Vol.2012-SE-176 No.12
Vol.2012-EMB-25 No.12
2012/5/22

# Android Open Source Projectを対象としたパッチレビュー活動の調査

濵﨑 一樹[1,a)]　藤原 賢二[1,b)]　吉田 則裕[1,c)]　Raula Gaikovina Kula[1,d)]　伏田 享平[1,†1,e)]
飯田 元[1,f)]

**概要**：ソフトウェア開発におけるレビューとは，設計文書やソフトウェアのソースコードを人が読み，設計の誤りやコードの記述ミス，コーティングルールの違反などの問題がないかを検査するプロセスのことである．レビューにより欠陥の早期発見，修正を行うことができ，欠陥のおよそ 60 %を発見可能であることが報告されている．近年，レビュー管理システムが登場し，レビューの追跡，管理，レビューアの割り当てなどが実現できるようになった．本研究では，Android Open Source Project（AOSP）が採用しているレビュー管理システム Gerrit に蓄積されたレビュー履歴の分析を行う．Gerrit に蓄積された，パッチ投稿者やレビュー実施者に関する情報を利用し，セルフレビュー（パッチ投稿者とレビュー実施者が同一のレビュー）の割合を分析した．調査の結果，全体のうち 15.3 %のレビューがパッチを投稿した本人によるセルフレビューであることが判明した．

**キーワード**：レビュー，ソフトウェアインスペクション，リポジトリマイニング

# An Analysis of Patch Reviews in the Android Open Source Project

Kazuki Hamasaki[1,a)]　Kenji Fujiwara[1,b)]　Norihiro Yoshida[1,c)]　Raula Gaikovina Kula[1,d)]
Kyohei Fushida[1,†1,e)]　Hajimu Iida[1,f)]

***Abstract:*** Software review is a process to discover violations of coding rules, and defects involved in design documents and source code. It promotes to discover and correct defects in early stage of software development. It is reported that 60% of defects can be discovered by software review. Recently, several code review systems have been proposed for tracking, management and assign reviews. In this study, we analyze the review history of Android Open Source Project. In our analysis, we use information on developers who review or submit patches, and calculate the ratio of self-reviews, completed by one developer who submitted the patch. The analysis shows that 15.3% of reviews were self-reviews.

***Keywords:*** Peer review, Software Inspection, Mining Software Repositories

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[†1] 現在，株式会社エヌ・ティ・ティ・データ
Presently with NTT DATA CORPORATION
[a)] kazuki-h@is.naist.jp
[b)] kenji-f@is.naist.jp
[c)] yoshida@is.naist.jp
[d)] raula-k@is.naist.jp
[e)] kyohei-f@is.naist.jp
[f)] iida@itc.naist.jp

## 1. はじめに

ソフトウェア開発におけるレビューとは，設計文書やソフトウェアのソースコードを人が読み，設計の誤りやコードの記述ミス，コーティングルールの違反などの問題がないかを検査するプロセスのことである．レビューにより欠陥の早期発見，修正を行うことができ，欠陥のおよそ 60 %を発見可能であることが報告されている [1].

オープンソースソフトウェア（OSS）の開発において，機能追加や不具合修正が行われる場合，直接ソースコードが編集されることは少ない．多くの場合，開発者が作成したパッチ（ソースコードの差分）をもとに行われる．機能追加や不具合修正は投稿されたパッチをレビューし問題が無いことを確認した上で，パッチをソースコードに反映することで行われる．OSS では多数の開発者がパッチを投稿しており，レビューはソースコードの品質を保つ上で重要な役割を果たしている [2]．多くの OSS プロジェクトではパッチレビュープロセスを規定している．パッチレビュープロセスでは，パッチの妥当性に関する議論や動作検証といったパッチレビューにおいて実施すべき作業をどのように行うべきか，その手順や作業内容が規定されている．一般的な OSS の場合，開発者が作成したパッチはプロジェクトのメーリングリスト（ML）やバグ管理システム（BTS）へ投稿され，レビューが行われる．たとえば OSS プロジェクトである Linux の場合，メール中にパッチとその解説を記述し，ML に投稿する．その後，他の開発者によって ML 上でレビューが行われる．問題がなければソースコードを直接編集する権限を持った開発者によりパッチの内容がソフトウェアに反映される．欠陥が見つかるなどしてパッチが拒否された場合，開発者はパッチの内容を更新，再投稿し，再び ML 上で議論が行われる[*1]．

