NAIST-IS-MT1551085

修士論文

コードレビューでの意見の衝突要因に基づく合意形成プロセスの分析

平尾 俊貴

2017 年 3 月 16 日

奈良先端科学技術大学院大学
情報科学研究科 情報科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士（工学）授与の要件として提出した修士論文である.

平尾 俊貴

審査委員：

松本 健一 教授　（主指導教員）
飯田 元 教授　（副指導教員）
伊原 彰紀 助教授　（副指導教員）

コードレビューでの意見の衝突要因に基づく合意形成プロセスの分析*

平尾 俊貴

内容梗概

本論文では，オープンソースソフトウェアプロジェクトにおけるコードレビューで発生する検証者間の合意形成プロセスを分析する．コードレビューは，修正されたソースコードを検証者が精読して品質を評価する手法である．検証者は建設的なフィードバックをソースコード作成者に与えるために，ソースコードが統合されるべきか否かを評価するスコアを投票する．複数の検証者がコードをレビューする際，検証者間で意見が衝突する．本論文では，意見の衝突がどのくらい発生して合意形成に至るのかを分析した結果，検証者間で意見が衝突する議論が 26% 存在することを確認した．特に，強い肯定意見と否定的意見が衝突した議論のうち，54% は検証者間で合意に至ることを確認した．また，合意形成に至った議論では，どのように意見の衝突を解決するのかを分析した結果，54% ± 5% はソースコードを再度修正せずに検証者間の議論だけで合意形成に至ったことを明らかにした．オープンソースソフトウェアプロジェクトでは，意見の衝突が頻繁に発生しているが，全ての議論が合意形成に至るとは限らないことを認識する必要がある．意見の衝突を解決する際，再度修正が必要であるのか，もしくは議論のみで意見の衝突を解決できるのかを適切に見極めることで，意見が衝突した議論を効率的に合意形成へ導くことが期待される．

キーワード

オープンソースソフトウェア，コードレビュー，合意形成，検証者，パッチ

*奈良先端科学技術大学院大学 情報科学研究科 情報科学専攻 修士論文, NAIST-IS-MT1551085, 2017 年 3 月 16 日.

An Analysis of Consensus Building Process based on Factors of Contentious Opinions in Code Review*

Toshiki Hirao

Abstract

I analyze a consensus building process among reviewers through a code review in open source software projects. Code review is a broadly adopted software quality practice, where reviewers critique each others' source codes. Reviewers are asked to provide a score to indicate whether the source code should be integrated into projects. Reviewers may disagree with each other, yielding contentious discussions. In this thesis, we analyze how many contentious discussions occur and how much these discussions eventually end up to an agreement. I found that 26% of discussions that receive multiple review scores are contentious, and contentious discussions that elicit strong opinions both in favour of and against integration have an integration rate of 54%. I analyze how reviewers address contentions. I found that $\mathbf{54}\% \pm \mathbf{5}\%$ of highly contentious discussions that are eventually integrated are indirectly addressed. OSS organizations should be aware of the potential for contention in discussions, and not all of contentions eventually end up to an agreement. I suggest that an appropriate decision that a contention should be addressed either indirectly or directly helps reviewers to lead to an agreement effectively.

Keywords:

Open Source Software, Code Review, Consensus Building, Reviewer, Patch

*Master's Thesis, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1551085, March 16, 2017.

目次

図目次

第 1 章　序章

1.1　研究背景

現在，ソフトウェアは社会生活の基盤となっている．特に，ソフトウェアの導入コストや運用保守コストの削減を目的として，誰でも無償で使用できるオープンソースソフトウェア（以降，OSS）を自社システムの一部に利活用する企業が増加している．多くの利用者が OSS を導入するようになったため，利用者から膨大な不具合が発見されている．OSS の社会的重要性が高まっている現在では，OSS 開発者は報告された不具合を迅速に解決することが求められている．

ソフトウェア開発現場では，報告された不具合を解決するために，ソースコードを改変する．そして，改変されたソースコード（以降，パッチ）を対象に，不具合が正しく修正されているか否かを検証するために，コードレビューを実施する．コードレビューとは，パッチ作成者以外の開発者（以降，検証者）がパッチの内容を精読して，パッチの正当性や無謬性を検証する方法である．コードレビューでは，複数人の検証者が実施して，パッチの品質を評価する議論を進める．

複数人で協調的に実施されるコードレビューの議論では，検証者間で意見の衝突が発生する．意見の衝突が発生した議論を解決することは容易ではない [13]．さらに，意見の衝突によって議論が進められたパッチは合意に至らずに破棄される場合もある．しかし，既存研究では，どのような議論は合意形成に至るのかを分析されていないため，検証者間で効率的に合意形成に至るプロセスを提案できていない．

1.2　本研究の目的とリサーチクエスチョン

本研究では，合意形成に至る議論の内容を理解することが目的である．本研究の目的を達成するために，３つのリサーチクエスチョンを設定した．また，本研究では，OpenStack プロジェクトの 2011 年から 2014 年で実施されたコードレビュー履歴を用いて合意形成内容を分析する．OpenStack は大規模でかつ開発コミュニティが成長し続けている OSS である．

1.2.1 RQ1: 意見が衝突したパッチは存在するのか？

動機

パッチ開発者は複数の検証者に対して，パッチのレビューを依頼する [29, 36]. 複数の検証者で議論を進める際，検証者間で意見の衝突が発生する可能性がある. しかし，実際のコードレビューで意見の衝突が発生するのか否か明らかにされていない．RQ1 では，大規模 OSS の一つである OpenStack プロジェクトにおいて，検証者間で意見の衝突がどのくらい発生するのかを調査する.

実験結果

OpenStack プロジェクトでは，意見が衝突するパッチが約 26% 存在することを明らかにした．さらに，開発期間に伴って，意見が衝突するパッチの数は増加する傾向にあることを確認した.

1.2.2 RQ2: 意見が衝突したパッチは合意形成に至るのか？

動機

異なる開発経験を持つ検証者間で発生した意見の衝突を解決することは，容易ではない．そのため，全ての議論が合意形成されて，パッチが統合されるとは限らない．しかし，実際のコードレビューで合意形成に至っているか否かは明らかにされていない．RQ2 では，意見が衝突した場合，どの程度合意形成に至るのかを調査する.

実験結果

強い否定的意見と肯定的意見が衝突したパッチでは，約 54% が検証者間で合意に至っていた．さらに，OpenStack の開発が進むに連れて，意見が衝突したパッチの割合は一定の値に収束することを確認した．例えば，弱い肯定的意見と強い否定的意見が衝突したパッチで合意が形成される割合は，開発初期の 2011 年中旬か

ら 2012 年下旬にかけて，約 90% から約 75% まで減少している．しかし，それ以降は約 75% に収束していく傾向があることを確認した．

1.2.3 RQ3: 合意が形成されたパッチの内容とは？

動機

RQ2 より，強い否定的意見と肯定的意見が衝突したパッチの 52% は合意形成に至った．合意形成に至った議論の内容を理解するために，合意形成に至ったパッチ集合から，検証者はどのように意見の衝突を解決して合意形成に至ったかを分析する．

実験結果

合意形成されたパッチの 54% ± 5% は，パッチを再修正せずに検証者間の議論だけで意見の衝突を解決できていることを確認した．さらに，パッチを再修正して合意形成を図ったパッチの 66% はシステム内部の技術的な問題に関して検証者間で意見が衝突しているが，再修正を繰り返しながら合意形成に至っている．

1.3 論文構成

本論文の構成は以下の通りである．続く，2 章でコードレビュープロセスの説明を，3 章では実験方法を説明して，4 章では，実験結果を示す．5 章では，実験結果に対する考察を示し，6 章でコードレビュープロセスを対象とした従来研究を紹介する．最後に 7 章で本論文の内容をまとめる．

第 2 章　コードレビュープロセス

OpenStack コミュニティは，Gerrit Code Reveiw 上でコードレビュープロセスを管理している．図 2.1 は，Gerrit Code Reveiw 上で実施される OpenStack コミュニティのコードレビュープロセスの概要を示す．

1．パッチの投稿：　パッチ作成者は，作成したパッチもしくは修正を加えたパッチを Gerrit Code Reveiw へ投稿する．そして，OpenStack コミュニティに存在する検証者からレビューを依頼する検証者を選定する．選定された検証者は議論を通して，そのパッチの品質を評価する．

2．Sanity Bot テスト：　検証者がレビューを開始する前に，Sanity Bot と呼ばれる自動テストシステムがコード規約やコンパイルの正常動作確認を検証する．もし Sanity Bot が何か問題を指摘した場合，当該パッチは再修正されて Sanity Bot の検証を通過しない限り，プロジェクトに統合されない．このプロセスでは，軽微な問題の指摘などの機械でも自動判定できる問題をレビュー開始前に予め解決する．

