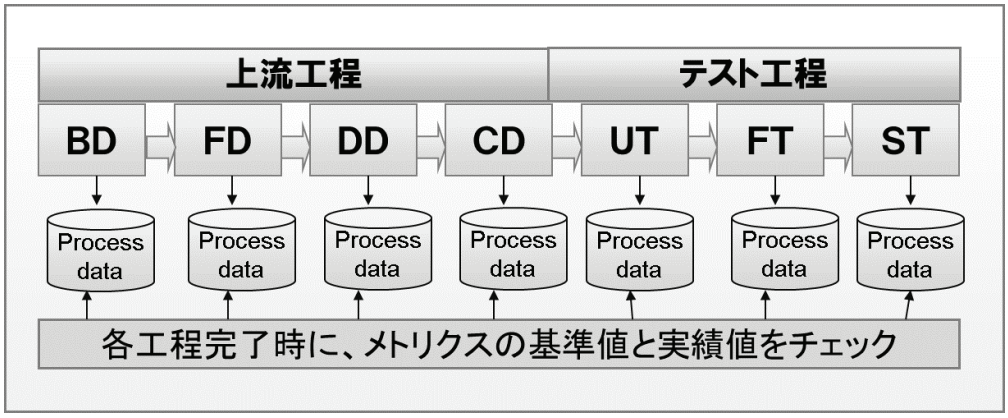


図 3-3 定量的品質管理

ソフトウェアファクトリを活用したプロセスメトリクスやプロダクトメトリクスの品質管理では，各工程の活動から抽出される各種のメトリクスの基準に達しているかどうかについて各工程の移行判定時にチェックされる（図 3-3）．したがって，各工程において実施されるべき品質向上のための作業や成果物が各工程で確実に実施されていることを確認する．できる限り早期にリスクを検出して後戻りの作業を削減するために，開発者や品質保証チームは，ソフトウェアファクトリの活用により短期間のサイクルで品質管理活動を実施することができる．

3.2.5　ソフトウェアファクトリによるプロダクトメトリクス管理

最初に，プロダクトメトリクス管理の運用手順を示す．まず，開発者はソースコードのチェックインとチェックアウトを毎日実施する．プロダクトメトリクス管理は，チェックアウトされたソースコードを入力として，毎日夜間に各種のソースコード診断機能を自動実行して結果を出力する．開発者はチェックアウトした翌日には，プロダクトメトリクス管理が診断した結果をもとに，ソースコードを適切に修正する．プロダクトメトリクス管理の診断機能は，ソースコードの脆弱性の指摘や，プロダクトメトリクスの基準値との比較を実施する．その結果，初期化漏れやバッファオーバーフローの危険があるソースコード，1 モジュールが大きすぎるソースコード，分岐条件・ネスティングの深さ・サイクロマチック数などから導出される複雑すぎるソースコード，およびコメントが少なすぎるソースコードなどを，リスト形式に出力する．ここで出力されたリストは，ソースコードの可読性や保守性の観点において組織で決めた基準を満たしていないため，ソースコードの適切な修正が必要なものである．このリストは，プロダクトメトリクスの組織基準を達していないモジュール一覧や，全てのソースコードのメトリクスの値の中で悪い順にモジュールを表示する．開発者は，毎日の診断結果をもとにソースコードを適切に修正する．また，開発プロジェクトのリーダや品質保証チームは，定期的なプロジェクト会議の中で，診断結果や修正の進行状況を確認し，開発者が適切にソースコードを修正することを促進する．

このようにして，プロダクトメトリクス管理のソースコード自動診断の結果を出力することにより，プロジェクト内の全ての開発者の作成したソースコードの特性が誰からも見える状態にする．開発プロジェクトのリーダや品質保証チームは，プロセスメトリクスを測定することにより，各工程のレビューなどのプロセスが適切に実施されているかを管理している．これに加えて，プロダクトメトリクスを測定することにより，ソースコードの診断結果をもとに全ての開発者の開発スキルレベルを推定することができる．

近年，オフショア開発が増加している．オフショア開発では，日本と比較すると開発者の交替が数多く発生する傾向にある．そのため，開発プロジェクトのリーダは，全ての開発者の技術レベルを把握することは不可能である．そこで，開発プロジェクトのリーダは，プロ

ダクトメトリクス管理が提供するソースコード診断結果を参照して，プロダクトメトリクスが悪い値を示すソースコードを直接参照することにより，オフショア開発の開発者のコーディング技術レベルを容易に推定できるようになる．また，開発者のコーディング技術レベルを一定の水準に引き上げて，開発プロジェクト全体のソースコードを一定のレベルに維持することが可能となる．

このように，従来は開発プロジェクトごとに設定されていた個別のルールでソースコードの可読性・保守性を確保していたが，ソフトウェアファクトリを使うことにより，組織的に統一した基準で可読性・保守性を確保できるようになる．

開発開始時には，開発者全員にプロダクトメトリクス管理の内容を教育する．その結果，開発者は，あらかじめソースコードの適切な書き方や組織特有のルールを理解する．これにより，プロダクトメトリクス管理が推進しやすくなる．

3.2.6　定量的品質マネジメントの効果

ソフトウェアファクトリが構築された 2010 年から，ソフトウェアファクトリを全面的に活用した定量的品質マネジメントの改善を行ってきた．特に，上流工程における設計レビューの定着化のために，プロセスメトリクスの中のレビュー工数密度とレビューにより摘出されたバグ密度について，組織が目標とする基準値を決めて，各工程移行判定時にチェックすることを重点的に実施してきた．図 3-4 は，組織全体のレビュー工数密度とレビュー摘出バグ密度の推移を示す．その結果，プロセス改善活動の実施開始から 2 年後の 2012 年にはレビュー工数密度とレビュー摘出バグ密度は組織の目標に到達し，レビューを確実に実施して上流工程における品質確保のプロセスが確立されてきた．

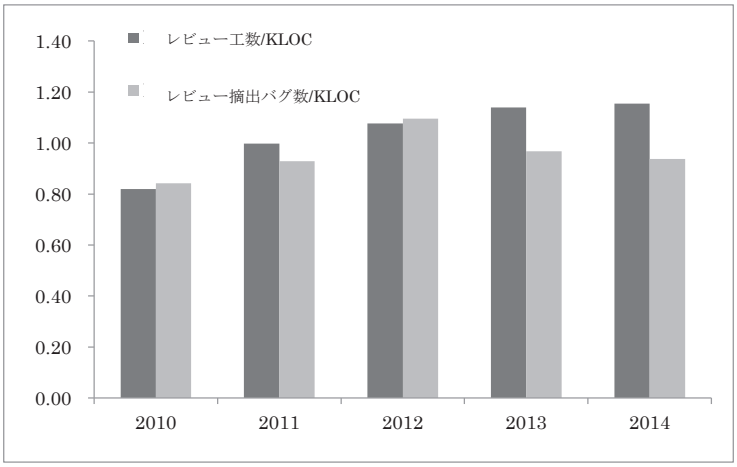


図 3-4 レビュー工数密度とレビュー摘出バグ密度推移
（各目標値に対する実績値比）

図 3-5 は，筆者の組織が開発している製品別の出荷後のバグ数の推移を示す．2010 年には，出荷後のバグ数が組織内で一番多かった（一番品質が悪かった）A 製品は 2014 年にはバグ数が 42%まで減少し出荷後の製品品質が改善した．2 番目に出荷後の製品品質が悪かった B 製品も，2014 年には約 3 倍の品質改善がみられた．この結果，2010 年には，製品ごとに出荷後の製品品質の差が非常に大きかった状態が，2014 年には，極度に品質が悪い製品がなくなり，組織内の出荷後の製品品質は，同じレベルに収斂してきた．

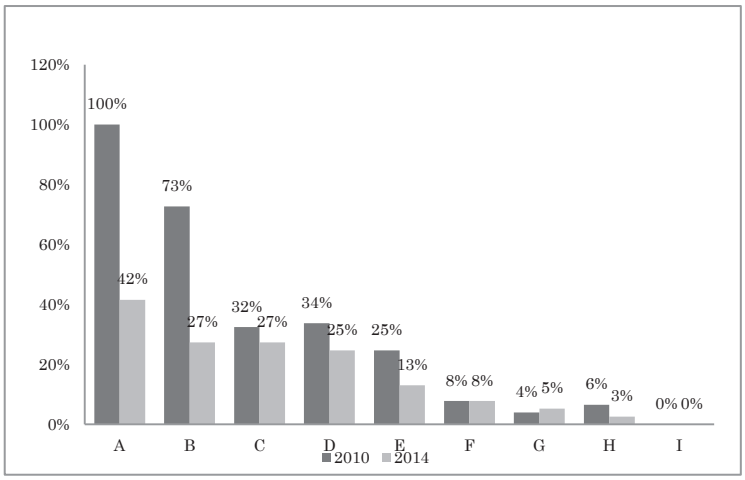


図 3-5 製品別出荷後バグ数の推移
（2010 年製品 A の出荷後バグ数を 100 とする）

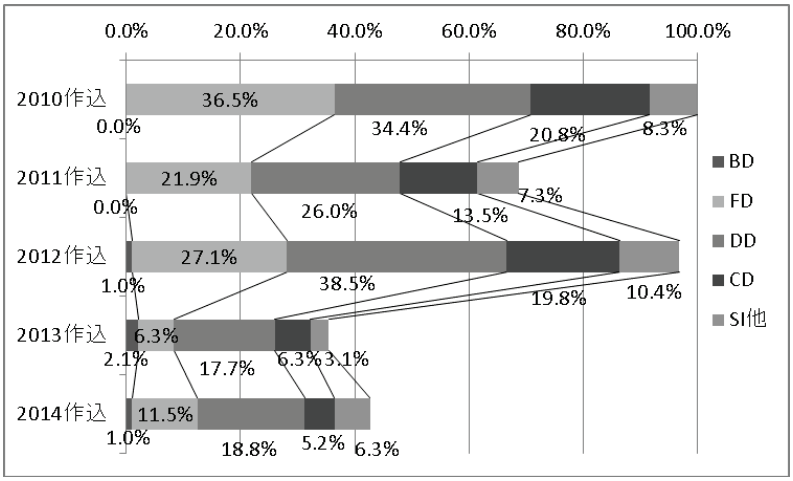


図 3-6 作り込み工程別出荷後バグ数の推移
（2010 年出荷後バグ数全体を 100 とする）

また，図 3-6 は，出荷後バグ数の作り込み原因工程別バグ比率の推移を示している．レビュー工数の確保により全体の出荷後バグ数が減少し，2014 年には 2010 年の約 40%まで出荷後バグ数が改善した．2014 年には，全ての上流工程の作り込み原因工程別のバグ数は減

少しているが，特に，2010 年から開始したソフトウェアファクトリのプロダクトメトリクス管理の推進により，全工程の作り込みバグ数の中で CD 工程の作り込みバグ数の占める割合が，2010 年の 20.8%から 2014 年の 5.2%に，1/4 まで減少し，作り込み工程別の出荷後バグ数の中で一番改善している．

次に，ソフトウェアファクトリを活用することにより，新たな品質向上対策であったプロダクトメトリクスの推移を図 3-7 および表 3-3 に示す．2011 年および 2013 年の各メトリクスの基準値に対する実績値の比率の変化を示している．

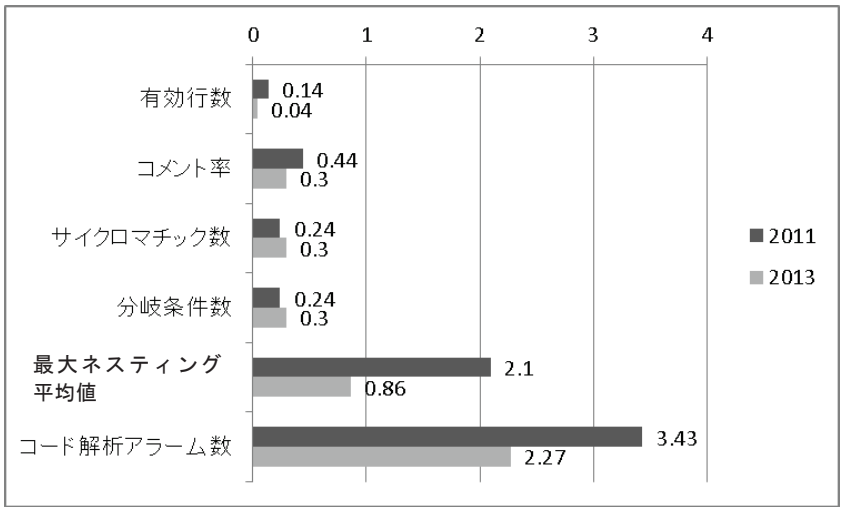


図 3-7 プロダクトメトリクス基準未達率推移
（組織の基準値を 1 とし，基準値を超えた場合には 1 より大きくなる）

表 3-3 プロダクトメトリクス基準未達率推移
（組織基準値を 1 とし，基準値を超えた場合には 1 より大きくなる）

基準未達率/KLOC	2011	推移	2013
有効行数	0.14	73%改善	0.04
コメント率	0.44	32%改善	0.30
サイクロマチック数	0.24	25%悪化	0.30
分岐条件数	0.24	25%悪化	0.3
最大ネスティング平均値	2.10	59%改善	0.86
コード解析アラーム数	3.43	34%改善	2.27

この結果，有効行数，コメント率，サイクロマチック数，および分岐条件数は，当初から基準値をクリアしていたが，最大ネスティング平均値やコード解析アラーム数は，2011 年には，基準値の 2 倍から 3 倍悪い状態であったことがわかる．改善活動が 2 年経過した 2013 年には，有効行数，コメント率，最大ネスティング平均値，およびコード解析アラーム数は改善したが，サイクロマチック数と分岐条件数は改善していない．この理由は，改善がみられたメトリクスについては，コーディング開発者に開発着手前にコーディング作法やプロダクトメトリクスの基準値等のルールを教育することで，開発者が自ら基準値を意識して作業を実施することができたからと考えられる．しかし，メトリクスの中でもサイクロマチック数や分岐条件数は，コーディング担当者がソースコードを開発しながら自分で測定することが他のメトリクスに比べて難しい（メトリクス値を知るには測定ツールの実行が必要になる）ため，適切な修正に至らなかったと考えられる．

3.3　プロセス成熟度の高い組織における定量的品質マネジメントの課題

筆者の組織では，品質向上の方策として，レビュー工数の基準値を定め，週次のプロジェクト進捗会議においてレビュー工数を監視し，工程移行判定において基準値の到達度をチェックする運用を通じて，レビュープロセスを組織的に安定化させてきた．その結果，レビュー工数の増加（図 3-4）に従い，出荷後の各製品のバグ数が減少（図 3-5）した．また，ソフトウェアファクトリを活用することで，プロダクトメトリクスの測定結果のフィードバックをほぼ毎日実施することができるようになった．出荷後の製品品質が特に悪かった 2 製品は，3 年間で出荷後品質が向上し，他の製品の出荷後品質のレベルに近づいた．レビューの改善に重点を置いたプロセスメトリクスやプロダクトメトリクスの管理を着実に実施する定量的品質マネジメントにより，出荷後の製品品質改善に効果をあげることができた．

しかし，複雑化・大規模化する IT システムにおいては，たとえ出荷後の製品品質問題が量的に減少したとしても，1 件の品質問題（バグ）が社会に大きな影響を与える可能性がある．改めて図 3-6 をみると，2013 年には出荷後の製品品質の改善が顕著にみられるが，2014 年には 2013 年ほどの改善がみられないばかりか，若干の製品品質の悪化を示している．これを踏まえて図 3-4 をみると，製品品質の向上に一番有効なメトリクスと思われるレビュー工数密度は，2012 年に組織の目標値を達成したものの，2013 年および 2014 年には増加傾向が鈍り横ばい状態である．これに対して，レビュー検出バグ密度は，2013 年および 2014 年には減少傾向に転じている．図 3-6 をみると，2014 年には出荷後の製品品質も改善がみられなかった．レビュー工数がある一定レベルまで到達すると，レビュー工数による品質向上の効果が鈍化したといえる．

別のメトリクスの視点から，レビュー工数の効果について考察する．図 3-8,図 3-9，および図 3-10 は，筆者の組織におけるレビュー工数密度，上工程バグ摘出率，および摘出バグ密度の 7 年間推移を示す．ここで，上工程バグ摘出率とは，上工程とテスト工程を通して摘

出された開発時の総バグ数に対する，上工程に摘出されたバグ数の比率を表す．上工程バグ摘出率が高いと上流工程で作り込まれたバグを上流工程で摘出している比率が高いことから，レビューによるバグの摘出の効果が高いと考えられる．ただし，上工程バグ摘出率が高い値であっても，開発時に摘出された総バグ数が少なければ，開発期間を通してバグの摘出不足の可能性があり，レビューによるバグの摘出の効果が高いとはいえない．したがって，開発期間全体における総摘出バグ数の状態も確認する必要がある．

各メトリクスの 7 年間の経年変化に注目すると，図 3-8 のレビュー工数密度の中央値が約 2 倍に増加している．これに対して，図 3-9 の上工程バグ摘出率の中央値は約 1.2 倍に増加しており，図 3-10 の開発時の総摘出バグ数の中央値も 1.2 倍に増加している．レビュー工数密度の増加の割合に対して，上工程バグ摘出率および開発時の総摘出バグ数の増加の割合は小さい．また，図 3-11 は，レビュー工数密度と上工程バグ摘出率の散布図である．図 3-12 は，レビュー工数密度と総摘出バグ密度の散布図である．どちらの図も，年々の回帰直線の傾きが小さくなっている．この結果より，レビュー工数の増加により，上流工程におけるバグの摘出が増加し，開発全体で摘出した総バグ数も増加していることから，レビューによるバグ摘出の効果は高いことがわかる．しかし，レビュー工数が毎年増加して，ある一定の量を超えると，レビュー工数の増加の割合に比べて，摘出できるバグ数の増加の割合は小さくなり，レビュー工数が増えるほどに，レビューによるバグの摘出効率が下がる．

このようにプロセス成熟度が高い組織において，さらなる出荷後の製品品質向上を追求し続けるには，これまでに効果があった上流工程におけるレビュー工数密度やレビュー摘出バグ密度以外のメトリクスに関する定量的品質マネジメントの改善が必要である．

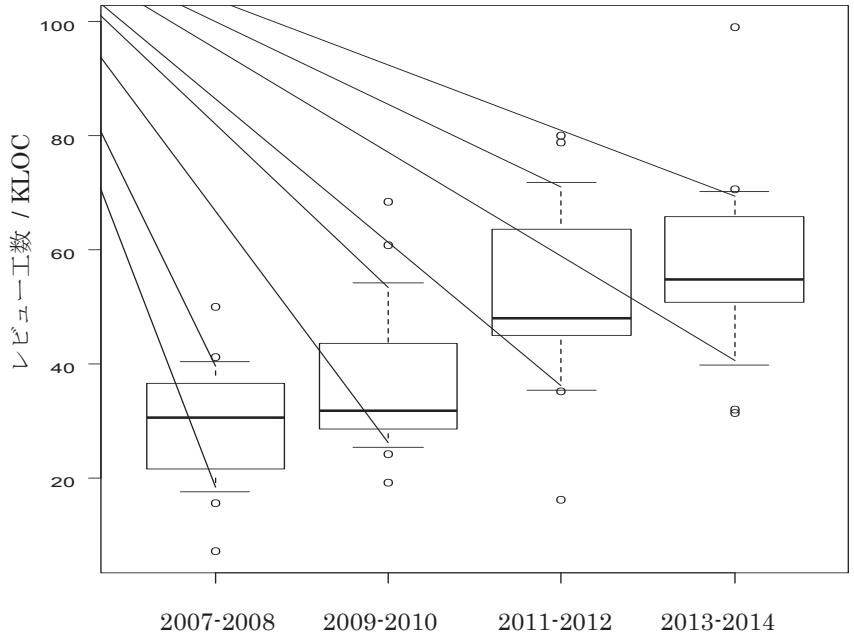


図 3-8 レビュー工数密度の推移

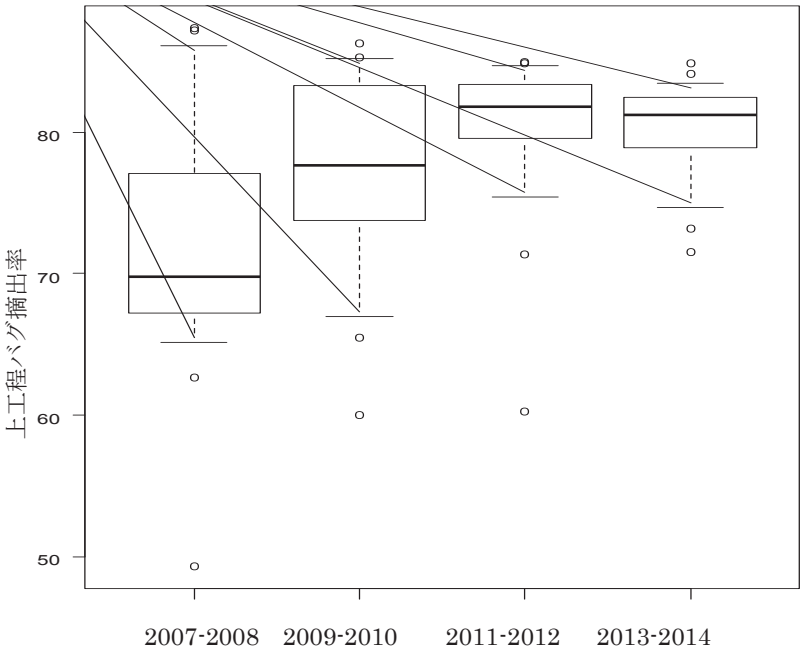


図 3-9 上工程バグ摘出率の推移

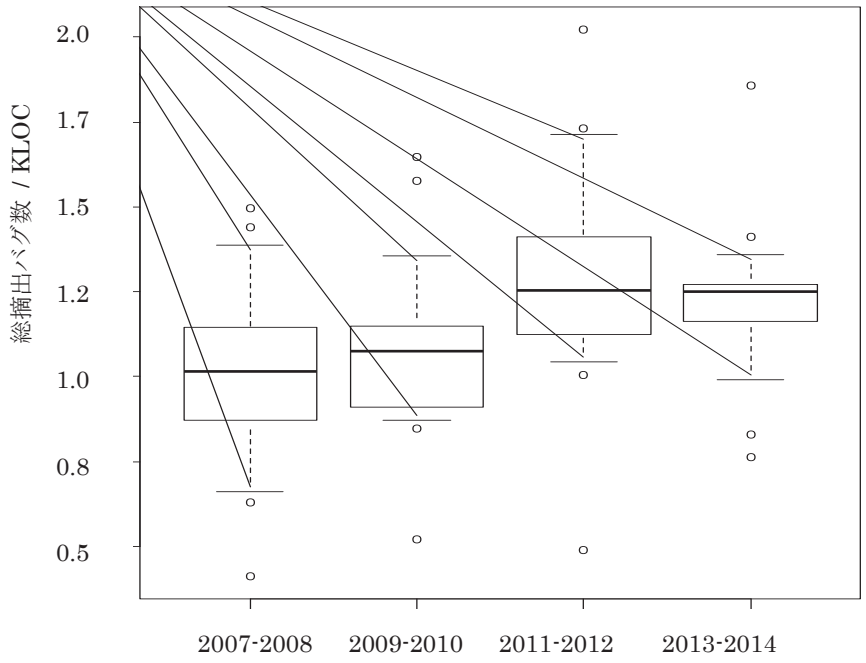


図 3-10 開発時の総摘出バグ密度の推移
(“2007-2008”の値を 1.0 とした相対値)

