

モジュールオーナーを特定するための 予測モデルの構築

M3 山谷 陽亮
2016/08/09

背景

- オープンソースソフトウェア(OSS)の普及によりOSSのユーザーが年々増加している
- 大規模OSSプロジェクトでは日々大量の不具合が報告されている[1]
- 報告された不具合はボランティアの開発者が中心となり修正が行われている

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should x this bug?
In Proceedings of the 28th International Conference on Software Engineering (ICSE'06), pp. 361--370, 2006.

不具合修正プロセス

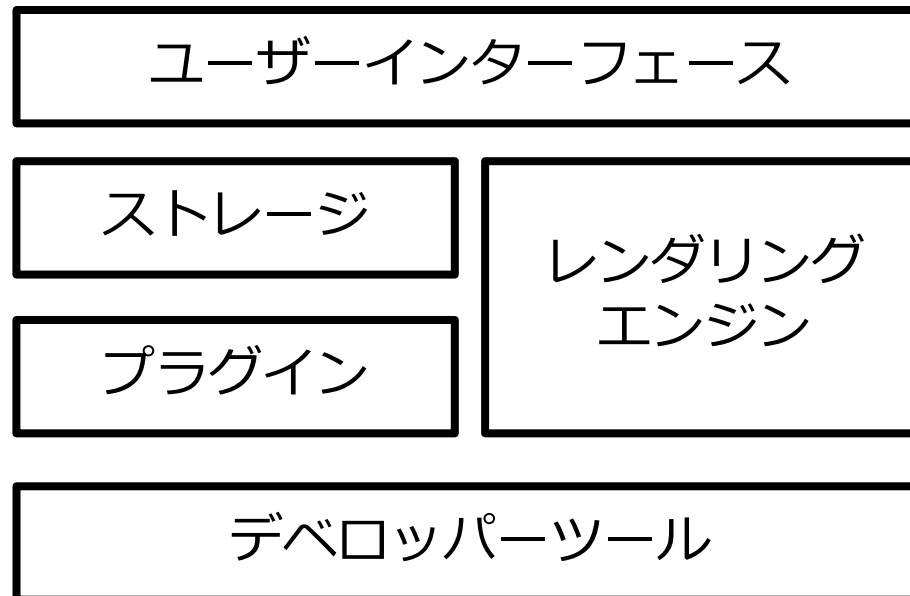
- 不具合を修正するためにパッチが投稿される
 - パッチ：コードの修正前後の差分を示すファイル
- モジュールオーナーがパッチを検証する
 - モジュールオーナー：コミット権限を持つ開発者



ソフトウェアモジュール

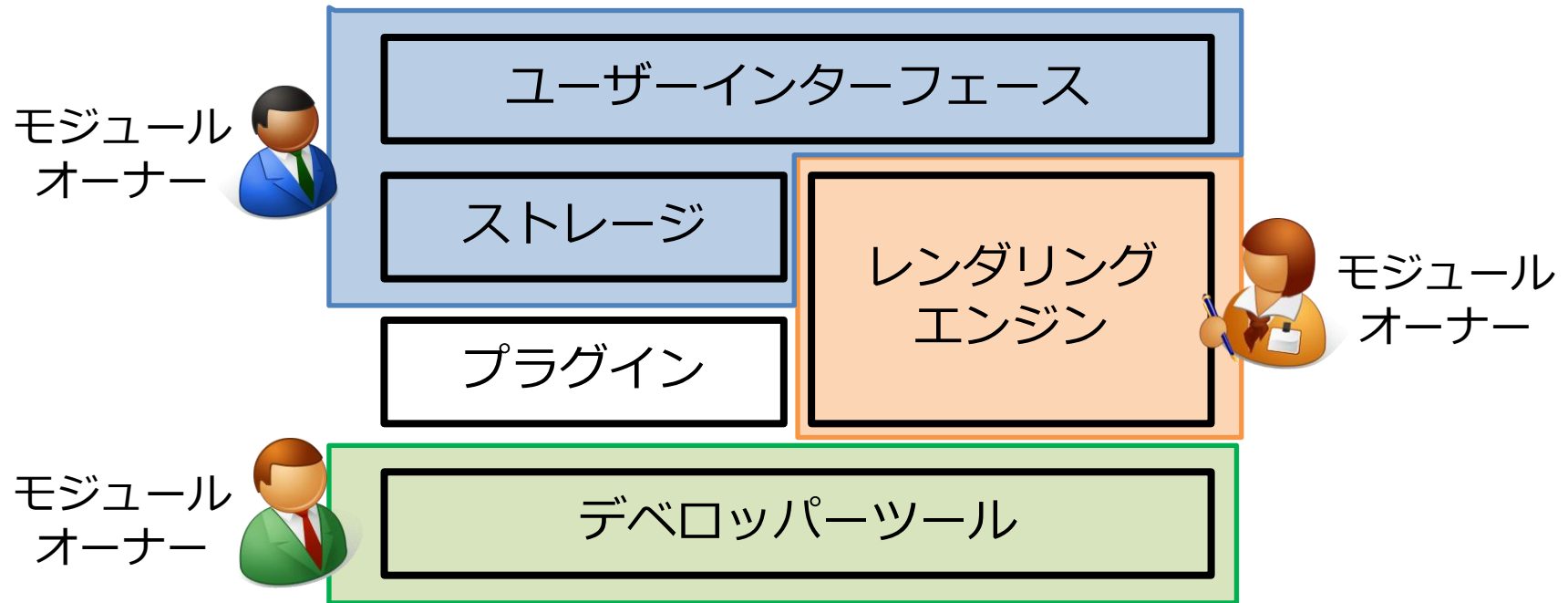
- ソフトウェアは保守性や再利用性を高めるために複数のモジュールが組合わさり構成される
 - モジュール：ひとまとまりの機能をもつ部品