3．検証者によるコードレビュー：　Sanity Bot による検証を終えた後，パッチ開発者はレビューを依頼する検証者を選択する．依頼された検証者は，投稿されたパッチに対してフィードバックと評価結果（スコア）が求められる．Gerrit Code Reveiw では，検証者は，次の 5 つのスコアを投稿する．“+2” はパッチの統合を強く賛成することを示す（Strong Positive）．“+1” はパッチの統合に賛成はしているが，他の検証者の意見も必要としている（Weak Positive）．“0” はパッチの統合について賛成・反対の意見を示していない．“-1” は，パッチの統合に反対しているが，他の検証者の意見も必要としている（Weak Negative）．“-2” はパッチの統合を強く反対している（Strong Negative）．

4．パッチの統合リクエスト　Gerrit Code Reveiw では，パッチが統合されるために満たすべきポリシーを定めている．例えば，OpenStack では，パッチの統合には，少なくとも 2 人以上から “+2” スコアを得る必要があ

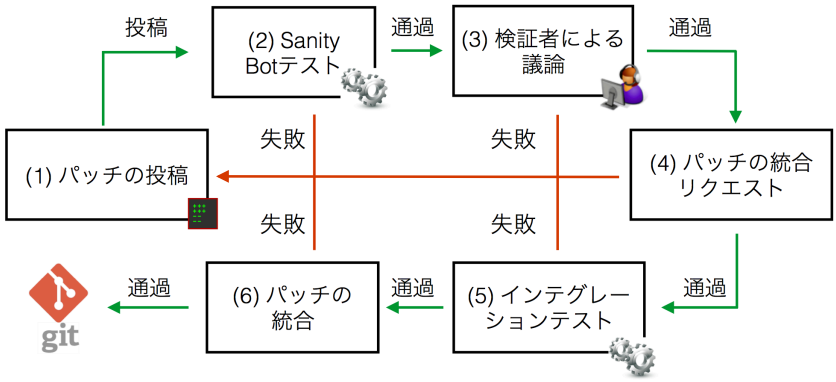

図 2.1　コードレビュープロセスの概要図

る.* このようなポリシーを満たしたパッチについて，パッチ作成者は統合リクエストをプロジェクトへ送信する.

5．インテグレーションテスト　統合リクエストがプロジェクトに受理された後，統合の際に問題が発生しないか否かを検証するために，インテグレーションテストを実行する．インテグレーションテストは Sanity Bot よりも厳格なテスト内容であり，リリース後にあらゆる問題が発生しないことを保証する必要がある．もしインテグレーションテストで問題が発見された場合，パッチ作成者がその問題を解決するまで統合されない.

6．パッチの統合　インテグレーションテストが何も問題を指摘しなかった場合，インテグレーションテストが成功したことになり，Gerrit Code Reveiw は自動的にプロジェクトへパッチを統合する.

*http://docs.openstack.org/infra/manual/developers.html#project-gating

第 3 章　実験方法

本研究では，OpenStack プロジェクトを実験対象プロジェクトとして扱う．本章では，OpenStack プロジェクトを選定した理由と，OpenStack プロジェクトのレビューデータを利用した実験の方法を説明する．

3.1　実験対象プロジェクトの選定方法

本研究では，大規模でかつ開発コミュニティが成長し続けている OSS を対象に実験を進めるために，以下の二つの指標を基に OSS プロジェクトを選定した．

指標 1：コードレビューポリシーの規律性　先行研究 [23, 24] と同様に，本研究においても，投稿されたパッチに対して厳格なプロジェクトポリシーを基にコードレビューを実施する OSS プロジェクトを選択する．

指標 2：複数の検証者によるコードレビューの実施率　意見の衝突が発生する場合，必ず二人以上の検証者がレビューする必要がある．しかし，複数の検証者によってレビューされたパッチの割合が低い OSS は意見の衝突を分析する上で，充分なデータ量を確保できない．従って，本研究では，複数の検証者によるコードレビューの実施率が高い OSS プロジェクトを選定する．

指標 1 を基に選定するために，本研究では Gerrit Code Review と呼ばれるコードレビューシステムで管理されている OSS プロジェクトを分析する．* Gerrit Code Reveiw は，バージョンコントロールシステムと自動テストシステムを利用してコードレビュープロセスを管理するシステムであり，当該システムを利用した現代のコードレビュー形態を Modern Code Review と呼ぶ．例えば，Qt プロジェクトでは，パッチがプロジェクトに統合される前に，少なくとも 1 人の検証者がレビューすることが義務付けられている [35].

本研究では，従来研究でデータセットが公開されている 5 つの OSS プロジェクトのレビューデータを対象に指標 1 と指標 2 を評価した．表 3.1 は，5 つの OSS プ

*https://code.google.com/p/gerrit/

表 3.1　実験対象データ候補となる 5 プロジェクトのコードレビュー活動履歴．
本研究では，OpenStack を実験対象プロジェクトとして選定した．

プロジェクト	開発期間	#総パッチ数	#複数検証者が検証したパッチ数	#総検証者数	1 パッチに対する平均検証者数
OpenStack	08/2011 - 05/2014	87,339	60,665 (70%)	2,636	2.57
Qt	05/2011 - 11/2013	68,072	14,565 (21%)	936	1.05
AOSP	10/2008 - 04/2015	63,610	16,457 (26%)	3,334	1.04
Eclipse	02/2012 - 05/2015	9,168	1,535 (17%)	795	0.76
Libreoffice	03/2012 - 05/2015	13,597	852 (6%)	407	0.55

ロジェクトに対して指標 1 と指標 2 を調査した結果である．Qt プロジェクト及び OpenStack プロジェクトについて，McIntosh ら [24] が公開しているレビュー履歴を使用する．また，Libreoffice，Eclipse そして AOSP プロジェクトについては，Yang らによって公開されているレビュー履歴を使用する [39]．表 3.1 より，Libreoffice，Eclipse，Qt，AOSP プロジェクトに対する複数の検証者がコードレビューを実施した割合であり，順に 6%，17%，21%，26% である．これら 4 つのプロジェクトでは，多くは 1 人の検証者によってレビューが実施されていることを確認した．一方で，OpenStack プロジェクトでは，約 70% のパッチは複数の検証者からレビューされているため，本実験では．OpenStack プロジェクトを対象とする．OpenStack はクラウド環境構築・管理するための OSS であり，IBM，VMware，NEC などの企業が開発に貢献する [36]．

3.2　実験対象データの整形方法

本研究では，McIntosh ら [24] が公開した OpenStack プロジェクトのデータを使用する．このデータセットは，2011 年 7 月から 2014 年 5 月の間で開発されたパッチに対するレビューの履歴情報が管理されている．図 3.1 は，このデータセットの整形方法の概要を示しており，3 つのステップで構成される．

3.2.1　意見が衝突したパッチの判定

はじめに，本実験対象となるデータセット中に存在する各パッチに対して，検証者間で意見が衝突したか否かを判定する．この判定では，同じリビジョン内で少なくとも 1 つの肯定的評価スコア（+2，+1）と 1 つの否定的評価スコア（-2，-1）が

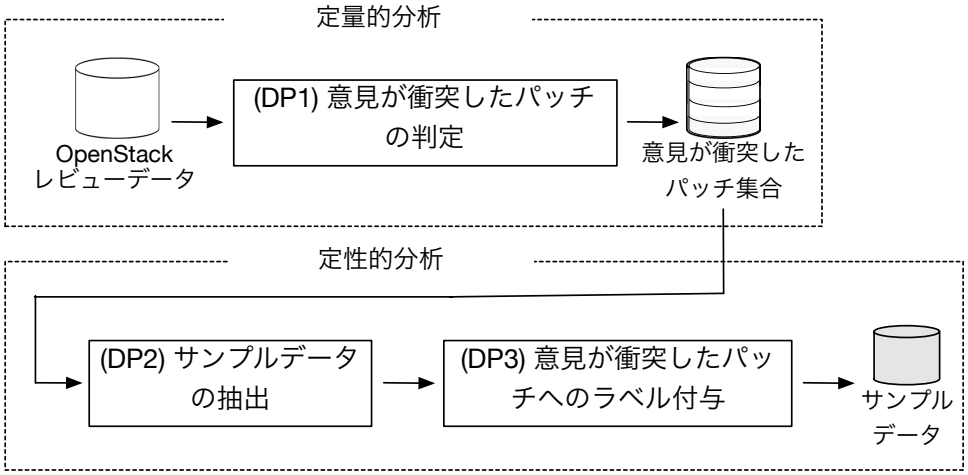

図 3.1　実験対象データの整形プロセスの概要図

投稿されたパッチを意見が衝突したパッチと定義する．リビジョンとは，作成/修正されたパッチの投稿時点から，再修正/統合された時点までの期間を指す．また，検証者は自分のスコアを更新する場合がある．本研究では，検証者の意見が変わる前の最初の評価内容を分析するため，本研究では検証者が最初に評価スコアのみを着目する．以上を踏まえ，本研究では検証者の評価スコア（+2，+1，-1，-2）を基に，意見が衝突する 4 つのパターンを定義する．また，0 スコアは肯定/否定的な意見を含まないため，本研究では対象としない．以下に，意見が衝突する 4 つのパターンを示す．

