情報処理学会論文誌 Vol. 50 No. 12 3074–3083 (Dec. 2009)

修正確認テスト規模の低減を目的としたコードレビュー手法

田村晃一[†1] 亀井靖高[†1] 上野秀剛[†2]
森崎修司[†1] 松本健一[†1]

ソフトウェア開発におけるテスト工程での欠陥修正には，修正部分の確認，および修正による新たな欠陥の混入がないことを確認するテストの両方が必要となるケースが多い．本論文では，欠陥の修正にともなって必要となる修正部分の確認ならびに再テスト規模の低減を目的としたコードレビュー手法を提案する．提案手法では，テスト規模が想定できる情報をレビューアに与えることにより，潜在的に修正確認テスト規模が大きくなる欠陥を予想しながら優先的に検出する．商用開発の実務経験者 6 名を含む 18 名の被験者の間で，提案手法と Test Case Based Reading（TCBR），Ad-Hoc Reading（AHR）を比較したところ，TCBR と比較して平均 2.1 倍，AHR と比較して平均 1.9 倍の修正確認テスト規模の削減が確認できた．

A Code Review Technique to Reduce Fix Assurance Test Size

Koichi Tamura,[†1] Yasutaka Kamei,[†1]
Hidetake Uwano,[†2] Shuji Morisaki[†1]
and Ken-ichi Matsumoto[†1]

In testing phases of software development projects, detected defects are fixed by modifying artifact including source code. Most of the detected defects require both test cases to confirm that the modification is correct and the modification does not cause new defects. In this paper, we propose a code reading technique in order to reduce size of such testing. The proposed method preferentially detects defects that potentially require larger size of testing by giving information that helps reviewer to estimate size of testing. In an evaluation experiment, the proposed technique reduces size of testing 2.1 times compared with test case based reading and 1.9 times compared with ad-hoc reading among 18 subjects including 6 commercial software developers.

1. はじめに

ソフトウェア開発における成果物もしくは中間成果物に対して作成者またはその他の開発者がその内容を確認することを目的として，レビューが実施されている．特に，欠陥を検出および除去することを目的として実施されるレビューは，テストと比較して早い段階での実施が可能であるため開発効率の向上につながることが報告されており[2),7),11),19)]，これまでにレビュー手法やその効果についての様々な研究が行われてきた[5),6),20)]．

レビューによる開発効率向上の理由の 1 つは，欠陥を早期に検出することによる修正工数と確認工数（成果物の欠陥を修正する工数，欠陥修正が期待どおり行われたかを確認する工数，欠陥修正にともない新たに別の欠陥が混入されていないかを確認する工数などの合計）の低減である．これは，テスト実施前にレビューを実施した場合の修正工数と確認工数の合計が，レビューを実施しなかった場合よりも小さくなることを前提としている．そのため，たとえばエラーメッセージの誤字脱字といった，早期検出により低減される修正工数と確認工数が小さい欠陥ばかりがレビューで検出されると，レビューによる効率化の効果が小さくなるおそれがある．

しかしながら，従来提案されているレビュー手法は欠陥の広範囲な検出に寄与することを主目的としており，検出される欠陥数が多くなることが期待される一方で，必ずしも欠陥が見逃された場合の修正工数と確認工数の大小について考慮されていない．そのため，レビューによる修正工数と確認工数の低減の効果が小さくなるおそれがある．たとえば，Checklist-Based Reading（CBR）[2)] では，見つけるべき欠陥を過去の事例や経験に基づいて事前にチェックリスト化し，それを確認しながらレビューを行うが，チェックリストの作成時やレビュー時に考慮すべき点として，欠陥の修正工数と確認工数への明示的な言及はない．

本研究の最終的な目的は，修正工数と確認工数を効率的に削減することにある．本論文では，文献 21) の調査結果をふまえ，特にレビュー時に見逃した場合に必要となる修正確認テスト規模に着目し，修正確認テスト規模が大きくなると推定される欠陥（確認工数が大

†1 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
†2 奈良工業高等専門学校情報工学科
Department of Information Engineering, Nara National College of Technology

© 2009 Information Processing Society of Japan

きくなる欠陥）を優先的に検出するコードレビュー手法を提案する．実験では，ソースコード，外部仕様書，内部設計書に加えて，テスト計画書を用いることによりレビュー時に修正確認テスト規模をレビューアが実際に見積もり，優先的にレビューすべき部分の特定および欠陥の検出ができるか否か確認し，提案手法により潜在的に軽減された修正確認テストの規模を調べる．比較対象の手法として，テストに関するドキュメントとあらかじめ用意した欠陥の種類（欠陥型）の一覧を用いてレビューする Test Case Based Reading（TCBR），および特定のレビュー手法を用いることなくレビューアの知識や経験に基づいてレビューを実施する Ad-Hoc Reading（AHR）を用いる．

以降，2 章では関連研究について述べる．3 章では修正確認テストの低減を目的としたコードレビュー手法を提案する．4 章では実験概要，レビュー対象，実験手順，評価尺度，実験結果について説明し，5 章で考察する．6 章でまとめと今後の課題について述べる．

2. 関 連 研 究

レビューにおける欠陥検出を効率的，効果的にするために様々な手法が提案されている．代表的なレビュー手法として Checklist-Based Reading（CBR）[2),12)]，Scenario-Based Reading（SBR）[9)] がある．SBR には，Perspective-Based Reading（PBR）[8),16)]，Defect-Based Reading（DBR）[13),14)]，Usage-Based Reading（UBR）[18)] などの手法が含まれる．