ML 上でのレビューの場合，レビューアは明確に割り当てられない，ML を購読している開発者が自分でレビューを行いたいパッチを選択し，レビューを行っている．そのためパッチを投稿したメールへの返信がない場合，それが単に無視されているだけなのか，レビューにおいて問題が発見されなかったため意見が述べていないだけなのか判断できない [3]．また，ML 上に分散したパッチの情報を追跡，管理するのは難しいため，多くのパッチが無視された状態になっていることがある[*2]．

このように ML を用いたメールベースでのパッチレビューでは，その追跡，管理に問題があった．一方近年では，Gerrit[*3]や，Rietveld[*4]，Review Board[*5]といったレビューを管理，支援するためのシステムが登場し，使われ始めた．これらのレビュー管理システムの特徴としてレビューアの割り当てと役割の明確化が実現できることが挙げられる．レビュー管理システムではレビューアの割り当てがレビュー前に行われ，レビューの結果をスコアといった数値やコメントという形で記録する．これによりそれぞれのレビューに対するレビューアが明確になる．

本研究では，Android Open Source Project（AOSP）が採用しているレビュー管理システム Gerrit に蓄積されたレビュー履歴の分析を行う．Gerrit に蓄積された，パッチ投稿者やレビュー実施者に関する情報を利用し，セルフレビュー（パッチ投稿者とレビュー実施者が同一のレビュー）の割合を分析した．その結果，下記に述べる特徴が観測された．

- 全レビューのうち 15.3% はセルフレビューが行われており，そのうち 80%は 1 日以内にレビューが完了していた．
- AOSP 内のそれぞれのプロジェクトカテゴリごとのセルフレビュー率と，そのプロジェクトに関わる開発者数との間には強い負の相関があることが分かった．例えば `device` カテゴリは 31 人の開発者がおり，セルフレビュー率は 53.6%であった．一方で，`platform` カテゴリは 1,088 人の開発者がおり，セルフレビュー率は 15.3% であった．

## 2. Android におけるレビューシステム

AOSP では Gerrit を利用してパッチレビューを行なっている．Gerrit はバージョン管理システムである Git と密接に連携して動作する．Gerrit はソースコードの変更差分を並べて簡単に表示することができ，レビューアはソースコード中にインラインコメントを残すことができる[*6]．このように Gerrit はソースコードのレビューに関して，開発者間のより効率的な共同作業を可能にする．

AOSP へ投稿されたパッチは全て Gerrit 上でのレビューを受けることが決められている．投稿されたパッチを承認するかどうかは，全て Gerrit 上で議論が行われる．レビューの結果，承認されたパッチは，システムが自動的に Git へ反映する．

AOSP のレビュー承認までのレビュープロセスを**図 1** に示す．開発者は，はじめに AOSP のソースコードリポジトリをローカル環境にコピーし，その上で欠陥修正や機能追加のためにパッチを作成する．パッチをソースコードリポジトリ上へコミットすると，Gerrit はコミットされたパッチに対してそれぞれレビューを自動的に作成する．その後，それぞれの開発者が割り当てられた役割に基づきレビューや検証作業を行い，パッチを承認するか議論を行う．