Strong Positive, Strong Negative (*SP-SN*)　少なくとも 1 つの “+2” スコアと 1 つの “-2” のスコアを含む．また，“+2” 及び “-2” は他の検証者から投票されている．

Strong Positive, Weak Negative (*SP-WN*)　少なくとも 1 つの “+2” スコアを含むが，“-2” のスコアを含まずに少なくとも 1 つの “-1” を含む．また，“+2” 及び “-1” は他の検証者から投票されている．

Weak Positive, Strong Negative (*WP-SN*)　少なくとも 1 つの “-2” スコアを含むが，“+2” のスコアを含まずに少なくとも 1 つの “+1” を含む．また，“-2” 及び “+1” は他の検証者から投票されている．

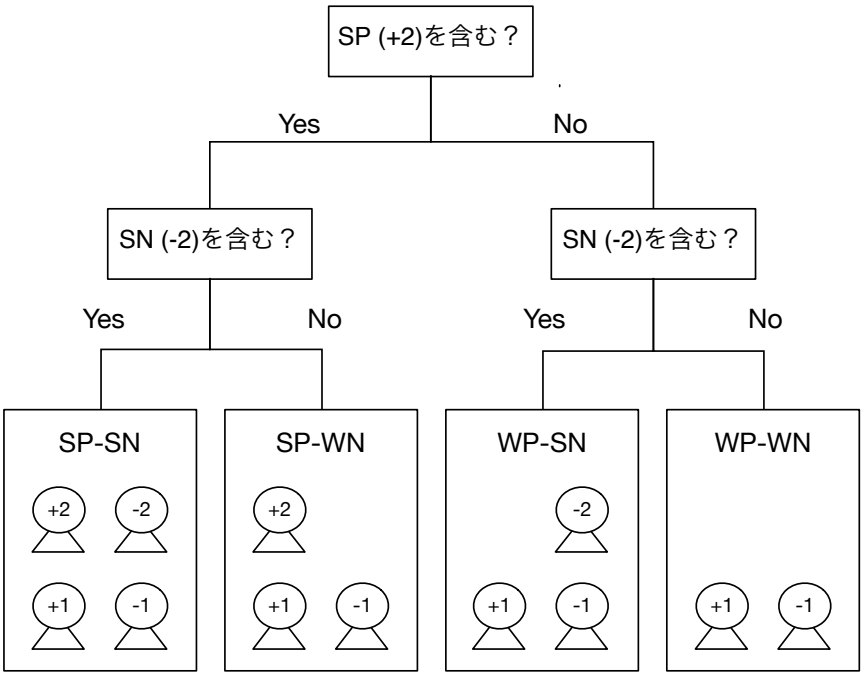

図 3.2　意見が衝突したパッチの判定方法の概念図

Weak Positive, Weak Negative (*WP-WN*)　“+2” のスコアと “-2” のスコアを含まず，少なくとも 1 つの “+1” のスコアと “-1” のスコアを含む．また，“+1” と “-1” は他の検証者から投票されている．

図 3.2 は，意見が衝突する 4 つのパターンの分類方法の概要を示す．例えば，あるパッチに対して，3 人の検証者 A，B，C がそれぞれ +2，-1，+1 を投票したとき， はじめに +2 スコアを含むか否かを判定する．このとき，パッチは +2 を含むため，左のパスに進む．続いて，パッチが-2 スコアを含むのか否かを判定する．このとき，パッチは-2 を含まないため，右のパスに進み *SP-WN* パターンに分類される．

3.2.2　サンプルデータの抽出方法

本研究の RQ3 では，意見の衝突したパッチがどのように合意形成されるのか理解するために，検証者が議論した内容を定性的に分析する．全てのパッチを定性的

に評価することは膨大な時間を要するので，本研究ではサンプルとなるパッチを抽出する．サンプルとなるパッチを信頼度 95%，信頼区間± 5% で抽出するために，サンプルデータ抽出ツール †を用いる [21, 18].

3.2.3 意見の衝突したパッチへのラベル付与

RQ3 では，意見の衝突したパッチがどのように合意形成されるのか分析するために，サンプルデータから得られたパッチそれぞれに対して，意見が衝突した主な要因を示すラベルを付与する．ラベル付けでは，パッチに対するインラインコメント，もしくは議論スレッドのレビュー内容を読む．合意形成されたパッチについて，意見が衝突した主な要因と，意見の衝突に対してどのように解決したのかを示すラベルを付与する．

†http://www.surveysystem.com/sscalc.htm

第 4 章　実験結果

本章では，本論文のリサーチクエスチョン RQ1 から RQ3 についての実験結果を示す．各リサーチクエスチョンについて，実験方法と実験結果を説明する．

4.1　RQ1：意見が衝突したパッチは存在するのか？

実験方法

RQ1 では，OpenStack プロジェクトに投稿されたパッチの中で，どのくらい意見が衝突したのかを調査する．はじめに，複数の検証者によってレビューされたパッチ全体の中で，意見が衝突したパッチがどのくらい存在するのかを調査する．次に，2011 年から 2014 年の期間において，月単位で意見が衝突したパッチ数の推移を調査する．

実験結果

実験結果 1— OpenStack では，意見が衝突するパッチは約 26% 存在する．図 4.1 は，複数の検証者によってレビューされたパッチ全体の中で，意見が衝突したパッチがどのくらい存在するのかを調査した結果である．図 4.1 より，複数の検証者がレビューしたパッチ 60,665 件中，26%（1% + 13% + 1% + 11%）は意見が衝突していたことを確認した．26% の中，ほとんどは WN を含むパターンである一方で，8 %のパッチ（$\frac{1\% + 1\%}{26\%}$）は SN を含むパターンであることを確認した．

実験結果 2— 意見が衝突するパッチの数は，急速に増加している．図 4.2 は，2011 年から 2014 年の期間において，意見が衝突したパッチ数の推移を月単位で調査した結果である．横軸は 2011 年から 2014 年までを月単位で分割した時系列を表し，縦軸は意見が衝突したパッチの累積数を対数変換した値である．図 4.2 より，全てのパターンにおいて，意見が衝突したパッチの数は指数関数的に増加していることを確認した．OpenStack において存在する割合が高い *SP-WN* と

図 4.1　意見が衝突したパッチの発生件数

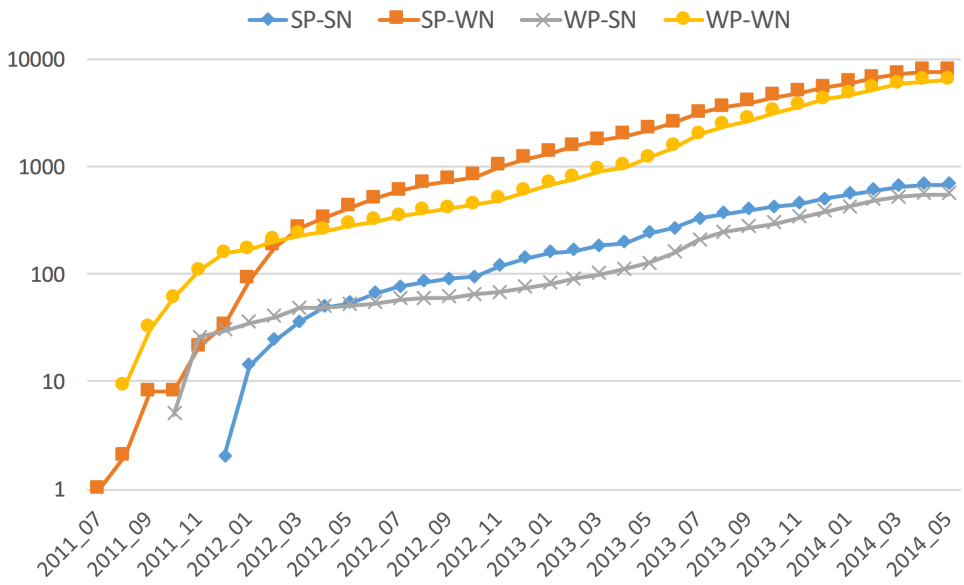

図 4.2　2011 年から 2014 年までにおける意見が衝突したパッチ数の推移

WP-WN パターンは急速に増加している一方で，存在する割合が低い *WP-SN* と *SP-SN* パターンは穏やかに増加している．これらの結果より，OpenStack コミュニティの成長に伴って，意見の衝突数が増加していることを明らかにした．

