

Scaling Open Source Communities: an Empirical Study of the Linux kernel

Xin Tan, Minghui Zhou (Peking University) ,
Brian Fitzgerald (University of Limerick)

論文情報

Proceedings of the ACM/IEEE 42nd

International Conference on Software Engineering (ICSE '20).

Association for Computing Machinery, New York, NY, USA, 1222–1234.

紹介者：堀口日向

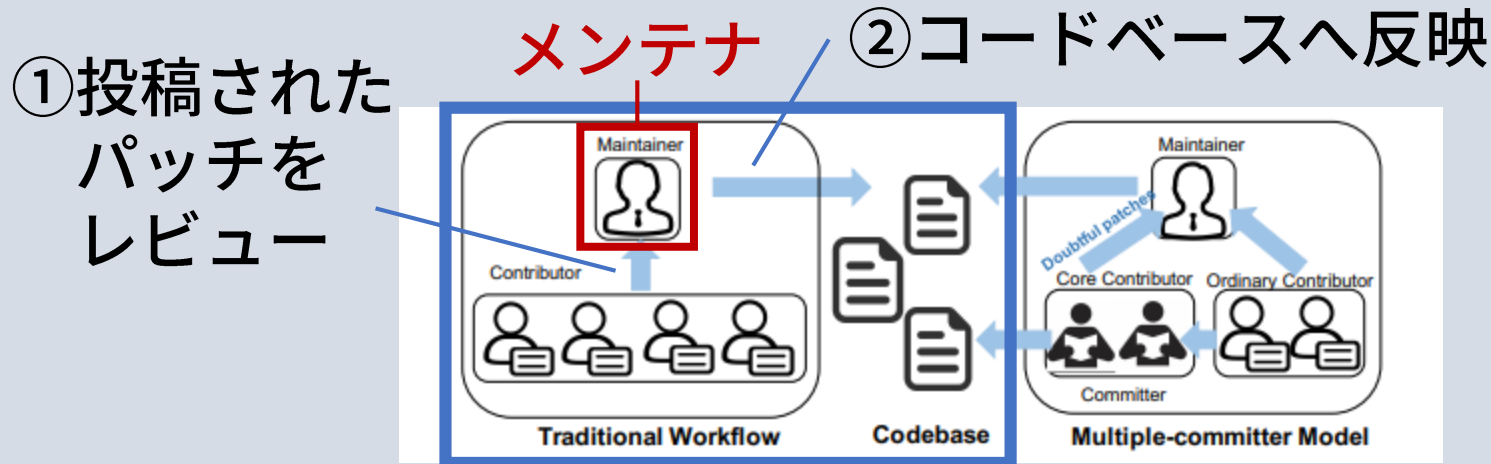
背景 | OSSプロジェクトのスケーリング

- Linux kernelやApache HTTP Serverなどの有名プロジェクトは、年々大規模化，複雑化
 - 例：Linux kernelの場合
 - 1991年（バージョン0.01）： 1万行
 - 2018年（バージョン4.19）：2,558万行
- 一般的なOSSプロジェクトの開発形態
少数のコア開発者 + 大人数の周辺開発者

現状の開発形態で大規模プロジェクトが運営できるのか？

背景 | Linux kernelのワークフローのボトルネック

既存のワークフロー



高品質を維持するため
ほぼすべてのパッチにメンテナが目を通す

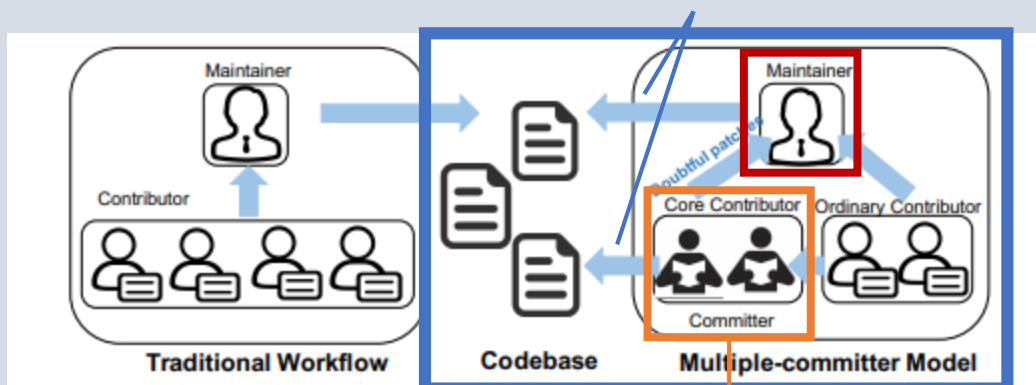
多くの場合メンテナの負担大
⇒ ボトルネック

背景 | ボトルネックへの対処

マルチコミッタモデル (Linux kernelの一部サブシステムで導入)