CBR は過去の経験などに基づいて，見つけるべき欠陥をチェックリスト化し，それを用いてレビューを行う手法である．PBR は対象システムの利用者，設計者，テスト担当者などの立場別に着目すべき観点を明示し，多面的に欠陥を検出する手法である．それぞれのレビューアが異なる観点を持つことで，対象システムに対して網羅的なレビューができる．DBR は定義された欠陥の種類ごとにレビューを行うことで効率的に欠陥を検出する手法である．異なる種類の欠陥を複数のメンバが個別にレビューすることで重複を減らし，PBR と同様に網羅的に欠陥を検出することができる．CBR，PBR，DBR はより多くの欠陥を検出することに焦点を当てているが，欠陥の優先度および修正確認テスト規模に関して明示的には言及されていない．

UBR はユースケースを用いることで，システムの利用者に大きな影響を与える欠陥を優先的に検出する手法である．提案手法では UBR と同様に欠陥に優先度を割り当てレビューを行うが，UBR ではユースケースに記述されていない動作についてはレビュー対象とならない場合がある．また，UBR はユースケースの優先度を考慮しているが，修正確認テスト規模とユースケースの優先度は必ずしも一致していない．

テストに関するドキュメントを用いたレビュー手法として Test Case Based Reading（TCBR）が提案されている[22)]．TCBR は，開発作業前にあらかじめブラックボックステスト用のテストケースを作成しておき，各テストケースを参考に，設計書またはソースコードの実行フローを追跡し，欠陥型（たとえば，文献 3) の欠陥型）に基づいて欠陥の検出を行う手法である．しかし，TCBR は優先的にレビューすべき部分や優先的に検出すべき欠陥，および必要となる修正確認テスト規模に関して明示的には言及されていない．

3. 提 案 手 法

3.1 概　　要

提案手法は，修正確認テスト規模の低減を目的とし，テスト工程で検出された場合に大きな修正確認テスト規模を必要とする欠陥をレビューで優先的に検出する．修正確認テストは，(1) 欠陥の修正後に当該機能の動作を確認するためのテスト（確認テスト），および (2) 欠陥の修正にともなって新たに欠陥が混入されていないか（デグレード）を確認するためのテスト（回帰テスト）からなる．

提案手法では，修正確認テスト規模を修正確認に必要なテストケース数（修正確認テスト件数）によって計測する．修正確認テスト件数は，確認テスト件数，回帰テスト件数の和である．確認テスト件数および回帰テスト件数はテスト計画書から得る．テスト計画書 I には次のテスト項目 I_k が含まれる．

$$I_k = <Description_k, c_k>$$

$Description_k$ は自然言語で記されたテスト内容，c_k は予定確認テスト件数である．

なお，テスト計画書は，テスト観点の設定とテストの所要時間を見積もることを目的としたドキュメントであり，提案手法ではテスト計画書がコードレビューに先立って作成されていることを前提とする．テスト計画書のテスト項目は，テスト項目の実施フェーズ（単体テスト，結合テスト，システムテスト），テスト視点（論理テスト，機能テスト），テスト方法（ブラックボックス，ホワイトボックス）を問わない．ただし，提案手法は，修正確認テスト削減によるテスト工数の低減を目的としているため，結合テスト，システムテスト，テスト視点においては，機能テストにおいて特に効果が得られることが期待される．

3.2 手　　順

以下に示す手順で対象システムのソースコードのレビューを行う．

(1)　ソースコードユニットへの分割

レビュー対象のソースコード U をソースファイル，クラス，関数，メソッドなどの

© 2009 Information Processing Society of Japan

構文上の区切りで分割し，それぞれをソースコードユニット u_i とする．分割の区切りは，ソースコードの規模，レビュー時間の制約によって決定する．

(2)　テスト項目 I_k のテスト対象となるソースコードユニットの列挙

$Description_k$ をもとに，テスト項目 I_k のテスト対象となるソースコードユニットの集合 U_k を求め，I_k に追加し I'_k とする．

$$I'_k = <Description_k, c_k, U_k>$$

(3)　ソースコードユニットごとの修正確認テスト件数の算出

ソースコードユニット u_i（$i = 1, \ldots, |U|$）の修正確認テスト件数 n_{u_i} を次の手順で求める．

for u_i in U **do**
  for I'_k in I' **do**
    if $u_i \in U_k$ **then**
      $n_{u_i} = n_{u_i} + c_k$
    end if
  end for
end for

(4)　レビューの実施

与えられたレビュー工数の範囲で n_{u_i} が大きいソースコードユニット u_i から順番に，欠陥が含まれないかレビューする．

なお，具体的なテスト計画立案が完了していないなど，何らかの事情で，修正確認テスト件数 c_k が得られない場合でも，手順 (2) において c_k どうしの大小関係を相対的に求めることができれば，本手順を実施できる．具体的には，テスト内容 $Description_k$ から c_k の確認テスト件数の大小関係を決定し，文献 4) に示されている Analytic Hierarchy Process (AHP)[15] を利用する．AHP の代わりに Cumulative Voting (CV)[10] を利用することもできる．ただし，AHP や CV はテスト項目数 $|I|$ が大きくなると適用が難しくなることが考えられるため，I_k を抽象化することによって Hierarchical Cumulative Voting [1] を適用したり，複数人で手分けすることによって大規模 AHP [23] を利用したりすることが考えられる．

