コード行数を用いない品質分析技術と開発速度を落とさない品質管理手法の提案

A novel quickly methodology for software quality control not depending on LoC

株式会社ＮＴＴデータ　技術革新統括本部
○熊谷 尚俊 1)　　熊川 一平 2)
○Hisatoshi Kumagai1)　　Ippei Kumagawa2)

Abstract We basically use several densities by line of code for software quality control such as the number of bugs per LoC, or the number of test cases per LoC. While at the same time, the number of package software or SaaS developments are increasing, thus cannot apply the above current densities. Recently system developments without coding are more & more required in addition to speed. The current ordinary methodology, using several densities, makes our developments slow. Instead we propose a novel quickly methodology for software quality control not depending on LoC.

1. はじめに

ソフトウェア開発の定量的な品質管理では，プログラムコード行数あたりのバグ摘出数・テスト数といった指標を使うことが多い[1]．昨今は業務パッケージソフトウェアの利用などでプログラムコードを直接書かないプロジェクトが増え，プログラムコード行数を用いた指標を利用することが難しくなってきている．また上限・下限値といった水準値を定めて指標値を評価するが，統計的に意味のある類似したプロジェクトの情報の入手が前提となっている．昨今は技術の多様化が進んでいるため類似したプロジェクトを見つけるのが厳しくなっており，この点からもプログラムコード行数を用いた指標を利用することが難しい．そこで本論文ではテストプロセスを対象とした，プログラムコード行数を用いない品質分析技術を提案する．

また現代のビジネスにとってソフトウェアの重要性は増しており，その開発を高速に行うことが市場に求められている．迅速に提供できたとしてもソフトウェアの品質が悪ければ提供価値が下がるため品質管理の重要性は変わらない．品質を保ちつつ品質管理にかかる作業は高速であることも求められている．プロジェクトでは次のフェーズに進んで良いか判断するために，各フェーズの終了時に品質を確認するクオリティゲートを設定することがよくある[2]．そのための品質報告情報のとりまとめや，次のフェーズに進んで良いか判定している期間がソフトウェア開発を停滞させてしまうことがある．この点に品質管理を高速にするための改善余地があると考え，解決するための手法を提案する．

定量的な品質管理は開発中に日々行うことが理想ではあるが，実現できているプロジェクトは多くはない．そこで日々の品質管理のために何を確認しているのかヒアリングを行った．その結果，テスト観点に対してテストを網羅的に実施できているかと，すり抜けバグの発生状況の確認を行っていることが多かった．この 2 つを定量的に確認できるような品質分析技術として整理し，無理なく品質管理を日々行うことで開発速度を落とさない品質管理手法を策定した．

本論文では 2 章で解決したい課題を明確にし，3 章で改善策の提案を行い，4 章で提案手法の適用結果を紹介し，5 章で今後の展開を述べる．

株式会社ＮＴＴデータ 技術革新統括本部 システム技術本部 生産技術部 ソフトウェア技術センタ
NTT DATA Corporation, Software Engineering Center, Production Engineering Department, System Engineering Headquarters
東京都江東区豊洲 3-3-9 豊洲センタービルアネックス
Toyosu Center Bldg. Annex, 3-9, Toyosu 3-chome, Koto-ku, Tokyo 135-8671, Japan
1) ソフトウェア技術センタ　主任　　e-mail: Hisatoshi.Kumagai@nttdata.com
2) ソフトウェア技術センタ　テクニカルグレード イノベータ（ソフトウェアプロセス）

【キーワード：】品質管理，品質評価，ソフトウェアテスト，バグ摘出率

2. 品質管理における課題

本論文で解決を図る 2 つの課題を示す.

2.1 プログラムコード行数を用いた品質管理指標の課題

ソフトウェア開発の定量的な品質管理ではプログラムコード行数あたりのバグ摘出数・テスト数である，バグ摘出密度・テスト密度といった指標を使うことが多い[1]. 昨今は業務パッケージソフトウェアや SaaS(Software as a Service)等の技術を活用したシステム開発も多くみられる. こうした開発では，プログラムコードを直接書くことが少ないため，プログラムコード行数を用いた指標値を利用することが難しくなっている. プログラムコード行数を取得しやすいスクラッチ開発であっても以下の課題がある.