例：ウェブブラウザのモジュール



モジュールオーナー

- 大規模OSSプロジェクトでは各開発者が担当すべきモジュールが決められている
 - モジュールオーナー**：特定のモジュールを担当するコミット権限をもつ開発者



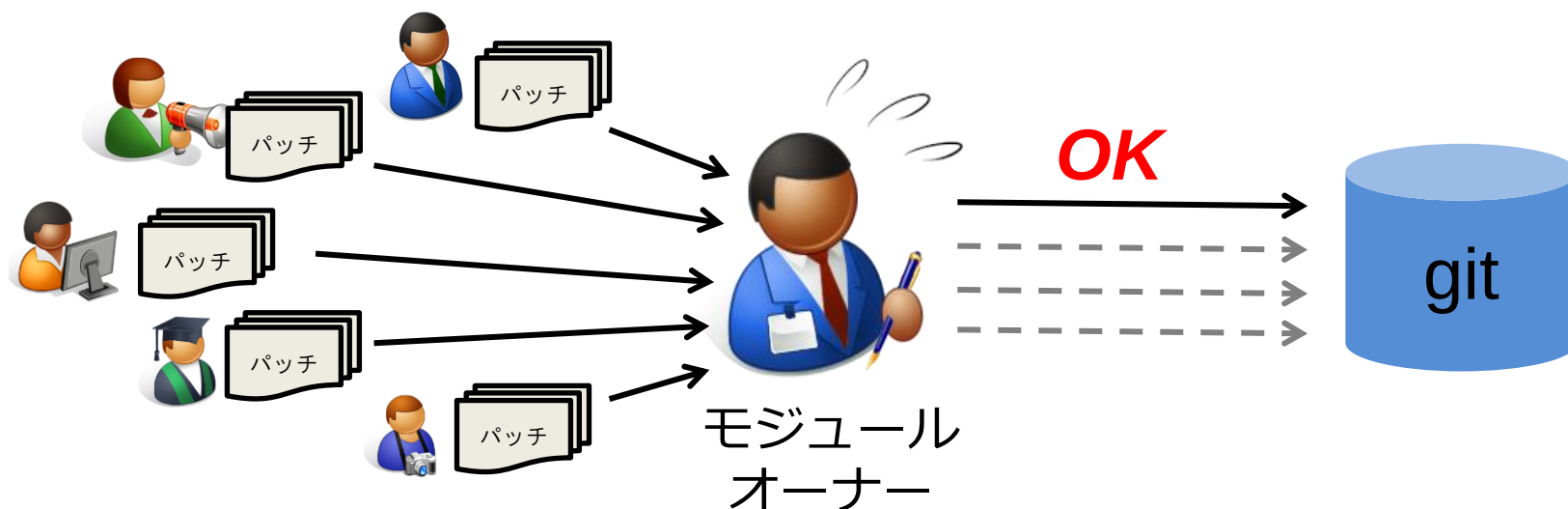
モジュールオーナー昇格

- モジュールオーナーはソフトウェアの品質管理の面で重要な役割を担っている
- 開発者にコミット権限を付与するかどうかの判断は既存のモジュールオーナーが慎重に行う [2]
- コミット権限を持たない開発者に比べモジュールオーナーは少人数で構成されている

[2] C. Jensen and W. Scacchi. Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. In Proceedings of the 29th International conference on software engineering (ICSE'07), pp.364-374, 2007