一部のコントリビュータにコミット権を付与
“コミッタ”にメンテナの負担を分散

メンテナ以外にもパッチをコードベースへ反映できる



コミッタ

コミット権を持つ人が増えると品質低下につながる懸念

目的，リサーチクエスション

目的

マルチコミッタモデルの導入がプロジェクトに与える影響を定量的明らかにし，導入のための支援を行う

リサーチクエスション

- RQ1: マルチコミッタモデルはメンテナの負担をどの程度軽減できるか？
- RQ2: マルチコミッタモデルをどのように導入すればよいか？
 - RQ2.1: 導入を成功させるために重要な要素は何か？
 - RQ2.2: コミッタはどのように選ぶべきか？

RQ1: どの程度を負担を軽減できるか？ | 分析方法

データ

Linux kernelのi915サブシステムのリポジトリ
intel内蔵GPUのドライバを開発

メトリクスの観点

マルチコミッタモデル導入前後でメンテナの作業負担が
どのように変化したかどうかを表す指標（1か月ごとに計測）

- ・メンテナの負荷
- ・レビューの待ち時間
- ・レビューの複雑さ
- ・レビューの品質

（詳細な説明は結果のスライドで）

RQ1: どの程度を負担を軽減できるか？ | 分析方法

回帰分析

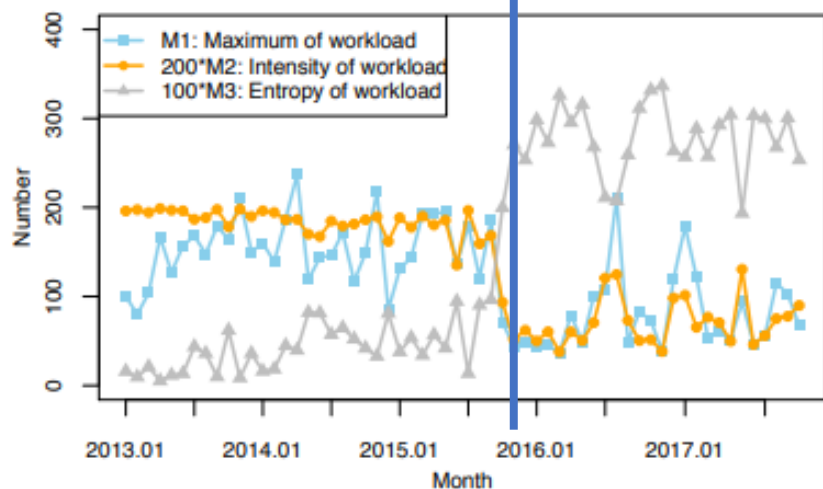
コミットが増えると、メトリクスにどのような影響があるか

- ・説明変数：先述のメトリクス
- ・目的変数：コミット数

RQ1: どの程度を負担を軽減できるか？ | 結果

メトリクス：メンテナの負荷

導入前 導入後



(a) Pressure

統計データ

Metrics	Throughput		
	M1	M2	M3
Mean	154.76, 79.92	0.92, 0.35	0.43, 2.79
Median	150.00, 63.50	0.93, 0.32	0.40, 2.81
P-value/r	7.18E-08/0.69	3.13E-11/0.85	3.03E-11/-0.85

