社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SE－163 (25)
2009／3／19

ソースコードの差分情報を用いたコードレビューコストの分析

保田 裕一朗 †, 森崎 修司 †, 松本 健一 †

† 奈良先端科学技術大学院大学情報科学研究科 〒630-0192 奈良県生駒市高山町 8916-5

あらまし プログラマは変更したソースコードに欠陥がないかを確認するために，レビュアにコードレビューを依頼する．しかし，複数のレビュー業務を抱えているレビュアの場合，それぞれのレビューを行うために必要な労力や時間（すなわちコードレビューコスト）を手際よく見積らなくてはならない．そこで本研究では，変更前後のソースコードの差分情報から計測できる値を元に，コードレビューコスト見積り手法の提案を目指す．本稿はその第一歩として，オープンソースソフトウェアの開発過程で作成された様々な差分ファイルをレビューすることで，コードレビューコストに影響を与えている要因を調査した．その結果，変更の種類，変更量，複雑度，影響範囲が主な要因として挙げられた．

Code Review Cost Analysis using Difference Information from Source Code

Yuichiro YASUDA † Shuji MORISAKI † Ken-ichi MATSUMOTO †

† Graduate School of Information Science, Nara Institute of Science and Technology 8916-5, Takayama, Ikoma, Nara, 630-0192 JAPAN

Abstract Programmers request the code review from reviewers to confirm that there are no bugs in changed source code. However reviewers who are engaged in several review tasks must estimate efficiently how much time or labor (i.e. code review cost) it will take to conduct each review. We aim to suggest the method of estimating code review cost by measuring difference information of source codes. In this paper, as the first step we researched factors to influence code review cost by conducting reviews of various difference files made in the process of developing open source software. As a result, we found that several metrics are main factors — type, amount, complexity, impact.

1 はじめに

不具合や矛盾の早期発見を目的として，ソフトウェア開発の各工程における成果物もしくは中間成果物に対し，作成者または他の開発者がその内容を目視により確認するレビューが実施されている．レビューはソフトウェアの開発工数削減と品質向上を目指す重要な工程と位置付けられ，その有効性が報告されている[3) 15)]．また，レビューの欠陥検出の効率化に向けて様々なレビュー手法が提案され，それらの効率性を単位時間当たりの欠陥検出数などを基準にして実験的に評価する研究も行われてきた[7) 13)]．

レビューの中でも特に，ソースコードを対象としたレビューをコードレビューと呼ぶ．プログラマはソースコードを変更した際，新たに欠陥が混入していないかを確認するためにコードレビューをレビュアに依頼する．しかし，依頼されたレビュアが複数のレビュー業務を抱えていた場合，レビュアはそれぞれのレビュー依頼を処理するのに必要な労力や時間，すなわちコードレビューコストを手際よく見積り，定められた期日内に効率的にレビュー業務を進めていく必要がある．

そこで本研究では，ソースコードの変更内容に着目し，変更内容から計測できる値を元にしたコードレビューコスト見積り手法の提案を目指す．具体的には，変更箇所に関連するソースコードからコードレビューコストの要因となる様々なメトリクスを計測し，それらを組み合わせることでコードレビューコスト見積りモデルを構築する．本稿ではその第一歩として，オープンソースソフトウェアの開発過程で作成された様々な差分ファイルを実際にコードレビューすることにより，コードレビューコストに影響を与えている要因を調査した．

以降，2章ではコードレビューと差分情報について説明し，3章ではコードレビューコスト要因の調査について述べる．4章では考察を述べ，5章では関連研究を紹介する．最後に6章でまとめと今後の課題を示す．

2 コードレビューと差分情報

2.1 コードレビュー

図1にウォーターフォールモデルにおける中間成果物とそのレビューを示す．各工程で実施されるレビューの中でも特に，コーディング工程の成果物であるソースコードを対象としたレビューをコードレビューと呼ぶ．コードレビューでは，プ