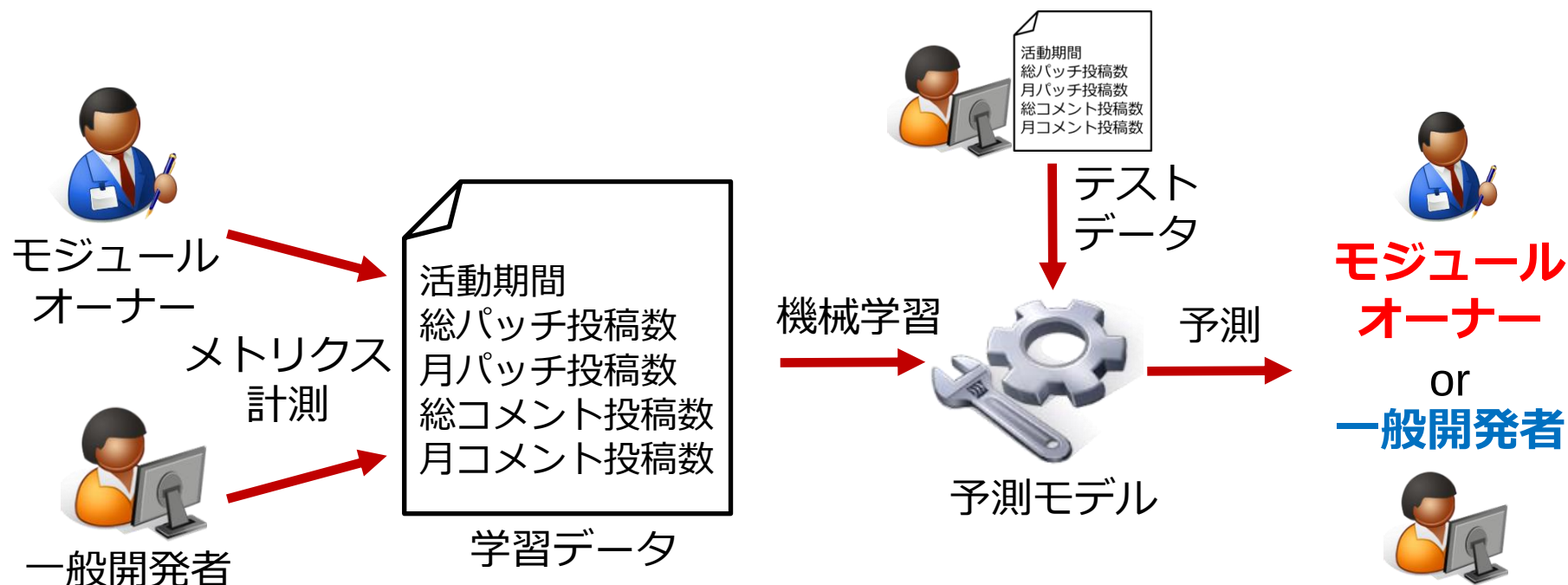
動機：モジュールオーナー不足

- OSSプロジェクトでは少人数で構成されるモジュールオーナーに過度な負担がかかっている
- モジュールオーナーに過度な負担がかかるとパッチの検証を効率的に行うことができない



先行研究

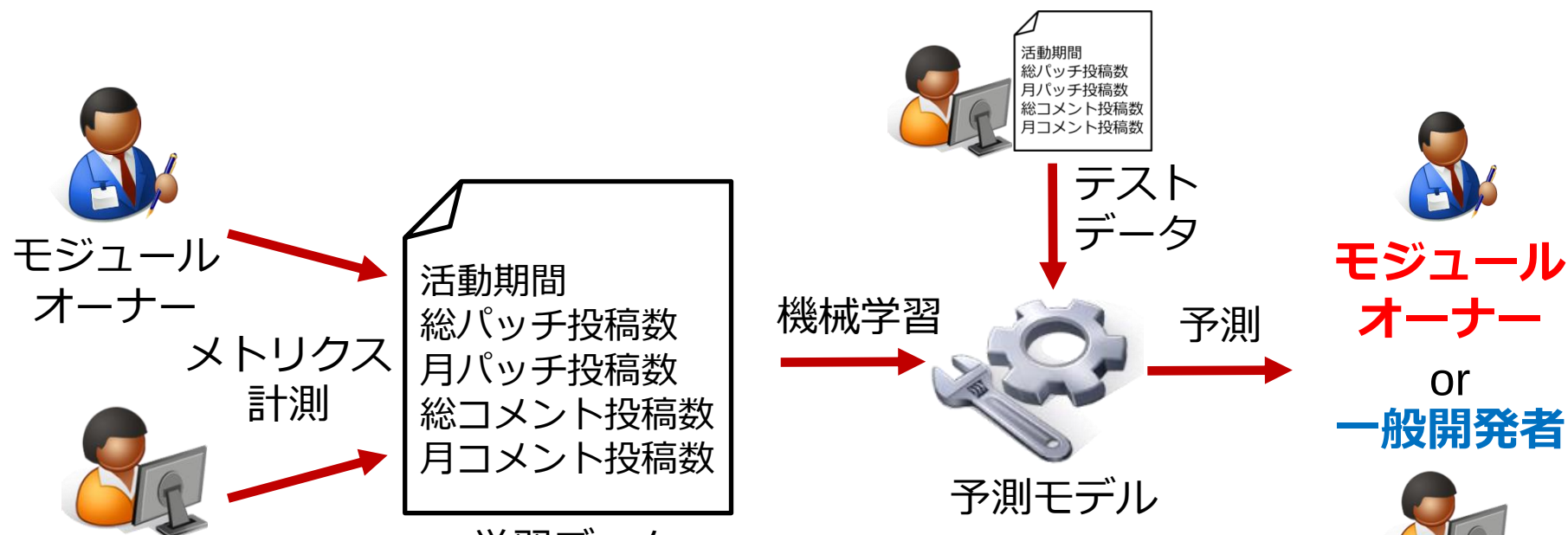
- モジュールオーナー不足を解消するためにモジュールオーナーに昇格可能な開発者を予測 [3]