バグ摘出密度・テスト密度は上限・下限値といった水準値を定めて指標値を評価するが，この手法は製造業における生産プロセスの考え方を元にしている. 製造業の生産プロセスの多くは大量生産であるため，統計的な不良品率を取得しやすい特徴がある. バラツキの要因には 5M(部品・設備・作業方法・作業者・測定精度)[4]があると言われており，極力揃えておかないと分析は非常に困難となる. 5M をソフトウェア開発で置き換えると，フレームワーク・開発ツール・開発プロセス・開発者・評価測定基準などに当たる. どの要素も多様化しており，バグ摘出密度・テスト密度の統計的に意味のある水準値を設定することが難しくなっている. 条件が異なる他のプロジェクトの水準値を安易に採用することは，品質管理の効果が出ないことが懸念されている[3]. プログラムコード行数の代わりにファンクションポイントを利用しようとしても，パッケージや SaaS も多様化しているためバグ摘出密度・テスト密度の利用が難しいことは同様である.

当社のプロジェクトにおいて，類似プロジェクトの実績情報を適切には入手できず期待していた品質管理の効果を得られないことや，プログラムコードを直接書かない開発の成果物を適切にプログラムコード行数換算できず比較ができないことが問題となることがあった. そのためプログラムコード行数や，類似プロジェクトの実績情報との比較に依存しない品質分析技術の確立が必要となっていた.

本論文ではテストフェーズにおける品質管理の課題解決を図る.

2.2 門番型品質管理の課題

ソフトウェア開発を高速に行うことが市場に求められている. 迅速に提供できたとしてもソフトウェアの品質が悪ければ提供価値が下がるため品質管理の重要性は変わらないものの，品質管理作業は高速であることも求められている. プロジェクトでは次のフェーズに進んで良いか判断するために，各フェーズの終了時に品質を確認するクオリティゲートを設定することがよくある[2]. これを本論文では門番型品質管理と呼称する. 品質報告のためのデータ収集及び資料作成や，次のフェーズに進んで良いか判定している期間がソフトウェア開発を停滞させてしまうことがある. クオリティゲートの直前になって品質管理を初めて行うのではなく，開発中の日々に品質管理を行い，その結果を改めて確認する場で本来はあるべきだ. しかし，実態ではクオリティゲート対応が独立してしまい開発を停滞させてしまっている事例が散見される.

当社のプロジェクトにおいて，品質評価報告ではバグ摘出密度・テスト密度をよく採用していが多いが,これを日常的に把握し利用することは難しい. 開発中は別の方法で品質を確認しており,門番型の対応のために改めて品質確認作業を行うということがある. 同じ目的の作業を 2 重に行うのは非効率であり，さらに２度行うという開発者への精神的負担により門番型で実施している定量的な品質管理が形骸化されてしまう懸念もある. より素早くソフトウェア開発を行うために,開発速度を落とさないソフトウェア品質管理手法の確立が必要となっていた.

3. 提案手法の解説

提案手法の方針と具体的な内容を解説する.

3.1 提案手法の方針

品質管理では定量的な確認により問題を検知したら終わりではなく，何故そうなったのか定性的に確認をする. 定性的な確認のみでは局所的な確認しかできていない可能性があるため，定量的な品質管理は必須だと考える[3]. 定性的確認は従来通り行うことを前提とし，本論文では定量

的な品質管理の改善を扱う.