M1: 最も忙しいメンテナが
レビューしたパッチ数

M2: パッチ全体のうち、最も忙しい
メンテナがレビューした割合

⇒ 減少

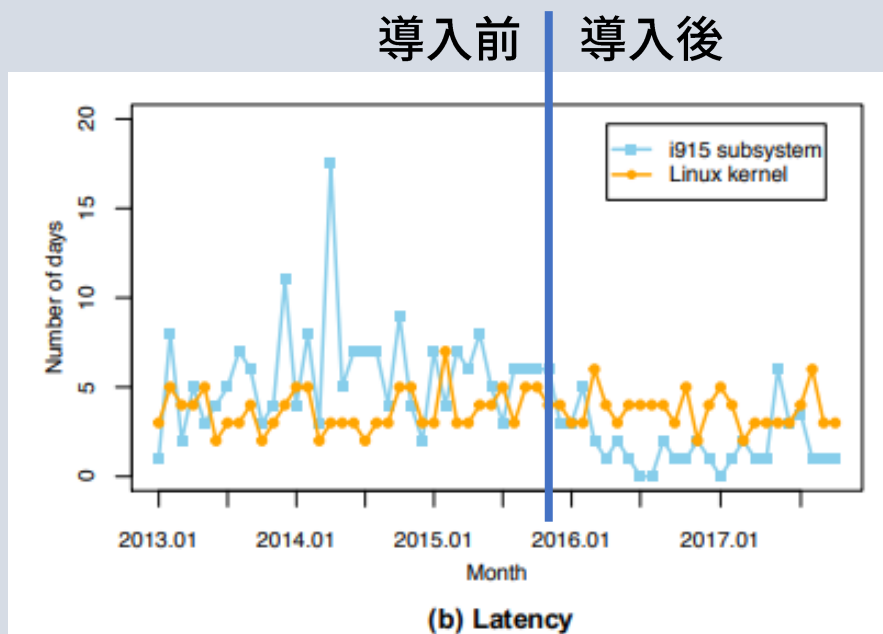
M3: レビューがメンテナまたは
コミッタによって行われるときの
エントロピー

⇒ 増加（つまり、レビューを行う
開発者の偏りが減った）

マルチコミッタモデル導入後
メンテナの負荷は分散された

RQ1: どの程度を負担を軽減できるか？ | 結果

メトリクス：レビューの待ち時間



統計データ

Metrics	Latency
	M4
Mean	5.71, 1.94
Median	5.00, 1.00
P-value/r	1.55E-08/0.68

M4: パッチのレビューにかかる時間の中央値

Linux kernel全体（導入していない）
⇒ 変化なし

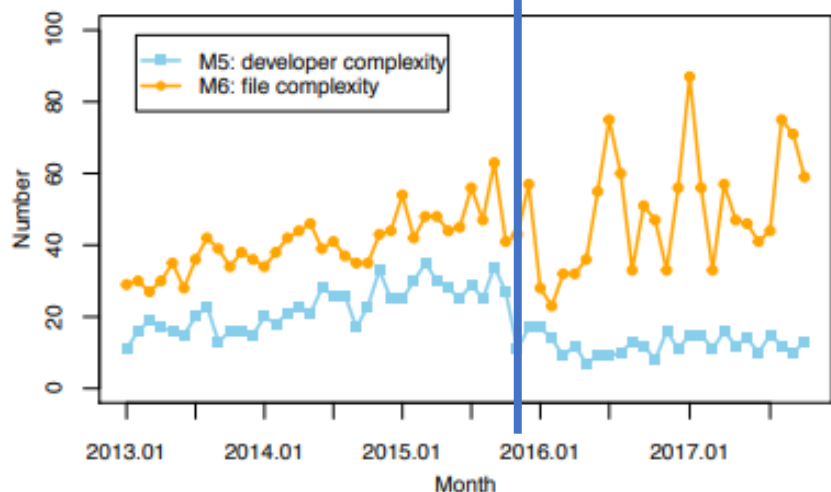
i915サブシステム（導入した）
⇒ 減少

マルチコミッタモデル導入後
レビューの待ち時間は減った

RQ1: どの程度を負担を軽減できるか？ | 結果

メトリクス：レビューの複雑さ

導入前 導入後



(c) Complexity

統計データ

Metrics	Complexity	
	M5	M6
Mean	22.39, 12.23	40.27, 49.12
Median	23.00, 12.00	39.00, 47.00
P-value/r	4.06E-09/0.75	0.019/-0.27