[3] Akinori Ihara, Yasutaka Kamei, Masao Ohira, Ahmed E. Hassan, Naoyasu Ubayashi, and Kenichi Matsumoto. Early identification of future committers in open sourcesoftware projects. In Proceedings of the 14th International Conference on Quality Software (QSIC'14), pp. 47--56, 2014.

先行研究

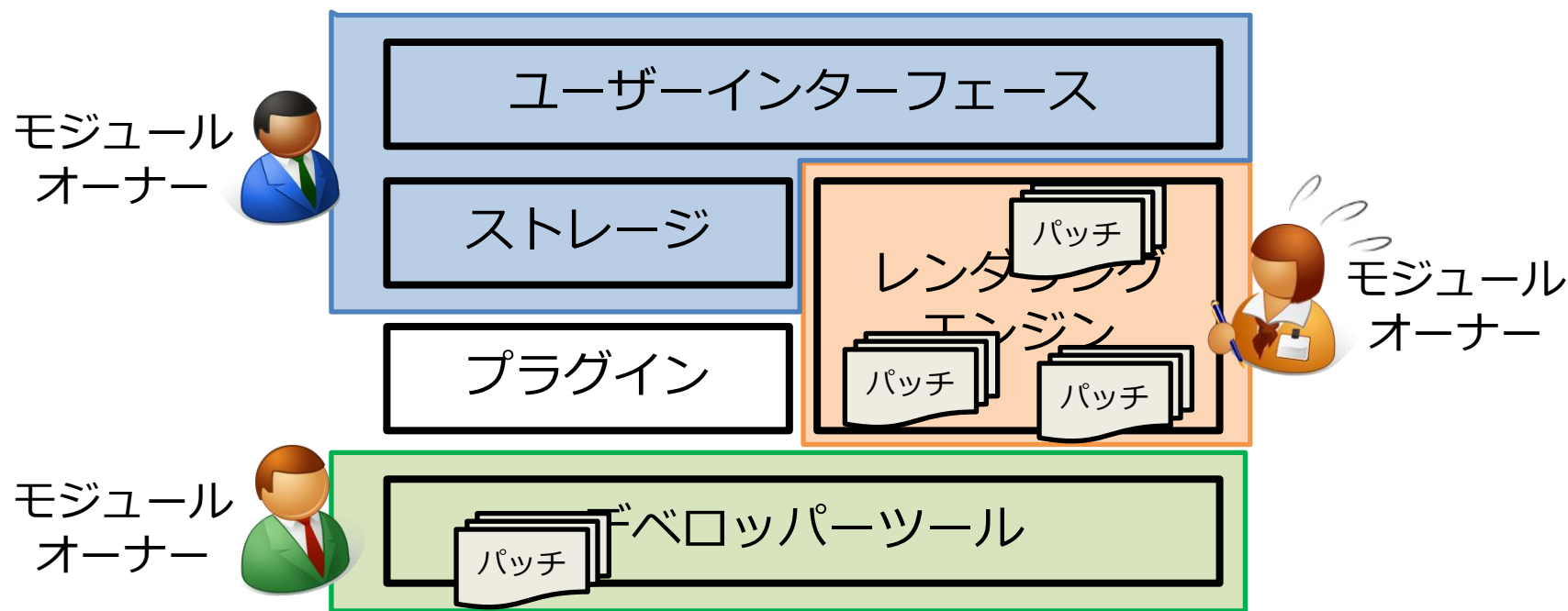
- モジュールオーナー不足を解消するためにモジュールオーナーに昇格可能な開発者を予測 [3]