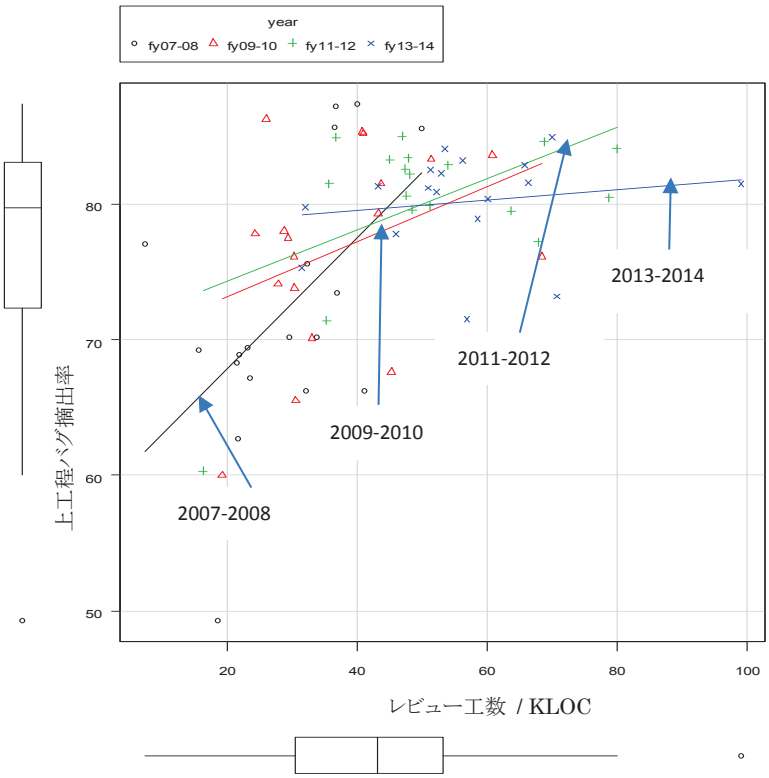


図 3-11 レビュー工数密度と上工程バグ摘出率の推移

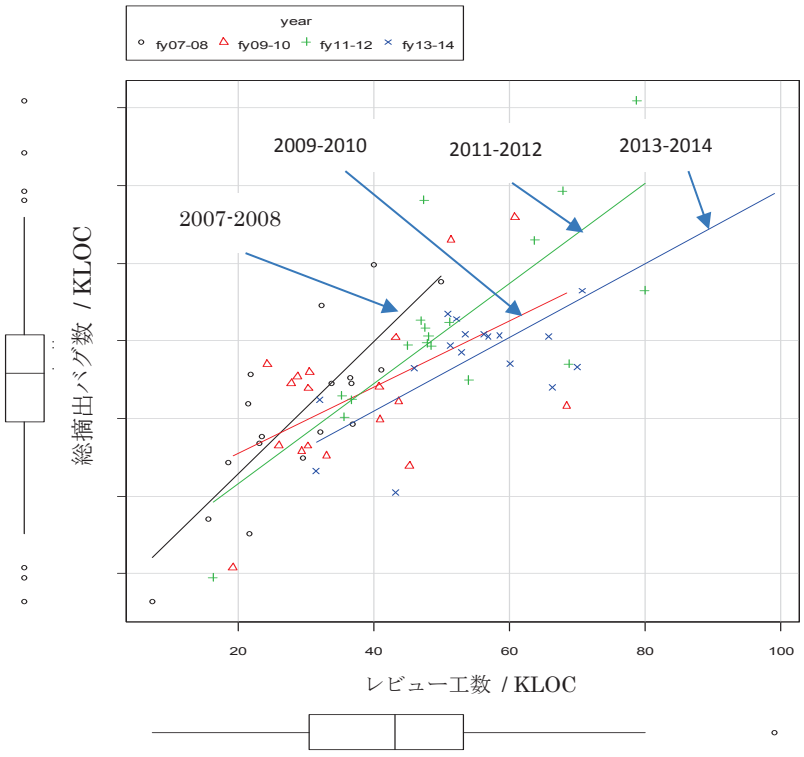


図 3-12 レビュー工数密度と開発時の総摘出バグ密度の推移

3.4 まとめ

本章では，プロセス成熟度の高い組織における定量的品質マネジメントの実際について，筆者の組織の開発プロセスの改善に関する実践研究を取り上げた．高度化された開発環境であるソフトウェアファクトリを組織内で統一して活用することにより，プロセスメトリクスやプロダクトメトリクスのデータ測定結果の集約および診断結果の出力をほぼ毎日実施することができ，短いサイクルでの定量的品質管理のフィードバックを実施することができている．

その結果，品質に大きく影響を与える上流工程におけるレビュー工数密度が増加するとともに，レビュー摘出バグ密度も増加し，出荷後のバグ数は減少した．また，プロダクトメトリクスにも改善がみられ，組織内で最も品質が悪かった 2 製品も各メトリクスが改善され，出荷後の製品品質も飛躍的に向上した．

最後に，定量的品質マネジメントによる品質改善の効果は得られたものの，企業の基幹システムや社会インフラシステムでは常に高い品質が要求されるために，出荷後の製品品質のさらなる改善が期待される．繰り返し述べているように，高品質ソフトウェア開発を実現するために，さらなる定量的品質マネジメントの高度化を目指す必要がある．定量的品質マネジメントの「測る」ことと，測った結果から科学的に「判断する」ことの両面でさらに定量的品質マネジメントを高度化することについては，第 4 章から第 8 章で論じる．第 4 章では，科学的な「判断する」ことの高度化の手法について，また，第 5 章から第 8 章では「測る」ことの高度化について，それぞれの実践研究を取り上げる．

第 4 章　定量的品質マネジメントのためのソフトウェア品質マップの構築と実践

4.1　はじめに

前章では，プロセス成熟度の高い組織における定量的品質マネジメントの実際と課題について，筆者の組織のプロセス改善の実践研究を取り上げ，定量的品質マネジメントの具体的な方法を論じた．その結果，レビュー工数密度の増加に伴いレビュー摘出バグ密度の増加と出荷後の製品品質向上の効果を確認した．しかし，さらなる品質向上のためには，レビュー工数密度による定量的品質マネジメントには限界があり，定量的品質マネジメントを高度化すること，すなわち，メトリクスの改善により「測る」ことを高度化し，品質分析の手法改良により測った結果から科学的に「判断する」ことを高度化することが必要であると論じた．

本章では，品質分析手法の改良により測った結果から科学的に「判断する」ことを高度化することについて，筆者の組織において定量的品質マネジメントの改善施策として考案した「ソフトウェア品質マップ」の具体的な手法と適用効果について論じる．

4.2　ソフトウェア品質マップ構築の背景

筆者の組織では，主に IT システムの高可用性を実現するコンピュータシステムに組み込まれる運用管理等のミドルウェア汎用パッケージ製品の開発や保守を担当している．製品は，企業の基幹システムや社会インフラシステムなどに組み込まれるため，常に高い製品品質が要求される．ミッションクリティカルなシステムに組み込まれた製品においては，出荷後のバグ１件が，お客様の業務に多大な影響を与える場合があるため，常に高い製品品質を目指すことが要求されている．一方，開発の現場では，製品の機能の肥大化による改造作業の複雑性の増大や，オフショア開発の担当者の入れ替わりによる製品技術スキル継承の問題などにより，お客様から要求される高い品質を確保することがますます困難な状況にある．したがって，従来の定量的品質マネジメントによる品質管理プロセスを徹底することに加えて，新たな定量的品質マネジメントとしてソフトウェア品質マップという品質分析手法を考案し実施してきた（以降は，ソフトウェア品質マップを単に品質マップと記述する）．なお，文献[4-1]に紹介されている品質マップは，テスト段階において，単体テスト完了までに摘出したバグを現象別・原因別にマトリクス表にして，どんな現象がどんな原因でバグが発生したのかの偏りを分析することにより，テストの十分性を評価するという用途であり，本事例の品質マップとは異なるものである．

従来の品質管理プロセスは，各工程のメトリクスの基準値を設定し，工程移行判定時に基準値の達成を確認することにより，各工程の成果物や実施作業の品質を確保する仕組みで

ある．この仕組みにより，理屈では開発終盤には高品質が確保されているはずである．しかし，現実には，開発途中の作業の小さな漏れや見逃しの積み重ねや，開発事情や開発環境の変更への対応漏れなどにより，開発終盤になっても品質が安定しなかったり，出荷後の品質に影響を及ぼす事態になることがある．品質マップは，このような開発途中における品質確認の関門をすり抜けてきた問題を抽出するために，開発の終盤段階において開発工程全体の品質状況を見直し，品質上の弱点対策を講じる手法である．

品質マップは開発終盤に実施するため，開発プロジェクトの納期も迫っている時期でもあり，多くの工数を掛けられない事情がある．したがって，的確な品質分析により，品質対策を効率よく進めなければならない．例えば，バグ分析とは，摘出されたバグの原因を分析して潜在する類似のバグを摘出する方法であるが，もし，分析の方向性を間違えると，本来の弱点ではない対象に品質対策の工数をかけてしまい，結果的に品質向上の効果が少なくなる場合がある．品質マップは，開発全体を俯瞰して，より多くの情報をもとに弱点を分析するため．バグ分析による品質対策の考え方にも適切な方針を示すことができる．

4.3 ソフトウェア品質マップの概要

4.3.1 適用対象と適用時期

品質マップの適用対象は，極めて高い品質が要求されるために出荷前に徹底的に品質を確保しておく必要のある開発プロジェクトである．開発計画時に開発管理方法を検討する段階で，品質マップを適用するかどうかを決定する．品質マップを適用する場合でも，従来の品質管理は実施する．

品質マップを実施する時期は，開発プロセスの最終工程であるシステムテストが完了した直後に開始する．

したがって，品質マップの適用対象となる開発プロジェクトは，従来の厳格な品質管理に加えて，開発の終盤段階に品質マップを作成し，各メトリクス間の関連性を多角的に分析して，開発全体を俯瞰して開発途中の品質確認の関門をすり抜けてきた問題がないかを見直し，必要な品質向上対策を実施することにより，さらに高い品質の達成を目指している．

4.3.2 品質保証部門の役割

品質保証部門は，全ての開発プロジェクトの品質管理を担当している．主な品質管理の作業は，品質データの収集と分析，工程移行判定，および出荷判定である．品質マップの作成と分析も品質保証部門が実施する．開発部門は，品質保証部門が提示した品質マップによる分析結果をもとに，品質向上対策を実施する．

4.3.3　品質マップ表の構成

品質マップの全体構成を表 4-1，表の横軸の項目を表 4-2，品質マップを使った分析の考え方を表 4-3，および適用例を表 4-4 に示す．

品質マップ表の縦軸は，品質マップの対象とする製品が開発する機能を配列する．配列する機能は，“端末接続台数拡大”や“レスポンス性能向上”のように独立した開発機能項目を最小単位で表し，1 単位の開発規模は数千 LOC（コード行数）程度である．表の横軸は，定量的データと定性的データで構成される．定量的データは，プロセスメトリクスとプロダクトメトリクスがあり，定性的データは工程移行時の開発リーダによる品質見解である．横軸の各項目を表 4-2 に示す．

表 4-1 品質マップ表の骨格

	定量的データ		定性的データ
	プロセス メトリクス	プロダクト メトリクス	工程移行時の品質見解
機能 A			
機能 B			
機能 C			
・・・			
・・・			

● 使用するプロセスメトリクス

プロセスメトリクスには，レビュー工数や摘出バグ数などの従来の品質管理プロセスで扱うメトリクスを使用している．ただし，従来の品質管理プロセスで扱うメトリクス以外には，「開発終盤バグ数」(表 4-2 の No.7)を使用していることが特徴である．このメトリクスは，品質マップによる品質分析において重要な役割を持っており，詳細は後述する．

全ての開発プロジェクトは，品質マップを適用するか否かに関わらず，従来の品質管理プロセスを実施している．したがって，品質マップを作成する開発終盤の段階では，表 4-2 に記載されているメトリクスは，従来の品質管理プロセスの仕組みによりデータを収集済みであり，品質マップを作成するために，改めて開発部門からデータを収集する必要はない．

開発プロジェクトは，従来の品質管理プロセスを実施しているので，各工程のプロセスメトリクスの基準値を達成できているかを工程移行判定時に確認するため，ほとんどのメトリクスは基準値をクリアできている状態である．品質マップでは，基準値をクリアしているかどうか以外に，他のデータの組み合わせ条件の中でメトリクスを判定するところに特徴があり，4.4 で詳細を述べる．

表 4-2 品質マップ表の横軸の項目

項目	No	内容	意味
プロセスメトリクス	1	上工程摘出バグ数（/KLOC）	（総摘出バグ数のうち,テスト開始前の上工程で摘出されたバグ数）/ 開発規模(KLOC)
	2	上工程レビュー工数（人・時/KLOC）	（設計およびコーディングに対するレビューに費やした工数（人・時））/開発規模(KLOC)
	3	上工程バグ摘出率（%）	（上工程摘出バグ数/総摘出バグ数）*100
	4	テスト工程摘出バグ数（/KLOC）	（総摘出バグ数のうち,テスト工程で摘出されたバグ数）/開発規模(KLOC)
	5	テスト項目数（/LOC）	テスト項目数 / 開発規模（LOC）
	6	総摘出バグ数（/KLOC）	出荷前に摘出した総摘出バグ数 / 開発規模(KLOC), 総摘出バグ数＝上工程摘出バグ数＋テスト工程摘出バグ数
	7	開発終盤バグ数	システムテスト以降に検出されたバグ数(バグ数は出荷後の製品品質を代替していると考える)
プロダクトメトリクス	8	コード解析アラーム数（/KLOC）	(指定のコード解析ツールが検出したアラーム数）/ 開発規模(KLOC)
	9	サイクロマチック数（/LOC）	サイクロマチック数 / 開発規模(LOC)
	10	分岐条件数（/LOC）	分岐条件数 / 開発規模(LOC)
	11	ネスティング基準外割合(%)	（ネスト基準値を超えた関数の数 / 関数の総数）*100
	12	最近コード修正回数	機能テスト完了以降にコードを修正した回数
工程移行時の品質見解	13	上工程完了時の品質見解	上工程完了時の工程移行判定で,開発リーダが上工程の品質を分析した見解
	14	機能テスト完了時の品質見解	機能テスト完了時の工程移行判定で,開発リーダが機能テストまでの品質を分析した見解

表 4-3　8 種類の条件適合判定表

条件	抽出した項目の組み合わせ条件	条件の解釈	対策の考え方
1	開発終盤バグとプロセスメトリクス（レビュー工数 or テスト項目数 or 総摘出バグ数）	プロセスメトリクスにマークされた項目が示す内容について開発途中に問題があったため，開発終盤バグが摘出された可能性がある．	開発終盤バグを修正するだけでは足りない．開発途中の問題により残存するバグを摘出すべきである．
2	開発終盤バグのみ	プロセスメトリクスには問題がないため，偶然に開発終盤バグが検出された．	ケース 1 と異なり，プロセスの問題が少ないため，開発終盤バグの原因分析による対策を優先する．設計上の問題まで分析が必要．
3	上工程バグ摘出率とその他のプロセスメトリクス（上工程摘出バグ数 or レビュー工数 or 総摘出バグ数）	プロセスメトリクスにマークされた項目が示す内容について開発途中に問題があったため，上工程バグ摘出率が低くなった可能性がある．	上工程のプロセスの問題を分析する．レビューが不足している可能性が大きい．レビュー内容を分析してレビューの追加が必要．
4	上工程バグ摘出率のみ	他のプロセスメトリクスが問題ないため，矛盾する．メトリクスでは見えない問題が存在する可能性がある．	ケース 3 と異なり，上工程プロセスの問題が少ないため，テスト工程のバグの傾向分析を優先する．その結果，レビューの量か質に不足がないかを分析する．
5	プロダクトメトリクスとプロセスメトリクス（レビュー工数 or テスト項目数 or 総摘出バグ数）	プロダクトメトリクスとプロセスメトリクスにマークされた項目が示す内容について開発途中に問題があった可能性がある．	プロダクトメトリクスの悪い箇所とコードレビューの因果関係を分析する．その結果，コードレビューの不足がないかの分析する．
6	総摘出バグ数のみ	レビュー工数，テスト項目数，プロダクトメトリクスは問題ないため，レビューやテストが十分にできており．作り込み品質も良いと判定する．	品質に問題のある可能性は極めて小さい．現時点では特にアクションの必要はなし．
7	最近コード修正回数と上工程バグ摘出率	テストの終盤でバグが多くなる傾向に問題がないかを確認する．	テスト工程の修正内容とテスト工程摘出バグに関連性がないか，品質上の問題がないかを確認する．
8	プロダクトメトリクスとレビュー工数とテスト項目数と総摘出バグ数	結果的にメトリクスが基準を達成していないことから，レビューやテストに問題があった可能性がある．	移行判定の品質分析との現時点のメトリクス結果に矛盾がないか，本当にレビューやテストに不足がないかの分析が必要．

表 4-4　品質マップの適用例

	プロセスメトリクス							プロダクトメトリクス					工程移行時の品質見解	
No	1	2	3	4	5	6	7	8	9	10	11	12	13	14
機能	上工程摘出バグ数	上工程レビュー工数	上工程バグ摘出率	テスト工程摘出バグ数	テスト項目数	総摘出バグ数	開発終盤バグ数	コード解析アラーム数	サイクロマチック数	分岐条件数	ネスティング基準外割合	最近コード修正回数	上工程完了時の品質見解	機能テスト完了時の品質見解
A	+4	+12	+3	0	+0.4	+4	0	0	-0.1	-0.2	0	1	（省略）	（省略）
B	+6	+14	+5	+6	+0.3	+12	2	0	-0.1	-0.2	0	4	レビューを基準以上行った結果バグも基準以上出したので，品質は良い	テスト項目数もテスト摘出バグ数も問題なし
C	+4	-2	+2	0	+0.5	+6	3	0	-0.1	-0.2	0	5	レビューアのスキルが高いのでレビュー工数が少なくても問題なし	テスト項目数もテスト摘出バグ数も基準クリアし問題なし

※No. 1–No. 6 および No. 8–No. 11 の数値は基準値に対する相対値を表す．No. 1–No. 6 の基準値は下限閾値のため，相対値が負の場合はプロセス品質が悪いことを示し，No. 8–No. 11 の基準値は上限閾値のため，相対値が負の値はプロダクト品質が良いことを示す．（No. 7 および No. 12 は絶対値を表す．）

● 上工程バグ摘出率が重要な理由

品質マップで使用しているメトリクスの中で重要な項目は，上工程バグ摘出率と開発終盤バグ数である．上工程バグ摘出率を重要な項目とする理由を説明する．上工程バグ摘出率は，上工程摘出バグ数の総摘出バグ数に対する割合で表す．すなわち，上工程摘出バグ数とテスト工程摘出バグ数との比率を表している．バグは，設計工程やコーディング工程などの上工程で作り込まれるので，作り込まれたバグをできる限り早い段階に摘出する必要がある．バグの摘出が，作り込まれた工程よりも遅くなれば，バグを修正するための作業の後戻り工数が大きくなるため，生産性の低下につながる．また，バグ修正による従来機能への悪い影響を及ぼすリスクを増大させることになる．したがって，上工程バグ摘出率が低いということは，テスト工程摘出バグ数が多いということに等しく，上工程でのレビュー活動の不完全さを表すことになる．

筆者の所属組織は，製品開発プロセスの中で，上工程の設計工程やコーディング工程を重要な工程と考え，メトリクスを定義して，設計レビューやソースコードレビューを十分に実施できるように品質管理を行っている．上工程バグ摘出率の基準値は 80%である．上工程バグ摘出率が 80%以上という高い値は，筆者の所属組織が 30 年以上に渡り蓄積してきた上工程の作業を重視する開発プロセスの経験により実現してきたものである．

