社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2004-53 (2005-01)

プログラム理解度がコードレビュー達成度に及ぼす影響の分析

栗山 進† 大平 雅雄† 門田 暁人† 松本 健一†

† 奈良先端科学技術大学院大学 情報科学研究科 〒630-0101 奈良県生駒市高山町 8916-5
E-mail: †{susu-ku,masao,akito-m,matumoto}@is.naist.jp

あらまし 本研究の目的は，コードレビューにおけるレビューアのプログラム理解度と，コードレビュー達成度 (バグの発見率) との関係を分析することである．レビューアのプログラム理解度を調べるために，コードレビュー実施後，レビュー対象コードを複数の側面から問う試験問題を課す実験を行った．実験の結果，ロジックやインタフェースに関する理解度との相関は見つからなかったものの，データ構造やデータ操作に関する理解度は，それらと関連するバグの発見率との相関が高いことがわかった．また，プログラム理解度は，バグを見つけるための必要条件となっていることがわかった．

キーワード コードレビュー，試験，定量化，ソフトウェアプロセス，ソフトウェア品質

Analysis of program comprehension that has effects on the code review achievement

Susumu KURIYAMA†, Masao OHIRA†, Akito MONDEN†, and Ken-ichi MATSUMOTO†

† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama-tyou, Ikoma-city, Nara, 630-0101 Japan
E-mail: †{susu-ku,masao,akito-m,matumoto}@is.naist.jp

Abstract The goal of this study is to analyze the relationship between the level of program comprehension and the bug detection rate in a code review process. We examined the reviewer's comprehension level of a program after the review process was held by the reviewer. As a results of examinations, we have found that the bug detection rate correlated with the comprehension level of data structure and its manipulation while it did not correlated with the comprehension level of logic and interface. We also found the program comprehension was a necessary condition for detecting bugs.

Key words code review, examination, quantification, software process, software quality

1. はじめに

ソフトウェアの社会的重要性が高まる中で，ソフトウェアを開発する企業や組織にとって，ソフトウェアの品質向上は，近年，特に重要な課題の一つである．ソフトウェアの品質向上のための取り組みの一環として，多くのソフトウェア開発の現場では，ソフトウェアライフサイクルの各工程で作成される作業成果物に対してピアレビュー (peer review) が実施されている．

ピアレビューとは，ソフトウェア作成者以外の 1 人以上の人間が，要求定義，設計，テスト計画，テスト項目，ソースコードなどの作業成果物を調べ，欠陥を発見したり改善を行う活動を指す [1]．ピアレビューには，捕らえにくいバグを突き止めるために同僚の開発者に即席でソースコードをチェックしてもらうといったアドホックレビューや，教育的・訓練的側面の強い構造化ウォークスルー [2]，最も厳格なレビュープロセスが定義されているインスペクション [3] など，柔軟性と効果によって様々な種類がある．

ピアレビューは，作成者以外の人間によって複数の視点から各種作業成果物を調べることになるため，作成者本人の視点では気づくことができない欠陥を発見する機会を提供する．しかしながら，レビューを行う人物の能力や経験，あるいはレビュー対象物への理解力の違いによって，発見可能な欠陥の数や種類に大きな違いが生じる場合も少なくない．このため，作業成果物に含まれる欠陥の数や種類が未知の時点で実施されるピアレビューは，どれ程の労力をかければどの位の欠陥を見つけることができるかを，容易には予測することができないという問題がある．

ピアレビュー (または単にレビュー) という用語は，種々の作

業成果物中に含まれる欠陥を発見するための活動を指すものであるが，一般的にはソースコードを入念に調べバグを発見するためのコードレビューを指す場合が多い[1]．本稿でも特に断りがない場合は，コードレビューを指すこととする．

これまで，我々の研究グループでは，レビューの質の低下を早期に発見しプロジェクト管理者を支援することを目的として，レビューの質を定量化するための試験方法の提案を行い，試験結果 (プログラム理解度) とレビュー達成度とに相関が見られることを確認した[4]．本稿では，バグの種類をより細かく分類した上で，これまで提案した試験方法を拡張し，分類 (データ，インタフェース，ロジック，仕様) 毎にプログラム理解度を調べた実験結果について報告する．