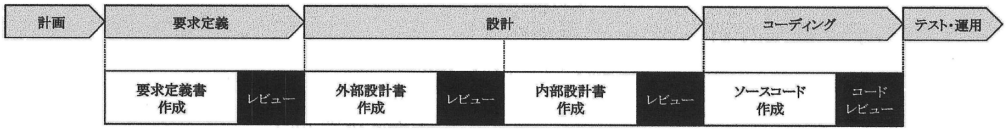

図 1　ウォーターフォールモデルにおける中間成果物とそのレビュー

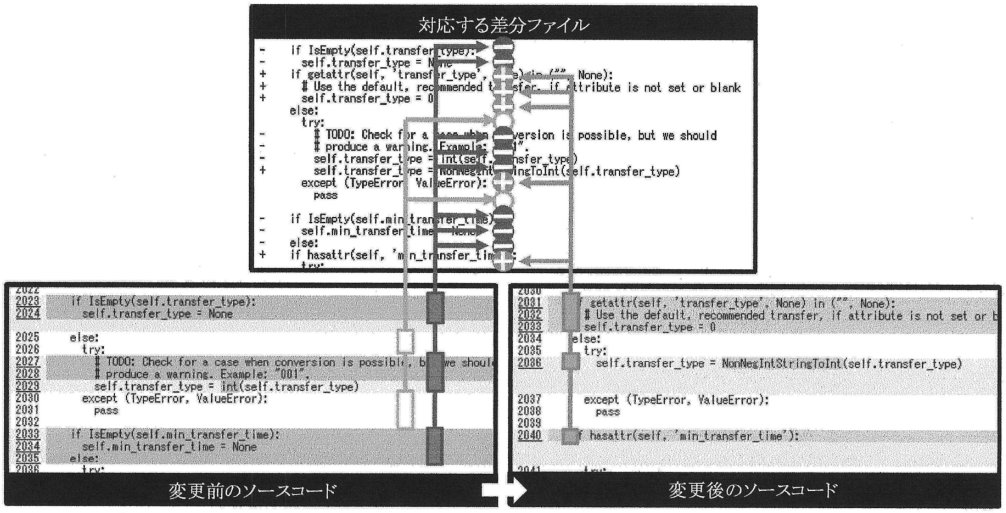

図 2　ソースコードの差分情報

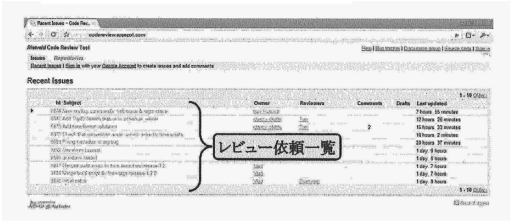

図 3　Rietveld に投稿されたレビュー依頼一覧

ログラマが新たにコードを追加した際や，既存のコードを修正・削除した際に，それらのソースコード中に欠陥が混入していないかを精査する．テストに先立って欠陥を発見することで手戻り工数を減らし開発コストの削減を図ると共に，コーディング規約の統一やソースコードの可読性向上など，ソースコード自体の品質向上にも効果をもたらす．

本研究では，コードレビューに必要な労力や時間を数値的に表したものをコードレビューコストとした．

2.2　差分情報

本研究で述べる差分情報とは，ソースコードの差分情報のことを示し，プログラマがソースコードを変更した際の変更前後のソースコードの相違点のことである．差分情報をファイルに記録したものを差分ファイルまたはパッチファイルと呼ぶ．

図 2 にソースコードの差分情報を示す．図 2 の下側が変更前後のソースコードであり，上側がその変更に対応する差分ファイルである．差分ファイルには，変更前のソースコード（左端が空白文字の行）に対して，新たに追加されたコード（左端が+記号の行）と削除されたコード（左端が-記号の行）が記述される．

また，ある程度の固まりになった変更箇所をまとめてチャンク (Chunk) という単位で呼ぶ．図 2 の変更は 1 チャンクである．

2.3　コードレビューシステム

差分情報とソースコードリポジトリを利用した，多人数が共同でコードレビューを行えるオンラインコードレビューシステムが注目を集めている．様々なプログラミング言語やプラットフォームに対応したものがあり，代表的なものとしては，Google 社で利用されているコードレビューシステム Mondrian のオープンソース版 Rietveld[9] や，仮想化ソフト VMware の開発で利用されている Review Board[1]

表 1　レビューした差分ファイルの詳細

対象システム	Rietveld[9)] http://codereview.appspot.com/
対象期間	2008 年 11 月 17 日～26 日の間に上記システムに投稿された差分ファイル
差分ファイル数	30 Issues (=difference files)
詳細	17 Projects, 174 Files, 643 Chunks, 15758 Lines
使用されている言語	Java(15), Python(8), C++(5), Ruby(2)
OSS 名	Google Transit Data Feed(5), Testability Explorer(5), Openbravo(2), Google Enterprise Connector Manager(3), OneBusAway(2), Caja(2), Protocol Buffers(1), Rietveld(1), Chromium(1), Planeshift(2), 他 (6)