● 開発終盤バグ数が重要な理由

次に，開発終盤バグ数を重要な項目とする理由を説明する．品質管理プロセスは，各工程のメトリクス基準を達成することで各工程の成果物や実施作業の品質を確保することを推進している．したがって，品質管理プロセスが成熟した組織では，開発の終盤においてバグが出ることは少ない．したがって，開発終盤にバグが摘出されるということは，そのバグを開発途中に摘出することができなかったという開発プロセス上の問題を探るヒントが隠されている．単にバグを修正するだけでなく，開発プロセス上の問題点を分析して他にも同様の問題が潜んでいないかを調査することが，品質向上作業では重要である．開発終盤バグと各工程におけるメトリクスとの関係を分析することで，問題となる開発プロセスを見つけ出すことができる．つまり，開発終盤バグは製品の出荷後バグと同じレベルであり，製品の出荷後品質を代替しているといえる．

● 使用するプロダクトメトリクス

プロダクトメトリクスは，ソースコード静的解析ツールから得られるデータを利用している．

ソースコードの複雑性を表すメトリクスの中で，サイクロマチック数や分岐条件数は，開発規模との正の相関が強い[4-2],[4-3]ので，開発規模で正規化した基準値を設定している．開発言語は C 言語が多いため，CK(Chidamber and Kemerer)メトリクスのようなオブジェクト指向言語用の複雑性メトリクスは適用していない．筆者の所属組織は，クラウド型の開発環境（第 3 章で説明したソフトウェアファクトリ）によりオフショア開発も含めて全てのソースコードを統一リポジトリで一括管理しているので，プロダクトメトリクスも毎日収集できる環境にある．

近年増加しているオフショア開発は，日本と比較すると開発者の交替が数多く発生する傾向にある．プロジェクトリーダは，開発者の技術レベルを把握するための方法として，プロダクトメトリクスを参考にする場合がある．リーダはプロダクトメトリクスが悪い値を示すソースコードを参照して，オフショア開発者等のコーディング技術レベルを把握することにもメトリクスを活用している．

● 使用する定性的データ

プロセスメトリクスやプロダクトメトリクスなどの定量的データに対して，定性的データは工程移行判定時の開発部門の品質見解を使用する．品質見解とは，開発リーダが上工程完了時と機能テスト完了時における品質を分析した見解である．

通常の品質管理プロセスは，工程移行判定時にメトリクスが基準値を達成しているかどうかをチェックしているので，基本的には基準値に達していない場合は工程移行判定にて合格できない仕組みになっている．しかし，現実には，開発の特性や事情によりメトリクスが基準値を達成できない場合もある．この場合，開発チームリーダが基準値を達成できない理由を品質保証部門に提出し，品質保証部門が理由を認めた場合は工程移行判定を合格にするように運用している．したがって，品質管理プロセスを厳格に運用しても，結果的に基準値を達成していないメトリクスも存在する．この場合は，後で述べる工程移行判定時の品質見解の定性的情報が品質マップによる弱点分析の重要なポイントになる．

4.4 ソフトウェア品質マップの適用手順

品質マップの適用手順は，(1)データ抽出，(2)条件適合判定，(3)最終分析，である．

4.4.1 データ抽出

品質マップ表に記載したデータの中からメトリクス値の悪いデータを抽出してマークする．抽出する方法は，基準値比較による抽出と，相対比較による抽出の 2 種類がある．

基準値比較では，各メトリクスが基準値以下のものを全て抽出する．相対比較では，表 4-1 の縦軸の項目の中で相対的に数値の低い 20%のデータを抽出する．相対比較をする理由は，開発全体の中で相対的に弱点となる機能を探すためである．縦軸の機能別のメトリクスの相対比較により弱点となる機能は，当該機能を担当する開発チームの技術力が影響している可能性があると考えている．

また，表 4-2 の No.7 の開発終盤バグ数は，データが 1 件以上の場合には全て抽出する．これは，前述のとおり開発終盤バグは出荷後品質を疑似的に表しているため，1 件以上のバグが発生している場合には，品質問題の可能性が高いという判断をしている．

4.4.2 条件適合判定

次に品質マップ表の各行が，表 4-3 に示す 8 種類の判定条件のどの条件に適合するのかを判定する．8 種類の判定条件は，これまでの品質管理活動の経験に基づき，各メトリクスの組み合わせの因果関係を考慮した条件になっている．品質マップ表の各行について，抽出したデータ項目の組み合わせが判定条件表の条件 1 から順に判定条件に適合しているかどうかを判定する．

4.4.3 最終分析

最終分析では，品質マップ表の各行の条件の判定結果について，工程移行時の品質見解や，開発終盤バグのバグ分析結果と比較して総合的に分析する．分析の考え方は，条件適合判定が示す悪いメトリクスと，開発終盤バグの分析結果や工程移行時の品質見解に矛盾がないか，または，なぜレビューの指摘観点の漏れやテスト項目の観点不足が発生したのかなどの品質観点の弱点を分析する．その結果，開発プロセスや品質観点の弱点を抽出する．

最終分析の例を表 4-4 に示す．機能 B の行は，データの抽出では開発終盤バグ数のみが抽出された．この場合，条件適合判定では表 4-3 の条件判定表の条件 2 に適合する．最終分析では条件判定表の条件 2 に記載された条件の解釈と，工程移行時の品質見解および開発終盤バグのバグ分析結果を比較して矛盾がないか等を総合的に分析する．その結果，条件判定表の条件 2 に記載された対策の考え方を参考に品質対策を検討する．同様に機能 C の行は，データの抽出では上工程レビュー工数，開発終盤バグ数，および最近コード修正回数が抽出されたので，表 4-3 の条件判定表の条件 1 に適合する．したがって，条件 1 に記載された対策の考え方を参考に品質の弱点分析と対策を検討する．

このように品質マップでは，メトリクスによる定量的データの判定結果と，開発終盤バグの分析や品質見解の定性的な情報との因果関係を分析することで，品質上の弱点を最終的に判定する．

最終分析の結果により，開発部門は抽出した弱点に対して品質向上対策を実施する．対策を実施する場合には，基本的に設計レビューかコードレビューの実施を推進している．対策としてテストを追加する場合もあるが，テストは想定どおりに動作することを確認する方法としては有効であるが，バグを摘出するには，網羅性の観点や効率の面から，十分とはいえないからである．

4.5　ソフトウェア品質マップの適用効果

筆者の所属組織では，品質改善活動の一つとして品質マップを一部の開発グループに適用してきた．品質マップを適用したグループについて，品質マップを適用する前と適用後の品質と生産性を比較した．

品質指標には，出荷後バグ密度を使用した．出荷後バグ密度は，評価対象グループの出荷後バグの中で当該年度に作り込んだバグ数を当該年度の開発規模で除した値で算出する．全評価対象グループの品質マップを適用前のデータ群と適用後のデータ群の t 検定では p 値が 0.0332 となり，品質マップの適用前後では出荷後バグ密度に 5%有意の有意差がみられた．この結果，品質マップを適用すると出荷後バグ密度が減少し，中央値比で約 4 倍の品質向上がみられた（図 4-2）．

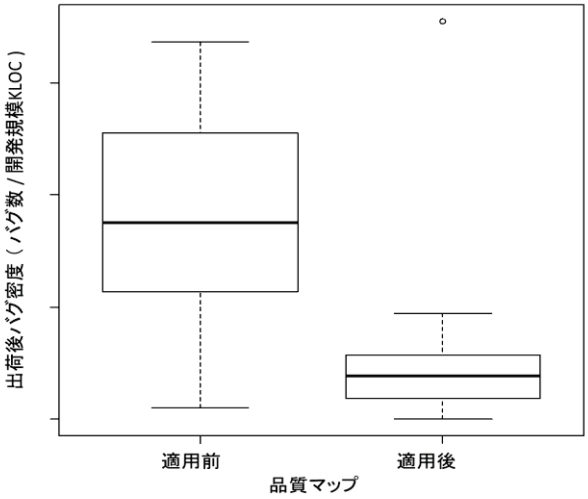


図 4-2 出荷後バグ密度比較

一方，生産性の指標は，評価対象グループの当該年度の開発工数を当該年度の開発規模で除した値で算出した．全評価対象グループの品質マップを適用前のデータ群と適用後のデータ群の t 検定では p 値が 0.160 となり有意差はみられなかったが，品質マップを適用す

ると生産性は中央値比で約 30%低下した(図 4-3)．品質マップを適用すると，開発終盤において品質分析と品質向上の対策を実施するための工数が増加し，生産性が低下した．

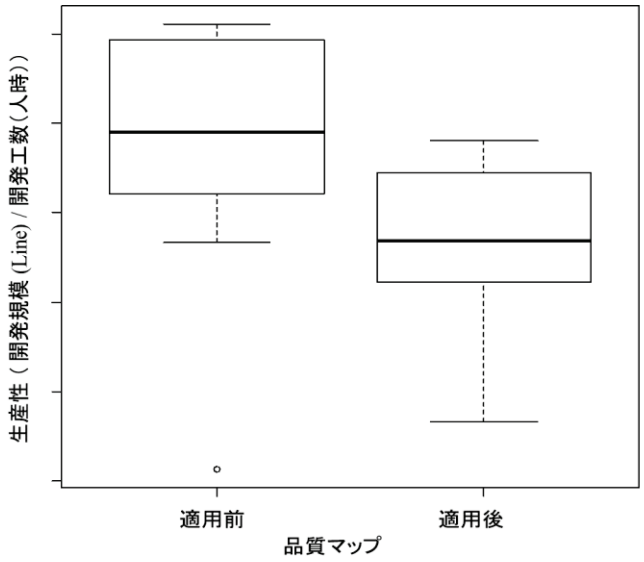


図 4-3 生産性比較

従来の品質管理手法と比較した品質マップのメリットは，開発終盤段階において各種の品質データを組み合わせて開発全体を俯瞰することにより新たな問題を抽出できることである．品質マップの定性的な適用効果は，以下のとおりである．

(1) 品質弱点の分析精度を高め，より効果的な品質対策が可能になった．

通常，バグ分析結果から品質対策を実施する場合には，バグの発生条件を弱点とした対策を実施するが，弱点を局所化してしまうリスクがある．また，対策の対象範囲も曖昧になる．しかし，品質マップによる弱点分析では，バグ分析による弱点の品質観点を，さらにバグを作り込んだ工程のメトリクス値や品質見解から評価することで，実施すべき対策の内容と対象範囲がより明確になる．

(2)製品を構成する機能間の比較による問題抽出が可能になった．

従来の品質管理手法では，経験上，メトリクスの基準値が機能単位でばらつく場合や，メトリクスが基準値を下回っていても品質に問題がない場合や，メトリクスが基準値を上回っていても品質が十分とは言い切れない場合などの弱点があると考える．しかし，品質マップでは，メトリクスの基準値による評価に加えて，開発全体を俯瞰した視点から妥当であるかを判定する．また，工程移行時の品質見解や開発終盤バグの結果を踏まえて，

メトリクスによる判定結果を再評価することができるため，従来の品質管理手法の弱点を補完することができる．

品質マップを適用すると，出荷後バグ密度が低下し品質が向上することを検証できた．ただし，高品質を確保するために掛けた工数分の生産性が低下した．一方で，品質保証部門と開発部門が共同して，より深く品質を分析するようになった．また，品質保証部門の要員が，定量データを一律・機械的に処置しがちな状況を防ぎ，測定データの目的の再確認や関連するデータ間の的確な品質分析を考える機会となり，品質マップで使用している条件適合判定表の強化や要員の育成につながった．

4.6　ソフトウェア品質マップの課題

品質マップの課題は，品質マップで使用する条件適合判定表の内容の充実である．条件適合判定表の内容は，これまでの豊富な定量的品質管理の失敗や成功の経験の知見であり，筆者の組織の財産であると考える．特に，プロセス成熟度の高い組織においては，開発プロセスの安定的な運用によりプロセスメトリクスの弱点が減少してくるため，条件適合判定表によりメトリクスや定性的データを組み合わせた総合的な分析が重要になってくる．しかし，現状の条件適合判定表では，プロダクトメトリクスに関する知見が少ないため，プロダクトメトリクスに関する経験と研究による知見を反映するなど，常に現場の実態に沿った内容に改良することが必要である．

4.7　まとめ

本章では，高品質ソフトウェア開発を実現するための定量的品質マネジメント技術の高度化の一つと考えている品質分析手法の改善による科学的に「判断する」ことの高度化について，ソフトウェア品質マップの実践研究を取り上げた．従来の定量的品質マネジメントでは，バグ修正等の後戻り工数を最小限に抑えるために，可能な限り作り込み品質を自工程完結[4-4]する考え方であった．しかし，自工程内でバグを完全に摘出することは難しく，または，摘出したと判断することは難しい．したがって，本方策は，自工程内の完結から漏れてしまうような問題を，開発工程の全体を見直す機会を設けることにより，メトリクスや定性的な品質見解などの相互の因果関係や矛盾点などを過去の経験をもとに知見化した条件適合判定表を用いて，総合的に品質を分析して弱点をあぶりだすという定量的品質マネジメント技術の科学的に「判断する」ことの高度化を実践したものである．本実践研究の結果，品質マップを適用すると出荷後バグ密度が減少し，適用前に比べて約 4 倍品質が向上した．一方，生産性は品質マップを適用すると約 30%低下した．品質マップを適用すると，開発終盤において品質分析と品質向上の対策を実施するための工数が増加したためである．

品質マップの課題は，品質マップで使用する条件適合判定表の内容の充実であり，特に，プロダクトメトリクスの改良が必要である．これは，定量的品質マネジメントのもう一つの高度化技術と考えているメトリクスの改善による「測る」ことの高度化を意味する．この課題の対策について次章以降にて論じる．

品質マップの考え方は，結局，ソフトウェアのバグの混入は不確定要素が大きく，確率論的な要素を考慮しないと品質の見極めが難しいということを示唆している．開発者スキルや開発の難易度をはじめ，開発途中に発生した不測の事態や，その対処の十分性などのような定量化の定義が難しい要素も，バグの混入やバグの摘出作業に大きな影響を与えていると考えられる．品質マップの狙いは，工程移行時における開発リーダーの品質見解のような定性的な情報とメトリクスとの因果関係や時系列による変化などを読み取って，品質の弱点を探るという考えであり，バグが潜在する確率が高そうな箇所を効率よく探す手順を常に考えることである．

第 5 章　定量的品質マネジメント改善のためのプロダクトメトリクスの研究 1 ～有効なプロセスメトリクスとプロダクトメトリクスの抽出～

5.1　はじめに

本章では，定量的品質マネジメント技術の「測る」ことを高度化するために，メトリクスの改善について検討する．まず，プロセスメトリクスとプロダクトメトリクスの相関関係を調べる．次に，プロダクトメトリクスの中で，品質に影響を与えるメトリクスを調べる．プロダクトメトリクスには，ソースコードの有効行数，コメント率，分岐条件，サイクロマチック，ネスティングなどのソースコードの複雑性を測るメトリクスを用い，各メトリクスと出荷後の製品品質との相関関係を分析した結果を論じる．

5.2　有効なプロセスメトリクスとプロダクトメトリクスの抽出

第 3 章で述べた筆者の組織において使用しているメトリクスの 5 年間のデータを用いて，プロセスメトリクスとプロダクトメトリクス（表 5-1 参照）間の相関関係を分析した．それぞれのメトリクスの意味は，表 3-1 に示した通りである．図 5-1 は，分析に使用したメトリクス間の散布図行列を示す．また，表 5-1 は，それらの相関係数行列を示す．

プロダクトメトリクスの分析対象には，ソースコードの複雑性を示すプロダクトメトリクス[5-1]から，分岐条件数密度，サイクロマチック数密度，最大ネスティング平均値を選択した．プロセスメトリクスの分析対象には，コーディング工程におけるレビュー工数密度と単体(UT)テスト項目数密度を選択した．その理由は，ソースコードの複雑性を示すプロダクトメトリクスの値が大きい場合には，ソースコードの品質を確保する労力を表すプロセスメトリクスであるレビュー工数密度や単体(UT)テスト項目数密度も大きくなるであろうと推定したからである．

表 5-1 主なメトリクスの相関係数行列

	branch	cyclo	nest	review	ut.item
分岐条件数密度(branch)	1.000	0.946***	-0.075	-0.285	-0.273
サイクロマチック数密度(cyclo)	0.946***	1.000	-0.352	-0.292	-0.405
最大ネスティング平均値(nest)	-0.075	-0.352	1.000	0.113	0.368
レビュー工数密度(review)	-0.285	-0.292	0.113	1.000	0.508*
単体テスト項目数密度(ut.item)	-0.273	-0.405	0.368	0.508*	1.000
*5%有意，　**1%有意，　***0.1%有意					

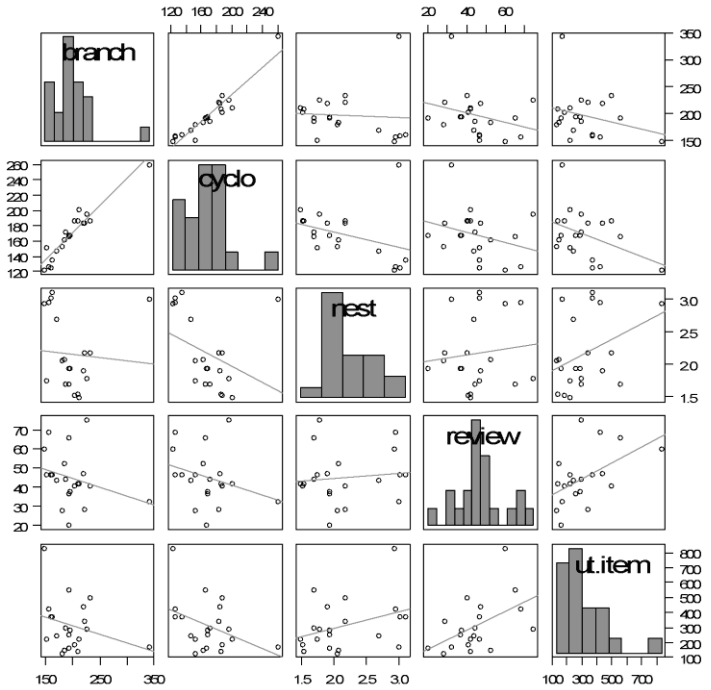


図 5-1 主なメトリクスの散布図行列

この分析の結果，選択したソースコードの複雑性を示すプロダクトメトリクスとソースコードの品質を確保する労力を表すプロセスメトリクスの間には，強い相関を示すメトリクスを見つけることができなかった．表 5-1 から，3 種類のプロダクトメトリクスと 2 種類のプロセスメトリクスとの相関係数は，-0.40 から 0.37 の間にあり（表 5-1 の太枠内），強い相関を示す関係はみられなかった．プロセスメトリクス間の相関については，レビュー工数密度と UT テスト項目数密度では相関係数が 0.51 (5%有意) と正の相関があり，レビュー工数密度が大きい場合には UT テスト項目数密度も大きくなることを示している．しかし，これらのソースコードの品質を確保する労力の大きさを示すプロセスメトリクスの値が大きくなるのは，ここで選択したソースコードの複雑性を示すプロダクトメトリクス以外の要因が考えられるようである．

プロダクトメトリクス間の相関については，分岐条件数密度とサイクロマチック数密度が 0.95(0.1%有意)と強い正の相関を示している．これは，表 3-1 で示すように，分岐条件数密度とサイクロマチック数密度の算出方法の特性が似ているためと考えられる．他方では，最大ネスティング平均値と，分岐条件数密度またはサイクロマチック数密度との相関係数が，-0.075 と-0.35 とほぼ相関がないことを示しており，ソースコードの複雑性を示すメトリクスでも特性が異なると考えられる．

次に，これらのソースコードの複雑性を示すプロダクトメトリクスの中で，出荷後の製品品質に影響を与えているメトリクスがないかを検証した．過去にプロダクトメトリクスに基準値を設けてソフトウェア品質を判断する方法論[5-2][5-3]が報告されているが，実際の製品データに適用した実践研究は多くない．実製品データを用いてメトリクスによる品質評価が行われた実践研究として，ソフトウェア設計品質の評価[5-4]や，Fault-Prone 予測[5-5]などが行われている．本研究においても実製品を用いて分析を行っている．

表 5-2 は，組織内の各出荷製品の出荷後の製品品質が高い層と低い層に層別した場合の t 検定の結果である．このとき，層別における“品質が高い層（良群）”とは，当該製品の出荷後バグがゼロであった場合であり，“品質が低い層（否群）”とは，当該製品の出荷後バグが 1 件以上あった場合である．検定結果では，最大ネスティング平均値の p 値が 0.0431 で，有意水準 5%で有意差ありとなり，ソースコードのネスティングの深さが出荷後の製品品質に関連があることが判明した．また，表 5-2 より，最大ネスティング平均値は，出荷後の製品品質が高い層は 1.98 であり，出荷後の製品品質が低い層は 2.522 である．それぞれの標準偏差は，0.48 と 0.52 であった．図 5-2 に最大ネスティング平均値と出荷後品質の高い層と低い層を層別した箱ひげ図を示す．

表 5-2 出荷後品質を目的変数とした主なメトリクスの t 検定

説明変数	出荷後品質 (目的変数)	平均	標準偏差	t 値	p 値
分岐条件数密度	良	196.3	26.27	-0.1186	0.907
	否	198.6	65.56		
サイクロマチック数密度	良	170.8	23.47	0.4301	0.672
	否	164.5	45.28		
最大ネスティング平均値	良	1.98	0.4835	-2.2808	0.0431*
	否	2.522	0.5278		
レビュー工数密度	良	48.28	13.03	1.6406	0.127
	否	38.27	13.24		
単体テスト項目数密度	良	313.4	129.5	0.003	0.998
	否	313.1	243.1		
良: n = 14, 否: n = 7, *5%有意, **1%有意, ***0.1%有意					