定量的な品質管理は開発中に日々行うことが理想ではあるが,バグ検出密度・テスト密度を日々算出して確認するには稼働が多くかかり,実現できているプロジェクトは多くはない.しかし日々の品質確認をしていないわけではなく，定量的ではない方法で行われていることが多い．この確認事項を計測可能な定量的データに落とし込めれば，コード行数を用いない品質分析技術であり品質管理が効率化できると考えた．当社のプロジェクトにヒアリングした結果，以下２つを確認している傾向が見られた．これらをどのような分析技術に仕立てたかは 3.2 以降で示す.

・テスト観点に対してテストを網羅的に実施できているかの確認

・すり抜けバグの発生状況を確認

具体的な分析技術の内容を説明する前に，どのような方針で整備したのかその考え方を示す.JISX0129-1 における品質モデルの定義では，プロセス品質が内部品質，外部品質，利用時の品質に影響を与えていることがわかる．高品質なソフトウェアを生み出すためには，プロセス品質が重要であると言える.

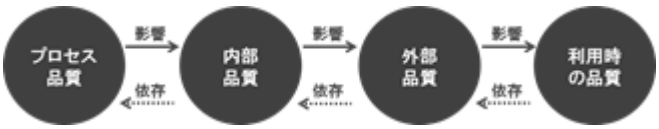

図 1 品質モデル(JISX0129-1 より)

プロセス品質を上げるためには，プロセス改善が重要である．プロセス改善のための手段としては PDCA サイクルの利用が非常に多い．PDCA サイクルでは，Plan が重要になる．計画がある程度の正確さを持たないと，Check が正しく機能しない．そのため，多くの時間をかけて Plan が行われることになる．もしくは時間がないため，計画の妥当性について確認が行われないままソフトウェア開発が始まってしまう．また，Check を通過することは多くの場合，プロセスの終了条件もしくは開始条件に設定される．厳重な PDCA は，開発を中断させる特徴があると考える.

そこで本提案では，軽量な PDCA サイクルと軽量な OODA ループを組み合わせた品質管理手法を提案する．OODA ループは Observe，Orient，Decide，Action からなるプロセス改善手法であり，Plan から始まる PDCA サイクルとは異なり，変化の激しい環境において迅速に対応方法を判断することのできるプロセス改善手法である．PDCA の Check を軽量化することで開発の中断期間を短縮しつつ，OODA ループを併用することで，プロセス品質の状態を監視し，問題があった場合にすぐ対処ができるようにする．従前の門番型品質管理と比較し，品質は維持しつつ開発速度を上げる品質管理手法を目指す.

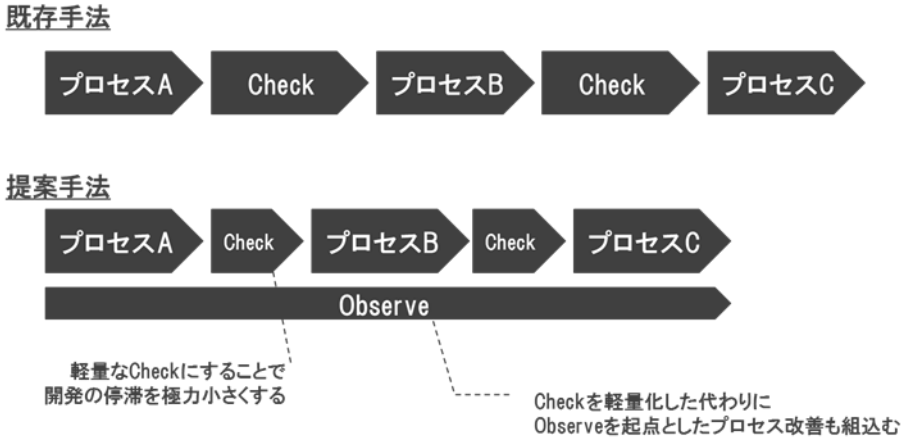

図 2 提案手法のイメージ

3.2 軽量な PDCA サイクルでの品質分析技術

PDCA サイクルにおいてはプロジェクトが開発中の品質管理で．テスト観点に対してテストを網羅的に実施できているかの確認をしていることを起点に，観点カバレッジという品質評価技術を策定した．観点カバレッジとはテスト観点にテスト項目数を対応付けた表である．(表 1 を参照)この表を用いることで，テスト観点に対するテスト項目の有無や明らかな偏りが確認できる.