本稿の構成は以下の通りである．続く 2 章では，コードレビューにおけるプログラム理解度の影響と問題点について述べる．3 章では，ソースコードに対する理解度を定量的に測りレビュー達成度とそれらの関係を調べるための実験を詳述し，4 章でその実験結果を示す．5 章では，今回の実験結果を考察し，最後にまとめを述べて本稿を結ぶ．

2. コードレビューにおけるプログラム理解度の影響と問題点

ソフトウェアの品質向上を目的としてこれまで提案されてきた様々なツールや技法の中でも，Fagan が開発したインスペクション[5] は，作業成果物中に含まれる欠陥を発見する方法として最も有効性が高い手法であるとされている．インスペクションによって摘出可能な欠陥の割合は，60%～90% にも達すると報告されている[6],[7]．

このように，最も高い欠陥発見率を期待することができるインスペクションではあるが，欠陥の発見率に大きな差異が見られるのも事実である．この原因は，インスペクションプロセスにおけるレビューの実施方法やミーティングでの対人関係・社会的関係などがあるとされており[8]，現在でも議論が続いている．

本研究は，レビュー対象となるプログラムへの理解度が，レビューの質にどのように影響するのかを明らかにすることを目的としている．プログラム作成能力 (生産性) については，従来から数倍以上の個人差あると指摘されている[9]．同様に，ピアレビューで対象とする各種作業成果物に対する理解力にも個人差があるのは明らかである[10],[11]．

最終プロダクトであるソフトウェアの品質を管理する責任のある品質管理者などの立場では，レビューの質をある水準以上 (重大な欠陥がないことなど) に保つことが必須の課題であり，どれくらいのレビュー能力のある人間をレビュー担当者として確保するべきかについては大きな関心事である．成果物の品質確保のみならず，できるだけ早い段階で欠陥を除去することは，プロジェクトの遅延を防止 (手戻り作業などで必要となる工数は，欠陥の発見が後れれば遅れるほど膨大になる) する意味でも大きな問題である[12]．

レビューの質をある程度確保するために，インスペクションや構造化ウォークスルーでは，複数人でレビューを実施する前に，各個人で仕様書，設計書，ソースコードなどの完全性をチェックする「準備」の段階がある[1]．

「準備」の段階では，様々なソフトウェア作業成果物に見られる欠陥の典型例を洗い出した欠陥チェックリストやルールセットを使用する．欠陥チェックリストを用いることによって，プログラムに障害を引き起こす可能性のある誤りの原因候補に注意を集中しやすくなる．ルールセットは，タスクを実施したり成果物文書を作成したりする際に特定の方法で行うように作成者に文書で指示するものであり，文章の内容，構造，表記法，構成などを指定するルールからなる．

チェックリストやルールセットは，最低限の水準以上のレビューの質を確保するために提供されるものであって，個々人の経験や能力による個人差が完全に解消される訳ではない．欠陥あたり工数 (欠陥を 1 つ見つけるのに必要な平均工数) と準備速度 (一人のレビューアが準備の段階で 1 時間で調べられる平均コード行数) との関係や，インスペクションの速度と発見される欠陥数との関係など，レビューの投資効果や最適なレビュー速度を計測し，インスペクションプロセスを定量的に管理するためのメトリクスも提案されている[13],[14] が，個々人のプログラム理解度の違いがレビューの質にどのように影響するのかに着目したものは，文献[4] 以外には見当たらない．

そこで次節では，プログラム理解度とレビューの質との関係を定量的に測定する試験方法をについて述べる．[4] の試験方法をバグの分類を細分化し拡張することによって，プログラム理解度とレビューの質との関係をより詳しく調べるためのものである．

3. 実　　験

3.1 レビュー対象のプログラム

実験に使用したプログラムは，酒屋倉庫管理問題のプログラムであり[4]．大阪大学情報工学科の演習で学生によって作成された．プログラムは C 言語で記述されており，604 行の大きさであった．コンパイラが発見できる構文上の誤りは，あらかじめ除去した．このプログラムは 17 個のバグを含んでいたが，今回の実験では，さらに著者が 12 個のバグを追加した．1 つのバグを追加するに際し，多数の文を大幅に変更することはせず，文の入れ換えや削除，式の変更にとどめた．また可能な限り，不自然なバグを追加しないように努めた．