M5: 特定のメンテナがレビューした
パッチ投稿者の最大人数

⇒ 減少

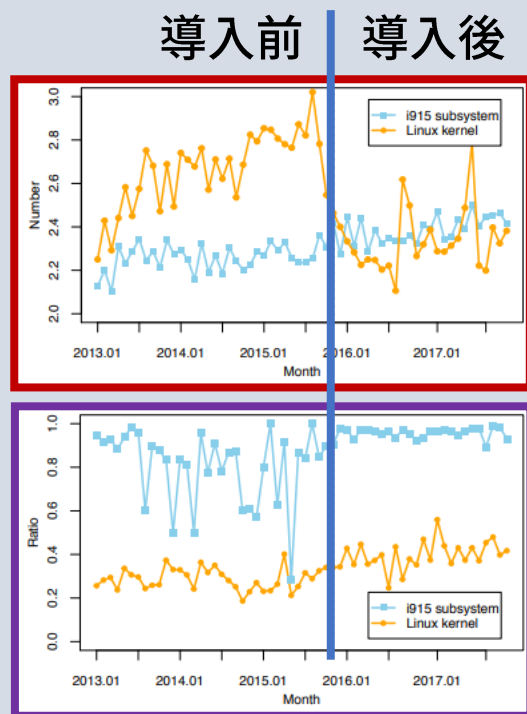
M6: 特定のメンテナがレビューした
ファイル数の最大値

⇒ 変化なし（ばらつきが増えた
ようには見える）

マルチコミッタモデル導入後
レビューの複雑さは下がった

RQ1: どの程度を負担を軽減できるか？ | 結果

メトリクス：レビューの品質



統計データ

Metrics	Quality Assurance	
	M7	M8
Mean	2.26, 2.39	0.80, 0.96
Median	2.26, 2.39	0.87, 0.97
P-value/r	6.45E-09/-0.74	1.44E-06/-0.61

M7: 特定のパッチをレビューするメンテナの平均人数

i915サブシステム ⇒ 増加

Linux kernel全体 ⇒ 減少

M8: メンテナによって投稿されたパッチのうち、ほかのメンテナによってレビューされた割合

i915サブシステム ⇒ 増加

Linux kernel全体 ⇒ 増加

マルチコミットモデル導入後
レビューの品質は上がった

RQ1: どの程度を負担を軽減できるか？ | 結果

回帰分析

Metric	ln(#C)	
	Est	Std.Err
M1	-0.10	0.07
M2	-0.92	0.06
M3	0.92	0.04
M4	-0.64	0.13
M5	-0.91	0.10
M6	-0.11	0.12
M7	0.57	0.12
M8	0.60	0.14

コミット数(#C)が増えると、

- メンテナの負荷
- レビューの遅延 ⇒ 軽減
- レビューの複雑さ
- レビューの品質 ⇒ 向上

グレー：モデルに影響があるメトリクス
(有意差がある)

RQ2.1: 導入を成功させるために重要な要素 | 分析方法

1. Google検索で
 - “multiple committers”
 - “Linux kernel”2つのキーワードを含むWebページを検索
2. 上位50件から，ページ内に埋め込まれたリンクを再帰的にたどる
3. これらのドキュメントを読み，MAXQDA※を使って内容に対しテーマ分析（Thematic Analysis）

※MAXQDA
質的データの分析を
支援するツール



3つのテーマを発見

ほぼ自明なので説明省略 (論文には書いてあります)

RQ2.2に関連する1項目のみを説明

コミッタに求められる特徴

前提：コミット権を分散させることは、品質低下のリスク

- ・技術力の高さ

例：パッチを5~10個提出

- ・コミュニティとの信頼関係

- ・個人の利益より、コミュニティの利益を優先する姿勢

例：いろんな企業からハードウェアドライバなどの貢献
自分の企業を優遇しない

RQ2.2: コミッタの選び方 | 分析方法

コミッタ候補者を予測

- コミッタに必要な要素（RQ2.1より）
 - 技術力の高さ
 - コミュニティとの信頼関係
- コラボレーションネットワークを使った予測
 - ノード：開発者
開発者が作成したコミット数で重みづけ
 - エッジ：パッチ投稿者 – パッチをレビューした開発者
同じ開発者同士でパッチをレビューした，された回数で重みづけ