3.3 実　施　例

図 1 に示すソースコード，および図中のテスト $O1$，$O2$，$S1$，$S2$，$P1$，$P2$ を対象に提案手法の実施例を示す．本実施例では，ソースコードを関数単位で読み進めていき（ソースコードユニットを関数とする），テストで欠陥が検出された場合に確認テスト件数が最も大きくなる関数から順番にレビューを進めるものとする．また，本実施例では，レビューアはシステムの大まかな動作とテスト件数は把握できているが，ソースコードは事前には理解できていないことを想定している．

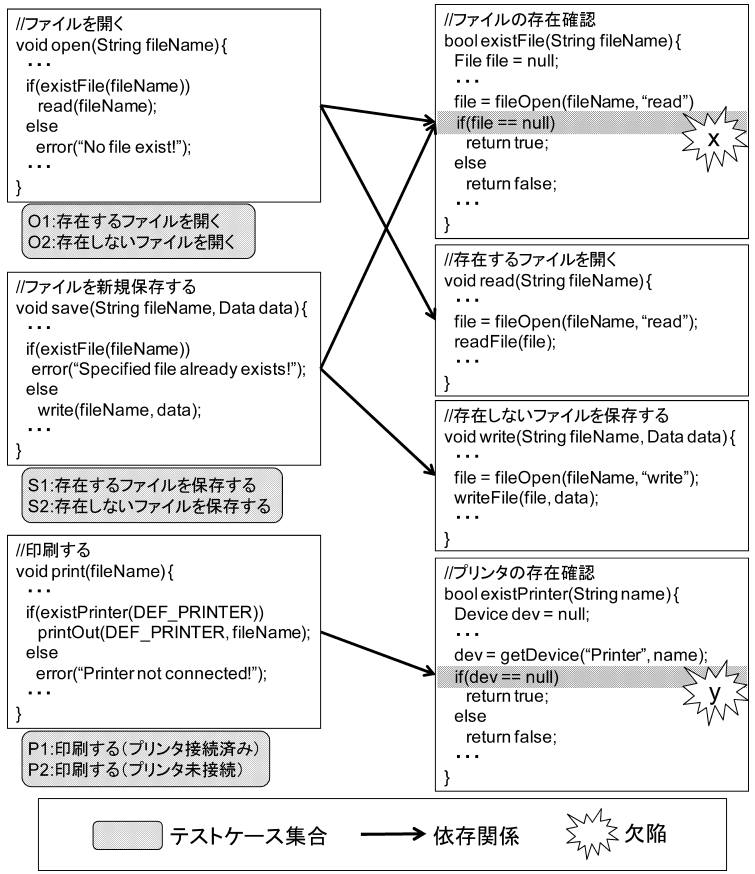

図 1　ソースコードの依存関係と修正確認テストの例
Fig. 1　The examples of the dependence of source codes and fix assurance tests.

© 2009 Information Processing Society of Japan

表 1　図 1 のテスト項目，テスト件数，対象ソースコードユニット
Table 1　Test items, number of test cases and source code unit in Fig. 1.

	テスト内容	予定テスト件数	テスト対象となるソースコードユニット
I'_1	O：ファイルを開くことができるかを確認する	2 件	open，existFile，read
I'_2	S：ファイルを保存することができるかを確認する	2 件	save，existFile，write
I'_3	P：印刷することができるかを確認する	2 件	print，existPrinter

図中の x，y は欠陥を表している．以降では，x，y の修正確認テスト件数は x のほうが大きいことを示し，その後，提案手法で x のほうが y よりも先に検出されることを手順に照らしながら示す．関数 existFile に含まれる欠陥 x を関数 open のテストで検出した場合，確認テスト件数が 2（O1 と O2），回帰テスト件数が 2（S1 と S2）であり，修正確認テスト件数は 4 である．一方，関数 print のテスト（P1 または P2）で関数 existPrinter に含まれる欠陥 y を検出した場合，確認テスト件数は 2（P1 と P2），関数 existPrinter は他の部分との依存関係を持たないため回帰テスト件数は 0 であり，修正確認テスト件数は 2 となる．以下は前節で述べた手順に沿って，図 1 に示すソースコードとテストケースを対象として提案手法を実施したものである．

(1)　ソースコードユニットへの分割
対象ソースコードをソースコードユニット（open，save，print，existFile，read，write，existPrinter）に分割する．

(2)　テスト項目 I_k でテスト対象となるソースコードユニットの列挙
テスト計画書 I から**表 1** に示される I' を求める．具体的には以下のとおり．
I'_1：ファイルを開くことができるかを確認する際に実行されるソースコードユニットを図 1 のソースコードから目視で確認する．open 関数，および，open 関数から呼び出される existFile 関数，read 関数をテスト対象となるソースコードユニット U_1 とする．
I'_2：ファイルを保存することができるかを確認する際に実行される save 関数，および，save 関数から呼び出される existFile 関数，write 関数をテスト対象となるソースコードユニット U_2 とする．
I'_3：印刷することができるかを確認する際に実行される print 関数，および print 関数から呼び出される existPrinter 関数をテスト対象となるソースコードユニット U_3 とする．

(3)　ソースコードユニットごとの修正確認テスト件数の算出
I' から，ソースコードユニット（関数）existFile の修正確認テスト件数 4 を得る．また，それ以外のソースコードユニットの修正確認テスト件数 2 を得る．

(4)　レビューの実施
最も修正確認テスト件数が大きい existFile に欠陥がないかレビューし，次に修正確認テスト件数が大きい open，save，print，read，write，existPrinter を与えられたレビュー工数の範囲でレビューする．