3.2 バグの分類

プログラムは複数の構成要素 (関数，データ構造など) からなり，また，複数の評価の側面 (アルゴリズムの正当性，仕様との適合性など) を持つものである．そこで，プログラムの理解度を調べるにあたり，理解の側面を幾つかに分類した．そして，その分類に従い，バグも分類した．バグの一般的な分類方法というのはなく，文献[15] を参考にしながら主観的に決定した．以下に，分類名と，その分類に対応するプログラムの構成要素や側面を挙げる．括弧内は，各分類毎のバグの個数である．

- データ (7): グローバル変数，構造体，配列，プログラム実行時に動的に確保されるデータ領域とそれらに対する操作．

- インタフェース (8): 関数のインタフェース (戻り値，引数，関数の呼び出し順の制限など).

- ロジック (6): ロジックの誤り (アルゴリズム，フラグの意味・条件，境界値条件に関する判定，ループ制御など).

- 仕様 (8): 仕様に記述されている機能が実装されていないものと，仕様に記述されていない機能が実装されているもの.

3.3 レビュー環境

前述のプログラムに対して，被験者一人ずつにコードレビューを行ってもらった．被験者は，奈良先端科学技術大学院大学情報科学研究科の 11 名 (博士前期課程 9 名，博士後期課程 1 名，助手 1 名) である．実験に先立ち，各被験者には，ソースコードのハードコピー，仕様書，実験者が作成したプログラム概要説明文書とがレビューの資料として渡された．このプログラムの概要説明文書には，プログラム中に定義されている各関数の概要が説明されている．

これらの文書をもとに，各被験者にコードレビューを実施してもらいバグを見つけてもらう実験を行った．レビューの方法については被験者に特別な指示は与えず，被験者の自由な方法で行ってもらった．レビュー時間についても制限は設けず，被験者の自由に任せ，被験者がすべてのバグを発見できたと思った時点 (自力で発見できるところまで) で終了した．

各レビューには，実験者が同席し，C 言語の仕様と標準ライブラリ関数に関して，被験者が不明だと思った事項に対して質問を受け付けた．ただし，プログラムの動作に関する質問は受け付けなかった．

3.4 試験手順

レビュー終了直後，2 回の試験を受けてもらった．レビュー終了直後に試験を行ったのは，試験成績が，コードレビュー終了から試験を受けるまでの時間に左右されることを防ぐためである．

レビュー終了直後に行った 2 回の試験の目的は，どちらの試験おいても，被験者がソースコードや仕様書を理解した度合いを調べる事である．試験問題は，被験者によらず同一であり，2 回の試験で同じ試験問題を与えた．

1 回目の試験においては，レビュー実施時に使用した資料 (ソースコード，仕様書，概要説明文書) を参照できない状態で試験問題を解いてもらった．2 回目の試験においては，再び同じ試験問題を解いてもらうが，被験者は資料を参照することを許可された．このような手順で試験を行った理由を以下に述べる．

1 回目の試験で資料への参照をできないようにした理由は，資料を参照することでレビュー時よりも試験中にプログラムを理解するための情報量が増えてしまう可能性を防ぐためである．ただし，1 回目の試験では，レビュー中に与えた資料の内容を覚えているかどうかという記憶力に依存した解答結果となる可能性が高い．

そこで，2 回目の試験においては，試験実施時の資料を閲覧することを許可した．ただし，2 回目の試験では，レビュー時に理解していなかった事柄に関する質問事項に対して，被験者はその場で確認したり理解してから解答する可能性がある．

表 1 それぞれの分類に対応する問題文と解答欄の一例

分類名	問題文と解答欄
データ	コンテナ構造体の領域は，必要時に動的に割り当てているか？ yes / no / わからない
インタフェース	出庫処理をする関数 SendLiquor は，引数の一つとして搬入日時を受け取るか？ yes / no / わからない
ロジック	指定した量の酒があるかどうか調べる isExistLiquor 関数は，コンテナ内の酒の構造体を調べるが，その際 MAX_LIQUOR 回だけ配列を調べているか？ yes / no / わからない
仕様	積荷票の搬入年月としてどれが正しいか選択せよ. (a) 199401 / (b) 9401 / (c) わからない