この結果，プロダクトメトリクスの基準値としては，最大ネスティング平均値は 2 (ただし，ネスト無しは 0 とカウントする)とするのが良いと考える．なお，文献[4-2]では，ソースコードの複雑性を示すプロダクトメトリクスとして，モジュールの凝集度(LCOM)や結合度(CBO)と欠陥修正モジュールとの関係性を調査する文献がみられる．しかし，これらのメトリクスは，ソースコードが JAVA 等のクラス定義によるオブジェクト型言語に限られる．筆者の組織では過去からの資産を含めると C 言語が主流であるため，ソースコードの複雑性を測るプロダクトメトリクスには前述のメトリクスを使用している．

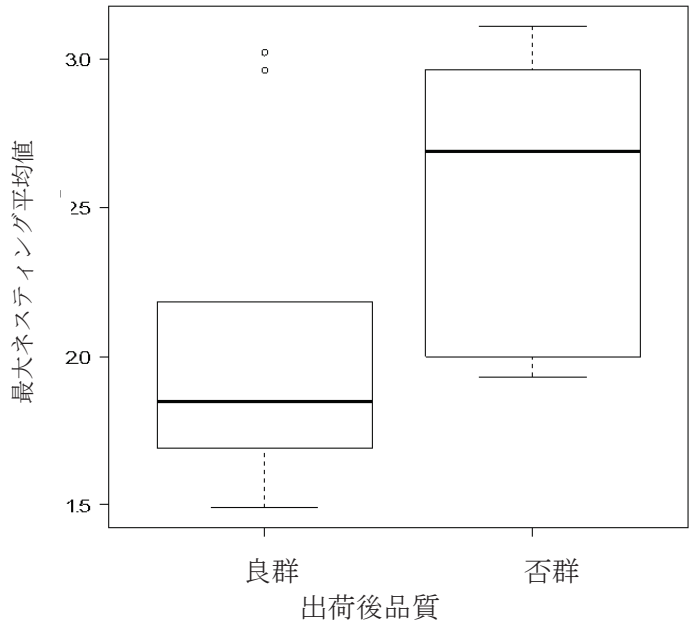


図 5-2 最大ネスティング平均値と出荷後品質の良群と否群の箱ひげ図

5.3 まとめ

本章では，ソースコードの複雑性を測るプロダクトメトリクスの中で，品質に影響を与えるメトリクスを抽出した．

初めに，ソースコードの複雑性を示すプロダクトメトリクスとソースコードの品質を確保する労力を表すプロセスメトリクスとの相関関係を分析した．しかし，両者に強い相関を示すメトリクスは見つからなかった．したがって，これらのソースコードの品質を確保する労力の大きさを示すプロセスメトリクスの値が大きくなるのは，ここで選択したソースコードの複雑性を示すプロダクトメトリクスの要因以外が考えられるようである．

次に，ソースコードの複雑性を示すプロダクトメトリクスの中で，出荷後の製品品質に影響を与えるメトリクスを分析した．その結果，最大ネスティング平均値が出荷後の製品品質に最も影響を与えていることが判明した．具体的には，最大ネスティング平均値が 2 を閾値として，2 を超えると，出荷後の品質が低くなる傾向があった．ただし，最大ネスティング平均値の出荷後の品質が高い群と低い群には有意差はみられなかったため，その原因について研究の継続が必要である．

本章で使用した最大ネスティング平均値は，表 3-1 で定義しているとおり，各メソッドの最大ネスティング数の総和をメソッド数で除した値であり，平均値である．上述の本メトリクスの閾値が 2 である意味は，コーディング時にすべてのメソッドのネストを 2 以下にす

べきというコーディング作法を示しているわけではない．コーディング全体の複雑性に関する性質を表す指標の一つと考えている．したがって，コーディング担当者にとっては，直観的には分かりにくい，自らコントロールしにくいメトリクスであり，この閾値を使ってプロセスをコントロールすることは難しいという課題が残る．

次章の研究において，さらに最大ネスティング数を含めたプロダクトメトリクスの有効性の分析と，分析結果からプロセス改善を導く指針を示している．

第 6 章　定量的品質マネジメント改善のためのプロダクトメトリクスの研究 2 ～ソースコード複雑性に関する有効なプロダクトメトリクスの評価～

6.1　はじめに

前章では，有効なプロセスメトリクスとプロダクトメトリクスの評価を行い，プロセスメトリクスとプロダクトメトリクスとの相関関係は認められなかったことと，出荷後の製品品質に影響を与えるプロダクトメトリクスとして，最大ネスティングが挙げられることが判明した．

本章においても，前章と同様に，実際の出荷製品について，出荷後の製品品質とプロダクトメトリクスとの相関関係を分析している．本研究は，市場で稼働中の製品について，開発時のプロダクトメトリクスと，出荷後の品質との相関関係を分析した結果を述べている．また，その結果から，開発プロセスへ適用するためのメトリクスの最適な閾値を考察している．

6.2　分析対象のデータ

本研究の分析の対象は，バージョンアップして機能を繰り返し強化しながら長期間にわたり市場に提供し続けている製品である．

対象製品の選定にあたっては，下記条件を満たすことを必須とした．

(1)　出荷後の一定量のバグ情報が取得可能である．
(2)　プロダクトメトリクスの計測対象が，分析に十分な個数である．
(3)　立案した方策を実行して，効果検証が可能である．

(1)の出荷後のバグ情報では，該当製品の複数バージョンにわたるバグ情報が蓄積されている．(2)のプロダクトメトリクスの計測対象では，該当製品の構成ファイルが 25,506 個あり，分析を実施するために十分なデータ量である．なお，プロダクトメトリクスは自社製ツールにて計測を行った．本ツールでは関数単位にメトリクスを計測するが，バグ情報はソースファイル単位に記録されているため，ソースファイル内の関数のメトリクスの中で最大値を当該ソースファイルのメトリクス値とした．(3)の立案した方策を実行して効果検証が可能であるかどうかについて，該当製品はバージョンアップ開発を継続しているので，立案した方策の効果の検証が可能である．

6.3　分析対象のメトリクス

本研究で分析対象としたメトリクスを表 6-1 に示す．品質を判断するメトリクスには，出荷後のバグ数を用いた．なお，以降の記述において，各メトリクスの計測値は，各計測値の平均値を 1 とした相対値で表記している．

表 6-1　分析対象のメトリクス

種別	メトリクス	記号	説明
品質を表すメトリクス	出荷後バグ数	DEF	出荷後に摘出されたバグ数
プロダクトメトリクス	有効行数	ELOC	プログラムの実行に必要な記述を含む行の数
	サイクロマチック数	CYCLO	プログラムの制御フロー上における独立した経路の数
	分岐条件数	BRANCHS	複合条件を考慮して，各々の条件に分離してカウントした分岐条件の数
	最大ネスティング数	NESTS	制御命令やブロック記述などによるロジックのネスト数の最大値

6.4　メトリクスの相関分析

対象製品のシステム全体のメトリクス間の相関分析を行った．図 6-1 にメトリクスの散布図行列を，表 6-2 に相関係数行列を示す．また，対象製品は開発チームごとに 4 つのサブシステムに分割される．開発チームにより傾向の違いがあるかどうかを確認するために，各サブシステムの相関係数を表 6-3 に示す．なお，一部のファイルは初期化パラメータのみの関数で構成されており，ELOC の値が大きいもののその他のメトリクスは非常に小さい値となった．このようなファイルは相関係数の分析において無視できない影響を与えるが，特殊なファイルと位置づけて分析対象から除外した．

相関係数は，システム全体およびサブシステム別のどちらも類似の傾向が確認できた．ELOC，CYCLO，および BRANCHS の 3 種のメトリクス間は相関が強い傾向のため，以降の分析では，このうち ELOC をメトリクスの代表とした．これは他の文献[4-2]にも同様の結果が見られる．しかし，NESTS は ELOC との相関が弱い傾向のため，以降の分析では，ELOC および NESTS の 2 つのメトリクスにて分析を実施することとした．なお，DEF と ELOC および NESTS の間には強い相関は確認できなかった．

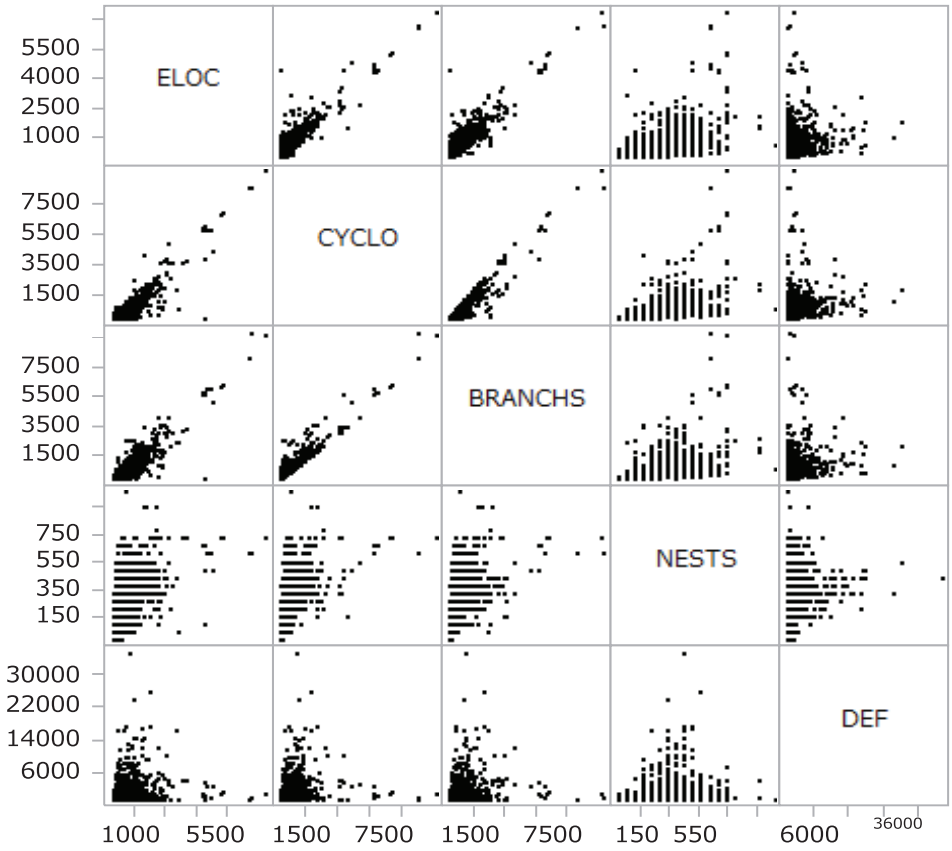


図 6-1　メトリクスの散布図行列（システム全体）

表 6-2　メトリクスの相関係数行列（システム全体）

	ELOC	CYCLO	BRANCHS	NESTS	DEF
ELOC	1.000	0.921	0.918	0.601	0.391
CYCLO	0.921	1.000	0.956	0.579	0.381
BRANCHS	0.918	0.956	1.000	0.559	0.385
NESTS	0.601	0.579	0.559	1.000	0.308
DEF	0.391	0.381	0.385	0.308	1.000

表 6-3　メトリクスの相関係数（サブシステム単位）

		ELOC	CYCLO	BRANCHS	NESTS	DEF
サブシステム A	ELOC	1.000	0.786	0.782	0.534	0.371
	CYCLO	0.786	1.000	0.937	0.465	0.402
	BRANCHS	0.782	0.937	1.000	0.452	0.442
	NESTS	0.534	0.465	0.452	1.000	0.235
	DEF	0.371	0.402	0.442	0.235	1.000
サブシステム B	ELOC	1.000	0.919	0.915	0.555	0.394
	CYCLO	0.919	1.000	0.949	0.551	0.391
	BRANCHS	0.915	0.949	1.000	0.517	0.376
	NESTS	0.555	0.551	0.517	1.000	0.353
	DEF	0.394	0.391	0.376	0.353	1.000
サブシステム C	ELOC	1.000	0.926	0.926	0.559	0.414
	CYCLO	0.926	1.000	0.956	0.549	0.371
	BRANCHS	0.926	0.956	1.000	0.536	0.396
	NESTS	0.559	0.549	0.536	1.000	0.312
	DEF	0.414	0.371	0.396	0.312	1.000
サブシステム D	ELOC	1.000	0.967	0.959	0.731	0.660
	CYCLO	0.967	1.000	0.977	0.698	0.657
	BRANCHS	0.959	0.977	1.000	0.684	0.637
	NESTS	0.731	0.698	0.684	1.000	0.570
	DEF	0.660	0.657	0.637	0.570	1.000

6.5　出荷後のバグ有無とメトリクスの分析

サブシステム別に，ELOC および NESTS の分布を，出荷後のバグ摘出の有無で層別した箱ひげ図を図 6-2 および図 6-3 に示す．相関分析では，DEF（出荷後バグ数）と ELOC および NESTS の間に強い相関は確認できなかったが，図 6-2 および図 6-3 の ELOC および NESTS の分布では，出荷後にバグ摘出が無い分布よりも出荷後にバグ摘出がある分布の方が各メトリクスの値が高くなっている．また，t 検定では 5%有意水準で 2 つの分布に有意差が認められる．

この結果より，出荷後に摘出されたバグの有無は，ELOC および NESTS と関係があると推測できる．

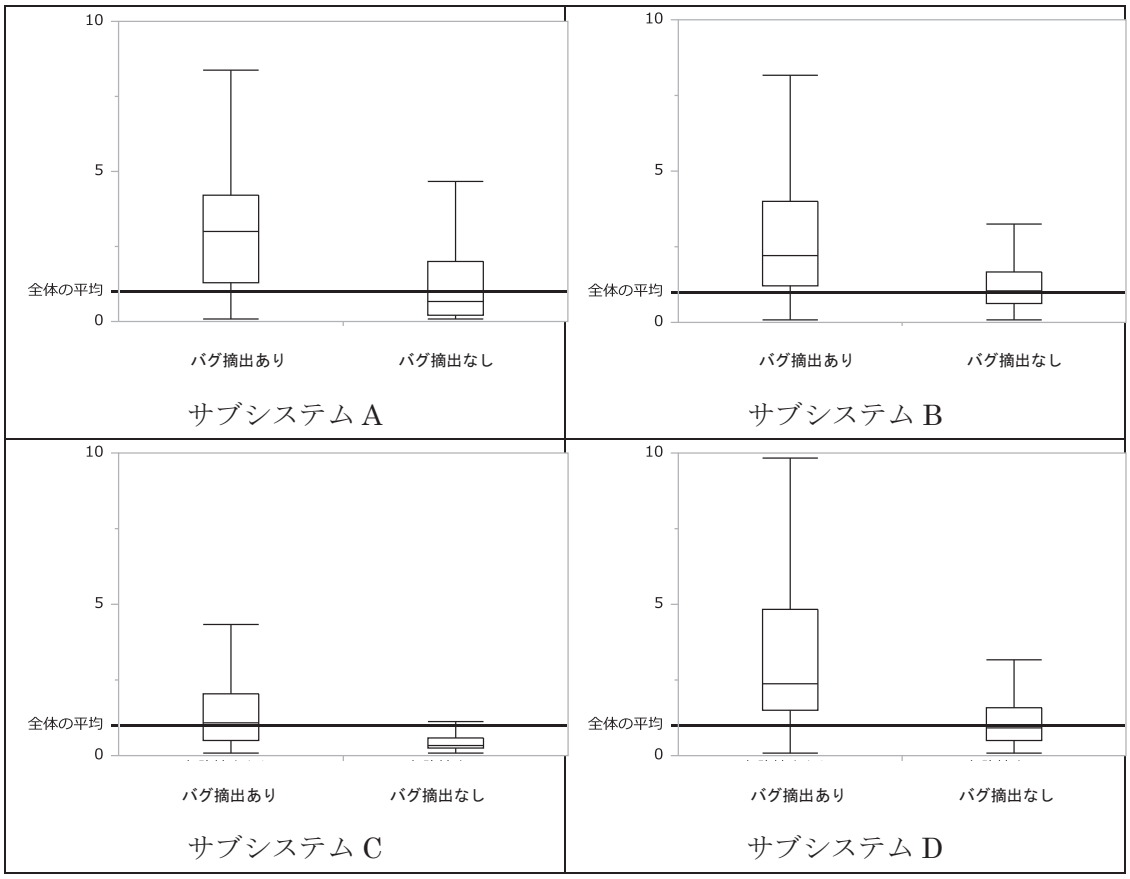
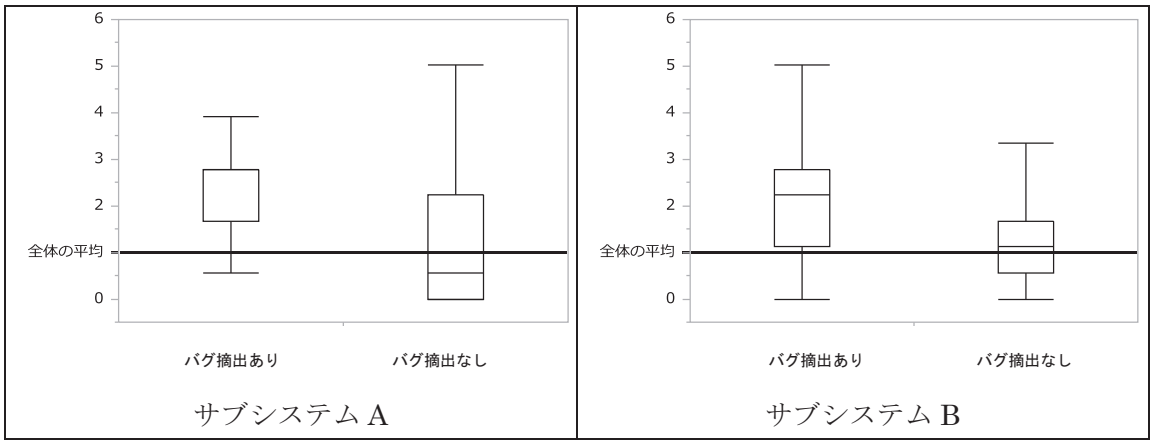


図 6-2　ELOC の箱ひげ図（全体平均を 1 とした相対表示）



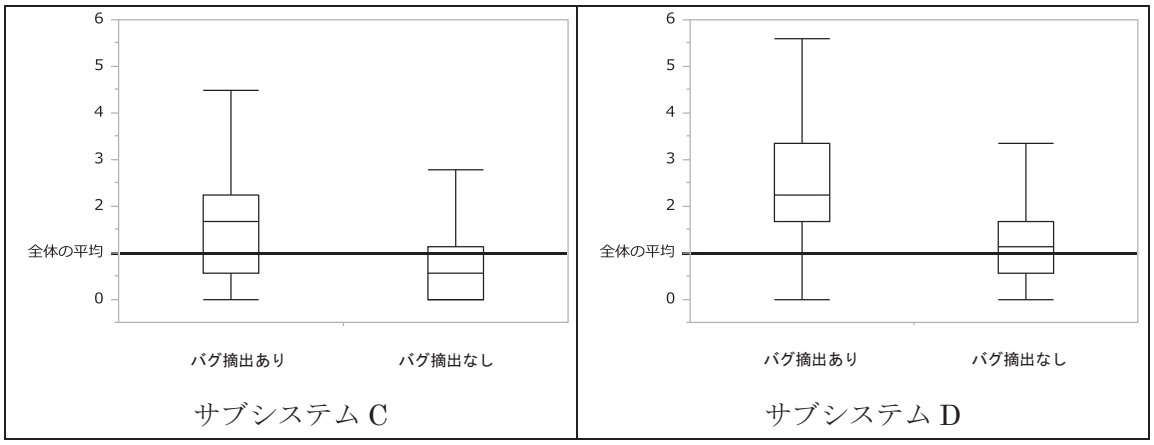


図 6-3　NESTS の箱ひげ図（全体平均を 1 とした相対表示）

6.6　ロジスティック回帰分析

ELOC および NESTS と，出荷後に摘出されたバグの関連をさらに検証するため，ステップワイズ法を用いたロジスティック回帰分析を行った．なお，ELOC は広い範囲に分布しているため，対数変換した後にロジスティック回帰分析を行った．回帰分析の結果を表 6-4 に示す．ELOC（対数値）および NESTS は共に p 値が 5%有意水準で有意であり，ELOC（対数値）および NESTS は出荷後のバグ摘出に影響を与えるメトリクスであることが示された．さらに，ELOC（対数値）および NESTS の回帰係数が正の値であることから，メトリクスの値が高いほど出荷後にバグが摘出される可能性が高くなることを示している．

表 6-4　ロジスティック回帰分析による回帰係数

パラメータ	回帰係数	p 値
切片	-1.581	0.0001 未満
ELOC（対数値）	1.039	0.0001 未満
NESTS	0.255	0.0001 未満

6.7　閾値設定によるバグ摘出の傾向分析

ELOC および NESTS の閾値を設定し，閾値以下のソースファイルと閾値を超えるソースファイルにおいて，出荷後のバグ摘出の有無に違いが見られるかどうかの傾向を分析した．暫定閾値として，ELOC を 200LOC（コード行数），NESTS を 4 に設定し，閾値の前後で層別した 2 つのグループのバグ摘出率を示す（表 6-5）．この場合のバグ摘出率とは，評価対象の全ソースファイル数に対する，出荷後に発生したバグによるソースを修正したファイル数の割合で算出する．なお，t 検定では 5%有意水準で層別した 2 つの分布の有意差が認められる．表 6-5 より，全てのサブグループについて，ELOC または NESTS の閾