本実施例は，関数の呼び出し依存関係を用いた見積りの例を示したが，観点は必ずしも呼び出し関係だけに依存するものではなく，ほかにも，データ構造の依存関係や計算機リソースの共有（性能，メモリ，デッドロックをはじめとするタイミング問題）に起因する修正は修正範囲が広く，修正確認テスト件数を増大させる傾向があるため，修正確認テスト件数の見積りを行う対象となる．

4. 実　　験

4.1 概　　要

提案手法が修正確認テスト件数の低減につながること，および多くの修正確認テスト件数が必要となる欠陥を数多く検出できることを確認するために，提案手法，TCBR，および AHR の比較実験を行う．すべてのレビューアには，レビュー対象である C 言語で記述されたソースコード，外部仕様書，および内部設計書を与える．これらに加えて，提案手法を実施するレビューアにはテスト計画書，TCBR を実施するレビューアにはテスト計画書および欠陥型の一覧を与える．

本実験におけるレビューアは，C 言語によるプログラミング経験のある学生 12 名と，商用のソフトウェア開発を実務で行った経験のある実務経験者 6 名である．提案手法がレビューアの経験の長短に強く依存するかどうかを調べることを目的として，実務経験者，学生を被験者に含めている．この 18 名を提案手法を実施するレビューア 6 名，TCBR を実施するレビューア 6 名，AHR を実施するレビューア 6 名（それぞれ学生 4 名，実務経験者 2 名）に分割し，比較実験を行った．レビューアの C 言語およびその他のプログラミング言語の経験年数が均等になるようグループの分割を行った．

4.2 レビュー対象

レビュー対象は次のようなイベント受付システム（Web サーバで実行される CGI）を実現するためのソースコードである．システムには「システム管理者」「イベント管理者（主

© 2009 Information Processing Society of Japan

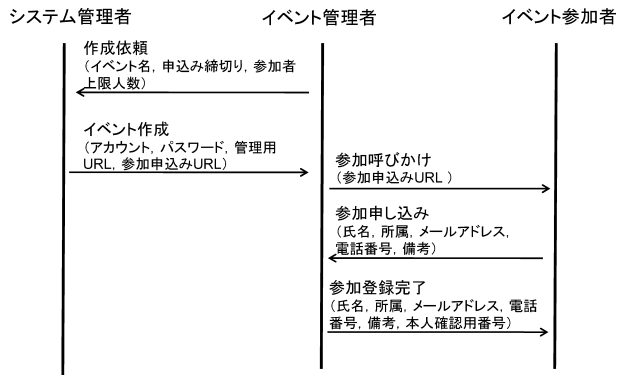

図 2　イベント受付システムのデータの流れ
Fig. 2　The data flow of the event reception system.

催者)」「イベント参加者」の 3 者が関わる．システムは，複数のイベントの参加登録を同時に受け付けることができる．イベントは URL によって区別され，システム管理者がイベントを登録した時点で URL が発行される．イベント管理者は自身が管理するイベントの URL を参加者に伝え，イベント参加者は住所，氏名といった情報をシステムに送信する．

図 2 はイベント受付システムの情報のやりとりを示したものである．個々のイベントを主催するイベント管理者がシステム管理者にイベント作成の依頼をし，作成画面からイベントを作成する．イベントを作成すると作成されたイベントの参加申込み用 Web ページの URL が生成され，システム管理者からイベント管理者に伝えられる．イベント管理者はこの参加申込み用の URL をイベント参加者に通知し，イベント参加者は参加申込み画面を通じて，参加に必要な情報をイベント管理者に送信する．イベント管理者は，イベント管理者画面を通じて，登録者リストを見ることができる．

対象としたソースコードは C 言語で書かれており，Linux 上で Apache Web サーバの CGI として動作する．規模は約 1,500 LOC であり，7 個のファイルからなる．本システムは本実験用に，類似システムの業務での開発経験を持つ実務経験者により開発されたものであり，類似システムの開発経験を持つ別の実務経験者により，現実的な開発や設計がされていること，ドキュメント（外部仕様書，内部設計書，テスト計画書）に不備がないことを確認した．

すべてのレビューアには外部仕様書，内部設計書，ソースコードが渡される．外部仕様書，内部設計書は自然言語で書かれている．加えて，提案手法でレビューを行うレビューアにはテスト計画書，TCBR でレビューを行うレビューアにはテスト計画書および欠陥型の一覧が渡される．テスト計画書は**表 2** に示すような表形式のものであり，機能大分類，画面の名称，機能小分類，予定テスト件数が記述されており，3 個の機能大分類と 43 個の機能小分類からなっている．欠陥型は文献 22) と同様に文献 3) の PSP 欠陥型標準を用いた．**表 3** に被験者に提示した欠陥型の一覧を示す．

表 2　実験に使用したテスト計画書の一部
Table 2　A part of the test plan document used in the experiment.

機能大分類	画面名称	機能小分類	予定テスト件数
参加登録機能	登録情報入力画面	イベント名ブラウザタイトル表示機能	4
		期限切れ確認機能	6
		参加人数超過確認機能	4
		ボタン押下によるページ遷移確認	1
	登録情報確認画面	「ご氏名」条件判断とエラーメッセージ	7
		「ご所属」条件判断とエラーメッセージ	7
		「メールアドレス」条件判断とエラーメッセージ	8
…	…	…	…

表 3　被験者に提示した欠陥型（文献 3)）
Table 3　The fault type used in the experiment.

型番号	型名	型番号	型名
10	文章	60	チェック
20	構文	70	データ
30	ビルド，パッケージ	80	機能
40	割当て	90	システム
50	インタフェース	100	環境

表 4　レビューでの検出により削減できる修正確認テスト件数
Table 4　The fix assurance test cases reduced by detection in the experiment.