不具合修正の効率化のためにはモジュールごとにモジュールオーナーを予測する必要がある

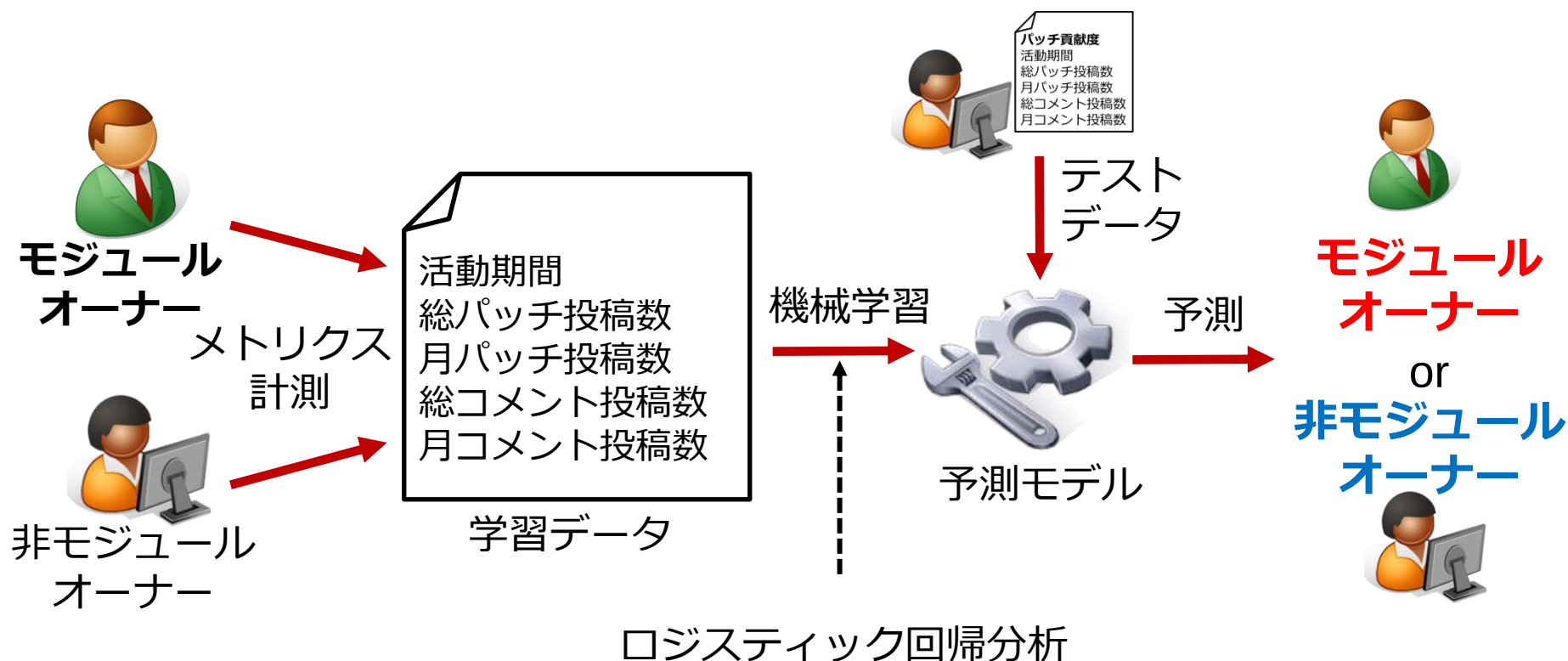
モジュールオーナー候補者

- 不具合修正の効率化には人手が不足している**モジュールのオーナーに適した開発者（モジュールオーナー候補者）**を推薦することが必要となる



モジュールオーナー候補者予測

- モジュールごとの活動量メトリクスを用いてモジュールオーナー候補者を予測する



モジュールごとの活動量メトリクス

- 先行研究で用いられている活動量メトリクスをモジュールごとに計測する
- モジュールごとに計測される活動量メトリクス

メトリクス名	内容
総パッチ投稿数	各開発者が特定のモジュールに対して投稿したパッチの回数
月パッチ投稿数	各開発者が特定の1か月にパッチを投稿した回数の中央値
総コメント投稿数	各開発者が特定のモジュールに対して投稿したコメントの回数
月コメント投稿数	各開発者が1か月にコメントを投稿した回数の中央値
活動期間	各開発者が活動（パッチ・コメントの投稿）した日数

統計処理

- 各メトリクス値の分布はべき分布に従うため、各メトリクス値の対数変換を行う

$$value' = \log_e(1 + value)$$

value : 各メトリクスの値

value' : 対数変換された各メトリクスの値

- モジュールオーナーに昇格した開発者と一般開発者の外れ値をもつデータを除去する
 - 第3四分位数に四分位範囲の1.5倍を足した値を超えている要素を外れ値として判断