AOSP は開発者に対して様々な役割が用意されており，レビュープロセスに関係のある役割は次のようになる．

**Contributor** Gerrit に登録されている開発者全体．AOSP のソースコードに貢献する人．

**Owner** パッチを投稿した人．

**Author** パッチのソースコードを実際に書いた人．多くの場合は Owner と同一人物になる．

[*1] How to Get Your Change Into the Linux Kernel or Care And Operation Of Your Linus Torvalds `http://www.kernel.org/doc/Documentation/SubmittingPatches`
[*2] How to Contribute Patches to Apache `http://httpd.apache.org/dev/patches.html`
[*3] `http://code.google.com/p/gerrit/`
[*4] `http://code.google.com/p/rietveld/`
[*5] `http://www.reviewboard.org/`
[*6] Gerrit Code Review `https://android-review.googlesource.com/`

**Approver**　パッチを承認するかどうかを決定する．

**Verifier**　実際に開発者の環境でパッチを適用させ，正常な動作をするか検証する．

パッチの承認に少なくとも一人の Approver による承認と Verifier による検証が必要である．Approver と Verifier のいずれかがパッチを拒否した場合は，パッチをソフトウェアに反映することができない．承認と検証が行われたパッチはソフトウェアへ反映される．パッチに問題が指摘されれば開発者はパッチを更新し，再びレビューが行われる．

## 3. 分析の観点

複数人のレビューアによるレビューが効果的であると言われている．Raymond は「目玉の数さえ十分あれば、どんなバグも深刻ではない」，つまり十分なレビューアがいれば、ほとんどの欠陥は発見でき、すぐさま修正できると主張している [4]．その点で，レビューはパッチの作成者とは別の開発者によってレビューされるべきものである．

本研究ではレビューアの割り当てと，パッチの作成者が直接レビューを実施しているセルフレビューに着目し分析を行った．具体的には以下の観点で調査した．

**観点 1:レビューアのレビューに対する貢献度**　レビューに割り当てられたレビューア数と，その中で実際にレビューをそのくらいのレビューアがレビューを行ったかについて調査した．

**観点 2:セルフレビューの発生頻度**　AOSP 内でセルフレビューがどの程度行われているか，その頻度を調査した．

**観点 3:セルフレビューの処理期間**　セルフレビューの場合と，そうでない場合でレビューの処理にかかる時間を比較した．

**観点 4: セルフレビューの発生箇所**　セルフレビューとコミュニティとの関係を調べるために，プロジェクトを大きなカテゴリに分け，それぞれのセルフレビューの発生状況を調査した．

**観点 5:レビューアの割り当て方法**　レビューに割り当てるレビューアをどのような方法で決めているかを調査した．

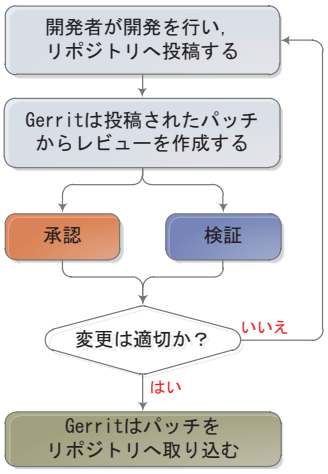


**図 1**　AOSP におけるパッチ承認までのレビュープロセスの概観

**Fig. 1**　Simplified Process for AOSP Review in the case of accepting a patch[*7]

[*7] http://source.android.com/source/life-of-a-patch.html

## 4. 分析対象プロジェクト

### 4.1 開発者コミュニティの構成

AOSP は Google が主導で運営しているプロジェクトであり，Approver と Verifier は Google の雇用者によって選ばれる．次期メジャーバージョンは Google 社内のプライベートなリポジトリ上で開発される．十分なテストが行われ安定すると，Google は次期バージョンのプラットフォームを公開する．このように，AOSP は他の OSS プロジェクトと比較して特殊であると言える．

AOSP のコミュニティ構成について調査を行った．ユーザのメールアドレスのドメイン部よりユーザの所属する組織を分類し，1024 の開発者と 173 の組織を得た．**図 2** に Contibutor が所属する組織の人数比を示す．