値を超える場合のバグ摘出率が，閾値以下の場合のバグ摘出率よりも約 2〜7 倍高いことが確認できた．

表 6-5　暫定閾値の前後で層別した出荷後のバグ摘出率

		閾値を超えるグループのバグ摘出率 a	比率 (a/b)	閾値以下のグループのバグ摘出率 b
ELOC（閾値は 200LOC）	サブシステム A	60.0%	2.0	29.7%
	サブシステム B	82.5%	2.2	37.6%
	サブシステム C	75.7%	6.8	11.2%
	サブシステム D	83.3%	2.3	35.7%
	システム全体	78.0%	5.0	15.7%
NESTS（閾値は 4）	サブシステム A	54.6%	2.6	20.9%
	サブシステム B	66.7%	2.0	34.2%
	サブシステム C	60.3%	6.2	9.8%
	サブシステム D	74.8%	2.6	28.7%
	システム全体	62.6%	4.7	13.2%

6.8　開発プロセスへの適用

前節の結果より，ELOC および NESTS に閾値を設定し，閾値を超えるグループが効率よく品質改善活動が行える見通しができた．本節では，開発プロセスへ適用する閾値として暫定閾値よりも適切な値が存在するかどうかの検討を行った．検討の手法は，暫定閾値を中心にその前後に閾値を変化させて，閾値の前後におけるバグ摘出率を算出し，閾値を超える場合のバグ摘出率が，閾値以下の場合のバグ摘出率の何倍であるかを計測した．この倍率が 1 の場合は，閾値の前後でバグ摘出率が同一であるため，閾値による品質の良し悪しの判断ができない．また，閾値を低く設定すると，全ファイルに対する閾値を超えるファイルの割合が高くなり，閾値の導入効果が低くなる．

図 6-4 に ELOC の閾値を変化させた場合のバグ摘出率の倍率の推移，図 6-5 に NESTS の閾値を変化させた場合のバグ摘出率の倍率の推移を示す．閾値を大きくした場合，最も倍率が低くなるのは ELOC および NESTS ともにサブシステム A であった．また，サブシステム A 以外の倍率は 2 以上で推移している．本研究では，倍率が全てのサブシステムで 2 を下回らない範囲で，ELOC および NESTS が最大となる値を閾値として採用し，開発プロセスに適用することにした．具体的には，図 6-4 および図 6-5 より，ELOC が 150LOC（コード行数），NESTS が 4 となる．

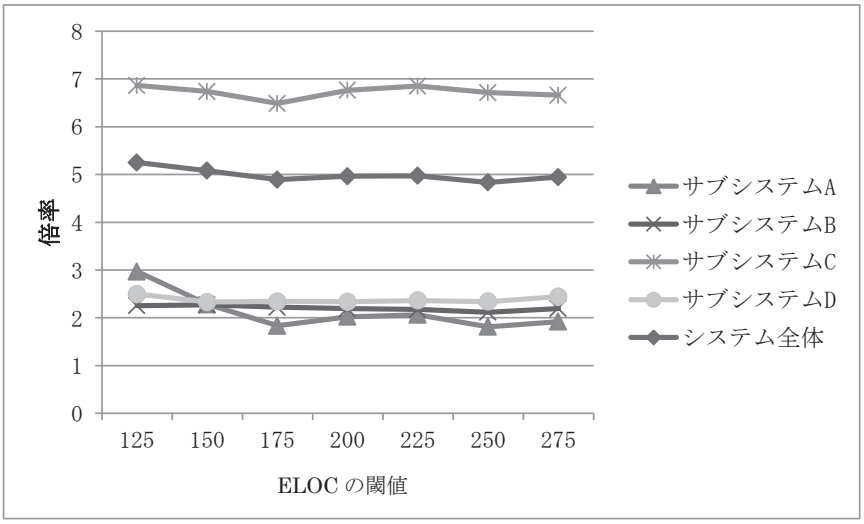


図 6-4　ELOC のバグ摘出率の倍率の推移

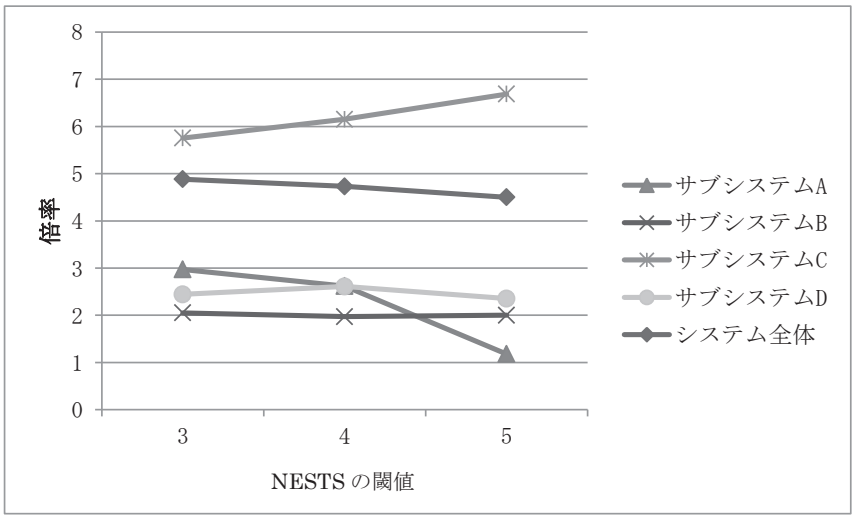


図 6-5　NESTS のバグ摘出率の倍率の推移

前節では, ELOC の暫定閾値を 200LOC(コード行数)に設定していたため, 改めて, ELOC の閾値を 150LOC（コード行数）に設定し直して，閾値の前後で層別した出荷後のバグ摘出率を分析した（表 6-6）．なお，t 検定では 5%有意水準で層別した 2 つの分布の有意差が認められた．表 6-6 より，ELOC の閾値を 200LOC（コード行数）から 150LOC（コード行数）に変更した場合も，全てのサブグループにおいて，ELOC の閾値を超える場合のバグ摘出率が，閾値以下の場合の欠陥摘出率よりも，約 2～7 倍高いことが確認できた．

表 6-6　変更後の閾値の前後で層別した出荷後のバグ摘出率

		閾値を超えるグループのバグ摘出率 a	比率 (a/b)	閾値以下のグループのバグ摘出率 b
ELOC（閾値は 150LOC）	サブシステム A	64.0%	2.3	28.0%
	サブシステム B	81.7%	2.3	36.0%
	サブシステム C	73.7%	6.8	10.9%
	サブシステム D	79.7%	2.3	34.2%
	システム全体	76.6%	5.1	15.1%

表 6-7 では，ELOC および NESTS の閾値前後の組み合わせによる 4 つのグループのバグ摘出率を示す． ELOC および NESTS の両方が閾値を超えるグループが，最もバグ摘出率が高く 80%を超える．次にバグ摘出率が高いのは，ELOC または NESTS のどちらか一方が閾値を超えるグループである．ELOC または NESTS のどちらか一方が閾値を超える場合のバグ摘出率は 50～60%であることから， 最もバグ摘出率が高い組み合わせとなる，ELOC および NESTS の両方が閾値を超える場合のみを，品質改善活動の対象とする．

表 6-7　閾値前後の組み合わせで層別したバグ摘出率

	ELOC	NESTS	バグ摘出率
システム全体	閾値を超える	閾値を超える	82.4%
	閾値を超える	閾値以内	56.2%
	閾値以内	閾値を超える	53.9%
	閾値以内	閾値以内	12.9%

これらの結果より，具体的に以下のような開発プロセスへの適用方策を設定し，実際に，ソフトウェア品質を安定的に確保する活動を推進している．

(1) ELOC は 150LOC（コード行数），および NESTS は 4 を閾値として設定し管理を実施する．
(2) ELOC または NESTS のどちらか一方でも閾値を超えた場合は，閾値を超えたソースコードについて開発部門に警告して，リファクタリングやネスティング階層の妥当性確認等の方策の検討を実施する．

6.9 まとめ

第 6 章では，第 5 章とは別の組織の製品におけるソースコードの複雑性を示すプロダクトメトリクスと出荷後の製品品質との相関関係を分析した．その結果，有効行数と最大ネスティング数のメトリクスの値が大きいほど出荷後の品質が低い傾向があった．したがって，モジュールの有効行数が 150LOC（コード行数），および最大ネスティング数が 4 を閾値として，閾値を超えるソースコードに対して開発部門に警告して，リファクタリング等を実施することによりメトリクスの値を低減することが，ソフトウェア品質を確保するためには有効であることを確認した．これらのことは，結局，適切なモジュール分割を推奨していることを意味する．モジュール行数が異常に大きかったり，ネストが異常に深かったりすると，ロジック上のバグを作り込みやすく，テスト項目が漏れやすく，さらにはコードの可読性の低下による担当者交代後の改造における品質リスクが高まると考えられる．特に筆者の組織が提供している製品は 10 年以上の維持や改造が必要になる場合が多く，可読性や改造のしやすさは品質維持の重要な要素になる．したがって，これらのプロダクトメトリクスを活用した定量的品質マネジメントが有効であると考える．

第 7 章　定量的品質マネジメント改善のためのプロダクトメトリクスの研究 3 ～ソースコード複雑性のプロダクトメトリクスの単体テストにおける検証～

7.1　はじめに

第 5 章および第 6 章では，定量的品質マネジメント技術の「測る」ことを高度化するための有効なプロダクトメトリクスの抽出について，ソースコードの複雑性と出荷後の製品品質の相関関係の分析を 2 つの実践研究で行った．いずれも，プロダクトメトリクスの中では最大ネスティングが，出荷後品質への影響が強いことが判明した．

本章では，同様にソースコードの複雑性のメトリクスの品質への影響を検証するが，前章までと異なるのは，品質への影響の検証を単体テストにおけるバグの見逃しリスクを用いて分析している点である．この理由は，出荷後の製品品質の影響を調べる場合に，出荷後の製品品質がメトリクス以外のさまざまな要因に左右されることが考えられるからである．例えば，製品の利用数やお客様の使い方の違いなどにより，潜在しているバグが顕在化する可能性が大きく変わる．したがって，単体テストにおけるバグの見逃しのリスクの検証は，出荷後のバグ数をもとにした検証よりも，検証結果としてはより有効であると考えられる．

筆者の組織は，第 3 章で述べたソフトウェアファクトリを利用することにより，ソースコードの複雑性に関するメトリクスを自動計測する仕組みと，単体テストの自動化による単体テストのメトリクス（単体テスト項目密度および単体テストカバレッジ）を自動計測する仕組みを実現している．そこで本研究では，単体テスト後のリスク評価に役立つメトリクスについて，実製品の分析結果を述べる．

まず，単体テストで見逃したバグ数とソースコードの複雑性に関するメトリクスの相関関係を分析した．次に，単体テスト項目密度および単体テストカバレッジのメトリクスを加えた分析結果を述べる．この分析結果をもとに，バグ見逃しリスクに応じた必要な対策について論じる．

7.2　分析対象データ

本研究では，ソフトウェアファクトリを利用している汎用的なシステムソフトウェア製品のうち，以下の条件に合致する製品を分析対象データとして選定した．

(1) 単体テスト（以下 UT と記す）がテストコードを用いて自動化されている．
(2) カバレッジを自動計測している．
(3) ソースコードが構成管理サーバに格納されている．
(4) UT 完了後から開発完了までに摘出したバグがバグトラッキングシステム（以下 BTS と記す）に登録されている．

(5) BTS から当該バグに対する修正ファイル，修正内容，および修正差分をトレースできる．

(1)のテストコードに記載されたテストメソッド数をテスト項目数としてカウントした．テストメソッド数をカウントすることで，通常のテスト項目数のカウント方法の曖昧さを排除した．(2)ではステートメントカバレッジとデシジョンカバレッジを採用した．(3)の構成管理サーバに格納されたソースコードから各種プロダクトメトリクスを測定した．(4)の BTS に登録されているバグから，UT で見逃したバグの数をファイル単位に集計した．ファイル単位のバグ数は，(5)の修正ファイル数を集計した．なお，UT で見逃したバグとは，詳細設計およびコーディング工程で作り込まれたバグが，UT で見逃されて，UT 完了後から開発完了時までに摘出されたバグのことである．なお，今回の分析対象規模は 24KLOC（1,000LOC（コード行数））であり，開発言語は Java である．

7.3 分析対象メトリクス

分析対象としたメトリクスを表 7-1 に示す．まず，UT の品質を判断するメトリクスとして，UT 見逃しバグ数を設定した．ソースコードの複雑性に関するメトリクスおよび UT メトリクスは，ソフトウェアファクトリを用いて組織的に自動収集可能なメトリクスを採用した．全てのメトリクスはファイル単位に集計を行った．本研究では，UT 見逃しバグに影響を与えるメトリクス，およびそれらの組み合わせのメトリクスの導出を実施する．なお，以下では，C0，C1 をまとめて表現する場合は UT カバレッジと称す．

表 7-1 分析対象メトリクス

種別	メトリクス	記号	説明
バグに関するメトリクス	UT 見逃しバグ数	DEF	UT で見逃され，UT 完了後から開発完了時までに摘出されたバグ数
ソースコードの複雑性に関するメトリクス	規模	ELOC	コメントを除く有効行数
	サイクロマチック数	CYC	McCabe のサイクロマチック複雑度[5-1]
	分岐条件数	BRA	複合条件を考慮した分岐条件の数
	最大ネスティング数	NEST	ファイル内の最大ネスティング数
UT に関するメトリクス	UT 項目数	UTI	UT 用のテストコードに記述されたテストメソッド数
	UT 項目数密度	UTD	UT 項目数 / 開発規模(KLOC)
	C0	C0	UT のステートメントカバレッジ
	C1	C1	UT のデシジョンカバレッジ

7.4 UT 見逃し欠陥のリスク評価

7.4.1 メトリクス間の相関分析

まず，収集したメトリクスの相関関係を俯瞰するため，各メトリクス間の相関分析を行った．図 7-1 に，メトリクス全体の散布図行列を示す．対象としたメトリクスは，開発規模，サイクロマチック数，分岐条件数，最大ネスティング数，UT 項目数，C0，C1，および UT 見逃しバグ数である．図 7-1 の各メトリクスは，表 7-1 の記号を用いて表現している．図 7-1 の右上は相関係数の組み合わせ，対角線はヒストグラム，左下は散布図の組み合わせである．目盛は平均を 100 とした場合の相対値である（ただし，UT 見逃しバグ数のみ絶対値）．

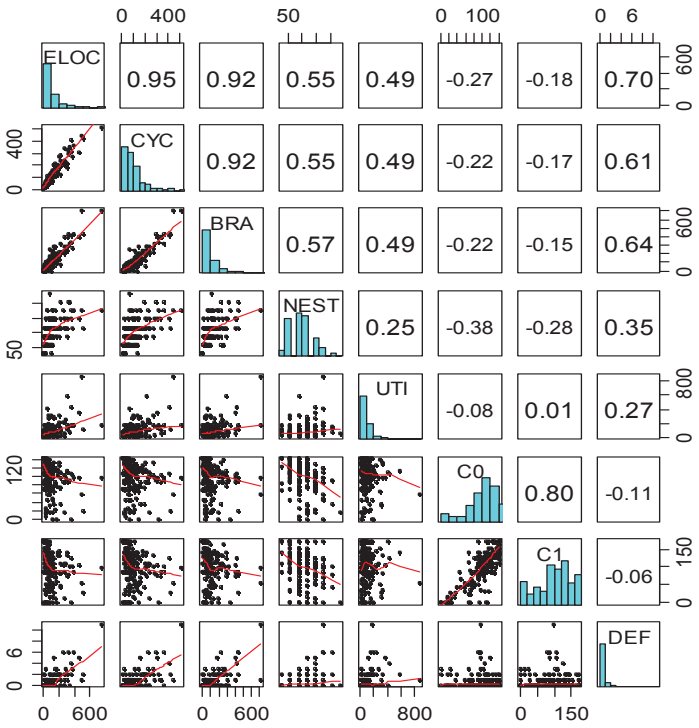


図 7-1 メトリクス全体の散布図行列

UT 見逃しバグ数と各メトリクスの相関を見ると，開発規模，サイクロマチック数，および分岐条件数は，中程度の相関がある．しかし，これら 3 つのメトリクスはお互いに強い相関がある．したがって，UT 見逃しバグ数との相関は，最もその値が高い開発規模を用いて説明できる．

また，最大ネスティング数は UT 見逃しバグ数と弱い相関があることが分かった．一方，UT 項目数，C0，および C1 などの UT に関するメトリクスについて，UT 見逃しバグ数の直接的な相関は見られなかった．

7.4.2　UT 見逃しバグの層別分析

図 7-1 の UT 見逃しバグ数のヒストグラムに着目すると，見逃しバグ数が 0 件の場合が，グラフが突出していることが分かる．そこで，UT 見逃しバグ数に関して，1 件ずつではなくデータ群として層別化を行った．具体的には，UT 見逃しバグ数が 0 件，1～2 件，3 件以上の 3 つに層別し，開発規模，最大ネスティング数，UT 項目数密度，C0，C1 に対して分析を行った．なお，UT 項目数ではなく，UT 密度を採用した理由は，開発規模による影響を排除するためである．各メトリクスの中で，最も特徴がみられた開発規模について，UT 見逃しバグ数の分布（箱ひげ図）を図 7-2 に示す．横軸が 0 の場合は，UT 見逃しバグ数が 0 件であったファイルの分布，1 の場合は UT 見逃しバグ数が 1 件または 2 件であったファイルの分布，2 の場合は UT 見逃しバグ数が 3 件以上であったファイルの分布を示す．縦軸は，ソースファイルの開発規模の平均値を 100 とした相対値である．

図 7-2 を見ると，UT 見逃しバグ数が 0 件のファイルは，そのほとんどが開発規模の相対値が 160（破線）以下に分布し，約 75%が開発規模の相対値 100(一点鎖線)以下に分布していることが分かる．また，UT 見逃しバグ数が 1 件または 2 件のファイルは，約 50%が開発規模の相対値=100 以下に分布しており，UT 見逃し欠陥数が 3 件以上のソースファイルは，その約 75%が規模の相対値 160 以上に分布していることが分かる．

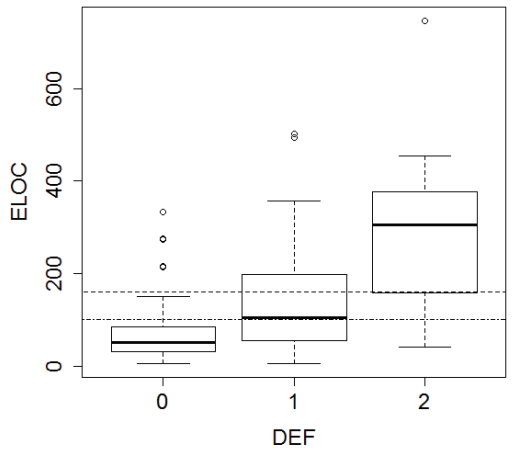


図 7-2 開発規模(ELOC 相対値)と UT 見逃しバグ数(DEF)の分布

7.4.3　分析結果の評価

前節で分析した開発規模に関して基準値を設けた場合の評価を実施する．評価指標は，表 7-2 の適合率，再現率，および F 値を用いる．

表 7-2 規模の評価指標

評価指標	説明
ファイルの適合率	基準値外ファイルのうち，UT 見逃しバグを（1 件以上または 3 件以上）含むファイルの割合
ファイルの再現率	UT 見逃しバグを（1 件以上または 3 件以上）含む全ファイルのうち，基準値外ファイルの割合
ファイルの F 値	ファイルの適合率と再現率の調和平均
バグの再現率	全 UT 見逃しバグ数のうち基準値外ファイルに含まれる UT 見逃しバグ数の割合

表 7-3 に前節で分析した開発規模の相対値（100～160）近傍で最も F 値が高かった値を基準値とした場合の評価結果を示す．本基準値を用いて基準値外のファイルを抽出した場合，以下のことが分かる．

(1) ファイルの適合率より，基準値外ファイルの 7 割強は，UT 見逃しバグを含んでいる．
(2) ファイルの再現率より，UT 見逃しバグを 3 件以上含むファイルの 9 割強が基準値外のファイルである．
(3) ファイルの F 値より，基準値の有効性が示されている（経験的に 55%以上が有効）
(4) バグの再現率より，基準値外ファイルには，全 UT 見逃しバグの 7 割強が含まれている．

(1)～(4)より，本基準値を用いれば高い確率で UT 見逃しバグのあるソースファイルを検出可能であることが分かった．

なお，他のメトリクス（最大ネスティング数，UT 項目数密度，C0，C1）を用いた場合，開発規模の適合率および F 値を超えるメトリクスは存在しなかった．以上より，UT 見逃しバグとの関係において，開発規模が最も有用なメトリクスの一つであることが分かった．

表 7-3 開発規模の評価結果

基準値	UT 見逃しバグ数	ソースファイル			バグ
		適合率	再現率	F 値	再現率
125	>0	**72%**	58%	64%	**74%**
	≥3	23%	**92%**	37%	

7.5 バグ見逃しリスク対策

前節では，UT におけるバグ見逃しリスク要因として，単純ではあるが開発規模が最も有用なメトリクスであることを確認した．本節では，さらにリスク対象ファイルを抽出後の方策について議論する．

7.5.1 開発規模に基づく分析

表 7-3 より，開発規模が大きいソースファイルに含まれる UT 見逃しバグは全体の 7 割であるが，開発規模が小さいソースファイルにも残りの 3 割の UT 見逃しバグが含まれることが分かる．場合によっては開発規模が小さいソースファイルに対しても方策が必要な可能性もあるため，本節では，開発規模の大小で分けた場合の対策について分析を実施する．

(1) メトリクス全体の分析

図 7-3 に開発規模の大小で分けた場合の散布図行列を示す．メトリクスは開発規模，UT 項目数密度，C0，C1，および UT 見逃しバグ数である．目盛は図 7-1 と同様に各メトリクスの平均を 100 とした場合の相対値である．ただし，UT 見逃しバグ数のみ絶対値である．