評価実験

- 実験の目的
 - モジュールごとにモジュールオーナー候補者を予測することができるのかを確かめること
- 対象プロジェクト

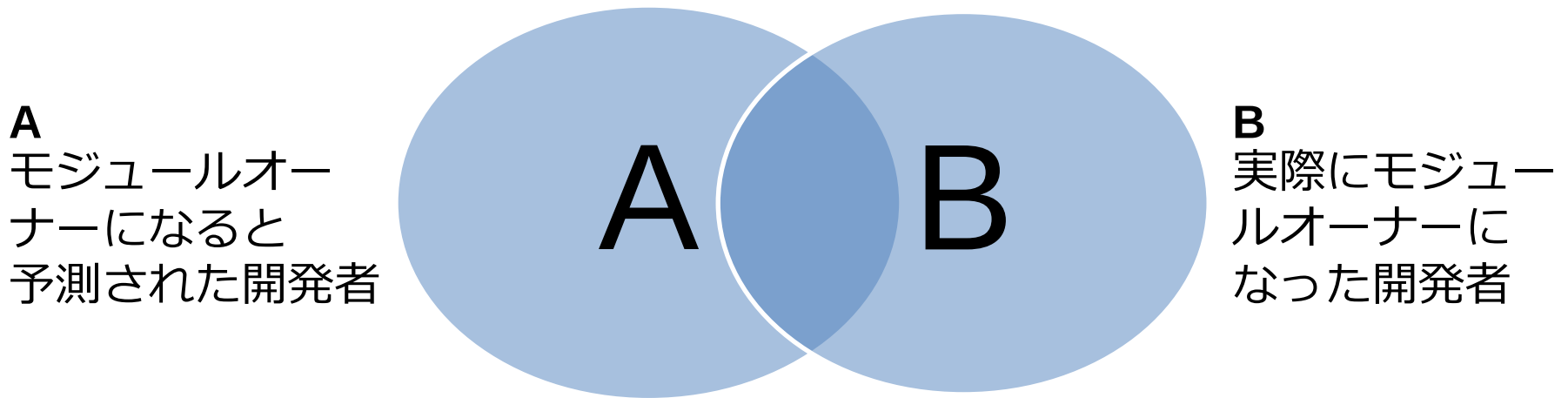
プロジェクト	期間	モジュール オーナー	一般開発者
Eclipse Platform	2001/10 ~2012/10	195	13,297
Mozilla Firefox	1999/07 ~2013/01	530	76,931
GCC	1999/07 ~2014/03	141	10,282
GIMP	2001/02 ~2013/09	87	4,068
WebKit	2000/12 ~2014/04	363	8,017

モジュールオーナーの特定方法

1. コミットログに含まれる不具合番号からどのモジュールに対する変更なのかを取得する
2. 1の結果からモジュールごとに変更されやすいソースファイルの位置を特定する
3. 2の結果から、モジュールごとに特定されたソースファイルをコミットしている開発者をモジュールオーナーとみなす

評価指標

- 予測モデルを適合率, 再現率, F1値で評価する



$$\text{適合率} = \frac{A \cap B}{A}$$

$$\text{再現率} = \frac{A \cap B}{B}$$

$$\text{F1値} = \frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}}$$

実験手順

1. モジュールオーナーと一般開発者それぞれのデータを2分割し、学習データとテストデータを作成する
2. モジュールオーナーの数だけ一般開発者の学習データからランダムに抽出し学習
3. テストデータの全てで予測精度を算出する
4. 1から3の手順を1,000回繰り返す

結果：予測精度

モジュール名	適合率	再現率	F1値
App	0.571	0.563	0.552
Bar	0.500	0.750	0.590
Bookmark	0.480	0.560	0.511
DeveloperTools	0.800	0.786	0.800
General	0.455	0.585	0.511
Preferences	0.552	0.613	0.587

- 適合率 0.455 ~ 0.800
再現率 0.560 ~ 0.786 という結果になった

予測に寄与するメトリクス

モジュール	活動期間	総パッチ	月パッチ	総コメント	月コメント	切片
App	0.420	1.780	NA	-0.459	27.477	-3.846
Bar	0.267	1.548	NA	0.834	-1.271	-7.077
Bookmark	0.037	1.224	0.041	0.419	-0.528	-3.048
DeveloperTools	11.461	7.975	-4.218	-8.126	8.114	-54.385
General	-0.063	0.741	-3.643	0.435	0.657	-2.212
Preferences	0.708	1.660	NA	-0.453	1.904	-5.839

- モジュールごとに結果は異なるが**総パッチ投稿数**および**月コメント投稿数**が予測に寄与する

考察

- 予測をはずした開発者に対して分析を行う
 - Q. モジュールオーナーである開発者をなぜモジュールオーナーでないと判断したのか？
 - 依存関係にあるモジュールで活発に活動していた
 - App : 5人 (7人中) , Bar : 2人 (4人中)
 - Q. モジュールオーナーではない開発者をなぜモジュールオーナーと判断したのか？
 - 不具合管理システムとコミットログで異なるメールアドレスを用いていることから, モジュールオーナーでないと判断した開発者の中にモジュールオーナーが存在していた