`android.com` および `google.com` は Google の雇用者による組織であり，全体の 17 %を占めている．他は `ti.com` や `sonyericsson.com` のようなデバイスやチップセットメーカーに所属する開発者や個人の開発者などである．

図 2 の `codeaurora.org` は携帯電話や無線システム向け OSS を推進する非営利法人であり，AOSP を Qualcomm のチップセットへ移植を行っている．このように多くのメーカーや個人の開発者が AOSP に活動を行っている．

パッチの作成者の割合は Google が 43.4%で他の開発者が 56.6%である．AOSP はコミュニティからのパッチを多く受け付けており，Google 主導のプロジェクトであるがコミュニティとの活発なコミュニケーションが行われている．

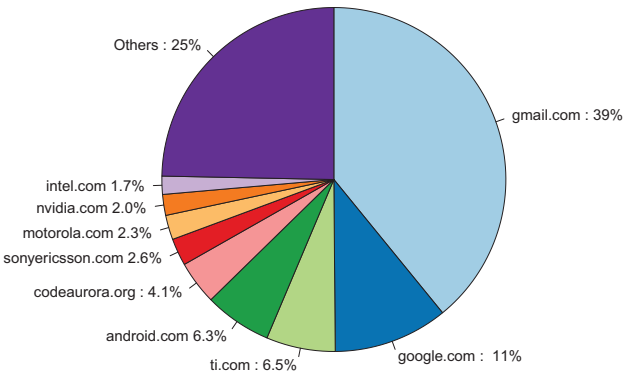


**図 2**　開発者の所属組織

**Fig. 2**　Contributor organizations

表 1　レビュー状態の内訳
**Table 1**　Review Status

| Status | #Reviews |
|---|---|
| Open | 1,608 |
| Merged | 7,120 |
| Pending | 1 |
| Abandoned | 2,903 |

表 2　開発者の役割
**Table 2**　Roles of Contributors

| Role | #Developers |
|---|---|
| Approver | 103 |
| Verifier | 91 |
| Committer | 675 |
| Author | 772 |
| Contributor | 1,204 |

### 4.2　レビュー履歴

Gerrit からレビュー履歴を抽出し，11,632 件のレビューを分析した．分析対象としたレビュー履歴の期間は 2008 年 10 月 21 日から 2012 年 1 月 27 日である．ただしサーバーダウンにより 2011 年 7 月 6 日から 2012 年 1 月 11 日までの履歴は含まれていない[*8]．

Gerrit ではそのレビューが進行中か否かといった状態が管理されている．状態は *Open*，*Merged*，*Submitted, Merged Pending*，*Abondoned* の 4 種類の状態がある．データ取得時におけるレビューの状態の内訳を**表 1** に示す．

**Open**　レビュー中の状態

**Merged**　レビューが承認され，ソースコードリポジトリへ取り込まれた状態

**Abandoned**　何らかの理由により，パッチが拒否された状態

**Submitted, Merge Pending**　レビューは承認されているが，リポジトリには取り込まれていない状態．リポジトリ上でのソースコードの競合などの理由により，Gerrit が自動的にパッチを取り込むことができない場合がある．レビュープロセス自体は正常に完了しているため，本研究ではこの状態を Merged と同じ状態として扱う．

**表 2** は Gerrit 上での活動ログからそれぞれの役割の人数分布を示した物である．

## 5.　調査結果

### 5.1　観点 1:レビュアーのレビューに対する貢献度

Liang と Mizuno らは *Assigned reviewer* と *real reviewer* の 2 つの用語を定義した [5]．本研究ではこれと同様に *Assigned reviewer* をレビューに招待された開発者，*Real reviewer* をレビューに招待された後，実際にレビューを行う，もしくはコメントを残すなどの活動を行ったレビューアと定義する．**図 3** は完了したレビュー (merged および abandoned 状態のレビュー) における Assigned reviewer と Real reviewer の分布を示している．