	最小値	中央値	平均値	最大値
修正確認テスト件数	1	7	10.6	71

ソースコード中には開発中に混入された欠陥がそのまま残されており，計 34 個の欠陥が確認されている．レビューで検出した場合に削減できる欠陥 1 件あたりの修正確認テスト件数の最小値，中央値，平均値，および最大値を**表 4** に示す．表中の値は，欠陥が存在するソースコードユニットの修正確認テスト件数から求めた．具体的には，ソースコードユニットを関数とし，3.2 節で示した手順と表 2 のテスト項目からソースコードユニットごとの修正確認テスト件数を算出した．

© 2009 Information Processing Society of Japan

修正確認テスト件数は原則として確認テストと回帰テストの合計件数とした．ただし，確認テストと明らかに重複する回帰テストのテストケースが存在している場合には，重複するテストケースの一方を省略可能と見なした．たとえば，文字列の長さをチェックするライブラリ関数の欠陥を修正した場合，文字列の長さをチェックするテストの中の「参加者氏名」「参加者住所」といった類似した（半角，全角文字で入力され最大長が設定された入力項目）複数のテストケースをすべて実施する必要はないため，必要のないテストケースを省略する．具体的には次式のとおりである．

修正確認テスト件数
＝確認テスト件数＋回帰テスト件数－省略可能テスト件数

ここで，省略可能テスト件数とは，ホワイトボックステストと考えた際に確認テストおよび回帰テストの中で他のテストケースと類似または重複しているテストケースの件数を表している．

4.3 手　　順

評価実験は以下の手順で行った．

(1) レビューアにレビュー対象としているシステムの説明を行う．説明では，ソースコード以外のドキュメントには欠陥は含まれないとし，ソースコードに含まれる欠陥の数は伝えていない．また，レビューア全員に，評価実験におけるレビューを「テストで検出する場合よりも，レビューで検出することにより，修正確認工数がより小さくなるような欠陥を目視で検出すること」であると説明した．

(2) 提案手法および TCBR を実施するレビューアには各レビュー手法についての説明を行い，AHR を用いるレビューアには特定のレビュー手法に関する説明は行わない．

(3) 提案手法を実施するレビューアにはテスト計画書を与え，ソースコードユニットを関数とすることを指示した．TCBR を実施するレビューアにはテスト計画書と欠陥型の一覧を与える．

(4) レビューアは手順 (2) で説明したレビュー手法を用いてレビュー作業を行う．欠陥を検出した際にはその内容と検出時刻を記録する．また，制限時間を過ぎた場合は，レビュー作業が途中であっても終了する．

実験に要する時間は，手順 (1) が 30 分，手順 (2) および (3) が 10 分（AHR のレビューアはなし），手順 (4) が 60 分の計 100 分（AHR のレビューアは計 90 分）であり，手順 (4) ではすべての欠陥を検出できなくても終了するものとする．

4.4 評 価 尺 度

テスト計画書に記述されている予定テスト件数に基づいて，削減できた修正確認テスト件数（削減件数）を評価する．また，削減される修正確認テスト件数が上位 50%（上位 17 個）に含まれる欠陥を優先的に検出すべき欠陥と考え，それらを検出した個数（優先欠陥検出数）も評価する．

これらの指標に加えて，文献 9) および 17) で定義されている次式の尺度も用いる．

$$\text{検出数} = \frac{\text{検出欠陥数}}{\text{レビュー時間（時）}}$$

$$\text{検出率} = \frac{\text{検出欠陥数}}{\text{全欠陥数}}$$

4.5 結　　果

実験結果を**表 5** に示す．レビューア全体では，削減件数の平均値は提案手法が，TCBR の約 2.1 倍，AHR の約 1.9 倍の値であった．優先欠陥検出数の平均値は提案手法が，TCBR および AHR の約 2.1 倍の値であった．検出数および検出率の平均値は提案手法が，TCBR の約 2.0 倍，AHR の約 2.6 倍の値であった．レビューア全体において，提案手法は TCBR および AHR よりも修正確認テスト件数を削減できたといえる．また，TCBR と AHR で削減できる修正確認テスト件数に大きな差はないが，TCBR の方が AHR よりも数多くの

表 5　実験における各レビュー手法の修正確認テスト削減能力と欠陥検出能力
Table 5　The performance of each reading technique of fix assurance test reduction and fault detection in the experiment.

		平均値		
		提案手法	TCBR	AHR
全体	削減件数	45.6	21.7	24.2
	優先欠陥検出数	2.5	1.2	1.2
	検出数	4.5	2.2	1.7
	検出率	13%	6%	5%
学生	削減件数	30.0	11.3	17.8
	優先欠陥検出数	2.3	1.0	1.0
	検出数	3.8	1.8	1.0
	検出率	11%	5%	3%
実務経験者	削減件数	76.5	42.5	37.0
	優先欠陥検出数	3.0	1.5	1.5
	検出数	6.0	3.0	3.0
	検出率	18%	9%	9%

© 2009 Information Processing Society of Japan

欠陥を検出できたといえる．

学生のレビューアでは，削減件数の平均値は提案手法が，TCBR の約 2.7 倍，AHR の約 1.7 倍の値であった．優先欠陥検出数の平均値は提案手法が，TCBR および AHR の約 2.3 倍の値であった．検出数および検出率の平均値は提案手法が，TCBR の約 2.1 倍，AHR の約 3.8 倍の値であった．学生のレビューアにおいても，提案手法は TCBR および AHR よりも修正確認テスト件数を削減できたといえる．また，AHR の方が TCBR よりも修正確認テスト件数を削減できたが，TCBR の方が AHR よりも数多くの欠陥を検出できたといえる．