表 1 観点カバレッジ表(テスト項目数)の例

テスト対象	観点 A	観点 B	観点 C
機能 α	10 件	2 件	**100 件**
機能 β	5 件	**0 件**	10 件
機能 γ	7 件	6 件	3 件

具体的には，まず Plan としてテスト観点を検討する．ここでのテスト観点は，テストレベルごとに設定するテストの目的とし，テスト観点を元にテストベースからテスト設計されるものと定義する．テスト観点の検討が完了した後，Do にあたるテスト設計およびテスト実行が行われる．Check では，テスト設計の成果物を確認し，テスト観点ごとのテスト項目数を計測して観点カバレッジ表を作成する．観点カバレッジを確認し，テスト項目数が 0 となっている箇所がないか，また 0 となることについて妥当な理由があるか，テスト項目数に異常な偏りがないか等の確認を行う．テスト項目の不足や，異常な偏りを検知した際は，Action としてテスト設計プロセスの改善を実施する．

また，Do でテスト実行を行った後，摘出されたバグに対しても，テスト観点へのマッピングを行う．その結果を Check することで，テスト観点ごとのバグの異常な偏りを検知することができる．いずれのテスト観点にもマッピングすることが困難なバグがある場合は，テスト観点に抜け漏れがあったと判断する．テスト観点に抜け漏れがあった場合は，テスト設計に立ち戻り，新たなテスト項目を追加する．

このように観点カバレッジを利用して PDCA サイクルを回し，プロセス改善を実現する．従来のバグ摘出密度・テスト密度を利用した品質分析とは異なり，Plan の時点でテスト項目数やバグ数などを統計的に算出する必要がなく，プログラムコード行数も用いていない．また，テスト観点ごとのテスト項目数などは，テスト項目を記載するドキュメントなどのフォーマットを工夫することで容易に自動算出することができ，check にかかる工数面から見た負担も少ない．

3.3 軽量な OODA ループでの品質分析技術

3.2 で提案した観点カバレッジは，プロセス品質におけるテスト観点に着目した問題点を検知できるが，プロセス品質の妥当性は判断できない弱点がある．この点は OODA ループの手法を用いることで補完する．OODA ループにおいてはプロジェクトが開発中の品質管理に，すり抜けバグの発生状況を確認していたことを起点に， DDP [5]を品質評価技術として採用する．DDP とは Defect Detection Percentage（欠陥摘出率）の略称で，すり抜けバグが増えると値が下がる特徴を持っている．

具体的には，Observe として DDP の推移をモニタリングし，Orient では週次のミーティングなどのタイミングで品質の低い可能性のあるプロセスを特定する．Decide では改善策を決定し，Action として確実に改善を行っていくことで品質改善を進める．

DDP は以下の式で算出し，評価対象のテストプロセスのバグを摘出する能力を示す．

$$DDP = n / (n + x)$$

n：評価対象のプロセスで検出できたバグ

x：評価対象のプロセスをすり抜けて検出されたバグ

例えば単体テスト，結合テスト，総合テストといった順で進むテストプロセスにおいて，単体テストの評価をする場合を考える．単体テストで 20 件のバグを摘出した場合，単体テスト完了時点での単体テストの DDP は 20/20 で 100%である．その後，結合テストにおいて，単体テストで見つかるべきであったバグが 5 件見つかると DDP は 20/(20+5)で 80%となる．提案手法では，DDP をモニタリングするにあたって，先行事例を参考に，DDP の評価閾値を予め定めている．