10,024 件の完了したレビューにおいて Assigned reviewer の平均値は 1.90 人, Real reviewer は 1.29 人であった．Real reviewer が 1 人以下であったレビューは，完了したレビューのうちの 49.1 %を占めた[*9]．多くのマージ済みレビューでは Approver と Verifier は同一であった．

Gerrit では，どの Contributor もパッチのコードレビューを行い，コメントすることができる．コードレビューはパッチの承認には必要ないが，Approver がパッチを取り込むかどうか判断する際にコードレビューの結果やコメントを参考にすることができる．**図 4** は完了したレビューにおけるコードレビューアの分布を示したものである．この図には Approver は含まれていない．83.1% のレビューにはコードレビューアがいないことから Approver はコードレビューアのコメントを参考にせず，パッチを承認することが多いと考えられる．

### 5.2　観点 2:セルフレビューの発生頻度

我々は Author と Owner，Approver，Verifier が同一である場合のレビューをセルフレビューと定義した．パッチを取り込むには Approver による承認と Verifier による検

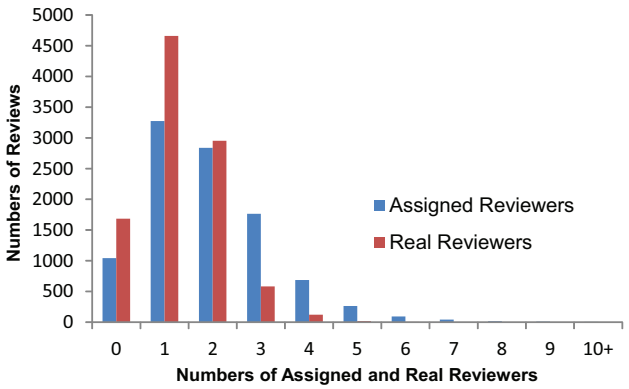


**図 3**　完了したレビューにおける Assigned Reviewer と Real Reviewer の分布
**Fig. 3**　Distribution of Assigned and Real Reviewers in closed reviews

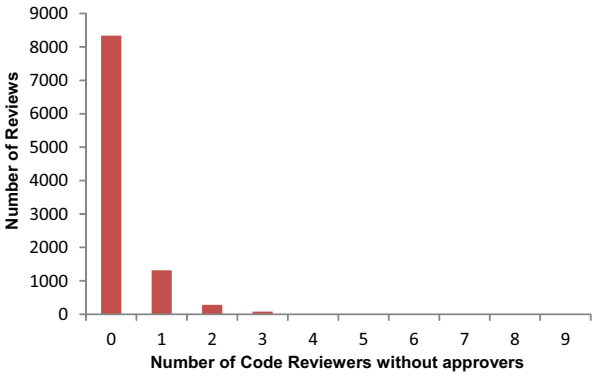


**図 4**　Approver 以外のコードレビュアの分布
**Fig. 4**　Distribution of Code Reviewers in closed reviews (exclude approvers).

[*8] `http://groups.google.com/group/android-contrib/browse_thread/thread/ca2bb0b4a4d5fc89`

[*9] Abandoned になったパッチでは Owner がレビューアを割り当てずに自らパッチを取り下げる例があったため，Real Reviewer が 0 になっているレビューがある．

証が必要である．セルフレビューを行うことができるのは両方の権限を持っている開発者に限られる．

我々はすべてのマージ済みレビューの中から，セルフレビューを抽出した．その結果 15.3% (108 件) のレビューがセルフレビューであることが判明した．

**図 5** はセルフレビューと非セルフレビューでの Assigned Reviewer 数を示したものである．セルフレビューのうち 66.0% のレビューは割り当てられたレビューアが一人だけ，つまり Owner がレビューアとして割り当てられただけであった．セルフレビューを行ったレビューアは他のレビューアをはじめから必要としていなかったことが分かる．