実務経験者のレビューアでは，削減件数の平均値は提案手法が，TCBR の約 1.8 倍，AHR の約 2.1 倍の値であった．優先欠陥検出数の平均値は提案手法が，TCBR および AHR の約 2.0 倍の値であった．検出数および検出率の平均値は提案手法が，TCBR および AHR の約 2.0 倍の値であった．実務経験者のレビューアにおいても，他のレビューアと同様に，提案手法は TCBR および AHR よりも修正確認テスト件数を削減できたといえる．また，TCBR の方が AHR よりも修正確認テスト件数を削減できたが，TCBR と AHR で検出できる欠陥数に差は見られなかった．

図 3 は，削減件数の累積（平均値）の推移を示している．横軸はレビュー開始からの経過時間を示し，縦軸は削減された修正確認テスト件数を示している．実線は提案手法，破線は TCBR，点線は AHR のテスト件数を表している．なお，すべてのレビューアは制限時間である 60 分間レビューを行った．開始から約 23 分後までは AHR の累積削減件数の平均値が最も大きかったが，それ以降は提案手法の累積削減件数の平均値が最も大きくなった．TCBR の累積削減件数の平均値は時間にかかわらず最も低かった．また，開始から 23 分以降と 25 分以降は，それぞれ提案手法と TCBR において，累積削減件数の増加の割合も大きくなった．

図 4 は，優先欠陥検出数の累積（平均値）の推移を示している．横軸はレビュー開始からの経過時間を示し，縦軸は累積優先検出欠陥数を示している．実線は提案手法，破線は TCBR，点線は AHR の累積欠陥数を表している．開始から約 26 分後までは AHR の累積数の平均が最も大きかったが，それ以降は提案手法の累積数の平均が最も大きくなった．開始から 59 分まで，TCBR の累積削減件数の平均値は最も低かった．また，開始から 23 分以降と 26 分以降は，それぞれ提案手法と TCBR において，累積優先欠陥検出数の増加の割合も大きくなった．

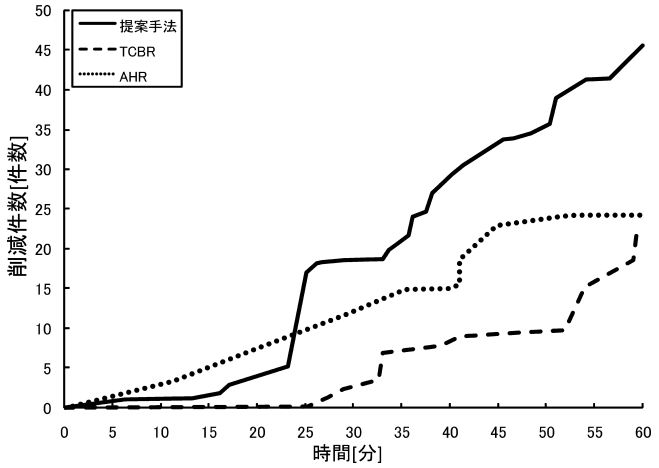

図 3　削減件数の累積（平均値）

Fig. 3　The cumulative fix assurance test cases by the detected faults in the experiment (mean value).

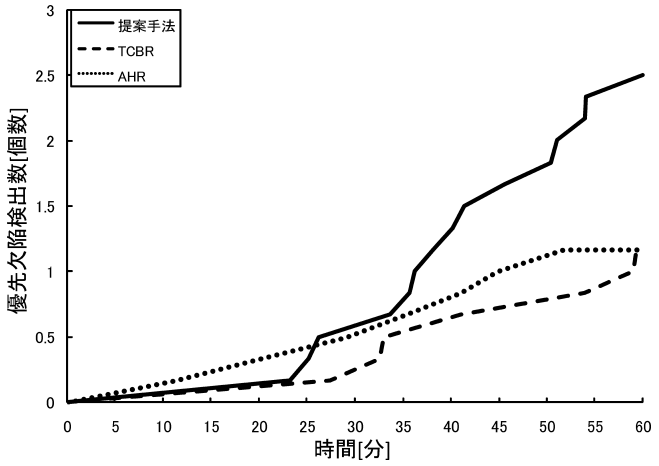

図 4　優先欠陥検出数の累積（平均値）

Fig. 4　The cumulative numbers of detected faults needed over the top 50% fix assurance test cases in the experiment (mean value).

© 2009 Information Processing Society of Japan

5. 考　　察

TCBR および AHR との比較実験により提案手法の有効性を示す結果が得られた．修正確認テストの削減件数，修正確認テスト件数上位 50%に含まれる欠陥が検出された数（優先検出欠陥数）ともに，提案手法は TCBR および AHR よりも大きな値を示した．提案手法によりレビューアは修正確認テスト件数が大きくなるような欠陥を優先して検出でき，それにともない多くの修正確認テスト件数を削減することができたと考えられる．学生，実務経験者によらず，提案手法の効果を確認できた．