まず，開発規模が小さい場合を分析する．図 7-3(a)を見ると，以下が読み取れる．

A) UT カバレッジが高く UT 項目数密度が高い場合，UT 見逃しバグ数が少ない．
B) UT 項目数密度が低い場合，UT カバレッジが高くても UT 見逃しバグは存在する．

開発規模が小さい場合は，A)より，UT カバレッジの向上が UT 見逃しバグを削減する一つの方策と考えることができる．また B)より，UT カバレッジが高くても UT 項目数密度が低いソースファイルは，UT 項目数密度を向上させることが一つの方策と考えられる．

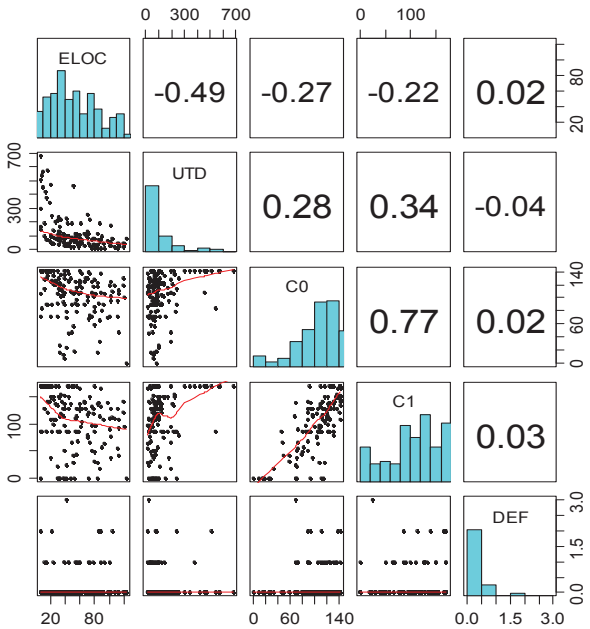
次に開発規模が大きい場合を分析する．図 7-3(a)と(b)を比較すると，以下が読み取れる．

C) 開発規模が小さい場合と比較して，全体的に UT 項目数密度が低い．

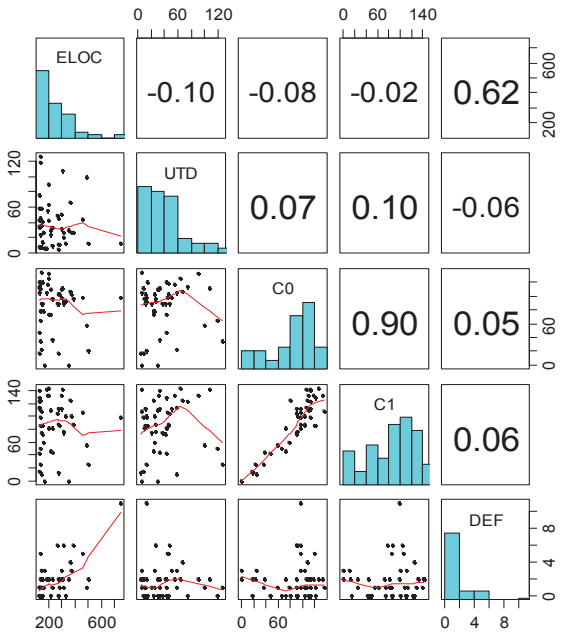
また，図 7-3(b)を見ると，以下が読み取れる．

D) C0，C1 共に，相対値 100 近辺で UT 見逃しバグが多い．

開発規模が大きい場合は，C)より，全体的に UT 項目数密度の底上げが必要であることと，D)より UT カバレッジだけでは判断できないことが分かる．



(a)開発規模小の場合



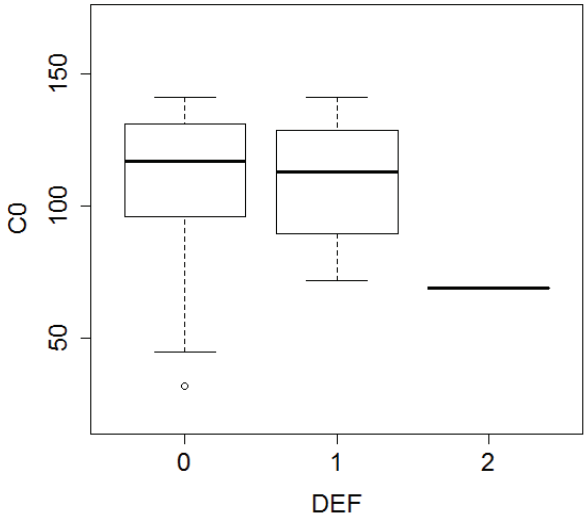
(b) 開発規模大の場合

図 7-3 開発規模の大小で分けた場合の散布図行列

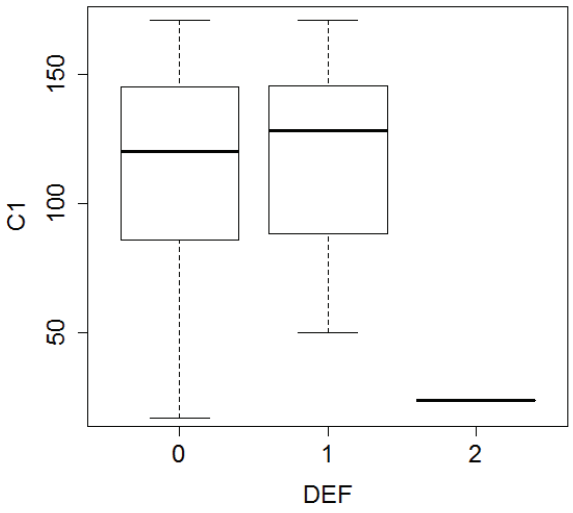
(2) UT カバレッジの分析

次に，開発規模が小さい場合の UT カバレッジについて，UT 見逃しバグ数の分布を分析する（図 7-4）．なお，UT カバレッジが極端に低いソースファイルは，別の要因が考えられるため外れ値として除外した．横軸は図 7-2 と同じであり縦軸は，(a)が C0，(b)が C1 である．なお，それぞれ平均値を 100 とした相対値である．UT 見逃しバグ数が 3 件以上の場合は，C0，C1 ともに低い値(平均以下)を示している．

したがって，開発規模が小さい場合，UT カバレッジを向上させる方策の必要性が裏付けられた．



(a) C0 の場合



(b) C1 の場合

図 7-4 UT カバレッジと UT 見逃しバグ数の分布（開発規模小）

(3) UT 見逃しバグの修正内容分析

次に，UT 見逃しバグのテスト漏れ観点を分析するため，UT 見逃しバグの内容を BTS より抽出し，UT にて摘出すべきテスト区分を調査した．テスト区分は，構造ベースドテストと仕様ベースドテスト [7-1]のどちらか一方に分類した. その結果を, 表 7-4 に示す. また，それぞれのテスト区分に分類した結果を表 7-5 に示す．具体的には，BTS から UT 見逃しバグの原因を分析し, その原因を除去する方策 (テスト技法) を導出した. さらに, そのテスト技法をテスト区分に分類した.

表 7-4 見逃しバグを摘出すべきテスト区分

	構造ベースドテストの割合	仕様ベースドテストの割合
修正ファイルが全て開発規模小のバグ	100%	0%
修正ファイルが全て開発規模大のバグ	22%	78%

表 7-5 テスト区分分類事例

テスト区分	バグの原因	テスト技法
構造ベースドテスト	クラス名称誤り，設定値誤り 判定ロジック評価漏れ	ステートメントカバレッジ デシジョンカバレッジ
仕様ベースドテスト	状態遷移評価漏れ 例外処理評価漏れ コマンドオプション評価漏れ 入力バリエーション評価漏れ	状態遷移テスト ユースケーステスト デシジョンテーブル 同値分割/境界値テスト

表 7-4 を見ると，開発規模小の場合は，構造ベースドテストが不足していることが分かる．一方，開発規模大の場合は，仕様ベースドテストが不足している割合が高いことが分かる．次に，各バグの修正内容を確認した．開発規模小のソースファイルの修正内容は，条件文を含む修正はなく，修正行数が実質 1～2LOC（コード行数）であったのに対し，開発規模大のソースファイルの修正は，条件文を含む修正，すなわちロジックに影響する修正がほとんどであった.

以上の確認結果は，開発規模小のソースファイルに対しては構造ベースドテスト，開発規模大のソースファイルに対しては仕様ベースドテストの必要性を示唆している.

7.5.2 定性的リスク分析と対策

前節で，ソースファイル規模の大小による，リスクと方策について議論した．この結果を定性的リスク分析として発生確率-影響度マトリクスにマッピングすると表 7-6 のようになる．ここで，発生確率とは，見逃しバグが UT 後に発生する確率であり，影響度とは，方策を実施する場合の手戻り工数を意味している．

表 7-6 発生確率・影響度マトリクス

		影響度	
		大	小
発生確率	高	開発規模大&UT 項目数密度小	開発規模大&UT 項目数密度大
	低	開発規模小&UT カバレッジ小	開発規模小&UT カバレッジ大

本研究では，表 7-6 に組織別・製品別の優先度をあてはめて，リスク管理を実施することを提案する．ここで，それぞれのリスクに対する方策は以下の通りである．

(1) 開発規模が小さい場合

UT カバレッジ（C0，C1）を向上させることを方策とする．

(2) 開発規模が大きい場合

UT 項目数密度の底上げが必要である．ただし，闇雲に UT 項目数を追加して UT 項目数密度を向上させても効果が薄く，仕様ベースドテスト観点の強化が必要である．

7.6 考察

本研究で得られた結果を以下に考察する．

(1) UT 品質に影響を与えるメトリクスについて

A) UT 完了時点のプロダクト品質に影響を与えるメトリクスとして開発規模が最も有用である．

B) 開発規模が一定未満の場合は，UT の品質を確保できる可能性が高い．

C) 開発規模が一定以上の場合は，UT の品質確保が困難になる可能性が高い．

(2) UT の十分性について

A) 単純にカバレッジだけでは UT の十分性を判断できない．ただし，開発規模が一定未満の場合は，カバレッジを用いて十分性を判断できる可能性がある．

B) 開発規模が一定以上の場合は，仕様ベースドテストが UT の十分性に影響を与える可能性が高い．

(3) UT の十分性を確保する対策について

A) 単体テスト対象の開発規模（本稿ではソースファイルの規模）を可能な限り一定以内に収める．

B) 一定開発規模未満の場合は，カバレッジ強化を重点的に実施する．

C) 一定開発規模以上の場合は，カバレッジだけでは不十分であり，仕様ベースドテストをより強化する必要がある．

7.7 まとめ

本章では，ソースコードの複雑性を示すプロダクトメトリクスと製品品質との相関関係を検証した．製品品質として UT 完了時点の品質リスクの関係性の評価を行った．様々なメトリクスを用いて評価した結果，単純ではあるが開発規模が最も有用なメトリクスであることを示した．開発規模の大小に応じたリスク評価を行ったあと，それぞれに必要な方策を導出した．この開発規模の基準値を活用すれば，UT 以降にリスクを含んだモジュールやファイルをある程度特定でき，それらに必要な方策を実施することが可能となることを確認した．

今後，ソフトウェアファクトリの利用拡大に伴い，UT が自動化された製品開発の増加が見込まれる．その結果，これらの製品に対しても今回対象としたメトリクスを今回と同じ粒度で測定可能となる．その際には，今回対象としたメトリクスに関する組織全体としての基準値の設定と，方策の立案をしたいと考えている．ただし，画一的な基準値と方策だけではなく，製品特性（事業領域，開発言語，使用するフレームワークなど）も踏まえた議論も，合わせて必要と考えている．

第 8 章　定量的品質マネジメント改善のためのプロダクトメトリクスの研究 4 ～設計レビューに関する有効なプロダクトメトリクスの検証～

8.1　はじめに

第 5 章から第 7 章までは，定量的品質マネジメント技術の高度化に向けて実施するべき「測る」ことと「判断する」ことの高度化のうち，「測る」ことの高度化について，プロダクトメトリクスの中でもソースコードの複雑性と品質との相関関係を評価してきた．

本章では，コーディング工程よりも上流に位置する設計工程において，レビュー記録票をもとにした新たなメトリクスについて検討する．初めに，プロセスメトリクスの課題について再度整理し，次に，設計レビューのレビュー記録票をもとにしたプロダクトメトリクスの評価と品質への影響の分析結果を論じる．

8.2　プロセスメトリクスの課題

ソフトウェアの品質を確保するためには，レビューやテストを着実に実施することが重要である．特に，上流工程における品質の確保手段である設計書のレビューは，各工程における成果物の完成度を高め，最終的な品質に大きな影響を与える．また，レビューを十分に実施することができれば，開発途中に検出されたバグの修正作業による後戻り工数を減らし，生産性向上にもつながる．

筆者の組織では，品質向上の方策として，プロセスメトリクスの一つであるレビュー工数の基準値を定め，レビュープロセスを着実に実行してきた．その結果，製品リリース後のバグは減少し，一定の成果を得た．しかし，レビューは，工数が投入されればされるほど品質が良くなるという性質のものではない．第 3 章で述べたとおり，レビュー工数がある程度確保された状態の開発プロセスを安定的に運用できるようになれば，レビュー工数はそれ以上には増加しない傾向にある．したがって，開発プロセスが成熟した組織では，レビュー工数というプロセスメトリクスによる品質確保の方策には効力の限界がくる．

そこで，レビュー工数以外にレビューの質を確保する要素には何があるかについて検討した結果，レビュー記録票に記載されている指摘事項に関するテキスト文から得られる情報をもとに，設計品質に影響を与えるレビューのプロダクトメトリクスを測る方法を考えた．なぜなら，設計担当者による設計書の完成度が設計品質を大きく左右する要因となるが，設計書の完成度は，設計者個人のスキルや開発難易度により大きく変動する．設計者のスキルや開発難易度をメトリクスとして品質への影響の大きさを定量化する試みもあるが，設計書の完成度を高いレベルに保つためのプロセス管理面の方策を講じにくい．しかし，設計者による設計書の完成度に関わらず，設計書のレビューという作業を高いレベルで実行す

ることができれば，設計者個人のスキルや開発難易度による影響を抑えて，設計書の最終的な完成度を高いレベルに保つことが可能であると考えられる．

レビューに関する過去の研究では，ピアレビュー会議の重要性を説く文献[8-1][8-2][8-3]などをはじめとして，ピアレビュー会議の実施方法や会議工数に関わる効率性を含めた分析が多い[8-4][8-5][8-6][8-7][8-8][8-9][8-10][8-11][8-12][8-13]．また，これらとは別の視点からレビューについて論じている研究には，人的要因分析の品質工学的アプローチ[8-14][8-15]や，開発プロセスの評価や改善によるアプローチ[8-16][8-17]の実践研究はあるが，レビュー記録からプロダクトメトリクスを使う手法の実践研究は少ない．

本研究では，レビュー記録の情報から得られるメトリクスから設計品質に影響を与える要因を分析した．

まず，主要なプロセスメトリクスの相関関係を調べてみた(図 8-1 および表 8-1)．なお，この分析では，9 つの開発チームの 2008 年から 2014 年までの 7 年間分の 63 個のデータセットを使用した．上工程摘出バグ密度とテスト工程摘出バグ密度を目的変数として，各メトリクスの相関関係を調べてみた．その結果，上工程摘出バグ密度は，レビュー工数密度との相関係数が 0.737 と強い正の相関を示している．しかし，テスト工程摘出バグ密度は，強い相関を示すメトリクスが見当たらない．例えば，テスト工程摘出バグ密度に直接的な影響を与えそうなメトリクスであるテスト項目数密度およびテスト工数密度は，相関係数が 0.24 および 0.14 であり，テスト摘出バグ密度との相関は強くなかった．

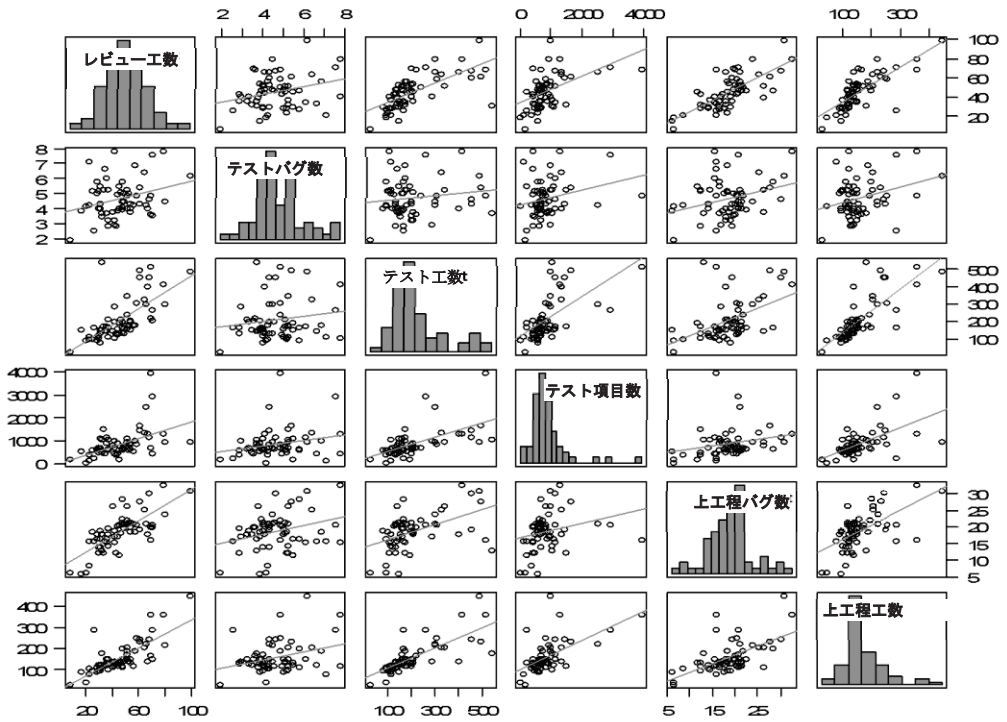


図 8-1 上工程摘出バグ数およびテストバグ数と他のメトリクスの散布図行列

したがって，レビュー工数が増加することで，上工程摘出バグ数が増加し，その結果，テスト工程摘出バグ数が減少するという論理は成立しない．つまり，レビューは品質を確保する重要な方策ではあるが，レビュー工数のみがレビューの質を表す指標とは言い切れない．レビュー工数以外のメトリクスを用いて設計品質を測る必要がある．筆者の組織は，日本のソフトウェア開発プロジェクトの平均レビュー工数の約 2 倍を確保している成熟したプロセスを持つ[2-11][3-4]ことからも，レビュー工数以外に設計品質を測るメトリクスが必要となる．

そこで，レビュー記録のテキストから得られる情報をもとに，設計品質に影響を与えるレビューに関するプロダクトメトリクスを測る方法を考察した．

表 8-1 上工程摘出バグ数およびテスト工程摘出バグ数と他のメトリクスの相関係数

	上工程工数密度	レビュー工数密度	テスト工数密度	テスト項目数密度	上工程摘出バグ密度	テスト工程摘出バグ密度
上工程摘出バグ密度	0.603***	0.737***	0.484***	0.248*	1	0.296*
テスト工程摘出バグ密度	0.305*	0.283*	0.146	0.244	0.296*	1
*5%有意， **1%有意， ***0.1%有意						

8.3 レビュープロダクトメトリクスの測定方法

設計品質の向上とは，設計工程の成果物となる設計書の品質が向上することである．設計品質を向上させる最適な方策は，設計書のドラフト作成工程において，高品質で完成度の高い設計書を初めから作成できることである．また，その方策は品質面の後戻り工数を最小限にできる．しかし，設計書のドラフト作成工程において，高品質で完成度の高い仕様書を作成することは，設計者のスキルや開発難易度にかなり依存すると考えられる．もちろん，設計者のスキルに依存しない設計を実施するための開発プロセス標準の高度化も重要である．例えば，設計の曖昧さを排除するための形式手法，UML 等の非言語系の仕様記述，構造化設計，およびオブジェクト指向設計などが開発プロセス標準の高度化の方策に該当する[2-3]．しかし，これらの技術の習得も設計者のスキルに依存する場合が多いと考える．

ところで，設計者による設計書のドラフト作成工程の次の工程にあたるレビュー工程では，少なくとも複数の技術者による多くの視点からレビューを実施するため，技術者個人のスキルに依存することは少なくなると考えられる．また，設計書のドラフト作成工程において，設計書の完成度や品質が低い場合でも，レビュー工程において，多くの技術者がレビューすることで，最終的には設計書の品質や完成度を高めることができる．

レビュー記録票

第4.1版

日付				処置内容承認		処置内容査閲	
時間							
場所							
開発部門				担当者			
プロジェクト名				開発管理番号			
開発項目(連携ｷｰ)				コンポーネント名			
文書番号				版数			
レビュー対象物							
レビュー方法		レビューフェーズ(連携ｷｰ)		レビュー回数			
工数(H)合計	0.00	予定チェック項目数		実施チェック項目数			
		指摘件数合計	0	内処置済み指摘件数合計			0
		摘出バグ数	0	内処置済みバグ数			0
		指摘件数(バグ以外)	0	内処置済み指摘件数(バグ以外)			0
		未判定件数	0	未完了件数			0
グループ名							
レビューア							
工数(H)							