RQ2.2: コミッタの選び方 | 分析方法

コミッタ候補者を予測

Goldbergが提案したアルゴリズム^[1]で密度が最大になる誘導部分グラフを求める

- ・ 誘導部分グラフ：一部のノードを取り出すとき，ノード間のエッジの有無を元のグラフと同じにする

ある誘導部分グラフPの密度

$$Density = \frac{1}{k} \left(\alpha \sum_{i \in P} w_i + (1 - \alpha) \sum_{i, j \in P} c_{i, j} \right)$$

↑
コミット数の総和
(技術力の高さ)

↑
共同作業回数の総和
(コミュニティとの信頼関係)

メンテナが存在する誘導部分グラフのうち，
密度が最大になる誘導部分グラフに属する開発者をコミッタ候補者と予測

[1] Andrew V Goldberg. 1984. Finding a maximum density subgraph. University of California Berkeley, CA, California, USA.

RQ2.2: コミッタの選び方 | 結果

i915サブシステムにアルゴリズムを適用

2015年9月までのデータを使用して，コミッタ候補者を予測
10月に実際にコミッタになった開発者と比較

実際にコミッタになった8人のうち，5人が一致

Precision	100.00 %
Recall	62.50 %

RQ2.2: コミッタの選び方 | 検証

Linux kernel内（i915以外）での検証

- 検証方法

1. Linux kernelのサブシステムの中から，最も忙しい上位50個を選択
 2. コミッタ候補者を予測するアルゴリズムを適用
 3. サブシステムのメンテナにアンケート
 - ・ マルチコミッタモデルを導入したいと思うか？
 - ・ アルゴリズムが予測したコミッタ候補者は適切か？
- 8つのサブプロジェクトが回答

RQ2.2: コミッタの選び方 | 検証

Linux kernel内（i915以外）での検証

- 結果
 - 8中5つ：コミッタ候補者は適切
 - 8中3つ：マルチコミッタモデルの導入に消極的
 - メンテナとは別にパッチ検証のチームが存在
 - メンテナ自身が過負荷と感じていない

i915以外のサブプロジェクトにも適用可能

RQ2.2: コミッタの選び方 | 検証

Linux kernel以外（Node.js）での検証

- Node.jsを対象にした理由
 - 急速なスケーリング問題を経験（2015年以降急速にスケール）
 - Linux kernelのようなパッチ方式ではなく，プルリクエスト方式
 - ⇒ 違う貢献方式のプロジェクトにも適用できるか検証可能
- 検証方法

2015年1月～3月（急速にスケールした時期）までのデータを使ってコミッタ候補者を予測し，
実際のコミッタと比較

RQ2.2: コミッタの選び方 | 検証

Linux kernel以外（Node.js）での検証

- 結果

35人のコミッタのうち，13人をコミッタ候補者として予測

Precision	100.00 %
Recall	37.14 %

Linux kernel以外のプロジェクトにも適用可能

- OSSプロジェクトの大規模化，複雑化の問題
現状のワークフローではメンテナの負担が大きく，
プロジェクト運営に影響
問題に苦しんでいるLinuxカーネルの実証的研究
- Linux kernelのi915サブシステムで採用されている
マルチコミッタモデルを分析（RQ1）
⇒ メンテナの負担を削減可能
- マルチコミッタモデル導入を支援するため，
コミッタ候補者予測のアルゴリズムを提案（RQ2.2）
⇒ 予測が適切であることを検証

- 新しいモデルの提案ではなく，OSS開発の現場で実践されているモデルの評価なので，研究はやりやすそう
ただし，そこまで普及しているわけではないが，
有用性がありそうなモデルを見つけるのは難しい・・・
- i915以外のLinux kernelにコミッタ候補者予測を適用した結果（PrecisionやRecall）が載っていなかった
アンケートの回答での評価が良かったから載せなかったのか，
不都合な結果が出たから載せなかったのか・・・

Appx. RQ1: どの程度を負担軽減できるか？ | 分析方法

メンテナ，コミッタの特定

- Linux kernelのコミットメッセージ末尾には，作業した人の名前とメアドを記した”Signed-off-by”行を含むのがルール

```
Signed-off-by: XXXX XXXX <xxx@xxx.com>
```

- パッチ投稿者以外の名前が記載されている場合
→ レビューが行われたと判定
- リポジトリのMAINTAINERSファイルに名前が，
 - 記載されている場合 → メンテナと判定
 - されていない場合 → コミッタと判定