DDP はテストレベルと紐づけられているため，DDP を算出するために，追加の情報を入力したり，確認したりする必要はなく，工数面の負担なく運用することができる．

表 2 DDP 閾値の参考情報

DDP の閾値	評価	
x ＞ 95%	改善検討（後工程）	すり抜けバグは一定量出るほうが自然である．長期間 DDP が 95%を超えている場合，後続のテストプロセスが狙い通りの効果を出せていない可能性がある．
95%≧ x ≧85%	順調	テストは狙い通りの効果を出している．
85%＞ x ≧60%	改善検討	今後，DDP が 60%を下回るか注意深く観察する必要がある．重要な機能のテストについては早期にプロセス改善を検討してもよい．
60% ＞ x	改善必須	テストは狙い通りの効果を出せていない．実施中のテストを停止し，すぐさまプロセス改善に取り組まなければいけない．

3.4 開発速度を落とさない品質管理手法

3.2 と 3.3 で説明した観点カバレッジと DDP モニタリングはコード行数を用いない品質分析技術である．この 2 つを組み合わせて用いることで開発速度を落とさない品質管理の実現を目指す．

表 3 2 つの分析技術

思想	品質分析技術
PDCA サイクル	観点カバレッジ
OODA ループ	DDP モニタリング

あるテストフェーズの品質確認は，そのテストフェーズ実施中は観点カバレッジを用い，そのフェーズが完了後は DDP モニタリングを用いる．例えば単体テストの品質確認であれば，単体テスト中は観点カバレッジを用い，結合テスト以降は単体テストの DDP の値を継続してモニタリング(Observe)していく．もし DDP の変化に問題の傾向が見られた場合，実施中のテストを一時中断して単体テストの強化テストを行う．このようにフェーズ終了後にも確認を継続するため，フェーズ終了時の品質確認（Check）は従来の門番型のように開発作業を停滞させてまで精緻に行うという必要がなくなる．フェーズ終了時に門番を置くのではなく，後のフェーズにガードレールを設置することで開発作業をスムーズに次フェーズに進めるイメージである．

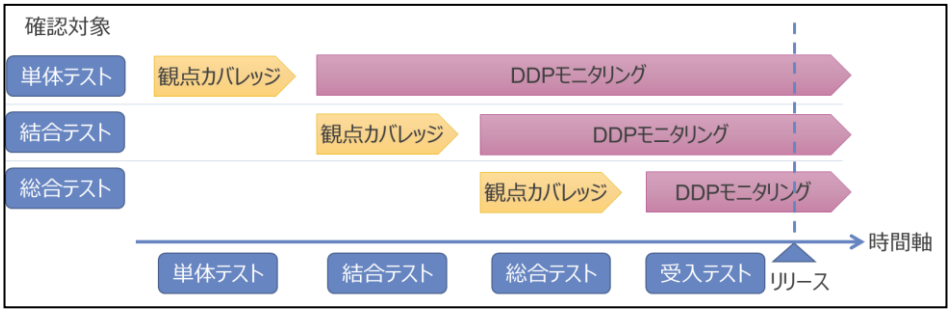

図 3 観点カバレッジと DDP モニタリングの実施タイミング

4. 検証結果と考察

当社の実プロジェクトで本手法を適用した結果を紹介する．

4.1 検証対象プロジェクトのプロファイル

デジタルコンテンツを検索する Java の Web システムの移行及び機能追加のプロジェクトで，機能追加部分に対して本手法を適用した．開発規模は約 4kStep であった．

単体テストと結合テストに品質評価に本手法を適用した．総合テストは従来手法を適用し，リリース判定は従来のコード行数を用いた品質評価と，本手法の観点カバレッジと DDP モニタリングを用いた品質評価の両方を行うことで評価結果を比較した．

4.2 観点カバレッジの適用結果

(1) テスト網羅性の確認

テスト観点は従前からありテストケース作成のインプット情報として利用していたが，どの観点をどのテストケースで満たせているか明確に分かる状態にはなっていなかった．そこでテストケース表の作成方法を変更し，観点カバレッジが自動で作成されるようにした．副次効果として，