項番	ページ／行	章、モジュール名	指摘事項	レビューア	指摘分類	バグ判定	作込フェーズ	重要度	影響範囲	修正判定	処置内容	完了予定日	完了日付

図 8-2 レビュー記録表テンプレート

表 8-2 レビュー記録表から得られるプロダクトメトリクス

メトリクス	意味と算出式
1 レビュー流出バグ密度	レビューで見逃したバグ数 / 開発規模(KLOC)
2 レビュー指摘密度	レビューで指摘した総数 / 開発規模(KLOC)
3 バグ指摘率	レビューでバグを指摘した割合（バグ指摘数 / レビュー総指摘数）
4 指摘文の長さ	指摘文の長さの平均;レビュー記録表に書かれた指摘文の総文字数 / レビュー総指摘数
5 レビュー指摘内容	レビューで指摘した内容の分類；各分類の指摘数/レビュー総指摘数
(1) 述語の種類	レビューで指摘した文に含まれる述語の種類を以下の 6 種類の述語に分類する（重複可とする）
A) 間違いの指摘	修正する,訂正する,改修する,改善する,変更する,更新する,反映する,対応する,対処する,処置する
B) 不足・漏れの指摘	不足している,抜けている,漏れている,加える,追加する,追記する
C) 要確認の指摘	確認する，検討する，調整する
D) 削除の指摘	削除する
E) 明確化の指摘	明確にする
F) その他の指摘	上記の 5 種類の分類に当てはまらない場合
(2) 質問	レビューの指摘ではなく，質問の場合（アクションは必要ない）
(3) コメント	レビューの指摘ではなく，単なるコメントの場合（アクションは必要ない）

そこで，筆者は，レビュー記録表から得られる情報をもとにレビューの質を測る手法を検討した．筆者の組織では，レビュー記録表は標準化されており，図 8-2 のレビュー記録表テンプレートを使用する．このレビュー記録表から得られる情報を整理して，レビュー指摘内容の文章の述語に着目して分類した．なぜなら，指摘内容の文の主語や目的語は，固有名詞が多く，その開発に特化した技術用語が多く頻出する．しかし，レビュー指摘内容の文章の述語は，「～が間違っている」，「～処理が漏れている」，および「～を追加する」など，指摘の結果や指摘により期待する行動を示す汎用的な表現が多く含まれるため，分類が可能と考えたからである．

表 8-2 に，レビュー記録表から抽出することができるプロダクトメトリクスを示す．レビューの質を示す目的変数には，レビューで見逃した「レビュー流出バグ密度」を定義した．レビュー流出バグとは，レビュー以降の工程である設計，コーディング，テストおよび出荷後に摘出されたバグであり，本来ならば，当該レビューで摘出すべきだったバグを意味する．

説明変数は，「レビュー指摘密度」，「バグ指摘率」，「レビュー指摘内容の述語の種類」，「レビュー時の質問」，および「レビュー時のコメント」の 5 種類のメトリクスを使用する．さらに，「レビュー指摘内容の述語の種類」は，レビュー記録表の指摘内容をテキストマイニングより得られた述語を表 8-2 の 6 種類に分類した．この 6 種類の述語には，設計書に記述された内容の誤り（設計誤り）を指摘するメトリクスの「間違いの指摘」や，本来成果物に記述すべき内容の不足や漏れ（設計不足）を指摘するメトリクスの「不足・漏れの指摘」などがある．

8.4　レビュー記録表の質的要因の分析結果

表 8-3 は，表 8-2 のレビュー記録表メトリクスの測定結果である．7 つの開発プロジェクトのレビュー記録表を測定した．レビュー記録表に記載された総計 3,439 個の指摘を分析した．測定対象の開発規模は約 210KLOC（1,000LOC（コード行数））であった．「レビュー指摘内容の述語の種類」は，「不足・漏れの指摘（設計不足）」が平均値 0.315 と一番多く頻出していることが分かる．2 番目に多く頻出している述語の種類は，「間違いの指摘（設計誤り）」が平均値 0.266 である．この 2 つの述語の種類の頻出が，全体の約 6 割を占める．

表 8-3 レビュー記録表メトリクスの測定結果

	開発規模 KLOC	レビュー総指摘数	レビュー流出バグ密度	レビュー指摘密度	バグ指摘率	指摘文の長さ	間違いの指摘	不足・漏れの指摘	要確認の指摘	削除の指摘	明確化の指摘	その他の指摘	質問	コメント
A	6.06	137	0.825	22.607	0.350	83.503	0.569	0.386	0.073	0.007	0.131	0.000	0.065	0.175
B	23.36	284	0.941	12.157	0.235	69.225	0.331	0.415	0.133	0.035	0.098	0.007	0.056	0.112
C	76.34	1723	3.942	22.570	0.252	82.948	0.200	0.240	0.074	0.036	0.082	0.007	0.178	0.193
D	32.77	678	1.983	20.689	0.435	59.498	0.231	0.377	0.070	0.092	0.054	0.007	0.064	0.138
E	46.80	45	5.491	0.961	0.133	80.000	0.177	0.222	0.000	0.000	0.133	0.288	0.044	0.133
F	8.77	376	1.710	42.873	0.199	68.031	0.183	0.308	0.069	0.042	0.101	0.021	0.188	0.085
G	16.00	196	1.687	12.250	0.744	61.326	0.168	0.255	0.051	0.030	0.015	0.306	0.066	0.107
s	210.10	3439	-	-	-	-	-	-	-	-	-	-	-	-
m	30.01	491.28	2.368	19.158	0.335	72.076	0.266	0.315	0.067	0.034	0.088	0.088	0.095	0.135
sd	24.83	580.24	1.717	13.013	0.205	10.081	0.144	0.078	0.039	0.030	0.042	0.143	0.061	0.038

s:総和, m:平均値, sd:標準偏差

表 8-4 レビュー記録表から得られたメトリクスとレビュー流出バグ密度との相関検定

	レビュー流出バグ密度との相関係数	t 値	p 値	95% 信頼区間	
レビュー指摘密度	-0.435	-1.0792	0.3298	-0.895	0.473
バグ指摘率	-0.411	-1.0074	0.36	-0.889	0.496
指摘文の長さ	0.415	1.0202	0.3544	-0.492	0.89
間違いの指摘	-0.54	-1.4349	0.2108	-0.919	0.359
不足・漏れの指摘	-0.792	-2.8973	0.0339	-0.968	0.0956
要確認の指摘	-0.725	-2.3527	0.06534	-0.956	0.0621
削除の指摘	-0.289	-0.67534	0.5294	-0.856	0.593
明確化の指摘	0.24	0.55338	0.6038	-0.626	0.841
質問	0.068	0.1525	0.8848	-0.722	0.781
コメント	0.27	0.62817	0.5575	-0.606	0.85

レビュー流出バグとの相関係数	P値
-0.435	0.3298
-0.411	0.36
0.415	0.3544
-0.54	0.2108
-0.792*	0.0339
-0.725	0.0653
-0.289	0.5294
0.24	0.6038
0.068	0.8848
0.27	0.5575

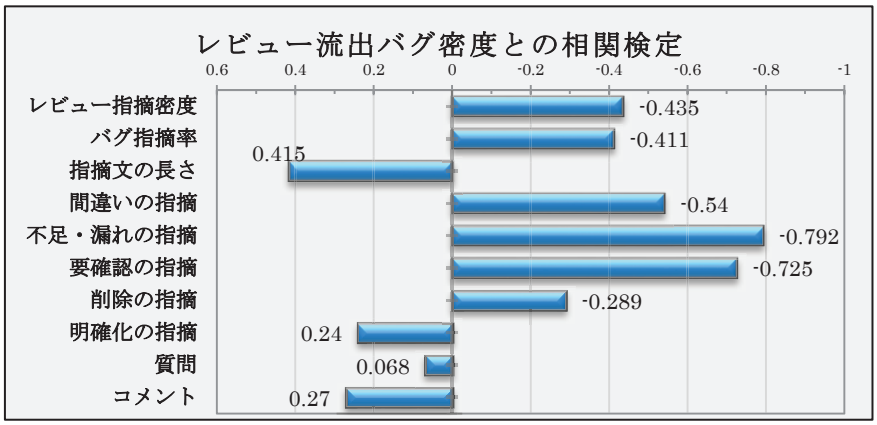


図 8-3 レビュー記録表から得られたメトリクスとレビュー流出バグ密度との相関検定

表 8-4 と図 8-3 は，目的変数のレビュー流出バグ密度と説明変数との相関分析の結果を示す．正の相関がある場合には，流出バグが多くなることを意味するので，レビューの質が低いと推測できる．逆に負の相関がある場合には，流出バグが少なくなることを意味するので，レビューの質が高いと推測できる．検定の結果は，レビューの質が高いとみられる負の相関が強い説明変数は，“不足・漏れの指摘（設計不足）”であり，その相関係数は -0.792 と 5%有意で有意差ありと判断された．設計の漏れや設計の不足など，仕様書に書かれていない内容を指摘することは，レビューの質が高いと考えられる．その他には，有意差のみられるような強い相関のある因子はなかった．逆に，レビュー流出バグ密度と正の相関があり，レビューの質が低いと推測できる因子は，レビューの指摘文の長さ(相関係数 0.415)である．レビューの指摘が長くなると，指摘内容を読み取るポイントが曖昧になることにより，レビューの質が低くなると推測できる．レビュー指摘密度やバグ指摘率は負の相関の傾向があるので，レビューの質が良いことを表すメトリクスになると考えられる．

この結果から，レビュー流出バグを少なくするような質の高いレビューや設計品質に良い影響を与える因子は，レビューにおいて設計不足や設計漏れを多く指摘している場合であると言える．

レビュー指摘密度やバグ指摘率も有意差が認められなかったが，メトリクスとしては使えそうである．逆に，流出バグが多くなるような，レビューの質が低く，設計品質に悪い影響を与える因子は，レビューの指摘文章が長い場合と言える．

8.5 まとめ

本章では，上流工程の有効な品質管理技術として，設計レビューを取り上げ，従来とは異なる新たなプロダクトメトリクスに関する研究結果とその有効性を論じた．レビュー記録票に記載された指摘文章などの情報から，設計品質に影響を与えるレビューのプロダクトメトリクスの有効性を評価した．その結果，設計品質に影響を与えるレビューに関する要因は，レビュー記録票に記載されているレビュー指摘の中で，設計仕様書に記述された内容の“設計不足”や“設計漏れ”を指摘しているレビューであることが判明した．これは，レビュー対象となる設計仕様書に書かれている内容の間違いを指摘することよりも，仕様書に書かれていない設計の考慮漏れや処理不足等を指摘することのほうが，より高度で効果的なレビューが実施されていることを示し，設計品質を向上させる要因であると考えられる．

第 9 章　高品質ソフトウェア開発のための定量的品質マネジメントの成功要因

前章までの論述を踏まえて，高品質ソフトウェア開発のための定量的品質マネジメントの成功要因を考察する．

定量的品質マネジメントは，大きく 2 つのプロセスに大別される．「測る」プロセスと，測った結果を科学的に「判断する」プロセスである．したがって，定量的品質マネジメント技術を高度化することは，「測る」技術の高度化と，測った結果を科学的に「判断する」ことの高度化であると考えてきた．以下に「測る」こと，および科学的に「判断する」ことについて，定量的品質マネジメントにおける意味とともに，それらを高度化することについて論じる．

まず，「測る」ことの意味について考える．ソフトウェア開発組織の定量的品質管理においてメトリクスを「測る」こと，特にプロセス成熟度が低い段階においては，プロセスメトリクスを「測る」ことにより，定義された開発プロセスを安定的に繰り返し実行することができるようになる．開発プロジェクトが定義された開発プロセスを遵守するということは，開発完了までの決められたプロセスにおいて何をやるべきか，その作業結果として何を報告すべきかを開発プロジェクトのメンバ全員が理解し，実践することである．これにより，開発プロジェクトは，ソフトウェア開発を個人の力量だけに頼らず，チームとしての力を発揮できる．すなわち，やるべき作業を個人の判断で勝手に省略したり，無意識ながらもやるべき作業を漏らすとか，不十分な状態で作業を完了してしまうなど，定められたプロセスの作業の欠損を，プロセスに関するメトリクスを測ることにより抑制することができる．また，定義された開発プロセスを遵守できているかどうかを「測る」ことにより，QCD (Q:Quality, C:Cost, D:Delivery) を見える化できるので，開発者全員が QCD の状態を客観的に把握して，プロジェクトを成功に導く意識がより強く働くことになる．

次に，科学的に「判断する」ことの意味について考える．科学的に「判断する」とは，測った結果をもとに品質を向上するための追加の方策などの新たなアクションが必要になるかを科学的に「判断する」ことである．「測る」ことにより，定義された開発プロセスの遵守状況が定量的データとして見える化されると，組織やプロジェクトは目標とするメトリクスを達成しようとする管理を実施する．測った結果が目標値を下回っていれば，目標値を達成するために，新たなアクションを実行することになる．「測る」ことの意味が「開発プロセスを守る」こととすれば，科学的に「判断する」ことの意味は，測った結果を判断して「開発プロセスを攻める（新たなアクションを起こす)」ことと表現できる．

第 2 章および第 3 章で論じたように，定量的品質マネジメントを確立することにより品質が向上することは明らかであった．「測る」ことにより，定義された開発プロセスを遵守することに効果を発揮していると考えられる．例えば，設計レビューのプロセスを遵守することにより，上流工程において開発者個人のスキルに依存する成果物の完成度を，チームの力で向上することができるようになる．しかし，このことは本来やるべきことをやっている

に過ぎないともいえるが，プロセス成熟度が低い組織や，近年のように複雑化・大規模化するソフトウェア開発の状況において，やるべきことを当たり前のようにやることが難しくなってきている現状では，定量的品質マネジメントの徹底により，やるべきことをきちんとやれる状態にすることは非常に重要である．第 2 章に示した定量的品質マネジメントの必要性を説く多くの文献も，「開発プロセスを守る」というステップまでのことを論じている．しかし，このステップから品質をさらにもう一段向上するためには，やるべきことを当たり前のようにやる，すなわち，定義された開発プロセスを守るだけでは，不十分であり，以下のような課題が出てくる．

- 開発プロセスはある程度成熟する．特に，レビュー工数の確保は，ある程度までは管理することが可能であり，品質向上に寄与する．しかし，レビュー工数を限りなく増加させることが品質を限りなく完璧（バグがゼロの状態）に近づけられるものでもない．
- 単一メトリクスでは見えないプロセス上の漏れが存在する可能性が高い．
- 定量的品質マネジメントの運用が，メトリクスの基準値を達成したかどうかに重点を置きすぎると，各工程の移行判定という側面で，メトリクスを個別に単純に判断するという作業に陥りやすい．
- 社会基盤を支えるようなミッションクリティカルな用途には，1 件のバグでも大きな社会的影響を与えかねない．
- 一方で，開発自体が複雑化・大規模化するなかで，短納期化や部品の流用化などで品質が見えにくくなっている．

そこで，測った結果より新たなアクションを実施するべきかを科学的に「判断する」ことが必要になる．「開発プロセスを守る」ことから「開発プロセスを攻める」状態にレベルを上げることである．「開発プロセスを攻める」とは，新たなアクションを起こして，潜在する品質問題を検出することにより，今よりもさらに品質を向上させることである．開発現場においてはこの行動は非常にハードルが高い．追加工数を確実に要することから，開発現場では受け入れ難いからである．したがって，この科学的に「判断する」ことによる品質の弱点の指摘がより効果的で確からしいものである必要がある．これを実現するための具体的な方策の提案が，第 4 章で述べたソフトウェア品質マップである．複数のメトリクスや各工程の断片における見解などを総合的に分析して，プロジェクトの品質上の新たな弱点を見つけ出し，品質向上の方策（新たなアクション）を実施する．また，総合的に判断するための過去の知見を条件適合判定表に形式知化することで，分析技術を高める．これが，定量的品質マネジメント技術の高度化を考える一要素である科学的に「判断する」ことの高度化である．

次に，「測る」ことの高度化については，品質マップで使用している条件適合判定表がプロセスメトリクスの組み合わせに関する知見に偏っているという課題があり，より適切な分析を実施するためには，メトリクスを改善する必要がある．特に，組織の開発プロセスの成熟につれてプロセスメトリクスの差が見られなくなる傾向や，プロダクトメトリクスについての知見がまだ少ないことから，有効なプロダクトメトリクスへの改善が必要である．この改善が，「測る」ことの高度化につながる方策である．

有効なプロダクトメトリクスへの改善について，第 5 章から第 8 章において，4 つの実践研究を取り上げた．ソースコードの複雑性のメトリクスについては，ソースのネスティングの深さと 1 モジュールの開発規模の大きさが品質に影響を与える要素であることが 3 つの実践研究で判明した．この結果を定量的管理や品質マップの改善に反映することで，より高度な定量的品質マネジメントに進化させることができると考えられる．また，第 8 章では，新たなプロダクトメトリクスを探究し，設計レビュー記録票から得られる情報から，設計仕様書の記述の漏れや不足を指摘しているほうが品質に良い影響を与えることが判明した．これまで，プロダクトメトリクスはソースコードの解析ツールから得られる情報を主に使用していたので，コーディング工程よりも上流にあたる設計工程におけるプロダクトメトリクスを設定して品質マップに反映することで，より高度な定量的品質マネジメントに進化させることができると考えられる．

以上のように，定量的品質マネジメント技術の高度化には，従来の定量的管理をさらに進化させたソフトウェア品質マップによる総合的な品質分析手法により科学的に「判断する」ことの高度化がまず必要である．つぎに，品質マップの重要な構成要素であるプロダクトメトリクスの最大ネスティング数を加えた品質分析のための知見の重視や，レビュー記録票から得られるメトリクスの導入などの改善による「測る」ことの高度化が，高品質ソフトウェア開発を実現するための定量的品質マネジメント技術の高度化を図るための成功要因である．

本研究では，レビュー工数による品質確保の方策がある程度の量を超えると有効性が薄れる，という課題に対する定量的品質マネジメントの高度化の方向性を示してきた．品質マネジメントを高度化する方法の特徴は，レビュー工数を安定的に確保するなどの，やるべき当たり前のプロセスを確実に運用するなどのプロセス成熟度を向上させる方法のようなシンプルで明快な方策とは異なり，複雑にからみあった開発途中のさまざまな事情を想定した，より繊細で緻密な分析が必要になると考えている．品質マップによる品質の弱点分析のように，その場面において的確に分析を行うことができるかが，潜在バグを摘出できるかどうかの見極めになり，現実的には工数との兼ね合いから幅広く適用するには難しい施策である．したがって，確率論に通じる不確定要素を探る方法について継続して研究する必要がある．

一方，従来の定量的品質マネジメントにおいて，レビューが重要であるという考え方に変わりはない．品質向上においては，レビューを確実かつ十分に実施するプロセスが基本である．その基本を無視して別の方法で品質を高めることは，これまでの経験から無理と考える．したがって，本研究では，レビューなどの基本的なプロセスを確立したうえで，さらに品質を高めるための方策を論じてきた．

第 10 章　結論

本研究では，ソフトウェアの品質を確保するための基本的な開発プロセスとなる定量的品質マネジメントの領域において，さらなる高品質ソフトウェア開発を実現するために，定量的品質マネジメント技術を高度化する仕組みの提案と実践，およびその効果と意義を論じた．

第 2 章では，主にソフトウェア開発組織の実態調査から定量的品質マネジメントの有効性を論じた．

はじめに，組織におけるソフトウェア開発では，個人のスキルだけに頼るようなソフトウェア開発が品質面において危険であることや，組織におけるソフトウェア開発を成功させるための方策として，定量的品質マネジメントが重要であることを論じた．

次に，ソフトウェア開発組織の実態調査として，独立行政法人情報処理推進機構(SEC)の調査データから日本のソフトウェア開発組織における定量的品質マネジメントの現状について論じた．具体的には，定量的管理の実施程度が高い方が，相対的に信頼性が高い傾向が見られ，プロジェクト成功率が約 2 倍高いという結果から，定量的品質マネジメントが品質向上に有効であることを確認した．また，定量的品質マネジメントのメトリクスと出荷後の製品品質の関係は，レビュー工数密度が高い，上流工程でのバグ摘出比率が高い，テスト摘出バグ密度が低い，およびテスト検出能率が低いほうが，相対的に出荷後のバグ密度が低いという結果であった．

さらに，この結果より，高品質ソフトウェア開発を実現するための定量的品質マネジメントはどうあるべきかについて論じた．

第 3 章では，プロセス成熟度の高い組織における定量的品質マネジメントの実際について，筆者の組織の開発プロセスの改善の実践研究を取り上げた．高度化された開発環境であるソフトウェアファクトリを組織内で統一して活用することにより，プロセスメトリクスやプロダクトメトリクスのデータ測定の集約および診断結果をほぼ毎日実施することができ，短いサイクルでの定量的品質管理のフィードバックを実施することができている．その結果，品質に大きく影響を与える上流工程におけるレビュー工数が増加するとともに，レビュー摘出バグ密度も増加し，出荷後のバグ数は減少した．また，プロダクトメトリクスにも改善がみられ，組織内で品質が悪かった 2 製品についても，メトリクスが改善され，出荷後品質も飛躍的に向上した．

しかし，定量的品質マネジメントによる品質改善の効果は得られたものの，企業の基幹システムや社会インフラシステムでは常に高い品質が要求されるために，出荷後の製品品質のさらなる改善が期待される．高品質ソフトウェア開発を実現するために，さらなる定量的品質マネジメントの高度化を目指し，定量的品質マネジメントの「測る」ことと，測った結果から科学的に「判断する」ことの両面でさらに高度化することが必要である．これについ

て，第 4 章では，科学的に「判断する」ことの高度化の手法について，第 5 章から第 8 章では「測る」ことの高度化について，それぞれの実践研究を取り上げることとした．