今後の予定

- 依存関係にあるモジュールでの活動量を考慮した予測モデルの構築
- 予測精度に関する議論
 - コミット権限を持たない開発者に比べてモジュールオーナーは少数であるため, 2人に1人の確率でモジュールオーナーを当たることは有用である
- 執筆 (今週中まで)

Done

- スライド作成
- 2章執筆
- 閾値を変えた場合での追加分析

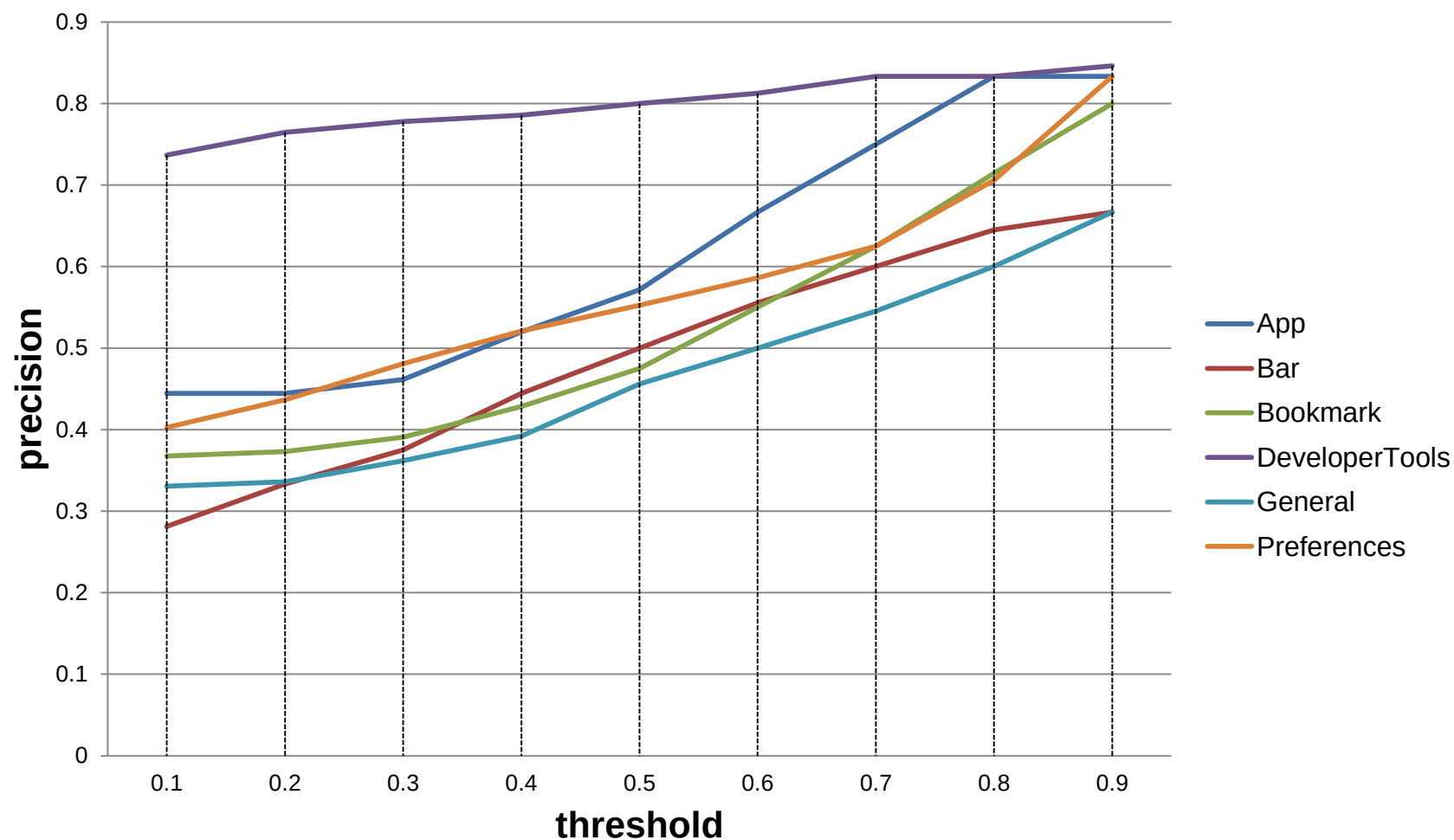
外れ値除去

- なぜ外れ値を除去するのか？
 - 外れ値がデータセットの中にあると外れ値に強い影響を受けて極端なモデルが構築されてしまうため
 - 外れ値をもつような開発者は、特殊な理由により昇格している可能性が高いため今回のデータから除く
- 外れ値除去方法
 - 一般的な外れ値除去の手段をとる
 - 箱ひげ図をプロットしひげの外側にあるデータを外れ値とみなす（Turkeyによって提案）