OpenStack では，複数の検証者によってレビューされたパッチ全体の約 26% で，意見の衝突が発生している．さらに，OpenStack コミュニティの成長に伴って，意見の衝突が発生するパッチの数は増加している．

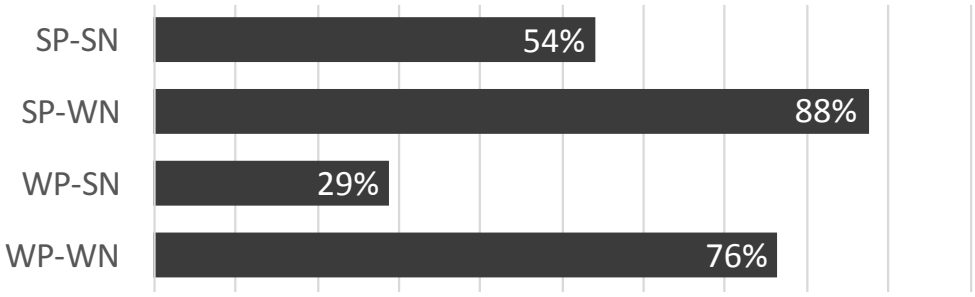

図 4.3　合意形成できたパッチの割合

4.2　RQ2：意見が衝突したパッチは合意形成に至るのか？

実験方法

RQ2 では，OpenStack プロジェクトにおける意見が衝突したパッチについて，検証者間でどのくらい合意を形成できたのか調査する．はじめに，意見が衝突したパッチ全体で，どのくらい合意が形成できたのかその割合を調査する．次に，OpenStack プロジェクトの 2011 年から 2014 年の開発期間において，どのくらい合意が形成できたのかその割合の推移を月単位で調査する．

実験結果

実験結果 3— 強い意見の衝突が発生したパターン（*SP-SN* ）において，過半数のパッチは合意が形成されている．図 4.3 は，意見が衝突したパッチの中，どのくらい合意が形成できたのかを各パターンに対して調査した結果である．もし肯定的スコアが否定的スコアより強いパターン *SP-WN* の場合，合意を形成しやすいと考えられる．図 4.3 より，肯定的スコアが否定的スコアより強いパターン *SP-WN* の場合，約 88% のパッチにおいて合意が形成されている．対して，肯定的スコアが否定的スコアより弱いパターン *WP-SN* では，約 71% のパッチが合意を形成できていなかったことを確認した．これらの結果より，強い否定的スコアが投票されたパッチについては，合意を形成することが難しいと考えられる．

次に，肯定的スコアと否定的スコアが同じ強さのパターン *SP-SN* と *WP-WN* に

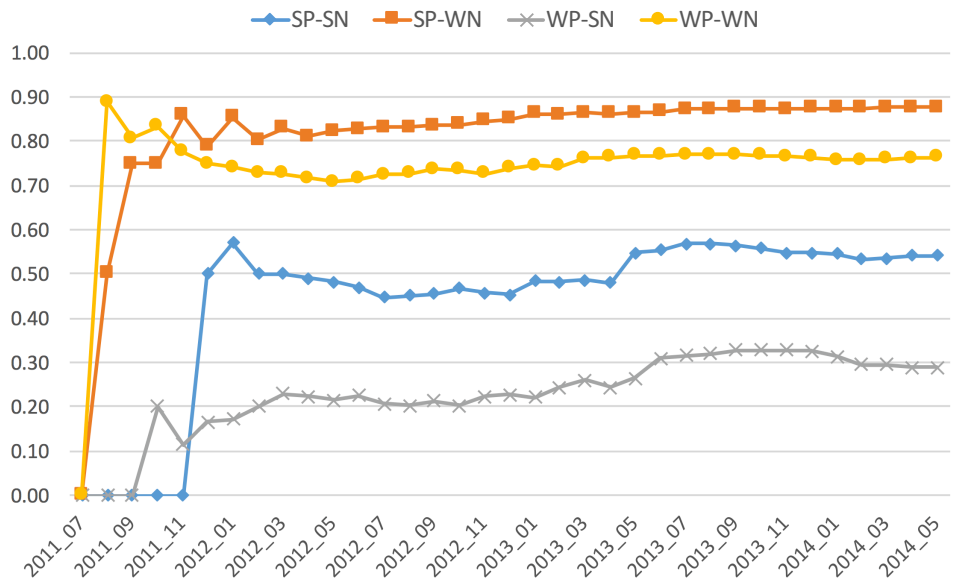

図 4.4　2011 年から 2014 年までにおける合意形成されたパッチの割合の推移

ついての実験結果を示す．*SP-SN* と *WP-WN* では，肯定的スコアと否定的スコアの強さが同じであるため，約 50% の割合で合意が形成できていると考えられる．しかし，図 4.3 より，*WP-WN* パターンでは，約 76% のパッチにおいて合意が形成されていたことを確認した．また，*SP-SN* においても，過半数以上である約 54% のパッチは合意が形成されていたことを確認した．この結果より，特に *SP-SN* パターンにおいて，強い肯定的スコアを投票する検証者はたとえ強い否定的スコアが投票されたとしても，合意形成へ導いていると考えられる．

実験結果 4— OpenStack 開発の経過に連れて，合意が形成されているパッチの割合は一定の割合に収束する．図 4.4 は，OpenStack プロジェクトの 2011 年から 2014 年の開発期間において，どの程度合意が形成できたのかその割合の推移を月単位で調査した結果である．図 4.4 より，意見の衝突が発生したパッチの数は増加しているにも関わらず（調査結果 2），合意が形成される割合（合意形成率）は開発が進むに連れて収束していることを確認した．また，開発初期において合意形成率が変動する傾向にあることを確認した．例えば，*WP-WN* パターンにおける合意が形成される割合は，2011 年中旬から 2012 年下旬にかけて，約 90% から約

75% まで減少している．しかし，それ以降は 75% 程度に収束していく傾向がある．開発初期の合意形成率の大きな変動の要因として，プロジェクトポリシー（2 人以上の検証者が参加する必要があるなど）に対する検証者の厳格性が低かったことが考えられる．しかし，プロジェクトが成熟していくに連れて，プロジェクトポリシーへの厳格性が高まり，意見衝突の各パターンで合意形成率が収束していったと考えられる．この結果より，OpenStack では，開発初期にかけては，合意形成された割合が急激に変動するが，開発が進むに連れてその割合は収束していくことを確認した．

最も強い意見の衝突が発生している *SP-SN* パターンでは，過半数以上のパッチにおいて合意が形成されている．さらに，合意が形成された割合が変動しやすい開発初期の後は，どのパターンにおいてもその割合は一定の値に収束する傾向がある．

4.3 RQ3：合意が形成されたパッチの内容とは？

実験方法

RQ3 では，検証者間で意見が衝突した議論を，どのように解決していくのか定性的に分析する．合意形成されたパッチから抽出したサンプルデータ中の各パッチに対してタグを付与する．タグは，強い肯定的評価スコア（+2）と衝突した強い否定的評価スコア（-2）を投票した検証者が指摘した内容と，その意見の衝突をどのように解決したのかを示す．

サンプルデータ中の各パッチへのタグ付けを終えた後，本研究では従来研究と同様に Open Card Sorting を用いる [3, 12, 17, 33]．Open Card Sorting とは，全てのタグに対して，類似したタグを分類する．そして，分類結果を基に，関係性のあるクラスタ同士を集約していき，階層構造を構築する．この分類法は，タグ付けされたデータから，明確でかつ一般性のある理論を構築することができる．本研究で進める Open Card Sorting では，はじめに類似したタグを集約して，それらの内容を表す短いタイトルを与える．続いて，それらの分類されたグループに対して，強い関係性を持つグループを集約して，階層構造を構築する．