第 4 章では，定量的品質マネジメント技術の高度化のひとつとして，品質分析手法の改善による科学的に「判断する」ことの高度化について，ソフトウェア品質マップの実践研究を取り上げた．従来の定量的品質マネジメントでは，バグ修正等の後戻り工数を最小限に抑えるために，可能な限り作り込み品質を自工程完結する考え方であった．しかし，自工程内の完結から漏れてしまうような問題さえも出荷前に見逃さないための方策として，品質マップを考案した．品質マップは，開発工程の全体を見直す機会を設けることにより，メトリクスや定性的な品質見解などの相互の因果関係や矛盾点などを，過去の経験を知見化した条件適合判定表を用いて，総合的に品質を分析して弱点をあぶりだすという，定量的品質マネジメント技術の科学的に「判断する」ことの高度化を実践したものである．

本実践研究の結果，品質マップを適用すると出荷後バグ密度が減少し，適用前に比べて約 4 倍品質が向上した．一方，生産性は品質マップを適用すると約 30%低下した．品質マップを適用すると，開発終盤において品質分析と品質向上の対策を実施するため，工数が増加し生産性が低下した．

品質マップの課題は，品質マップで使用する条件適合判定表の内容の充実であり，特に，プロダクトメトリクスの改良が必要である．これは，定量的品質マネジメントのもう一つの高度化技術と考えているメトリクスの改善による「測る」ことの高度化に相当する．この課題の方策について，第 5 章から第 8 章では，それぞれの実践研究を取り上げた．

第 5 章では，ソースコードの複雑性を測るプロダクトメトリクスの中で，品質に影響を与えるメトリクスを抽出した．

はじめに，ソースコードの複雑性を示すプロダクトメトリクスとソースコードの品質を確保する労力を示すプロセスメトリクスとの相関関係を検証した．しかし，両者に強い相関を示すメトリクスは見つからなかった．したがって，これらのソースコードの品質を確保する労力の大きさを示すプロセスメトリクスの値が大きくなるのは，ここで選択したソースコードの複雑性を示すプロダクトメトリクス以外の要因が考えられるようである．

次に，ソースコードの複雑性を示すプロダクトメトリクスの中で，出荷後の製品品質に影響を与えるメトリクスを分析した．その結果，最大ネスティング数が出荷後の製品品質に最も影響を与えていることが判明した．具体的には，最大ネスティング平均値が，2 を閾値として，2 を超えると，出荷後の品質が低くなる傾向があった．ただし，最大ネスティング数の出荷後の品質が高い群と低い群には有意差はみられなかったため，研究の継続が必要と考える．

第 6 章では，第 5 章とは別の組織の製品におけるソースコードの複雑性を示すプロダクトメトリクスと出荷後の製品品質との相関関係を分析した．その結果，有効行数と最大ネスティング数のメトリクスの値が大きいほど出荷後の品質が低い（本実践研究では出荷後のバグによりソースモジュールの修正がある）傾向があった．モジュールの有効行数が

150LOC（コード行数），最大ネスティング数が 4 を閾値として，有効行数および最大ネスティング数が閾値を超えるソースコードに対しては，追加のレビューやリファクタリング等を実施することによりメトリクスの値を低減することが，ソフトウェア品質を確保するためには有効であることを確認した．

第 7 章では，ソースコードの複雑性を示すプロダクトメトリクスと製品品質との相関関係を分析した．しかし，第 5 章および第 6 章では，出荷後の製品品質との関係を分析したが，本章では，製品品質として UT 完了時点の品質リスクの関係性の評価を実施した．この結果，単純ではあるが開発規模が最も有用なメトリクスであることが判明した．さらに，規模の大小に応じたリスク評価を実施したあと，それぞれに必要な方策を導出した．この規模の基準値を活用すれば，UT 以降にリスクを含んだモジュールやファイルをある程度特定でき，それらに必要な方策を実施することが可能となることを確認した．

第 8 章では，上流工程での有効な品質管理技術として，設計レビューを取り上げ，従来とは異なる新たなプロダクトメトリクスに関する研究結果とその有効性を論じた．レビュー記録票に記載されたテキスト等のさまざまな情報から，設計品質に影響を与えるレビューに関するプロダクトメトリクスの有効性を評価した．その結果，設計品質に影響を与えるレビューに関する要因は，レビュー記録票に記載されているレビュー指摘の中で，設計仕様書に記述された内容の“設計不足”や“設計漏れ”を指摘しているレビューであることが判明した．これは，レビュー対象となる設計仕様書に書かれている内容の間違いを指摘することよりも，設計仕様書に書かれていない内容，すなわち考慮漏れや処理不足などを指摘することのほうが，より効果的なレビューであり，設計品質を向上させる要因であると考えられる．

第 9 章では，第 4 章から第 8 章までの実践研究や検証結果を踏まえて，高品質ソフトウェア開発のための定量的品質マネジメント技術の成功要因について考察した．定量的品質マネジメント技術の高度化には，従来の定量的管理をさらに進化させたソフトウェア品質マップによる総合的な品質分析手法により科学的に「判断する」ことの高度化と，品質マップの重要な構成要素であるプロダクトメトリクスの最大ネスティング数を加えた品質分析のための知見の重視やレビュー記録票から得られるメトリクスの導入などの改善による「測る」ことの高度化が，高品質ソフトウェア開発を実現するための定量的品質マネジメント技術の高度化を図るための成功要因であると論じた．

最後に，今後の研究を続けるうえでの課題について論じる．

本研究における課題は，品質向上と生産性向上との関係性の解明である．本研究では，一貫してソフトウェア品質の向上のための改善について論じてきたが，開発生産性については言及していない．開発現場では短納期や低コストが常に求められ，品質と生産性の両立の研究が求められている．筆者の基本的な考えは，「品質を良くすれば生産性は後からついてくる」であり，良い品質のソフトウェア開発をすれば，開発中や出荷後の品質問題による障害の調査や修正などの後戻りコストが下がることから，開発時の生産性だけでなく，出荷後

の障害対応による後戻り工数を加味した総合的な生産性は上がるという考えである．しかし，この考えについては，まだ定量的に検証できていない．また，「顧客満足を最優先し，価値のあるソフトウェアを早く継続的に提供」[10-1]することを目指すアジャイル開発が開発現場にも本格的に普及の兆しがあり，アジャイル開発における定量的品質マネジメントや品質確保の手段をどのように考えるかについても，生産性の課題とともに研究が未成熟な領域であり，今後の研究課題と考える．

独立行政法人情報処理推進機構(IPA)の調べでは，2004 年から 2012 年までの生産性の中央値の経年変化に生産性向上または生産性低下の傾向はみられない．また，近年の開発プロジェクトの複雑さや難しさが増している一方で，開発要員のスキルレベルが追いついていないという側面もある．このような状況下では，開発総量が増大するに伴い，必要となる開発要員も増大するため，開発要員の平均スキルレベルが低下する可能性が大きくなる．また，多くの開発プロジェクトが改良型開発になってきており，新規開発による開発要員がスキルアップする機会が減少していると考えられる．したがって，如何にして開発要員のスキルレベルを維持し向上させるか，また生産性が向上する開発方法を実現するかが，IT ベンダーの大きな課題となりつつある[2-11]．

ソフトウェア品質を第一に考えることが，ゆるぎない開発理念であることには違いないが，これらの時代の変化に追随しながら，さらなる品質や生産性の向上が求められるため，高品質ソフトウェア開発について継続して実践的な研究を進めていく所存である．

参考文献

[1-1] 三縄俊信，加藤均，“重要インフラ等システム障害対策”，SEC journal, Vol.12, No.1, pp.19-23, 2016.

[1-2] T.DeMarco（著），渡辺純一（訳），「品質と生産性を重視したソフトウェア開発プロジェクト技法―見積り・設計・テストの効果的な構造化」，近代科学社，1987.

[1-3] 水野幸男監修，「ソフトウェアの総合的品質管理」,日科技連出版社，1990.

[1-4] 保田勝通，「ソフトウェア品質保証の考え方と実際」，日科技連出版社，1995.

[1-5] 山田茂，福島利彦，「品質指向ソフトウェアマネジメント-高品質ソフトウェア開発のためのプロジェクトマネジメント-」，森北出版，2007.

[1-6] 山田茂，「ソフトウェア信頼性の基礎-モデリングアプローチ-」，共立出版，2011.

[1-7] 山田茂，田村慶信，「ソフトウェア工学の基礎と応用-高品質ソフトウェア開発を目指して-」，数理工学社，2013.

[1-8] SQuBOK 策定部会，「ソフトウェア品質知識体系ガイド -SQuBOK Guide-(第 2 版)」，オーム社, 2015.

[1-9] Project Management Institute, Inc.，「プロジェクトマネジメント知識体系ガイド PMBOK®ガイド）第 5 版」, PMI 日本支部, 2015.

[2-1] V.R.Basili and H.D.Rombach, “The TAME project: Towards improvement-oriented software environment”, *IEEE Transactions on Software Engineering*, Vol.14, No.6, pp.758-773, 1988.

[2-2] ISO9000:2005，「Quality Management Systems –Fundamentals and Vocabulary (JIS Q 9000:2005, 品質マネジメントシステム –基本及び用語)」,日本規格協会, 2015.

[2-3] S. H. Kan, *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 2002.

[2-4] 中島毅, 東基衛, “ソフトウェア開発における品質プロセスのコスト最適化のためのモデルとシミュレーションツール”，電子情報通信学会論文誌，Vol.J91-D，No.5, pp.1216-1230, 2008.

[2-5] V.Basili, “Using Measurement to Build Core Competencies in Software”, *Seminar sponsored by Data and Analysis Center for Software*,pp.646-661,2005.

[2-6] 独立行政法人情報処理推進機構(IPA)技術本部ソフトウェア高信頼化センター(SEC)，「統計指標に基づく品質マネジメント実践集～ベンチマーキングによる信頼性向上のヒントと具体例～」，2016.

[2-7] 独立行政法人情報処理推進機構(IPA)技術本部ソフトウェア高信頼化センター(SEC)，「ソフトウェア開発データ白書 2014-2015」, 2014.

[2-8] 独立行政法人情報処理推進機構(IPA)技術本部ソフトウェア高信頼化センター(SEC),

「組込みソフトウェア開発データ白書 2015」, 2015.
[2-9] 独立行政法人情報処理推進機構(IPA)技術本部ソフトウェア高信頼化センター(SEC), 「統計指標に基づく品質マネジメントの実践集　トピックス概要」, 2016.
[2-10] 独立行政法人情報処理推進機構(IPA)技術本部ソフトウェア高信頼化センター(SEC), 「横断的アプローチによるソフトウェア開発データの分析～高信頼性定量化部会信頼性メトリクス WG 検討報告書～」, 2015.
[2-11] 独立行政法人情報処理推進機構(IPA)技術本部ソフトウェア高信頼化センター(SEC), 「ソフトウェア開発データが語るメッセージ 2015～プロジェクトや組織のマネジメントの指針と信頼性・生産性の傾向～」, 2015.
[2-12] 日経 BP 社, “プロジェクト実態調査 800 社”, 日経コンピュータ,pp.38～42,2008.

[3-1] 水野幸男監修,「ソフトウェアの総合的品質管理」,日科技連出版社,1990.
[3-2] 誉田直美,「ソフトウェア品質会計～ＮＥＣの高品質ソフトウェア開発を支える品質保証技術～」,日科技連出版社,2010.
[3-3] CMU Software Engineering Institute, *CMMI for Development, Version1.3 (CMMI-DEV, V1.3)*, CMU/SEI, 2010.
[3-4] N.Honda,“Beyond CMMI level5 – Comparative analysis of two CMMI level5 organizations -”, *Software Quality Professional*, Vol.11, No.4, pp.4-12, 2009.
[3-5] 誉田直美, 山田茂, “高品質ソフトウェア開発を実現する品質会計技法”,プロジェクトマネジメント学会誌,Vol.13,No.5, pp.9-14,2011.

[4-1] 独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター,「ソフトウェアテスト見積りガイドブック」,オーム社,pp.116-117,2008.
[4-2] 野中誠, 小池利和, 小室睦,「データ指向のソフトウェア品質マネジメント～メトリクス分析による「事実にもとづく管理」の実践～」,日科技連出版社,pp.158-159,2012.
[4-3] T.Shimomura, “Risk evaluation after unit testing using software metrics”, *Proceedings of the 7th International Conference on Project Management*, pp.1-8, 2013.
[4-4] 佐々木眞一,「自工程完結-品質は工程で造りこむ-」, 日本規格協会, 2014.

[5-1] T.J.McCabe, “A complexity measure*”*, *IEEE Transactions on Software Engineering*, Vol.SE-2, No.4, pp.308-320,1976.
[5-2] A.H.Watson and T.J.McCabe, “Structured testing: A testing methodology using the cyclomatic complexity metric”, *NIST Special Publicatio*n, 500(235),pp.1-123,1996.
[5-3] M.Lorenz, *Object-Oriented Software Development: A Practical Guide*, Englewood Cliffs, 1993.

[5-4] 倉下亮, 吉村博昭, 野中誠, 誉田直美, “CK メトリクスの分布に基づくソフトウェア設計の質の定量的評価”, ソフトウェア品質シンポジウム 2011 報文集, 2011.
[5-5] 纐纈伸子, 川村真弥, 野村准一, 野中誠, “プロセスメトリクス利用による Fault-Prone クラス予測精度の向上”, ソフトウェア品質シンポジウム 2010 報文集, 2010

[7-1] T.Muller, D.Friedenberg, and the ISTQB WG Foundation Level, *Certified Tester Foundation Level Syllabus*, International Software Testing Qualifications Board (ISTQB), version2011, pp.40-42, 2011

[8-1] T. Gilb and D. Graham, *Software Inspection*, Addison-Wesley,1993.
[8-2] P. M. Johnson, “Reengineering inspection”, *Communications of the ACM*, Vol.41, No.2, pp.49-52, 1998.
[8-3] R. L. Glass, “Inspection – some surprising findings”, C*ommunications of the ACM*, Vol.42, No.4, pp.17-19, 1999.
[8-4] 野中誠, “設計・ソースコードを対象とした個人レビュー手法の比較実験”, 情報処理学会研究報告 SE-140-4, Vol.2004, No.118,pp.25-31, 2004.
[8-5] 森崎修司, “ソフトウェアインスペクションの動向”, 情報処理学会誌, Vol.50,No.5, pp.377-380, 2009.
[8-6] 細川宣啓, “第三者インスペクションによる品質検査と欠陥予測”, 情報処理学会誌, Vol.50, No.5, pp.405-411, 2009.
[8-7] 小室睦他, “開発現場の実態に基づいたピアレビュー手法の改善と改善効果の定量的分析”, SEC journal, Vol.1, No.4, pp.6-15, 2005.
[8-8] 中野裕也, 水野修, 菊野亨, 阿南佳之, 田中又治, “コードレビューの密度と効率がコード品質に与える影響”, SEC journal, Vol.2, No.4, pp.10-17, 2006.
[8-9] 安達賢二, “レビュープロセスの現実的な改善手段の提案”, ソフトウェアテストシンポジウム, 2006.
[8-10] 久野倫義, 丹羽友光, 前川隆昭, “デザインレビューの効果的実施及び評価方法”, 三菱電機技報, Vol.83, No.5, pp.18, 2009.
[8-11] 野中誠, “ソフトウェアインスペクションの効果と効率”, 情報処理学会誌, Vol.50, No.5, pp.385-390, 2009.
[8-12] B.Robbins, “Cognitive factors in perspective-based reading (PBR): A protocol analysis study”, *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement*,pp.145-155, 2009.
[8-13] 久野倫義, 中島毅, 松下誠, 井上克郎, “ピアレビュー会議の改善と品質改善の効果”, SEC Journal Vol.10, No.1, pp.16-23, 2014
[8-14] 江崎和博, 山田茂, 高橋宗雄, “設計レビューにおけるソフトウェア信頼性に影響を

及ぼす人的要因の品質工学的解析”，電子情報通信学会論文誌，Vol.J84-A，No.2，pp.218-228，2001.
[8-15] 山田茂，冨高功介，“ソフトウェア信頼性に影響を及ぼす品質工学的アプローチに基づく人的要因分析と信頼性予測”，日本信頼性学会誌，Vol.27，No.7，pp.439-448，2005.
[8-16] 梶原誠，中條武志，“デザインレビューの成熟度評価方法の提案”，日本品質管理学会誌，Vol.43，No.2，pp.75-84，2013.
[8-17] 林章浩，片岡信弘，木野泰伸，津田和彦，“レビュープロセスの継続的改善方式”，日本品質管理学会誌，Vol.40，No.3，pp.71-80，2010.

[10-1] “アジャイルソフトウェアの 12 の原則”http://agilemanifesto.org/iso/ja/principles.html

謝辞

本研究を遂行するにあたり，多大なご指導・ご鞭撻をいただきました鳥取大学大学院工学研究科山田茂教授，北村章教授，小柳淳二准教授ならびに山口大学大学院創成科学研究科田村慶信准教授に深く感謝いたします．特に主指導教員の山田教授には，鳥取大学大学院工学研究科博士後期課程に在学中に，懇切なご指導と格別なご配慮，温かい励ましを賜り，心から感謝しております．

また，本研究を進めるにあたり，さまざまな面でご支援をいただきました井上真二助教授ならびに山田研究室の方々に深く感謝申し上げます．

本研究は，筆者が日本電気株式会社入社から現在に至る品質保証部門における業務が基礎になっています．ソフトウェア開発の現場において品質向上を推進する機会に恵まれたことを深く感謝するとともに，著者と一緒にソフトウェア品質向上とプロセス改善に取り組んだ職場の先輩，同僚，関係者各位にも改めて感謝申し上げます．

特に，入社以来，業務において常にご指導と励ましをいただき，さらには本研究のきっかけとなる山田茂先生にご指導いただくことを推薦いただきました日本電気株式会社誉田直美主席品質保証主幹に心より御礼申し上げます．

最後に，本研究を進めるにあたり，日頃から心の支えとなり励ましてくれた，妻和枝，ならびに息子勇作，娘玲美に感謝します．

研究業績一覧

主論文

1. T.Sato, and S.Yamada, “Application of Software Factory to Advanced Software Quality Management”, *Asia-Pacific Journal of Industrial Management*, Vol. VI, Issue 1, March 2017, 掲載決定済.
2. K.Ohno, T.Sato, and S.Kawamura, “Improving the Acceptance Inspection Process in Offshore Software Development Projects”, *Proceedings of the 10th International Conference on Project Management (ProMAC2016)*, pp.978-984, November 2016.
3. 佐藤　孝司, 山田　茂, “品質マップによるソフトウェア分析手法の提案”, プロジェクトマネジメント学会誌, Vol.18, No.5, pp.35-40, 2016 年 10 月.
4. 小角　能史, 佐藤　孝司, “アジャイルプロジェクトにおけるペアワーク適用の改善事例” , SQiP ソフトウェア品質シンポジウム 2016 発表報文集, CD-R, 6pp., 2016 年 9 月.
5. T. Sato and S. Yamada, “Analysis of Process Factors Affecting Software Quality based on Design Review Record and Product Metrics”, *International Journal of Reliability Quality and Safety Engineering*, Vol.23, No.4, pp.1650011.1-1650011.11, July 2016.
6. T.Sato and S.Yamada, “Software Quality Analysis Practice and Verification based on QUALITY MAP System”, *Proceedings of the 9th International Conference on Project Management (ProMAC2015)*, pp.380-386, October 2015.
7. 山崎　健司, 新海　昭一, 古村　仁志, 佐藤　孝司, “プロダクトメトリクスと品質の関係分析”, SQiP ソフトウェア品質シンポジウム 2015 発表報文集, CD-R, 8pp., 2015 年 9 月.
8. T.Sato, “Quality Improvement using the QUALITY MAP Technique”, *Proceedings of the 6th World Congress for Software Quality(6WCSQ)*, CD-R, 12pp., July 2013.

参考論文

1. T.Sato and S.Yamada, “Qualitative Analysis of Process Factors Affecting Software Quality Based on Design Review Record”, *Proceedings of the 9th Japan-Korea Software Management Symposium*, pp.36-41, November 2016.
2. T.Sato and S.Yamada, “Software Quality Management Based on Process and Product Metrics Analysis with Software Factory”, *Proceedings of the 13th International Conference on Industrial Management (ICIM2016)*, pp.332-339, September 2016.
3. 佐藤　孝司, 山田　茂, “ソフトウェア信頼性に影響を及ぼす設計レビュー記録の質的要因分析”, プロジェクトマネジメント学会 2016 年秋季研究発表大会予稿集, pp.189-194, 2016 年 9 月.
4. T.Sato and S.Yamada, “Qualitative Quality Analysis of Process Factors Based on Software Design Review Record Affecting Software Reliability”, *Proceedings of the 22nd ISSAT International Conference on Reliability and Quality in Design (ISSAT/RQD2016)*, pp.117-121, August 2016.
5. T.Sato, “Quality Improvement Practice and Software Quality Analysis based on the QUALITY MAP System”, *Proceedings of the 8th Japan-Korea Software Management Symposium -Trends in Management Technologies and Human Resource Development-*, pp.7-20, November 2015.
6. 佐藤　孝司, 山田　茂, “品質マップによるソフトウェア品質分析手法の提案”, プロジェクトマネジメント学会 2015 年秋季研究発表大会予稿集, pp.487-492, 2015 年 10 月.
7. 佐藤　孝司, “NEC におけるソフトウェア品質向上への取り組み”, 日本科学技術連盟クオリティフォーラム 2014 報文集, pp.76-81, 2014 年 11 月.
8. T.Shimomura, T.Mori, M.Nonaka, and T.Sato, “Software Quality Risk Evaluation on Unit Testing Product Metrics”, *Proceedings of the 7th International Conference on Project Management (ProMAC2013)*, CD-R, 8pp., November 2013.
9. 下村　哲司, 森　岳志, 野中　誠, 佐藤　孝司, “ソフトウェアメトリクスを用いた単体テストの品質リスク評価”, SQiP ソフトウェア品質シンポジウム 2013 発表報文集, CD-R, 8pp., 2013 年 9 月.