今回の実験では，それぞれの条件においての試験結果を総合的に分析することによって，被験者のプログラム理解度を調べることとした．

3.5 試験問題

試験問題は全部で 77 個の小問題からなる．試験問題は 1 問毎に 3.2 節で述べた分類の中の 1 つに深く関係したものとなっている．データ，インタフェース，ロジックの分類に対応する問題がそれぞれ 19 個，仕様の分類に対応する問題が 20 個である．表 1 に問題文と解答欄の一例を示す．

試験問題は全てこのような選択解答形式となっている．1 つの問題は，問題文と，解となりうる選択肢 (主に yes と no) と，'わからない' という選択肢からなっている．被験者には，正しいと思う選択肢がある場合にはそれに印をつけてもらい，わからない場合は 'わからない' に印をつけてもらった．

自由記述で解答する形式の試験問題にした場合，理解しているが明示的に答えられない事柄は記述できないため，理解度をより定量的に計測するためにすべての設問を選択式で解答できるようにした．選択解答方式では，理解できなかった事柄も解答可能であるが，'わからない' という項目が選択できることと，前節で述べたように同じ試験を 2 回実施し両方を分析することで選択解答形式の試験問題の欠点を補うことが可能であると考えた．

3.6 採点方法

試験の成績として以下のようにバグの分類毎の正答率を計算し，プログラム理解度を測る尺度とした．選択肢 'わからない' の場合は，不正解とした．

$$\text{ある分類に対する成績} = \frac{\text{分類毎の正答した数}}{\text{分類毎の小問題の総数}}$$

4. 実験結果

図 1 に，全体のバグ発見率と，試験全体の正答率との相関を示す．図 2-5 は，分類別のバグの発見率と，対応する試験の正

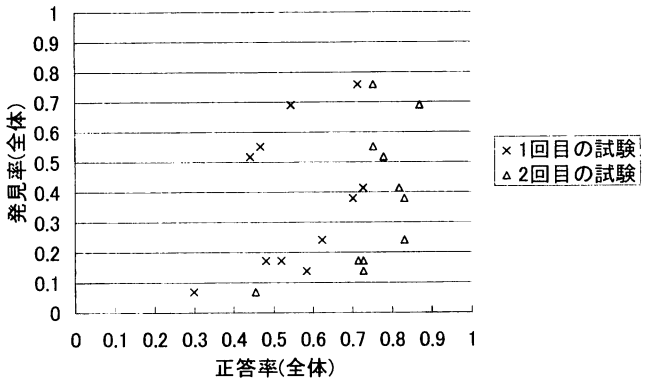

図 1　バグ発見率 (全体) と試験の正答率との相関

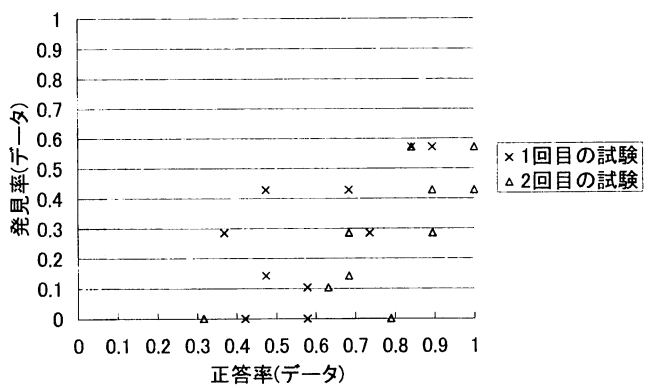

図 2　バグ発見率 (データ) と試験の正答率との相関

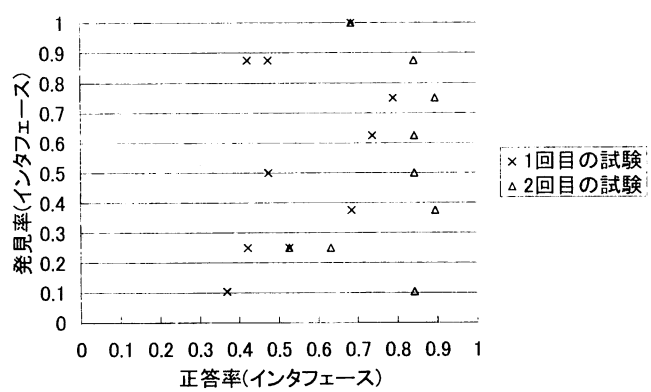