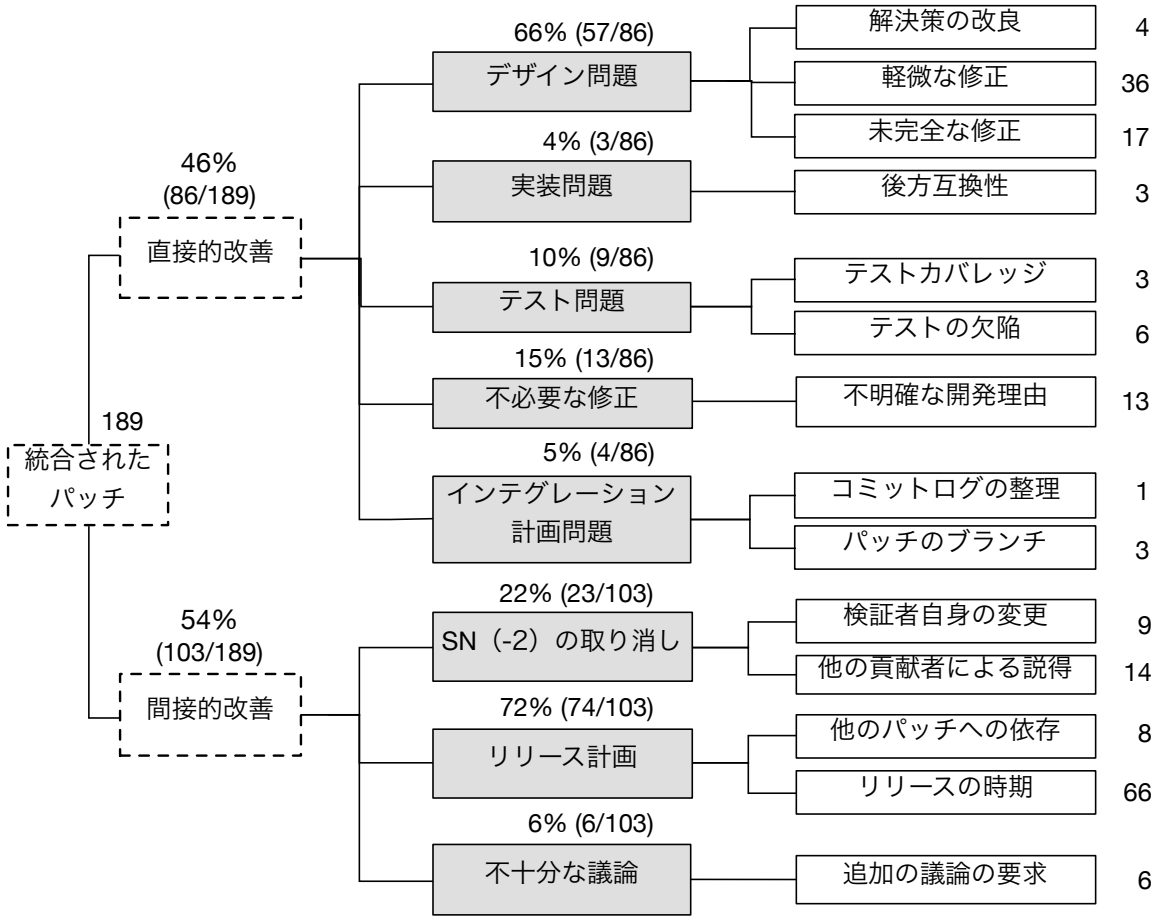

図 4.5　RQ3 の定性分析の結果

実験結果

図 4.5 は，合意形成されたパッチのレビューにおいて，意見が衝突した内容と，それらをどのように解決したかを示している．各タグの合計をタグの横に記述している．意見の衝突したパッチは，直接的改善と間接的改善に分類された．直接的改善とは，意見の衝突の要因となった問題を，検証者がパッチ作成者にフィードバックを与えて，そのフィードバックを基にパッチ作成者がパッチを再修正するという作業を繰り返す方法を指す．間接的改善とは，意見の衝突の要因となった問題を，パッチを再修正せずに検証者間の議論だけで解決する方法を指す．

実験結果 5— 合意形成されたパッチの 54% ± 5% は，間接的改善によって合意形成に至る．合意形成されたパッチの中で，過半数以上のパッチはパッチ作成者が再修正せずに検証者間の議論だけで解決していたことを確認した．これらのパッチは，「否定的スコア（-2）の変更」，「リリース計画」，「不十分な議論」に分類される．

否定的スコアの変更: 間接的改善に分類されたパッチの 22% は，強い否定的評価スコア（-2）を投稿した検証者が自らそのスコアを取り下げた．間接的改善では，“検証者自身の変更” と “他の貢献者による説得” の二つのパターンを確認した．一つ目は，強い否定的評価スコアを投稿した検証者が自身の否定的評価スコアを議論せずに引き下げるパターンである (39%)．たとえば，パッチ ID 68845*では，検証者は，投稿されたパッチが他の機能へ悪影響を与える問題があることを指摘した．しかし，ソフトウェアテストを実行して何も問題がないことを確認できた後，自身の-2 スコアを取り下げた．二つ目は，-2 スコアを投票した検証者に対して，パッチ作成者もしくは他の検証者が説得して，-2 を取り下げたパターンである (61%)．たとえば，パッチ ID 71401†では，検証者は投稿されたパッチは統合する必要がないと指摘した．しかし，パッチ作成者がその検証者に対して，投稿されたパッチの貢献度を伝えた結果，-2 は取り下げられた．

リリース計画: 直接的改善に分類されたパッチの 72% は，パッチの統合する上で発生する問題に対して，意見の衝突が発生していた．直接的改善では，「他のパッチへの依存’」と「リリースの時期」の 2 つのパターンを確認した．「他のパッチへの依存」とは，投稿されたパッチが他のパッチへ機能的に依存しているため，依存するパッチの統合を待機するパターンである．たとえば，パッチ ID 70126‡では，投稿されたパッチがパッチ ID 69614§と依存関係にあるため，パッチ ID 69614 が統合されるまで待機するために，-2 スコアが投稿されている．「リリースの時期」とは，投稿されたパッチが統合されるブランチの次のリリースに合わせて統合するため，待機するパターンである．たとえば，パッチ ID 17015¶では，-2 を投稿した検証者は，他の検証者たちに統合する予定となる次のリリース時期まで待機させている．

*https://review.openstack.org/#/c/68845/
†https://review.openstack.org/#/c/71401/
‡https://review.openstack.org/#/c/70126/
§https://review.openstack.org/#/c/69614/
¶https://review.openstack.org/#/c/17015/

不十分な議論: 直接的改善に分類されたパッチの 6% は，必要な議論がまだ十分に行われていないと判断した検証者が，否定的なスコアを投票して意見が衝突したパターンである．例えば，パッチ ID 13837[‖]では，投稿されたパッチを統合する時，他の開発タスクにも影響を及ぼすことを言及している．そのため，統合に対して賛成した検証者に対して，議論を継続するように促している．

実験結果 6— 直接的改善では，66% は内部の技術的問題による意見の衝突を解決した．合意形成されたパッチの中で，30% のパッチはパッチ開発者が検証者からフィードバックを受け，それらを基に再修正して合意形成を図っていたことを確認した．これらのパッチは，「デザイン問題」，「実装問題」，「テスト問題」，「不必要な修正」，「インテグレーション計画問題」に分類される．

デザイン問題: 直接的改善において，66% はコード内に存在する技術的問題について意見の衝突が発生している．デザイン問題には，「解決策の改良」，「軽微の修正」，「未完全な修正」がある．

- **解決策の改良**: 修正する問題へのより良い解決策が議論されている．たとえば，パッチ ID 74225[**]では，検証者はパッチ作成者が提案した問題の解決策を改良する必要があることを指摘している．
- **軽微の修正**: パッチが軽微の修正（コミットログの修正，スペルミス，関数名の修正など）に関して議論されている．たとえば，パッチ ID 26684[††]では，検証者はパッチに含まれるコードではなく，ドキュメントの内容について軽微な修正点を議論している．
- **未完全な修正**: 投稿されたパッチが，修正すべき箇所を全て修正できていない未完全な部分に関して議論されている．たとえば，パッチ ID 36350[‡‡]では，検証者は投稿されたパッチについて，セキュリティの問題があることを指摘している．

不必要な修正: 直接的改善において，15% は不必要な開発である．「不明確な開発理由」とは，検証者がパッチ作成者がなぜパッチを作成したのかの意図が理解で

[‖]https://review.openstack.org/#/c/13837/
[**]https://review.openstack.org/#/c/74225/
[††]https://review.openstack.org/#/c/26684/
[‡‡]https://review.openstack.org/#/c/36350/

きていないパターンである．たとえば，パッチ ID 2915[§§]では，検証者は投稿されたパッチの開発理由を理解できていないため，なぜこのパッチが必要なのかパッチ作成者に質問をしている．

テスト問題: 直接的改善において，10% は自動テストに関する問題である．「テストカバレッジ」とは，自動テストによるテストの網羅性の高さを指摘して意見の衝突が起きたパターンである．たとえば，パッチ ID 29549[¶¶]では，検証者はテストの網羅性の欠如を指摘して，網羅すべきコード部分を提案している．「テストの欠陥」は，テストコード内に欠陥がある，もしくは自動テストの異常動作などの問題で意見の衝突が起きたパターンである．たとえば，パッチ ID 33211[***]では，検証者は自動テストが正常に動作していないことを指摘して，そのような対処方法をとるべきか議論している．

インテグレーション計画問題: 直接的改善において，5% はパッチをプロジェクトへ統合する際に起こる問題によって意見が衝突する．「コミットログの整理」とは，OpenStack コミュニティーでは，パッチを統合する前に開発過程で発生したコミットを必要最低限に統合してインテグレーション時に発生する可能性がある問題を回避するために，開発履歴を整頓することを求められる. たとえば，パッチ ID 8721[†††]では，検証者はコミットログを一つに統合するように指摘して，開発者は再修正した．「パッチのブランチ」は，パッチ開発者がパッチを適切なブランチ上で作成されていなかったパターンを指す．たとえば，パッチ ID 62980[‡‡‡]では，検証者は投稿されたパッチで修正された内容は，他のブランチで修正されるべきであることを指摘している．