実際にレビューアが提案手法により修正確認テスト件数を考慮し優先的にレビューすべき部分を特定できたかどうかをレビューアへのインタビューにより確認したところ，提案手法が期待どおりの効果をもたらしていることを裏付ける次のような回答が得られた．「提案手法を用いることで優先的にレビューすべき部分の特定ができ，レビューが円滑に行えたように感じた」，「レビュー中に，削減できる修正確認テスト件数が小さい（削減に寄与しない）と予想される欠陥を検出した場合において，その欠陥と類似した別の欠陥を見つけることに固執せず，修正確認テスト件数が大きくなると予想される欠陥を優先的に検出することに注力することができた」．これらの回答は提案手法により修正確認テストが大きくなるような欠陥の検出を優先できることを示していると考えられる．提案手法を実施したレビューアは 3.2 節の手順 (2) の作業を，テスト内容に対応する入出力画面を類推し，テストを実施した際に最初に実行されるソースコードユニットを探すことからはじめていた．手順 (3) では，最初に実行されるソースコードユニットから呼び出されている関数を類推し，データ構造などそのほかにも，依存関係のある部分を探していた．一方，AHR を実施したレビューアのレビュー方針についてもインタビューしたところ，「他からよく呼び出されているライブラリ周りからレビューした」「設計書に書かれているシステムの利用手順に従ってレビューした」といった回答が得られた．

AHR と比較して提案手法は修正確認テストが大きくなる部分を特定するための時間を要する可能性を示す結果も得られた．図 3 が示しているように，レビューを開始してから 23 分後までは AHR の方が修正確認テスト件数の平均が大きな値となっている．レビューアへのインタビューにおいても「レビュー開始直後はテスト計画書および内部設計書を読み，各機能の呼び出し依存関係を理解し，優先的にレビューすべき部分を特定する作業を行っていた」という回答が得られた．

本評価実験では，提案手法の手順 (2) および (3) で必要となるソースコードユニットごとの修正確認テスト件数を見積もる時間をレビュー時間に含めている．プロジェクト計画時など，コードレビューに先立ってこの見積り作業を実施している場合には，修正確認テスト件数を見積もる作業を省略することができ，より効率的に欠陥を検出できる可能性がある．一方，本評価実験のようにレビュー対象に関する知識の少ないレビューアが提案手法を実施する場合，優先的にレビューすべき部分を特定する作業に AHR と比較して多くの時間が必要となる可能性がある．そのため，レビュー対象ソフトウェアの件数，レビューアの対象ソフトウェアの理解度を考慮し，事前に対象ソフトウェアを理解する時間を確保するなどの対策を検討する必要がある．

本実験のレビュー対象は C 言語で記述されており，機能とソースコードの対応が比較的明確なものが多い傾向にあった．レビュー対象がオブジェクト指向プログラミング言語で記述されている場合には機能とプログラムの構成要素との関係が必ずしも明確でない場合もあるが，この場合でもテスト対象となるソースコードユニット（クラスやメソッドなど）は推定可能と考える．テスト内容（$Description_k$）にはテスト対象が記述され，対応するソースコードユニットを推測することはそれほど困難ではないからである．ただし，プログラミング言語がもたらす影響は，今後検討すべき重要な課題の 1 つである．

6. おわりに

本論文では，修正確認テスト規模が大きくなると推定される欠陥を優先的に検出し，潜在的な修正確認テスト規模を低減させることを目的としたレビュー手法を提案した．提案手法を評価するために，Test Case Based Reading（TCBR）および Ad-Hoc Reading（AHR）との比較実験を行い，以下の結果が得られた．

- 提案手法の方が修正確認テスト件数を TCBR よりも約 2.1 倍，AHR よりも約 1.9 倍多く削減できた．
- 多くの修正確認テスト件数が必要となる欠陥について，TCBR および AHR よりも提案手法の方が約 2.1 倍多く検出できた．
- 提案手法には優先的にレビューすべき部分を特定する作業が必要なため，AHR を実施した場合と比較してレビュー開始からしばらくの間は欠陥検出数，削減される修正確認テスト件数が少なくなる可能性がある．

なお，本論文では修正確認テスト規模（修正確認テスト工数）と修正確認に必要なテストケース件数（修正確認テスト件数）との間に強い相関があるとし，修正確認テスト規模の低減を修正確認テスト件数の低減により計測した．対象ソフトウェアや対象プロジェクトに応じて，修

© 2009 Information Processing Society of Japan

正確認テスト規模を計測するための他の評価尺度を検討することは，今後の課題の 1 つである．

また，本論文ではソースコードをレビュー対象としたが，提案手法はテスト規模を考慮できる情報が存在する開発であれば，設計書レビューにも適用が可能であると考えられる．

今後は，提案手法の効果と制限を明確にするために，異なる規模のソースコードを用いて同様の実験を行う予定である．また，コードレビューに適用可能なその他の手法である CBR，PBR などと比較実験を行う予定である．さらに，CBR や PBR と同様に，提案手法ではレビュー実施前の準備が必要となるため，準備のコストも考慮してレビュー手法を評価する予定である．

謝辞　実験に参加してくださった学生ならびに実務経験者の皆様に感謝いたします．本研究の一部は，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．また，本研究の一部は，文部科学省科学研究補助費（若手 B：課題番号 21700033）による助成を受けた．