### 5.3 観点 3:セルフレビューの処理期間

レビューアが集まらなかったためにセルフレビューが生じているのか，それとも開発者がレビューを意図して避けているかを判別するために，レビュー期間の調査を行った．レビューアが集まらなかったために，結果としてセルフレビューになっているのであれば，しばらくレビューが行われることを期待して待機するため，レビュー期間は長いと考えられる．逆に，開発者がレビューを避けている場合は，レビュー期間は短いと考えられる．**図 6** にセルフレビューによるレビューとそれ以外のもので期間の分布を示した．

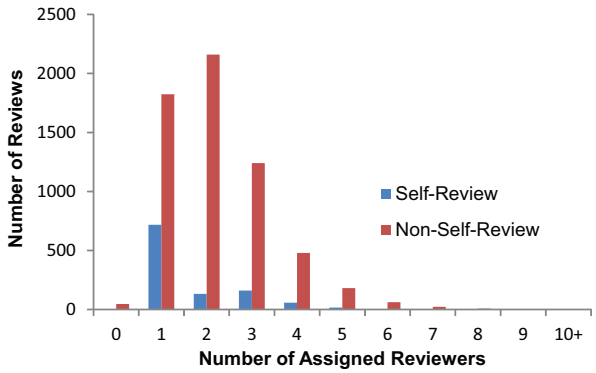


**図 5** セルフレビューと非セルフレビューでの Assigned reviewer 数
**Fig. 5** Assigned Reviewers in Self and Non-Self-Review

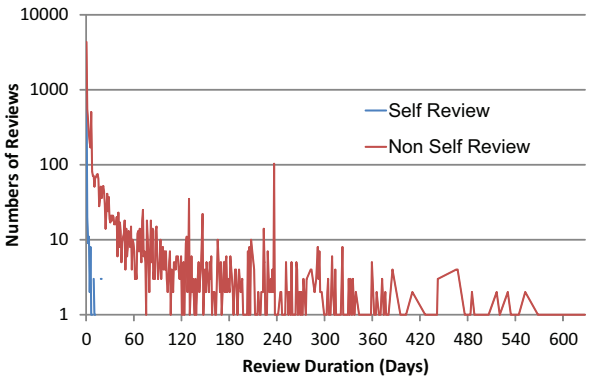


**図 6** レビュー期間
**Fig. 6** Review Duration in Closed Reviews

セルフレビューは 1 日以内に処理が終わる傾向にある．1 日以内に処理の終わってるセルフレビューの平均処理時間は 0.50 日であった．一方，非セルフレビューの平均期間は 28.1 日であり，1 日以内に処理が終わっているものは 48.5%であった．このことからセルフレビューは多くの場合，開発者がレビューを避けることの理由で行われていることが考えられる．

AOSP におけるセルフレビューは権限を持った主要な開発者の手によって行われており，必ずしも品質の低下を招くとは言えない．Apache Web Server[*10]の場合，主要な開発者であれば，小規模の変更に限られるが，先にソースコードリポジトリにコミットしてからレビューを受けるというプロセスが認められている [3]．

### 5.4 観点 4: セルフレビューの発生箇所

表 3 はプロジェクトごとの特徴をまとめたものである．我々はそれぞれのプロジェクトカテゴリごとに，規模と開発者コミュニティの違いに着目し，セルフレビューにおける差違を確認した．**図 7** と**表 3** はそれぞれのプロジェクトカテゴリごとのセルフレビュー比と開発者数を示したものである．

AOSP はソースコードリポジトリ上にある 189 のプロジェクトから構成される．それぞれのプロジェクトは階層構造で管理され，`toolchain` と `device`, `kernel`, `platform` の 4 つの大きなカテゴリに分けることができる．`toolchain` はビルドツールであり，`device` は特定デバイス向けの実装，`kernel` は Android が用いている Linux カーネル，`platform` は Android のプラットフォームに関連するプロジェクトであり，最も活発なカテゴリである．