図 3　バグ発見率 (インタフェース) と試験の正答率との相関

答率との相関である．表 2 には，1 回目と 2 回目の試験の正答率とバグ発見率の相関係数をそれぞれ示す．

ここで，分類別のバグ発見率とは，コードレビューに使用したプログラムに潜んでいた各分類別のバグの総数に対して，各被験者がレビューで発見したバグの割合を意味する．試験問題の正答率は，3.6 節において述べた計算式で算出した．

縦軸はバグの発見率を，横軸は試験の正答率を示している．グラフ上の各点は，1 人の被験者の試験の正答率とバグ発見率に対応している．本実験のように，同じプログラムを同じ条件の下でレビューしても，バグ発見率が被験者によってかなり異なることがわかる．また，各問題の正答率は，バグ発見率と関連があることがわかる．

図 2 において，試験の正答率が高い程，高いバグ発見率を示すという傾向が，1 回目と 2 回目の試験の両方で見て取れる．「データ」に分類されたバグに関しては，プログラムを良く理解している程，バグを多く発見するということである．この他の種類のバグにおいては同様の関係は見られない．

図 3 においては，1 回目の試験において，低い正答率を示しながらも，高いバグの発見率を示す被験者が 2 名存在していることが見て取れる．同様に，図 4，図 5 においても，このような傾向を示す被験者が存在していることがわかる．これついては 5 章で考察する．

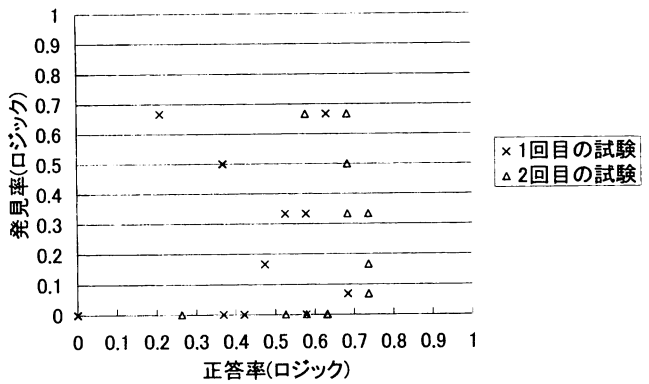

図 4　バグ発見率 (ロジック) と試験の正答率との相関

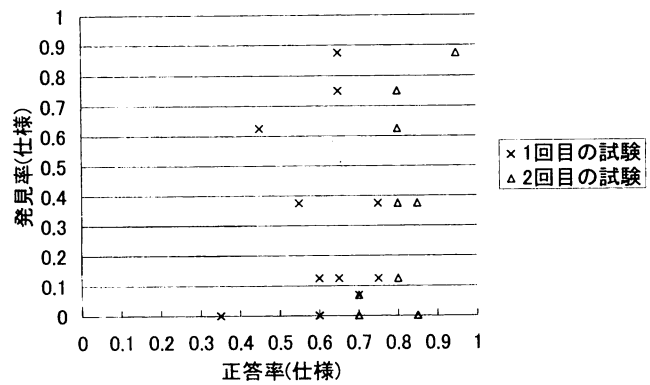

図 5　バグ発見率 (仕様) と試験の正答率との相関

表 2　バグ発見率と試験の正答率の相関係数

	1 回目の試験	2 回目の試験
全体	0.382000843	0.553229807
データ	0.633588261	0.733207061
インタフェース	0.222891853	0.052907429
ロジック	0.067301439	0.350269722
仕様	0.067168934	0.594145526