実装問題: 直接的改善において，4% はパッチ実装時に生じる問題で意見が衝突する．「後方互換性」は，投稿されたパッチが以前のバージョンでも整合性がある動作をするかどうかの問題である．たとえば，パッチ ID 17735[§§§]では，検証者は後方互換性について指摘して，パッチ作成者が再修正した後，意見の衝突は解決された．

[§§] https://review.openstack.org/#/c/2915/
[¶¶] https://review.openstack.org/#/c/29549/
[***] https://review.openstack.org/#/c/33211/
[†††] https://review.openstack.org/#/c/8721/
[‡‡‡] https://review.openstack.org/#/c/62980/
[§§§] https://review.openstack.org/#/c/17735/

第 5 章　考察

5.1　本研究による知見

5.1.1　複数の検証者によるレビューでは，意見の衝突が発生する．

実験結果 1 では，OpenStack コミュニティにおいて，複数の検証者が議論したパッチ全体のうち，約 26% で意見の衝突が起きていることを確認した．また，実験結果 2 では，意見の衝突が起きたパッチの数は，OpenStack プロジェクトの成長に伴って増加していることを確認した．実験結果 1 及び実験結果 2 より，パッチを統合するために満たすべき要件で，少なくとも 2 人以上の検証者がレビューする必要がある場合，検証者間で意見の衝突が発生することを想定する必要がある．

5.1.2　*SP-SN* パターンでは，約 54% は合意形成される．

実験結果 3 より，SP-WN 及び，WP-SN パターンでは，どちらかの評価スコアが一方のスコアより強い．そのため，肯定的意見の方が強い場合は統合されやすく，否定的意見の方が強い場合は採択されにくい．一方で，強い肯定的評価スコア（SP）と同等の強さの否定的評価スコア（SN）が衝突した場合，50% 以上が統合されることを確認した．最も激しい意見の衝突（*SP-SN* ）であっても，合意形成に至るパッチは少なくないことを確認した．従って，肯定的意見と否定的意見の強さが同じパターン（*SP-SN* ，*WP-WN* ）では，合意形成に至る可能性が高いため，パッチ作成者は積極的にパッチの改良に取り組むべきである．

5.1.3　検証者間の建設的な議論と再修正によって合意を形成する．

実験結果 5 より，合意形成されたパッチの 54% ± 5% は，間接的改善によって意見の衝突が解決されたことを確認した．パッチ開発者及び，レビューに参加する検証者は，議論で解決できる問題であるかを適切に見極める必要がある．また，実験結果 6 より，46% ± 5% は議論と再修正によって，合意形成に至っている．主に，デザイン問題に関する内容であり，パッチのデザインについて意見の衝突が起きた

場合は，積極的に再修正していくことで合意形成に至ることが期待できる．

5.2 本研究の展望

本研究では，合意形成内容には様々な種類が存在することを明らかにした．RQ3では，合意形成に至ったパッチの 54% ± 5% は，間接的改善によって意見の衝突が解決されたことを明らかにした．しかし，本研究では，合意形成の方法の妥当性を評価しておらず，意見の衝突が発生した時に検証者がどのように合意を形成していくべきかを推薦できていない．今後の展望として，意見が衝突した内容に対して，どのように合意形成を図っていくべきか分析して，最適な合意形成方法を推薦するモデルを提案する．合意形成方法推薦モデルを提案できれば，効率的に合意を図ることができ，これまで合意形成に費やしてきた時間を他の開発作業に割り当てることができる．最適な合意形成方法の推薦モデルを構築するためには，本研究結果から得た各々の合意形成内容に対して，どのように再修正したのか，どのように議論を進めていたのかを詳しく調査する必要がある．

5.3 妥当性脅威

本研究の定量的分析（RQ1 と RQ2）と定性的分析（RQ3）における，構成概念妥当性，内的妥当性，外的妥当性について議論する．

5.3.1 構成概念妥当性

本研究で実施した定性的分析は Gerrit Code Reveiw で記録された議論の履歴を用いた．しかし，検証者は必ずしも Gerrit Code Reveiw で議論するとは限らず，パッチに対する議論を直接検証者間で行うケース [8] や，他のチャットシステム [32]，メーリングリスト上 [12, 31]. で議論することがある．したがって，これらの方法で議論された内容をデータセットとして作成する研究が進められている [2, 9, 15]．一方で，OpenStack プロジェクトでは，全てのパッチに対して議論された内容が Gerrit Code Reveiw 上で記録されているため，膨大なパッチ

に対する議論履歴を分析することができる．

図 3.1 で示した通り，本研究で定性的に調査したサンプルは 189 件である．信頼度 95%，信頼区間 5% でサンプルを抽出したが，サンプルからは得られなかった結果を残りのパッチ集合が得られる可能性がある．今後は，サンプル以外のパッチにも調査を進め，研究結果の更新が必要である．

5.3.2 内的妥当性

本研究では，複数の検証者が評価スコアを投票して，かつそれらが少なくとも肯定的スコア（+2 もしくは +1）と否定的スコア（-2 もしくは-1）を含むパッチを意見が衝突したパッチとして定義した．しかし，実際には評価スコアを投票せずにコメント上でパッチを否定する内容を記載している場合がある．これらの事例に対して，自然言語処理技術を利用することで，コメント上から検証者の評価内容を推測することが可能である．しかし，自然言語処理技術による推測では，基本的に過去の議論履歴を学習する必要があり，誤った推測をする可能性もある．今後は，誤推測する可能性を考慮して，自然言語処理技術を活用したコメントから検証者の評価内容を推測できる技術を提案する．

本研究では，*SP-SN* パターンを対象として，定性的分析を進めた．しかし，他の 3 つのパターン *SP-WN*，*WP-SN*，そして *WP-WN* からも新たな知見を発見できる可能性がある．しかし，本研究では，厳格にかつ徹底的な open card sorting を実現するために，+2 と-2 のスコアを含む最も強い意見同士が衝突したパターンを対象とした．今後は，他の 3 つのパターンに対して定性的分析を進める．

5.3.3 外的妥当性

本研究は，OpenStack プロジェクトを対象として定量的/定性的分析を進めた．この理由は，OpenStack プロジェクトが厳格なプロジェクトポリシーにプロジェクト貢献者が従って開発が進められており，かつ 3 章で示した実験対象プロジェクトの選定指標を全て満たしたためである．今後は，他の OSS プロジェクトに対しても分析を進め，一般性を保証する分析を進めていく．

第 6 章　関連研究

本章では，コードレビューの有効性，コードレビューの最適化，そしてコードレビューにおける検証者間の議論分析に関する従来研究について説明する．

6.1　コードレビューの有効性

コードレビューは，重要なソフトウェア品質評価手法のひとつである．Raymond ら [28] は，検証者が多く存在する OSS プロジェクトでは，ソフトウェアをリリースする前にレビューを実施することで最も深刻な欠陥を発見できることを明らかにした． また，Rigby ら [30, 29] は，専門的な知識を有する少数の検証者は，OSS プロジェクトへ参加していることを明らかにした．そして，Modern Code Reveiw は，ソフトウェア内に潜む問題を発見/修正するための効率的な品質評価手法である．

厳格に実施されたコードレビューは，ソフトウェア品質と関係性がある．例えば，Kemerer ら [16] は，学生が開発したソフトウェアプロジェクトでコードレビューを実施したときに，ソフトウェア品質が向上したことを確認した．また，先行研究では，検証者の参加とリリース後のバグの関係性を明らかにした [23, 35, 24]．同様に，ソフトウェアのアンチパターンやセキュリティの脆弱性との関係性を明らかにした [26, 25].