※括弧内の数値は 30 個の Issue の内訳数を示す

などが挙げられる．プログラマは差分ファイルを Rietveld に投稿することでレビュアにレビューを依頼する．Rietveld に差分ファイルを投稿すると，リポジトリに存在するソースファイルと投稿された差分ファイルから変更前後のソースコードが可視化され，レビュアはソースコードの変更点をブラウザ上で確認し，気になる箇所にコメントを付けることでレビューを行うことができる．Rietveld の開発用の公式デモサイトには様々なオープンソースソフトウェアの差分ファイルが次々に投稿され，日々コードレビューが行われている．図 3 に，Rietveld に投稿されたレビュー依頼一覧画面を示す．本研究では Rietveld を利用してコードレビューの実験を進めていく．

3　コードレビューコスト要因の調査

3.1　調査概要

本調査では，差分情報を用いたコードレビューコスト見積り手法の構築に向けた第一歩として，コードレビューコストの要因となる差分情報の特徴を調査した．いくつかのオープンソースソフトウェアの開発過程で作成された様々な差分ファイルを実際にレビューすることにより調査を行った．

本調査では，どのような種類の差分ファイルがあるのかを調べるために，変更の種類を基準にした差分ファイルの分類を行った．さらに，レビューに労力や時間のかかった差分ファイルに対し，その理由を差分ファイルやソースコードから得られる特徴と結び付け，分析し，考察を行った．

また，本調査におけるコードレビューは，変更内容を理解し，変更により新たな欠陥が生じていないかを確認するという設定のもとで行った．

3.2　調査対象

表 1 に，調査対象に選んだ差分ファイルの詳細を示す．調査対象は，コードレビューシステム Rietveld の開発用の公式デモサイトに投稿されていた 30 個の差分ファイルである．Rietveld には，Issue と呼ばれる単位で差分ファイルが投稿される．対象とした差分ファイルの投稿期間は 10 日間である．30 個の差分ファイルは，17 個の OSS プロジェクトのものであり，合計 174 ソースファイル，643 チャンクの変更であった．また，それらの開発に利用されているプログラミング言語はいずれもオブジェクト指向言語であった．

3.3　調査結果

様々な差分ファイルをコードレビューした結果，コードレビューコストに影響を与えている主な要因として，変更の種類，変更量，複雑度，影響範囲が挙げられた．

3.3.1　変更の種類

調査目的の一つである，どのような種類の差分ファイルがあるのかを調べるために，変更の種類を基準に差分ファイルの分類を行った．三段階で変更の種類を分類した．

まず始めに，ソースコードや外部ファイルなどの，ファイル単位の追加，修正，削除である．これらの変更は以下のように定めた．レビューした 174 個のファイルに対して分類を行った結果を表 2 に示す．

- **追加：** プロジェクトに新規のソースファイルを追加
- **修正：** プロジェクトの既存のソースファイルを修正
- **削除：** プロジェクトの既存のソースファイルを削除

次に，チャンク単位の追加，修正，削除で分類を行った．これらの変更は以下のように定めた．また，追加，修正，削除を示すチャンクをそれぞれ図 4，図 5，図 6 に一例として示す．レビューした 643 個のチャンクに対して分類を行った結果を表 3 に示す．

表 2　ファイル単位の変更分類

変更の種類	個数 (全 174 個)
追加	25
修正	148
削除	1

表 3　チャンク単位の変更分類

変更の種類	個数 (全 643 個)
追加	134
修正	495
削除	14

- **追加：** 全ての変更行が追加であるチャンク
- **修正：** 追加と削除の両方の変更行を含むチャンク
- **削除：** 全ての変更行が削除であるチャンク

最後に，クラスの追加，メソッドの修正，フィールドの削除など，具体的な変更内容に着目して分類を行った．その結果を表 4 に示す．

今回レビューした差分ファイルの多くは修正を示すものであった．追加部分，修正部分，削除部分の中で，追加部分をレビューするのに最も時間を要した．その次に修正に時間がかかり，削除はそれほど時間がかからなかった．