`toolchain` と `device` カテゴリは開発者とレビューアが少なく，セルフレビュー率が高い．これとは対照的に `kernel` と `platform` カテゴリは多くの開発者とレビューアが関わっており，セルフレビュー率は低くなっている．Contributor 数とセルフレビュー率には相関係数 −0.72 の強い負の相関があった．この結果よりプロジェクトごとの開発者数が増えるとレビューが活発に行われ，結果としてセルフレビューが減ることが示唆される．

### 5.5 観点 5:レビューアの割り当て方法

レビューの被招待者や実施者が一部の開発者に偏っているかどうか調査した．レビューアごとのレビュー件数について，上位 10 名を**表 4** に示す．累計で 42,250 人のレビューアがレビューに招待され，そのうち上位 10 人のレビューアが全体の 50.3 %を占めていた．実際にレビューを行った人数は累計で 25,754 人で，そのうち上位 10 人のレビューアが全体の 60.8 %を占めていた．このように，活

[*10] `http://httpd.apache.org/`

表 3　プロジェクトカテゴリごとの開発者数とセルフレビュー比
**Table 3**　Self review ratio and # of developers for each project category

| Project | #Reviews | #Contributors | #Unique Domains | Self-Review Ratio |
|---|---|---|---|---|
| toolchain | 79 | 15 | 6 | 67.2% |
| device | 43 | 31 | 10 | 53.6% |
| kernel | 3,520 | 274 | 34 | 13.2% |
| platform | 7,990 | 1,088 | 171 | 15.3% |

表 4　レビューアごとのレビュー数，レビュー招待数
**Table 4**　Number of reviews and invited reviews for each reviewers

| Reviewer | #Invited Reviews | #Real Reviews |
|---|---|---|
| jbq@google.com | 5,151 | 3,836 |
| digit@android.com | 3,118 | 2,439 |
| xav@android.com | 2,925 | 1,726 |
| ccross@android.com | 1,865 | 1,477 |
| pickgr@pv.com | 1,426 | 1,383 |
| btmura@android.com | 1,469 | 1,305 |
| ralf@android.com | 1,995 | 1,133 |
| tnorbye@google.com | 1,432 | 947 |
| malchev@google.com | 907 | 705 |
| lockwood@android.com | 964 | 700 |
| Subtotal | 21,252 | 15,651 |
| All | 42,250 | 25,754 |

発に活動している特定のレビューアにだけレビューが集中している．

セルフレビューが行われる原因の 1 つとして，投稿されたパッチを担当可能な開発者を探すことが困難であることが考えられる．レビューを担当可能な開発者を探すために使用できる情報は，レビュー履歴やパッチ対象のソースコードの編集履歴のみであり，これらから，十分な数のレビューアを探すことは困難であると考えられる．

## 6. 関連研究

OSS プロジェクトにおけるパッチの取り扱いに関しては，これまでに多くの研究がなされている．Bird らはメール中に含まれているパッチの投稿とその承認を特定する手法を提案している [6]．また Phannachitta らは，投稿されたパッチのうち一部分だけが採用されるケースに着目し，そのような採用箇所を抽出する手法を提案している [7]．これらの手法はメールやバグ管理システムのデータを対象としている．そのため，パッチ特定することは可能であるが，これらのパッチが採用される過程を分析するには不向きである．レビュー内容に関する分析としては，レビュー中の欠陥の発見回数に着目した研究が多く行われてきた．しかし，このような分析はレビューの効果を測定するという観点からは限定された評価しか行えない．Mantyla らは欠陥の発見回数ではなく，欠陥の種類に着目して分析を行った [8]．また，Fujita らはパッチレビュープロセスの効率化に向けた試みとして，コミッタ と非コミッタ間でのパッチ編集や, レビューに要する時間について比較，分析を行っている [9]．本研究ではレビュー管理システムである Gerrit を分析対象とした．レビュー管理システムを分析対象とした研究としては，Liang らの研究が挙げられる [5]．Liang らはレビュー管理システムである Rietvelt を利用している Chromium プロジェクトのパッチレビュー活動を分析している．この分析では，レビューに招待される開発者の傾向やレビューア数とレビューの品質との関係について分析している．