機能的な欠陥の判定や修正はソフトウェア品質向上を目的とした有益な手法である一方で，Modern Code Review はそれら以外の目的でも実施される．例えば，Bacchelli ら [3, 5] は，Microsoft では専門的な知識の共有を目的としてコードレビューを実施している．Baysal ら [7, 6] は，WebKit と Blink という OSS プロジェクトにおいて，技術的な問題を指摘されたパッチは不採択されやすく，また検証者からの返答速度が遅くなる傾向にあることを確認した．

6.2　コードレビューの最適化

コードレビューの有効性は，近年の研究でも注目されている．Bosu ら [10] は，モジュールレベルで着目した検証者の専門性がコードレビューの有効性と関係性が

あることを明らかにした．Thongtanunam ら [34] は，専門知識を持つ検証者もしくはモジュールの変更を主に管理している検証者がモジュールの開発のレビューに参加していない場合，たとえ多くの検証者が参加していたとしても，欠陥を含みやすいモジュールとなってリリースされやすい．Kononenko ら [19] は，Mozilla プロジェクトでは，コードレビューに参加する検証者の専門性は，コードレビューの品質に関係することを明らかにした．Aurum ら [1] は，コードレビューの品質を保証するために，コードレビュー対象ファイルについて専門知識がある検証者にレビューを依頼する必要があることを明らかにした．

コードレビュー対象ファイルにふさわしい検証者を選定するために，従来研究では検証者推薦システムが研究されている．例えば，Balachandran ら [4] は，過去に検証者がレビューしたファイル内容を基にレビュー対象ファイルにふさわしい検証者を推薦する ReviewerBot システムを提案した．Thongtanunam ら [36] は，過去に検証者がレビューしたファイルのディレクトリパスを基にレビュー対象ファイルにふさわしい検証者を推薦する RevFinder システムを提案した．最近の研究では，従来手法の検証者推薦システムを改良している．改良された検証者推薦システムでは，検証者の専門性の他に技術的な問題に対するレビュー経験を基に，レビューを依頼すべき検証者を予測する [40, 27]．また，Xin ら [38] は，パッチの内容に着目した改良方法を提案した．

レビューすべき検証者がレビューする場合において，それら検証者全員がレビュー対象ファイルに対して，同じ評価結果を下すとは限らない．本研究では，どのくらい検証者間で意見が衝突するのか，そして，意見が衝突したパッチは検証者間でどのように合意を形成されたのかを分析した．

6.3 コードレビューにおける検証者間の議論分析

最近では，Modern Code Review で実施された議論を定量的に分析する研究が進められている．Jiang ら [14] は，Linux kernel プロジェクトにおいて，検証者による議論の量（コメントの数）はパッチが採択されるか否かを決定する重要な指標であることを明らかにした．Kononenko ら [20] は，レビュー活動を示す指標（レビューに参加した検証者の数）は，コードレビュープロセスの品質と関係があ

ることを明らかにした．実際にケーススタディを通して，McIntosh ら [23, 24] と Thongtanunam ら [35] は， 検証者による議論の量はパッチを採択するか否かを決定する際に，考慮すべき要素であることを確認した．

さらに，機能的な欠陥がコードレビューの議論を通して発見されている．Tsay と Dabbish [37] は，議論を通じて，欠陥やバグを解決するための方法の妥当性を検証者は考慮しており，また，他の方法を提案することもあることを明らかにした．Czerwonka ら [11] は，Microsoft でのコードレビューで発生した議論において，機能的な欠陥に関する内容は約 15% の割合を占めることを確認した．Rigby と Storey [31] は，コードレビューでは，主に幅広い技術的な問題，プロジェクトの拡張性，そしてプロジェクトで定めたポリシーに関する問題が議論されることを明らかにした．Mäntylä と Lassenius [22] らは，学生のソフトウェアプロジェクトと企業のソフトウェアプロジェクトにおいて，75% は保守/運用に関する欠陥について，25% は機能面に関する欠陥について議論されていることを明らかにした．Beller ら [8] は，企業のソフトウェアプロジェクト及び OSS プロジェクトにおいて，コードレビューで発生したコメントのうち，保守/運用及び機能面に関する欠陥について 75:25 の比率で議論されていることを明らかにして， Mäntylä らの研究成果と同様の結果を確認した．本研究は従来研究の研究成果に追加できる成果を確認している．具体的には，合意形成できたパッチの $54\% \pm 5\%$ は，パッチを再修正せずに議論を通して，意見の衝突の要因となった検証者が懸念する問題を解決した．

コードレビューで行われる議論では，複数の検証者が参加するため，検証者間で意見の衝突が発生すると考えれられる．先行研究では，意見の衝突が発生した議論では，レビューが開始してから完了するまでに要する時間が長期化することを確認した [13]．さらに，パッチの採択/不採択の決定は，必ずしも単純多数決法に従うとは限らないことを明らかにした．従来研究では，意見の衝突が起きたパッチに対してソフトウェア品質とコードレビューコストの側面から定量的に分析されてきたが，本研究ではどのように検証者間で合意を形成するのか定性的に調査した．

第 7 章　総括

コードレビューは，ソフトウェアの品質を評価するための重要な技術である．コードレビュープロセスでは，検証者は投稿されたパッチに対して，インラインコメントや公開掲示板によるコメントなどをパッチ作成者へ投稿する．コードレビューは一般的に複数の検証者によって，議論が進められるため，意見の衝突が発生すると考えられる．しかし，それがどのくらい発生するのか，また，検証者間でどのような議論がされているのか明らかにされていない．

本研究では，合意形成内容を明らかにするために，RQ1 から RQ3 に沿って実験を進めた．本研究の分析結果は以下の通りである．

- OpenStack プロジェクトでは，意見が衝突するパッチが約 26% 存在することを明らかにした．さらに，OpenStack プロジェクトの開発期間に伴って，意見が衝突するパッチの数は増加する傾向にあることを確認した．
- 強い否定的意見と肯定的意見が衝突したパッチでは，約 54% が検証者間で合意に至っていた．さらに，OpenStack の開発が進むに連れて，意見が衝突したパッチの割合は一定の値に収束することを確認した．たとえば，例えば，弱い肯定的意見と弱い否定的意見が衝突したパッチ（*WP-WN* ）で合意が形成される割合は，開発初期の 2011 年中旬から 2012 年下旬にかけて，約 90% から約 75% まで減少している．しかし，それ以降は 75% 程度に収束していく傾向があることを確認した．
- 合意形成されたパッチの 54% ± 5% は，パッチを再修正せずに検証者間の議論だけで意見の衝突を解決できていることを確認した．さらに，パッチを再修正して合意形成を図ったパッチの 66% はシステム内部の技術的な問題に関して検証者間で意見が衝突しているが，再修正を繰り返しながら合意形成に至っている．

OSS プロジェクトのコードレビューでは，検証者間で意見の衝突が発生することを認識する必要がある．合意形成内容は様々な種類があり，主に再修正せずに議論だけで意見の衝突を解決する．一方で，再修正が必要とするパッチも存在するため，検証者は意見が衝突した内容を的確に見極めて合意を形成する必要がある．

謝辞

本研究を進めるにあたり，多くの方々にご指導，ご協力，ご支援を賜りました．ここに謝意を添えてお名前を記させて頂きます．

主指導教員であり，本稿の審査委員を務めて頂いた奈良先端科学技術大学院大学 情報科学研究科 松本健一教授に対し，厚く御礼申し上げます．研究に対する直接的な指導に限らず，研究を取り組む上で必要な心構えから研究者としての在り方まで，あらゆる面でご指導頂きました．心より感謝申し上げます．また，約 3 ヶ月のカナダの McGill 大学への留学の機会を与えて頂いたことで，海外研究者と共同研究を進めることができました．重ねて，感謝申し上げます．本稿の副指導教員を務めて頂いた奈良先端科学技術大学院大学 情報科学研究科 飯田元教授には，学内発表問わずあらゆる面でご指導頂きました．心より感謝申し上げます．本稿の副指導教員を務めて頂いた奈良先端科学技術大学院大学 情報科学研究科 伊原彰紀助教授には，研究の進め方から，論文の執筆，プレゼンテーションの方法など，多くの時間を割いてご指導して頂きました．また，海外で共同研究する機会も与えて頂きました．さらに，私が博士後期課程に進学するか悩んでいたときも親身になって相談して頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 畑秀明助教授には，研究活動において，研究内容の相談，プレゼンテーションの方法などについてご指導頂きました．海外出張時も助言を多く頂きました．心より感謝申し上げます．カナダ McGill 大学 Electrical and Computer Engineering 学科 Shane McIntosh 先生は，私が博士前期課程 2 年次の時に訪問研究生として研究室へ受け入れて頂きました．共同研究を進めて頂き，ソフトウェア工学分野で最高峰の国際会議 The International Conference on Software Engineering 2017 に論文を投稿することができました．論文執筆に必要な技術などをご指導頂き，心より感謝申し上げます．

奈良先端科学技術大学院大学　情報科学研究科　ソフトウェア工学研究室の皆様には，日頃より苦楽を共にさせて頂きました．お陰様で，充実した 2 年間の研究活動を過ごすことができました．心より感謝申し上げます．

最後に，大学院進学及び博士後期課程への進学を受け入れ，多くの支援をして頂いた母，父，妹に心より感謝申し上げます．

参考文献

[1] A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years," *Software Testing, Verification and Reliability*, vol. 12, no. 3, pp. 133–154, 2002.

[2] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts." in *Proceedings of the 32nd International Conference on Software Engineering*, 2010, pp. 375–384.

[3] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 35th International Conference on Software Engineering*, 2013, pp. 712–721.

[4] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 35th International Conference on Software Engineering*, 2013, pp. 931–940.