3.3.2　変更量

ソースコードの変更量が多いほど，コードレビューに時間がかかった．約 30 行のコードの追加よりも，約 300 行のコードの追加の確認に労力がかかり，変更量の大きさに比例してレビューに要した時間も大きくなる傾向がみられた．ただし，変更部分が難解な場合や，依存関係が強い場合は，たとえ変更量が少なくても時間がかかった．

3.3.3　複雑度

変更部分のソースコードの複雑度が高いほど，コードレビューに時間がかかった．一連の変数の宣言や定義，コメント文，規則性や連続性のある処理などが長々と記述されている単純なコードは確認が比較的容易であったが，複雑な制御構造もしくは難解なアルゴリズムによって記述されている複雑なコードは確認するのに時間がかかった．

3.3.4　影響範囲

変更部分のソースコードに関与している範囲が広いほど，コードレビューに時間がかかった．クラスを変更した場合，変更したクラスが影響を与えているソースコードに，新たに欠陥が混入していないか，不具合が発生していないかを確認する必要があった．また，変更部分で呼び出しているメソッドやモジュールを理解していなければ，変更部分の挙動を理解することができないため，呼び出し先のメソッドを確認する必要があった．

```
@@ -27,7 +27,11 @@
   private SessionManagerInterface sm;
   private ArtifactResolver artifactResolver;
   private AuthzResponder authzResponder;
+  private ArtifactStore artifacts;

+  public BackEndImpl() {
+    artifacts = new ArtifactStore();
+  }
   public void setSessionManager(SessionManagerInterface sm) {
     this.sm = sm;
   }
```

図 4　追加を示すチャンク

```
@@ -1861,9 +1861,10 @@
         p.patchset = patchset
       for c in ps_comments:
         c.draft = False
-        # XXX Using internal knowledge about db package: the key for
-        # reference property foo is stored as _foo.
-        pkey = getattr(c, '_patch', None)
+        # Get the patch key value without loading the patch entity.
+        # NOTE: Unlike the old version of this code, this is the
+        # recommended and documented way to do this!
+        if pkey in ptches:
           patch = patches[pkey]
           c.patch = patch
```

図 5　修正を示すチャンク

```
@@ -52,10 +50,6 @@
   private HttpServletResponse res;
   private JsonRpcServlet servlet;

-  private BeanJsonConverter jsonConverter;
-  private BeanConverter xmlConverter;
-  protected BeanConverter atomConverter;
-
   private final IMocksControl mockControl = EasyMock.createNiceControl();

   private final ByteArrayOutputStream stream = new ByteArrayOutputStream();
```

図 6　削除を示すチャンク

4　考察

4.1　変更の種類

変更部分のソースコードの依存関係が無い場合は，追加，修正，削除の順に変更によって生じるリスクが大きいと言われている [6]．欠陥を含んでいる可能性が高いほど，レビューの必要性も増すため，変更の種類はコードレビューコストに影響を与えていると考えられる．削除に関しては，削除された部分に欠陥が含まれていないかを確認する作業が必要ないため，レビューの時間もそれほどかからなかったと考えられる．

表 4　変更内容に着目した変更分類

変更の種類		個数
追加	クラス追加	18
	クラス宣言	4
	メソッド追加	18
	メソッド宣言	6
	メソッド実装	22
	フィールド追加	8
	その他	11
修正	クラス修正	24
	メソッド修正	47
	フィールド修正	12
	移動	9
	その他	14
削除	クラス削除	6
	メソッド全部削除	5
	メソッド一部削除	5
	フィールド削除	1
	その他	2

4.2　変更量

変更量が多いほど，単純に確認する量が増えたため，コードレビューに時間がかかったと考えられる．本研究における変更量とは，プログラマが既存のソースコードを追加，修正，削除することによって変更された変更前後のソースコードの相対的な変更量のことを示す．変更量は，主に差分ファイルから計測することが可能であり，削除，修正，追加されたソースコード行数，トークン数，メソッド数，クラス数などを計測項目の候補に考えている．

4.3　複雑度

複雑度が高いほど，プログラムの処理内容を理解するのが困難になるため，コードレビューに時間がかかったと考えられる．本研究における複雑度とは，if 文や for 文などの分岐処理による制御構造の複雑度を示す循環的複雑度，式中の演算子と被演算子の複雑度を示す式の複雑度，ポインタや構造体などのデータ構造の複雑度など，ソースコードが難解になっている要因のことを示す．McCabe のサイクロマチック数や Halstead のソフトウェアサイエンスなどを複雑度の計測手法の候補に考えている．また，変更を加えた箇所の複雑度を計測することで，どれだけ複雑な箇所に変更を施したかを，変更を加えた部分の複雑度を計測することで，どれだけ複雑な変更を施したかを計測することができると考えている．