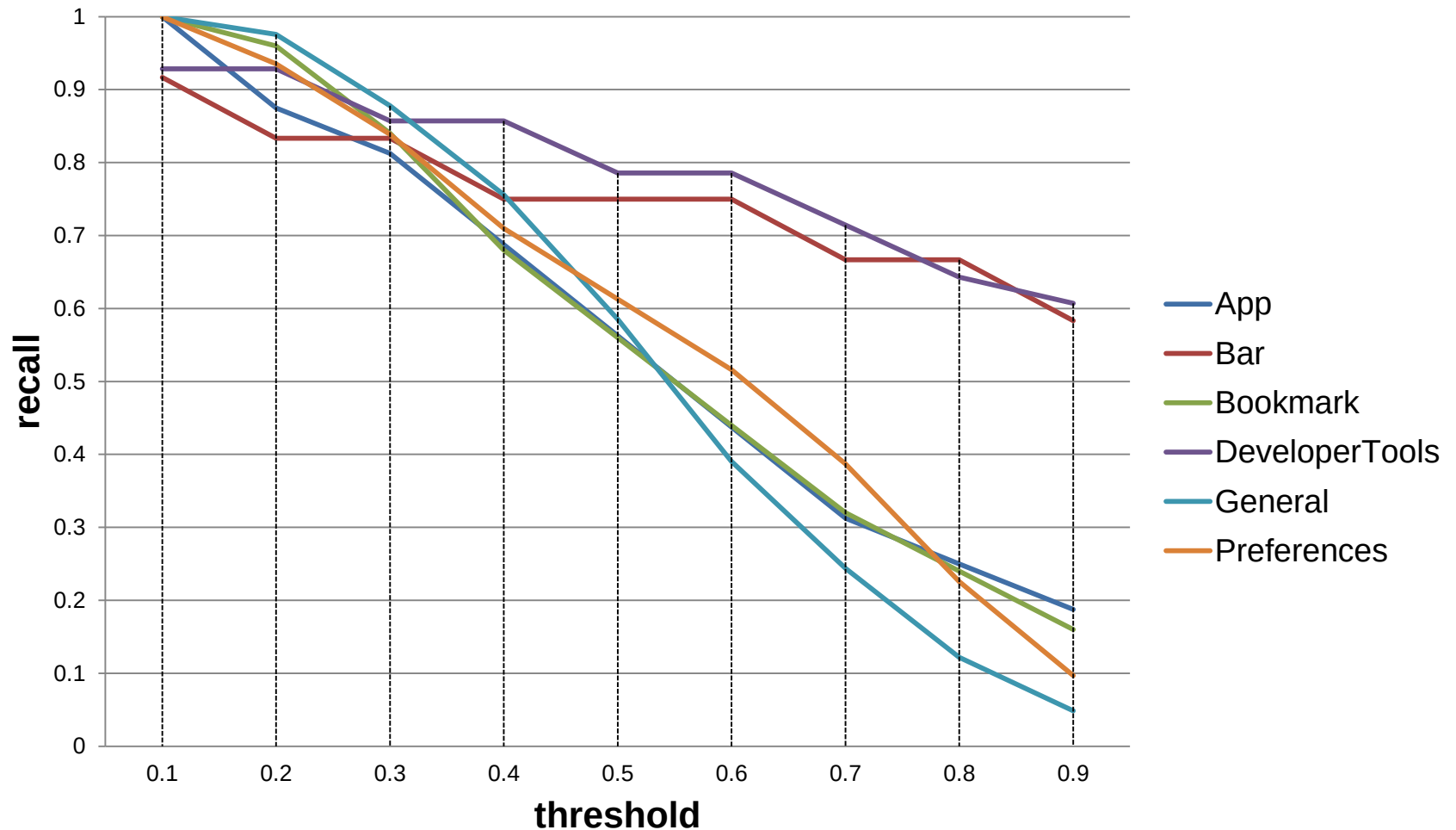
考察

- 予測精度に関する考察
 - ロジスティック回帰分析の結果, 0.5以上の値を返した場合, 昇格すると判断していたが, この閾値を変化させることによる予測結果を考察する
 - 方法: 閾値を変化させて同じように予測をする

結果：Precision



結果：Recall



考察

- 閾値の値を高く設定するとRecallが低くなるがPrecisionが高くなり、閾値の値を低く設定すると、Recallが高くなるが、Precisionが低くなる
- 閾値を0.9に設定すると、App, Bar, Bookmark, DeveloperToolsモジュールでPrecisionが0.8を超える

TODO

- 執筆