[5] N. S. Baum.Tobias, Liskin.Olga, "Factors influencing code review processes in industry," in *Proceedings of the 24th International Symposium on Foundations of Software Engineering*, 2016, pp. 85–96.

[6] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "Investigating technical and non-technical factors influencing modern code review," *Empirical Software Engineering*, vol. 21, no. 3, pp. 932–959, 2016.

[7] ——, "The influence of non-technical factors on code review," in *Proceedings of the 20th Working Conference on Reverse Engineering*, 2013, pp. 122–131.

[8] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 202–211.

[9] C. Bird, A. Gourley, and P. Devanbu, "Detecting patch submission and acceptance in oss projects," in *Proceedings of the Fourth International Work-*

shop on Mining Software Repositories, 2007, pp. 26–29.

[10] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the 12th International Working Conference on Mining Software Repositories*, 2015, pp. 146–156.

[11] J. Czerwonka, M. Greiler, and J. Tilford, "Code reviews do not find bugs: How the current code review best practice slows us down," in *Proceedings of the 37th International Conference on Software Engineering*, 2015, pp. 27–28.

[12] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in open source software development mailing lists," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 277–286.

[13] T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, and K. Matsumoto, "The impact of a low level of agreement among reviewers in a code review process," in *The 12th International Conference on Open Source Systems*, 2016, pp. 97–110.

[14] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast?: Case study on the linux kernel," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 101–110.

[15] Y. Jiang, B. Adams, F. Khomh, and D. M. German, "Tracing back the history of commits in low-tech reviewing environments," in *Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement*, no. 51, 2014.

[16] C. F. Kemerer and M. C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on psp data," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 534–550, 2009.

[17] N. Kerzazi, F. Khomh, and B. Adams, "Why do automated builds break? an empirical study," in *Proceedings of the 30th International Conference on Software Maintenance and Evolution*, 2014, pp. 41–50.

[18] L. Kish, "Survey sampling." *John Wiley and Sons*, 1965.

[19] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1028–1038.

[20] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, 2015, pp. 111–120.

[21] S. K. Lwanga and S. Lemeshow, "Sample size determination in health studies: A practical manual." *Geneva: World Health Organization*, 1991.

[22] M. V. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?" *IEEE Transactions on Software Engineering*, vol. 35, no. 3, pp. 430–448, 2009.

[23] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 192–201.

[24] ——, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2146–2189, 2016.

[25] A. Meneely, C. Tejeda, B. Spates, S. Trudeau, D. Neuberger, C. Whitlock, K. Ketant, and K. Davis, "An empirical investigation of socio-technical code review metrics and security vulnerabilities," in *Proceedings of the 6th International Workshop on Social Software Engineering*, 2014, pp. 37–44.

[26] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 171–180.

[27] M. M. Rahman, K. R. Chanchal, and J. A. Collins, "Correct: Code reviewer recommendation in github based on cross-project and technology experience," in *Proceedings of the 38th International Conference on Soft-*

ware Engineering, 2016, pp. 222–231.

[28] E. S. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.

[29] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 202–212.

[30] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 541–550.

[31] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 541–550.

[32] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying the use of developer irc meetings in open source project," in *Proceedings of the 25th International Conference on Software Maintenance*, 2009, pp. 147–156.

[33] M. Shridhar, B. Adams, and F. Khomh, "A qualitative analysis of software build system changes and build ownership styles," in *Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement*, no. 29, 2014.

[34] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1039–1050.

[35] ——, "Investigating code review practices in defective files: An empirical study of the qt system," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 168–179.

[36] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. ichi Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *Proceedings of the 22nd International Conference on Software Analysis,*

Evolution, and Reengineering, 2015, pp. 141–150.

[37] J. Tsay, L. Dabbish, and J. Herbsleb, "Let ' s talk about it: Evaluating contributions through discussion in github." in *Proceedings of the 22nd International Sym posium on Foundations of Software Engineering*, 2014, pp. 144–154.

[38] X. Xia, D. Lo, X. Wang, and X. Yang, "Who should review this change? putting text and file location analyses together for more accurate recommendations." in *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, 2015, pp. 261–270.

[39] X. Yang, R. G. Kulay, N. Yoshida, and H. Iida, "Mining the modern code review repositories: A dataset of people, process and product," in *Proceedings of the 13th Working Conference on Mining Software Repositories*, 2016.

[40] M. Zanjani, H. Kagdi, and C. Bird, "Automatically recommending peer reviewers in modern code review." *Transactions on Software Engineering*, vol. 42, no. 6, pp. 530–543, 2015.

発表リスト

国際会議発表

1. Toshiki Hirao, Akinori Ihara, Yuki Ueda, Passakorn Phannachitta, and Kenichi Matsumoto, "Towards a Better Reviewer Recommendation Using the Level of Agreement," The 7th International Workshop on Empirical Software Engineering in Practice (IWESEP2016), 2016.

2. Toshiki Hirao, Akinori Ihara, Yuki Ueda, Passakorn Phannachitta, and Kenichi Matsumoto, "The Impact of a Low Level of Agreement among Reviewers in a Code Review Process," In The 12th International Conference on Open Source Systems (OSS2016), pages 97-110, 2016.

3. Toshiki Hirao, Akinori Ihara, and Kenichi Matsumoto, "Pilot Study of Collective Decision-Making in the Code Review Process," In Proc. of the Center for Advanced Studies on Collaborative Research (CASCON2015), pages 248-251, 2015.

4. Toshiki Hirao, Akinori Ihara, and Kenichi Matsumoto, "How do core reviewers make a decision in Code Review Process - - A Pilot Study of Open Source Project Patches - - ," MSR Asia Summit 2015, 2015.

国内研究会

1. 平尾俊貴, 伊原彰紀, Shane McIntosh, 松本健一, "コードレビュープロセスにおける検証者間の合意形成方法に関する調査," ソフトウェアエンジニアリングシンポジウム (SES2016), 2016 年.

2. 平尾俊貴, 伊原彰紀, 松本健一, "コードレビューにおける誤評価する検証者の特定に向けて," グループウェアとネットワークサービスワークショップ 2016, 2016 年.

3. 平尾俊貴, 伊原彰紀, 松本健一, "コードレビュープロセスに基づくパッチ再投稿の予測," ソフトウェアエンジニアリングシンポジウム 2015(SES2015), number 26, 2015 年.

4. 平尾俊貴, 伊原彰紀, 松本健一, "コードレビュープロセスに基づくパッチ再投稿の予測," ソフトウェアエンジニアリングシンポジウム 2015(SES2015), pages 195-201 2015 年.

受賞

1. 平尾俊貴, 伊原彰紀, Shane McIntosh, 松本健一, "コードレビュープロセスにおける検証者間の合意形成方法に関する調査," ソフトウェアエンジニアリングシンポジウム (SES2016), 2016 年.（インタラクティブ賞）

その他の発表

1. Kenichi Ono, Akinori Ihara, Toshiki HIrao, and Kenichi Matsumoto, "Toward Identifying Responders to a Code Review Request -Case Study of Qt Project-," In The 7th International Workshop on Empirical Software Engineering in Practice (IWESEP2016), 2016.

2. 則兼卓人, 伊原彰紀, 平尾俊貴, 松本健一, "ソフトウェア工学分野における公開データとその活用 -ソフトウェア検証に関する公開データについて- ," 電子情報通信学会技術研究報告（PRMU, パターン認識・メディア理解）, pages 73-78 2016 年.

3. 小野健一, 伊原彰紀, 坂口英司, 平尾俊貴, 松本健一 , "OSS 開発のコードレビュー依頼に貢献する開発者の予測," 第 23 回ソフトウェア工学の基礎ワークショップ (FOSE2016), pages 157-162 2016 年.

4. 小野健一, 伊原彰紀, 平尾俊貴, 松本健一 , "オープンソース開発におけるパッチ検証者数の予測," ウィンターワークショップ 2016・イン・逗子 論文集, volume 2016, pages 53-54 2016 年.

5. 安藤聡志, 平尾俊貴, 伊原彰紀, 松本健一, 関浩之 , "OSS 開発におけるソースコード静的解析手法を用いたパッチ検証手法の提案," ウィンターワークショップ 2016・イン・逗子 論文集, volume 2016, pages 49-50 2016 年.

6. 安藤 聡志, 伊原 彰紀, 関 浩之, 平尾 俊貴, 則兼 卓人, 松本 健一, "OSS 開発におけるパッチの特徴量を用いた再投稿要求の予測," 第 194 回ソフトウェア工学研究発表会, 2016 年.

7. 則兼卓人, 伊原彰紀, 平尾俊貴, 松本健一, "コードレビューアの指摘記録に基づく開発者の実装技術の成長に関する調査," ソフトウェアエンジニアリングシンポジウム (SES2016), 2016 年.