4.4　影響範囲

影響範囲が広いほど，確認する箇所が増えたため，コードレビューに時間がかかったと考えられる．本研究における影響範囲とは，変更部分のソースコードが影響を与えていたり，影響を受けていたりする範囲のことである．変更部分で呼び出しているモジュールやメソッドを確認するという点については，レビュアの知識や経験に左右されるところが大きい．レビュアが呼び出しているモジュールやメソッドを熟知していればその必要性も無くなるためである．影響範囲は，差分ファイル以外の部分も計測する必要があり，クラスやメソッドの依存関係，クラスの継承関係などを計測項目の候補に考えている．

図 7 に，コードレビューコストの要因として挙げた変更の種類，変更量，複雑度，影響範囲をまとめた図を示す．

4.5　計測箇所

上記で挙げた変更量，複雑度，影響範囲の各メトリクスに対し，それらのメトリクスを計測する箇所と重みを変更の種類に応じて変える必要があると考えている．例えば，新規にモジュールを実装し追加した場合は，追加したモジュールを呼び出している部分よりも追加したモジュール自体にレビューの時間がかかった．新しく追加した部分自体に欠陥が含まれていないかを精査することが重要であったからである．この場合，追加したモジュール自体のメトリクスを重点的に計測する必要があるといえる．また逆に，既存のモジュールを削除した場合は，削除したモジュール自体よりも削除したモジュールを呼び出していた部分にレビューの時間がかかった．削除したモジュールを使用していた部分が適切に修正されているかを確認することが重要であったからである．この場合，呼び出していた部分のメトリクスを重点的に計測する必要があるといえる．つまり，追加と削除では重点的にレビューすべき箇所が異なりうるため，これを考慮した計測を行う必要がある．

5　関連研究

ソフトウェアメトリクスから作業量を見積る手法が現在までに提案されている [2) 10)]．例えば，Shepperd ら [10)] は，ソフトウェア開発プロジェクトの規模や業種から，プロジェクトの開発に必要な作業量（開発工数）を見積るための手法を提案している．これらの研究では，プロジェクトの初

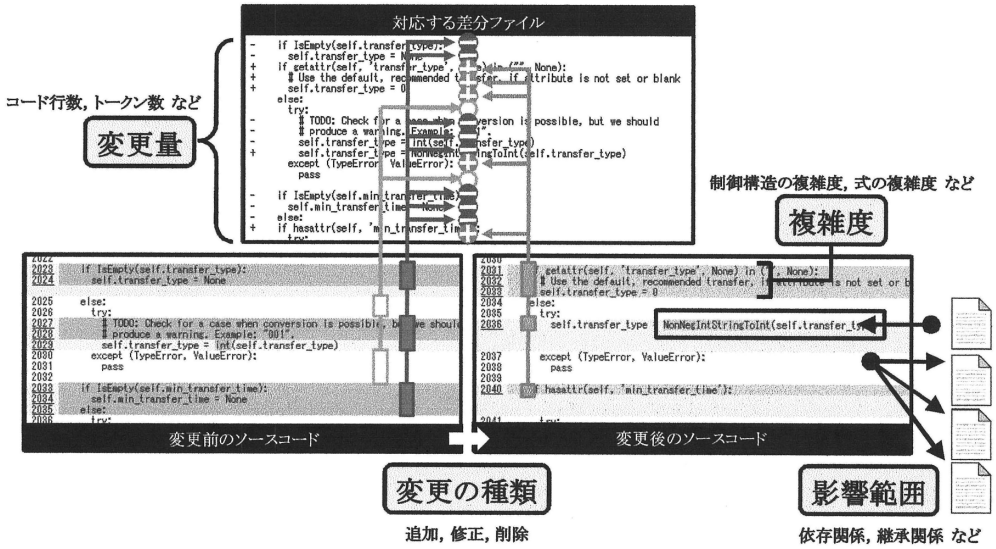

図 7　コードレビューコストの要因と計測

期に得られるメトリクスからプロジェクト全体の大まかな作業量を見積もることに焦点を当てているが，差分情報からコードレビューコストを見積ることに焦点を当てている本研究とは異なる．