テストケースでどのようなバグを検出したいのかが分かりやすくなったとの開発者からの声があがっている．

あるテスト対象のテストケースを作成が完了するとともに，そのテスト対象の観点カバレッジ表でテスト項目数が自動で集計されるようにフォーマットを工夫した．テスト項目が 0 件の箇所はテストケース作成者が 0 件で問題ない根拠を記載し，妥当な根拠がない場合はレビュー依頼に出す前にテスト項目の追加を行って是正する．セルフレビュー時点で修正できるため，レビューを受けてから修正する場合と比較し手戻りを削減できる．

レビュアーはテストケースの中身と共に観点カバレッジ表のレビューも行う．以下の表の例では，帳票様式の正当性のテスト観点に対する処理 A のテスト項目は 0 件であるが，当該処理では帳票出力を行わないため妥当な理由と考えられる．0 件で問題ない根拠の内容について意識齟齬があればテストケース作成者へテストケース追加指示を返した．

表 4 処理 A での観点カバレッジ表-テスト網羅性の結果（一部）

	入力バリエーションの網羅性	繰り返し処理の網羅性	帳票様式の正当性	…
テスト項目数	10 件	1 件	0 件	…
問題ない根拠	-	-	当該処理で帳票出力はないため	…

観点カバレッジ表はあるテスト対象のテストケース作成が終わる度に情報を追加していき，作成済みの他のテスト対象のテストケースの結果との比較を行いつつ進めた．あるテストフェーズの最後のテスト対象のテストケース作成が完了した時点で観点カバレッジ表は完成しており，品質評価者にはこれを提出した．従来であると開発作業とは別に品質評価者に提出するための資料を作成することが少なくなかったが，その工数を削減できた．

従来手法ではテスト密度があらかじめ設定しておいた下限値を下回っている場合に，テストが足りていない可能性があるため，それで問題ないか確認をしていた．本手法では観点カバレッジ表でテスト項目 0 件の箇所に対して，テストが足りていない可能性があるため，それで問題ないか確認することで代替できた．

(2) バグ網羅性の確認

単体テストでバグは 2 件検出し，観点カバレッジ表は以下のようになった．テストケースで狙ったバグの検出となっており，既存テスト観点と紐づかないバグの検出はなかった．仮に観点と紐づかないバグが検出されていたとしたらテスト観点及びテストケースの追加が必要となっていたことを補足する．

表 5 単体テストの観点カバレッジ表-バグ網羅性の結果（一部）

テスト対象	境界値の網羅性	入力バリエーションの網羅性	その他の観点
モジュール X	1 件	0 件	0 件
モジュール Y	0 件	1 件	0 件
その他モジュール	0 件	0 件	0 件

バグの内容が致命的なものではないこと，テスト網羅性の観点カバレッジ表で確認した通りテストが網羅的に実施できていることより，単体テストの品質評価結果は問題ないと判断した．

結合テストのバグは 9 件検出し，こちらもテスト観点と紐づかないバグはなかった．バグの内容が致命的なものではないこと，テスト網羅性の観点カバレッジ表で確認した通りテストが網羅的に実施できていることより，結合テストの品質評価結果も問題なしと判断した．

従来手法ではバグ摘出密度があらかじめ設定しておいた上限値を上回っている場合に，プロセス品質が悪い可能性があるため確認をしていた．本手法では偏ってバグの多いところが無いことの確認で代替している．精緻な定量データとしては従来手法に劣るが，その点はテストフェーズ終了後も DDP モニタリングを継続することで，もし問題を見逃していてもリリース前に発見できるようにすることで補っている．

4.3　DDP モニタリングの適用結果

単体テスト，結合テストそれぞれに対して当該フェーズ終了後から DDP モニタリングを行った．単体テストはリリースまですり抜けバグが検出されなかったため DDP100%のまま終了した．結合テストは 1 件すり抜けバグを検出し DDP が 90%となった．結合テスト中に検出したバグが 9 件と小さい数だったため 1 件すり抜けただけで 10%落ちたが，すり抜けたバグの横展開で新たなすり抜けバグは見つからず，これ以上すり抜けバグは出ないと判断してテストの中断はしなかった．結果，それ以上すり抜けバグは検出されず結合テストの DDP は 90%でリリースを迎えた．