また，図 3 において，2 回目の試験の正答率に着目すると，大部分のレビューアの正答率が，0.8–0.9 の間という比較的狭い範囲に分布していることがわかる．これと同様の関係が，図 4 や図 5 においても見て取れる．

5. 考　　察

5.1　先行研究との比較

文献 [4] においては，2 つの異なるプログラムに対して同様の分析がなされている(注1)．その結果によると，相関係数はそれぞれ，0.720，0.746 となっており，強い相関が認められる．しかしながら，表 2 で示したように，今回の実験においては強い相関は認められなかった．

(注1)：[4] は，1 回の試験による正答率とバグ発見率との関係を測定している点，2 種類のみのバグ分類で分析を行っている点，全体のバグ個数が 17 個である点で本研究とは異なる．

5.2　全体の傾向

試験の正答率とバグ発見率との間において見られる，全体的な傾向について考察する．図 2–5 を見ると，インタフェースやロジックにおける一部の例外を除いて，低い正答率を示した被験者のバグ発見率は低くなっていることがわかる．また，高い正答率を示した被験者であっても，バグの発見率が高いとは限らないことも見て取れる．以上の事から，試験の正答率，即ち，プログラム理解度は，バグを見つけるための必要条件になっていることが考えられる．

なお，1 回目の実験の，被験者の答案を見てみると，'わからない' という解答が多く見られた．試験後に被験者に，どうしてわからなかったのかと尋ねると，覚えていないからという返答が多かった．これは即ち，試験問題に，記憶を試す問題が少なからず含まれていたものと考えられる．それゆえ，プログラムを良く理解していてバグを良く発見しているにも関わらず，低い正答率となってしまったものと思われる．

5.3　バグの分類毎の傾向

5.3.1　デ　ー　タ

4 章で述べたように，データ構造やデータ操作について良く理解している程，その種のバグの発見率が高いことが実験結果からわかった．この種のバグについては，まずデータ構造やその操作をより理解した上でレビューにのぞむことが重要であるといえる．また，今回行ったような試験をレビュー終了後に実施することによって，レビュー達成度を定量的に管理すること

ができるものと期待される．

5.3.2 インタフェース

インタフェースに属するバグとして，今回のプログラムにおいては，関数の返り値に対する処理のバグと，関数へ適切な実引数を渡していないバグしか存在しなかった．これらのバグは，関数の返り値をきちんと確認すること，関数の仮引数の意味と実引数の意味を確認することによって，比較的容易に見つけることができると考えられる．この種のバグを探す際にはチェックリストを用いるのが有効であると思われる．試験の正答率とバグ発見率に相関がみられなかったのは，プログラムは理解していたが，レビュー時のチェックを怠っていたためと推測される．

5.3.3 ロジック

ロジックにおいては，ループを回る回数が1回異なるという簡単なバグが3つ含まれていたが，これらのバグが発見される率(そのバグを発見した人数/レビュー者の総人数)はそれぞれ，18%，18%，36%であった．これらのバグのうちの1つを以下に示す．

> for文によるループで，`i < MAX_REQBUF`と書かれているべき所が，`for (i = 0; i <= MAX_REQBUF; i++)`のように，`i <= MAX_REQBUF`と書かれており，ループを一回多く回してしまっている．

このようなバグの見逃しを防ぐためには，チェックリストを用いるのが有効であると思われる．試験の正答率とバグ発見率に相関がみられなかったのは，インタフェースのところで述べた理由とほぼ同じであると考えられる．

5.3.4 仕　　様

仕様においては，試験の正答率に大きな違いは認められなかったものの，仕様の未実装を指摘した被験者らと，指摘しなかった被験者らの2群に分かれることがわかった．後者の群に属する被験者らは，仕様書とソースコードとの間の対応に注意を払わなかったものと考えられる．仕様書とソースコードとの間の対応は，チェックリストを用いることにより支援が行えるため，この種のバグを探す際にはチェックリストを用いるのが有効であると考える．試験の正答率とバグ発見率に相関がみられなかったのは，インタフェースのところで述べた理由とほぼ同じであると考えられる．