参 考 文 献

1) Berander, P. and Jönsson, P.: Hierarchical Cumulative Voting (HCV) – Prioritization of Requirements in Hierarchies, *International Journal of Software Engineering and Knowledge Engineering*, Vol.16, No.6, pp.819–849 (2006).
2) Fagan, M.E.: Design and Code Inspection to Reduce Errors in Program Development, *IBM Systems Journal*, Vol.15, No.3, pp.182–211 (1976).
3) Humphrey, W.S.: *A Discipline for Software Engineering*, Addison-Wesley, Boston, MA, USA (1994).
4) Karlsson, J. and Ryan, K.: A Cost-Value Approach for Prioritizing Requirements, *IEEE Software*, Vol.14, No.5, pp.67–74 (1997).
5) Kusumoto, S., Matsumoto, K., Kikuno, T. and Torii, K.: A New Metric for Cost Effectiveness of Software Reviews, *IEICE Trans. Information and Systems*, Vol.E75-D, No.5, pp.674–680 (1992).
6) Laitenberger, O., DeBaud, J. and Runeson, P.: An Encompassing Life Cycle Centric Survey of Software Inspection, *The Journal of Systems and Software*, Vol.50, pp.687–704 (2000).
7) Laitenberger, O.: Studying the Effects of Code Inspection and Structural Testing on Software Quality, *Proc. 9th International Symposium on Software Reliability Engineering* (*ISSRE'98*), pp.237–246, IEEE Computer Society Press (1998).
8) Laitenberger, O. and Debaud, J.M.: Perspective-based Reading of Code Documents at Robert Bosch GmbH, *Information and Software Technology*, Vol.39, No.11, pp.781–791 (1997).
9) Lanubile, F., Mallardo, T., Calefato, F., Denger, C. and Ciolkowski, M.: Assessing the Impact of Active Guidance for Defect Detection: A Replicated Experiment, *Proc. 10th International Symposium on Software Metrics*, pp.269–279 (2004).
10) Leffingwell, D. and Widrig, D.: *Managing Software Requirements: A Use Case Approach*, 2nd edition, McGraw-Hill (2003).
11) Melo, W., Shull, F. and Travassos, G.H.: Software Review Guideline, Technical Report, COPPE/UFRJ Systems Engineering and Computer Science Program Technical Report ES556/01 (2001).
12) Myers, G.J.: *The Art of Software Testing*, 1st edition, John Wiley and Sons, New York (1979).
13) Porter, A.A. and Votta, L.G.: An Experiment to Assess Different Defect Detection Methods for Software Requirements Inspections, *Proc. 16th International Conference on Software Engineering*, pp.103–112 (1994).
14) Porter, A.A., Votta, L.G. and Basili, V.R.: Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment, *IEEE Trans. Softw. Eng.*, Vol.21, No.6, pp.563–575 (1995).
15) Saaty, T.L.: *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, McGraw-Hill (1980).
16) Shull, F., Rus, I. and Basili, V.: How Perspective-Based Reading Can Improve Requirements Inspections, *IEEE Computer*, Vol.33, No.7, pp.73–79 (2000).
17) Thelin, T., Andersson, C., Runeson, P. and Dzamashvili-Fogelstrom, N.: A Replicated Experiment of Usage-Based and Checklist-Based Reading, *Proc. 10th International Symposium on Software Metrics*, pp.687–704 (2004).
18) Thelin, T., Runeson, P. and Wholin, C.: An Experimental Comparison of Usage-Based and Checklist-Based Reading, *IEEE Trans. Softw. Eng.*, Vol.29, No.8, pp.687–704 (2003).
19) Wiegers, K.E.: *Peer Reviews in Software – A Practical Guide*, Addison-Wesley, Boston, MA, USA (2002).
20) 田村晃一，亀井靖高，上野秀剛，森崎修司，松村知子，松本健一：見逃し欠陥の回帰テスト件数を考慮したコードレビュー手法，電子情報通信学会技術研究報告，Vol.108, No.173, pp.61–66 (2008).
21) 松村知子，門田暁人，森崎修司，松本健一：マルチベンダ情報システム開発における障害修正工数の要因分析，情報処理学会論文誌，Vol.48, No.5, pp.1926–1935 (2007).
22) 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，Vol.2004, No.118, pp.25–31 (2004).
23) 八巻直一，関谷和之：複数の評価者を想定した大規模 AHP の提案と人事評価への適用，日本オペレーションズ・リサーチ学会論文誌，Vol.42, No.4, pp.405–421 (1999).

(平成 21 年 4 月 3 日受付)
(平成 21 年 9 月 11 日採録)

© 2009 Information Processing Society of Japan

田村　晃一

平成 19 年岡山県立大学情報工学部卒業．平成 21 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，奈良先端科学技術大学院大学博士後期課程在籍．ソフトウェアインスペクション，ソフトウェアメトリクスの研究に従事．

亀井　靖高（正会員）

平成 17 年関西大学総合情報学部卒業．平成 21 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．博士（工学）．エンピリカルソフトウェア工学，特にソフトウェア信頼性の研究に従事．電子情報通信学会，IEEE 各会員．

上野　秀剛

平成 16 年岩手県立大学ソフトウェア情報学部卒業．平成 21 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年奈良工業高等専門学校情報工学科助教．博士（工学）．ソフトウェア開発におけるヒューマンファクタの研究に従事．電子情報通信学会，IEEE，ACM 各会員．

森崎　修司（正会員）

平成 13 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年（株）インターネットイニシアティブ入社．オンラインストレージサービスの立ち上げ/企画/開発，RFID ソフトウェアの国際標準策定活動に従事．平成 17 年奈良先端科学技術大学院大学情報科学研究科研究員．平成 19 年同大学助教．ソフトウェアレビュー，エンピリカルソフトウェア工学の研究に従事．博士（工学）．IEEE 会員．

松本　健一（正会員）

昭和 60 年大阪大学基礎工学部情報工学科卒業．平成元年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 5 年奈良先端科学技術大学院大学助教授．平成 13 年同大学教授．工学博士．エンピリカルソフトウェア工学，特に，プロジェクトデータ収集/利用支援の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員，IEEE Senior Member．

© 2009 Information Processing Society of Japan