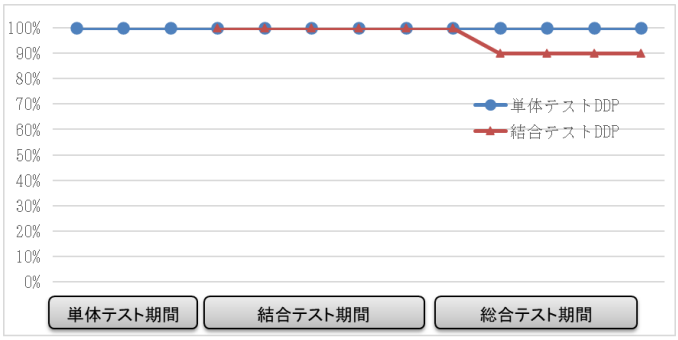

図 4　DDP グラフ

表 6　バグ検出

	テストフェーズ中のバグ	開発中のすり抜けバグ	リリース後すり抜けバグ
単体テスト	2 件	0 件	0 件
結合テスト	9 件	1 件	0 件

リリース判定において，DDP モニタリングの結果と観点カバレッジの結果より単体テストと結合テストの品質は問題ないと判断した．コード行数を用いた従来の品質評価も行ったが，その結果も問題なしで新手法での結論と一致した．リリース後にバグは発生しておらず実態としても問題なかった．そのため，このプロジェクトにおいてコード行数を用いない新手法のみの適用で品質管理を適正に行えていたものと結論する．

4.4　効率化の考察

新手法は従来手法と比較して以下の効率化が図れた。

- テストケース作成者がセルフレビューでテスト網羅性の問題を発見することにより，レビュアーが発見していたときと比べて手戻り削減
- １つのテストケース表の作成完了時点で品質の問題検知と修正対応が可能になり，フェーズ終了前に品質評価をしていたときと比べて手戻り削減
- 品質評価会議のために別途資料を作成していた工数の削減
- 単体テスト，結合テスト終了時点の門番型品質管理の撤廃による工期の短縮

5　今後の展開

観点カバレッジと DDP モニタリングはコード行数を用いた品質管理に代替できることを提案し，実プロジェクトにおいて適正に品質管理を行えたことを示した．また開発速度を落とさずに品質管理を実施できる見込みを示した．紹介した事例以外にも 2 プロジェクトでの適用も完了しており，それぞれ同様に効果を確認している．ただいずれも従来手法との併用での適用であったため，純粋に新手法のみを適用した実プロジェクトでの適用結果が得られたら追加報告したいと考えている．情報を公知にすることで，業界全体でこの課題に対処していけるように努めたい．

また本論文ではテストフェーズのみを改善対象としているが，それ以外のフェーズの品質管理の技術と手法に関しても検討中である．こちらも進展したら報告をしたいと考えている．

参考文献

[1] 小笠原秀人, 三浦邦彦，石田芳昭，小堺大亮, 篠田みゆき，富本達明，渡邊泰史，“ソフトウェア成果物の品質評価方法の研究”，Union of Japanese Scientists and Engineers(JUSE)，第 21 年度ソフトウェア品質管理研究会第 1 分科会

[2] 伊藤潤平，加藤大受，“クオリティゲートの通貨判断として品質特性を利用した受入テスト

の導入と効果”，ソフトウェア品質シンポジウム 2017
[3] 奈良 隆正，“ソフトウェアの品質保証の本質 技法の変遷から学ぶ”，日本ソフトウェアテストシンポジウム JaSST2017 招待講演，p.56
[4] 坂下嘉宏，“品質管理による継続的改善の必要性”，近畿アルミニウム表面処理研究会会誌 No311，2018
[5] Dorothy Graham，Challenges in Software Testing，JaSST'13 Tokyo，P.17，2014