6. まとめと今後の課題

本稿では，プログラム理解度とコードレビュー達成度との関係を分析することを目的とした実験について述べた．

実験結果を分析した結果，データ構造やデータ操作について良く理解している程，その種のバグの発見率が高いことが実験結果からわかった．この種のバグについては，レビュー達成度を試験で調べることができると考えられる．その他のバグ種別においては同様の関係は見られなかった．しかしながら，全体的な傾向から，試験の正答率，即ちプログラム理解度は，バグを見つけるための必要条件となっていることがわかった．

5.3.2-5.3.4項で述べたように，プログラム理解度以外にもレビューの質に影響を与える要因が存在すると考えられる．プログラムの細部をきちんと確認するといった行為，対応が取られているべき箇所をきちんと確認するといった行為がそれである．このことは同時に，チェックリストやルールセットを用いてレビューを行うことの有効性を示すものであると考えられる．

今回の実験においては，コード中に含まれていたバグの多くが，チェックリストやルールセットを用いてレビューすれば発見できた可能性がある．今後の課題として，被験者全員に同一のチェックリストやルールセットを用いてレビューを行ってもらい，その上で，プログラム理解度とレビューの質の関係を調べる必要がある．

データ構造やデータ操作について良く理解している程，その種のバグの発見率が高いことがわかったが，その原因理由についてはまだ不明な部分が多い．これについては，以下のような課題をクリアしながら明らかにしていく予定である．

- 別のプログラムでも同様の結果が得られるのか調べる
- 記憶に依存しにくい問題の作成方法を見つける
- レビューの質に影響を与える他の要因の考察

謝辞　本稿で行った実験に快く参加して下さった被験者の皆様に深く感謝します．本研究の一部は，文部科学省「e-Society基盤ソフトウェアの総合開発」の委託に基づいて行われた．

文　　献

[1] K.E. Wiegers：“ピアレビュー”, 大久保 雅一 (監訳), 日経 BP ソフトプレス (2004).
[2] E. Yourdon：“ソフトウェアの構造化ウォークスルー”, 伊土 誠一, 富野 壽 (監訳), 近代科学社 (1991).
[3] T. Glib, D. Graham：“ソフトウェアインスペクション”, 国友 義久, 千田 正彦 (訳), 共立出版 (1999).
[4] 的場 政明：“コードレビューの質を定量化するための試験方法の提案と評価”, 修士論文, 奈良先端科学技術大学院大学 情報科学研究科, NAIST-IS-MT35112 (1995).
[5] M. E. Fagan: “Advances in software inspections”, IEEE Trans. Software Engineering, **12**, 7, pp. 416–423 (1986).
[6] B. Boehm and V. R. Basili: “Software defect reduction top 10 list”, IEEE Computer, **34**, 1, pp. 135–137 (2001).
[7] R. L. Glass: “Practical programmer: inspections?some surprising findings”, Commun. ACM, **42**, 4, pp. 17–19 (1999).
[8] R. L. Glass：“ソフトウエア開発 55 の真実と 10 のウソ”, 山浦 恒央 (訳), 日経 BP 社 (2004).
[9] H. Sackman, W. J. Erikson and E. E. Grant: “Exploratory experimental studies comparing online and offline programming performance”, Commun. ACM, **11**, 1, pp. 3–11 (1968).
[10] A. J. Ko and B. Uttl: “Individual differences in program comprehension strategies in unfamiliar programming systems”, 11th IEEE International Workshop on Program Comprehension (IWPC'03), IEEE, pp. 175–184 (2003).
[11] S. Rifkin and L. Deimel: “Program comprehension techniques improve software inspections: A case study”, 8th International Workshop on Program Comprehension (IWPC'00), IEEE, pp. 131–138 (2000).
[12] B. W. Boehm: “Software engineering”, IEEE Trans. Computers, **25**, 12, pp. 1226–1241 (1976).
[13] J. Barnard and A. Price: “Managing code inspection information”, IEEE Software, **11**, 2, pp. 59–69 (1994).
[14] R. G. Ebenau: “Predictive quality control with software inspections”, Cross Talk, Vol. 7, no. 6, pp. 9–16 (1994).
[15] B. Beizer：“ソフトウェアテスト技法”, 小野間 彰, 山浦 恒央 (訳), 日経 BP 出版センター (1994).