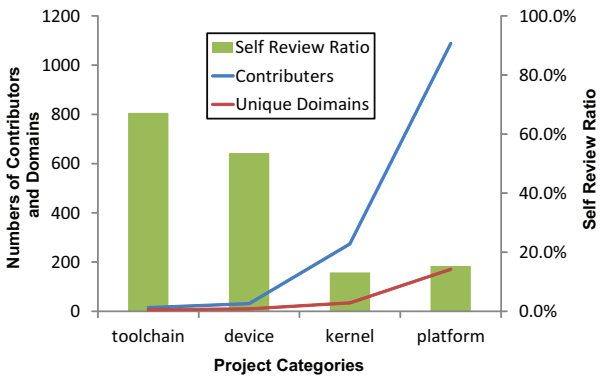


図 7　カテゴリ別のセルフレビュー比
**Fig. 7**　Self-Review Ratio for Each Category. Small-scale community is insufficient for frequent peer-reviews.

## 7. おわりに

本研究では AOSP におけるパッチレビュー活動の実態を調査した．パッチレビューは OSS 開発における品質維持のための重要な活動の一つである．しかしながら，全体の 15.3%のレビューでセルフレビューが，行われていることがわかった．

本研究で行った分析では，レビューのみを対象とし，プロダクトの品質については分析を行なっていない．今後，セルフレビューがプロダクトの品質に与える影響など，レビュープロセスとプロダクトの品質の関係を明らかにしたい．また，セルフレビューを減らすためには，レビューアを探すための支援をレビュー管理システムに用意する必要があると考えられる．今後，適切なレビューアを自動推薦する手法を考案したい．

**謝辞**　本研究は，日本学術振興会 科学研究費補助金 基盤研究（C）（課題番号:22500027）の助成を得た．

## 参考文献

[1] Boehm, B. and Basili, V. R.: Software Defect Reduction Top 10 List, *Computer*, Vol. 34, pp. 135–137 (2001).
[2] Rigby, P. C. and Storey, M.-A.: Understanding broadcast based peer review on open source software projects, *Proc. of ICSE 2011*, pp. 541–550 (2011).
[3] Rigby, P. C., German, D. M. and Storey, M.-A.: Open source software peer review practices: a case study of the apache server, *Proc. of ICSE 2008*, ICSE '08, pp. 541–550 (2008).
[4] Raymond, E. S.: *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media (1999).
[5] Liang, J. and Mizuno, O.: Analyzing Involvements of Reviewers Through Mining A Code Review Repository, *Proc. of IWSM/MENSURA2011*, pp. 126–132 (2011).
[6] Bird, C., Gourley, A. and Devanbu, P.: Detecting Patch Submission and Acceptance in OSS Projects, *Proc. of MSR 2007*, No. 26 (2007).
[7] Phannachitta, P., Jirapiwong, P., Ihara, A., Ohira, M. and Matsumoto, K.: An Analysis of Gradual Patch Application: A Better Explanation of Patch Acceptance, *Proc .f IWSM-MENSURA 2011*, pp. 106 –115 (2011).
[8] Mantyla, M. V. and Lassenius, C.: What Types of Defects Are Really Discovered in Code Reviews?, *IEEE Trans. Softw. Eng.*, Vol. 35, No. 3, pp. 430–448 (2009).
[9] Fujita, S., Ohira, M., Ihara, A. and Matsumoto, K.: An Analysis of Committers Toward Improving the Patch Review Process in Oss Development, *Proc. of ISSRE 2010*, pp. 369–374 (2010).