レビューの欠陥検出の効率化を目標として，様々なレビュー手法が提案されてきた．レビュー指針を特に与えずにレビュア各々の知識や経験に基づいて自由に欠陥を検出する手法である Ad Hoc Reading に対して，Checklist Based Reading は，過去の経験を基に作成した見つけるべき欠陥をまとめたチェックリストの質問に回答することにより欠陥を検出する手法である [3]．Scenario Based Reading は，欠陥を種類ごとに検出するためにレビュー実施シナリオを用いてレビューする手法である [5]．Perspective Based Reading は，ユーザや開発者やテスト担当者などの各々の観点に基づいてレビューを進めていく手法である [11) 4]．Usage Based Reading は，ユースケースを用いてユーザに大きな影響を与える欠陥から優先的に検出する手法である [14]．Defect Based Reading は，欠陥の種類ごとに専門のレビュアが個別にレビューを行うことにより欠陥検出の重複を減らし効率的に欠陥を検出する手法である [8]．Test Case Based Reading は，あらかじめ作成したテストケースに基づいて成果物をレビューする手法である [7]．Regression Test Based Reading は，見逃した場合に回帰テスト件数が大きくなると思われる欠陥を優先的に検出する手法である [12]．さらに，これらの効率性を単位時間当たりの欠陥検出数などを基準にして実験的に評価する研究も行われてきた [7) 13]．しかし，これらの手法はレビューの効率化を目標とした研究であり，見積りに関しては特に考慮されていない．

6　おわりに

本稿では，差分情報を用いたコードレビューコスト見積り手法の提案を目指すための第一歩として，オープンソースソフトウェアの開発過程で作成された様々な差分ファイルをコードレビューすることにより，コードレビューコストに影響を与えている要因となる差分情報のメトリクスを調査した．その結果，変更の種類，変更量，複雑度，影響範囲が主な要因として挙げられた．

今後は，数名の被験者を対象に，実験向けに作成したソフトウェアの開発過程の差分ファイルをコードレビューしてもらい，レビューに要した時間やレビューした箇所などを計測する．そして，差分情報から計測した各メトリクスに対し，被験者実験の結果を考慮した重み付けを行い，組み合わせることでコードレビューコスト見積りモデルを構築する．最終的には，完成した見積りモデルをより大規模なプロジェクトの差分情報に適用することにより，モデルの見積り能力を評価したいと考えている．

謝辞

本稿を執筆するにあたり，多くの御指摘・御助言を頂いた奈良先端科学技術大学院大学情報科学研究科の亀井靖高氏に深く感謝致します．また，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．

参考文献

1) Review Board: reviewboard - Google Code. `http://code.google.com/p/reviewboard/`.

2) Boehm, B. W.: *Software Engineering Economics*, Prentice Hall (1981).

3) Fagan, M. E.: Design and Code Inspection to Reduce Errors in Program Development, *IBM Systems Journal*, Vol. 15, No. 3, pp. 182–211 (1976).

4) Laitenberger, O., DeBaud, J. and Runeson, P.: Perspective-Based Reading of Code Documents at Robert Bosch GmbH, *Information and Software Technology*, Vol. 39, No. 11, pp. 781–791 (1997).

5) 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).

6) McDonald, M.: *Practical Guide to Defect Prevention*, Microsoft Pr (2007).

7) 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).

8) Porter, A. A., Votta, L. G. and Basili, V. R.: Comparing Detection Methods for Software Requirements Inspection - A Replicated Experiment, *IEEE Transaction on Software Engineering*, Vol. 21, No. 6, pp. 563–575 (1995).

9) Rietveld: rietveld - Google Code. `http://code.google.com/p/rietveld/`.

10) Shepperd, M. and Schofield, C.: Estimating Software Project Effort Using Analogies, *IEEE Trans. Softw. Eng.*, Vol. 23, No. 11, pp. 736–743 (1997).

11) 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).

12) 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).

13) 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).

14) Thelin, T., Runeson, P. and Wholin, C.: An Experimental Comparison of Usage-Based and Checklist-Based Reading, *IEEE Transaction on Software Engineering*, Vol. 29, No. 8, pp. 687–704 (2003).

15) Wiegers, K. E.: *Peer Reviews in Software - A Practical Guide*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002).