

OSS開発における 不具合修正タスクの割当支援

オープンソースソフトウェア工学研究室（大平研）

60140027

柏 祐太郎

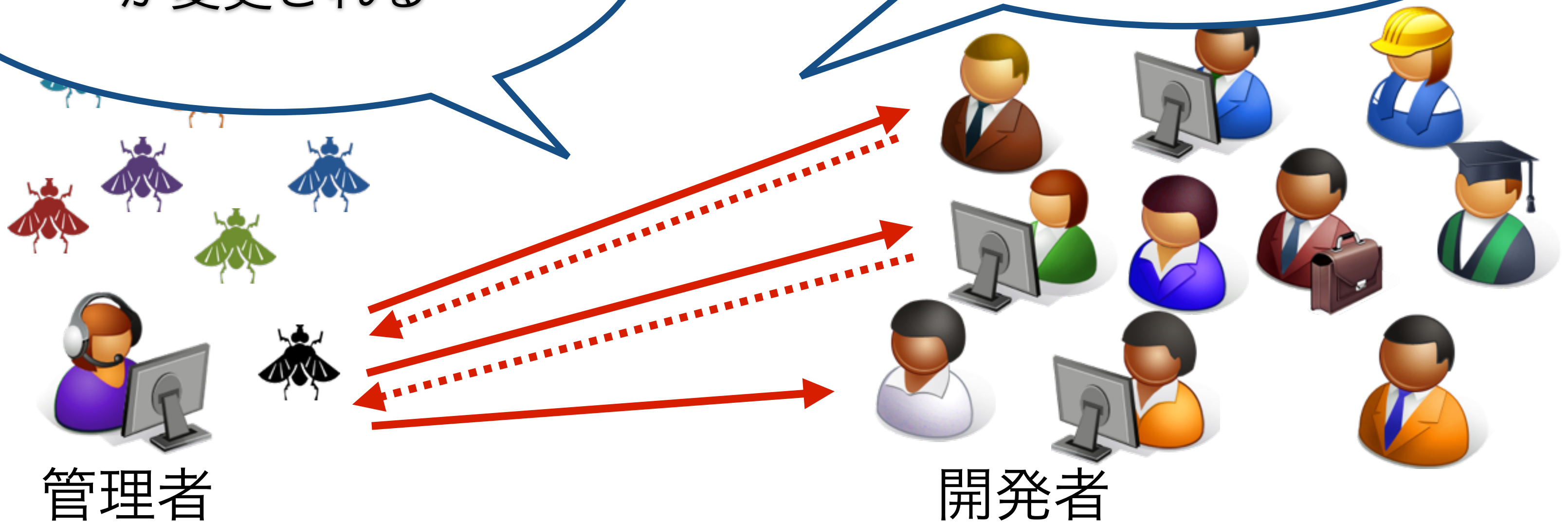
現状のタスク割当て手法と問題点

2

- 管理者が不具合報告の内容を精査し、担当の開発者を決定

44%の不具合が担当者が変更される

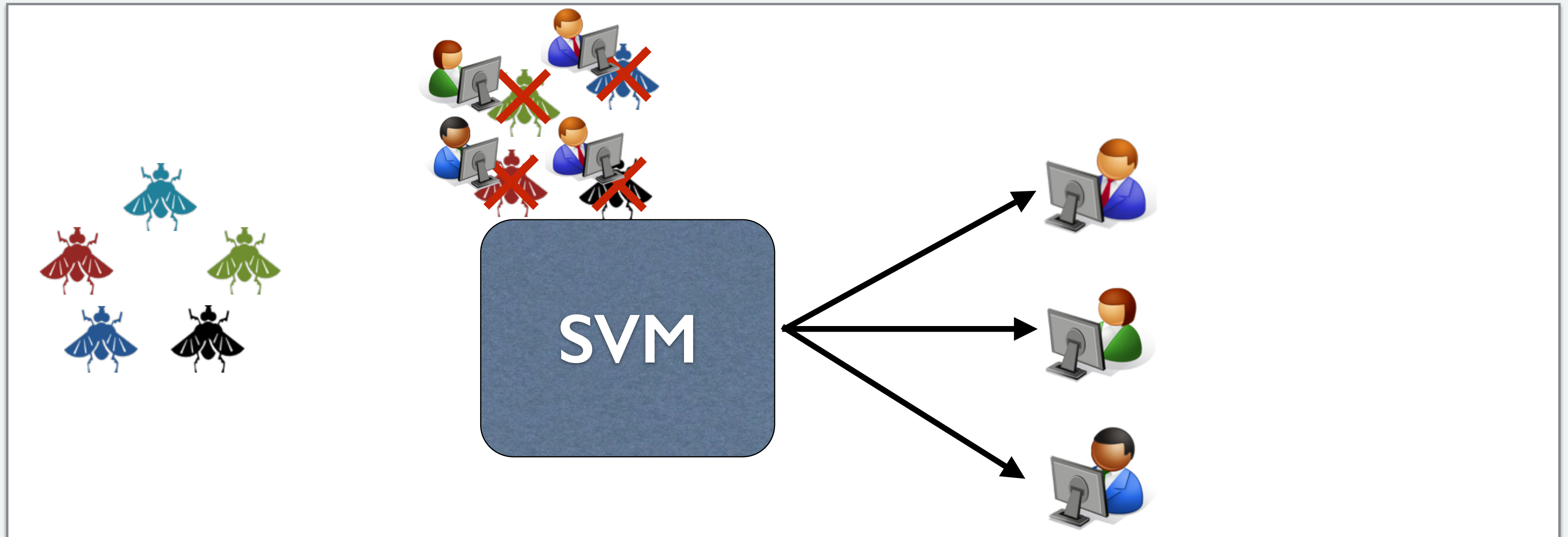
1度の担当者の変更で修正時間が平均50日遅延



不具合の担当者変更（再割当）の頻発により不具合修正が長期化

既存手法とその問題点

■ 機械学習（SVM）を用いたタスク割当て



○ メリット

- 適任の開発者に割当てることができる

× デメリット

- 一部の開発者にタスクが集中する傾向

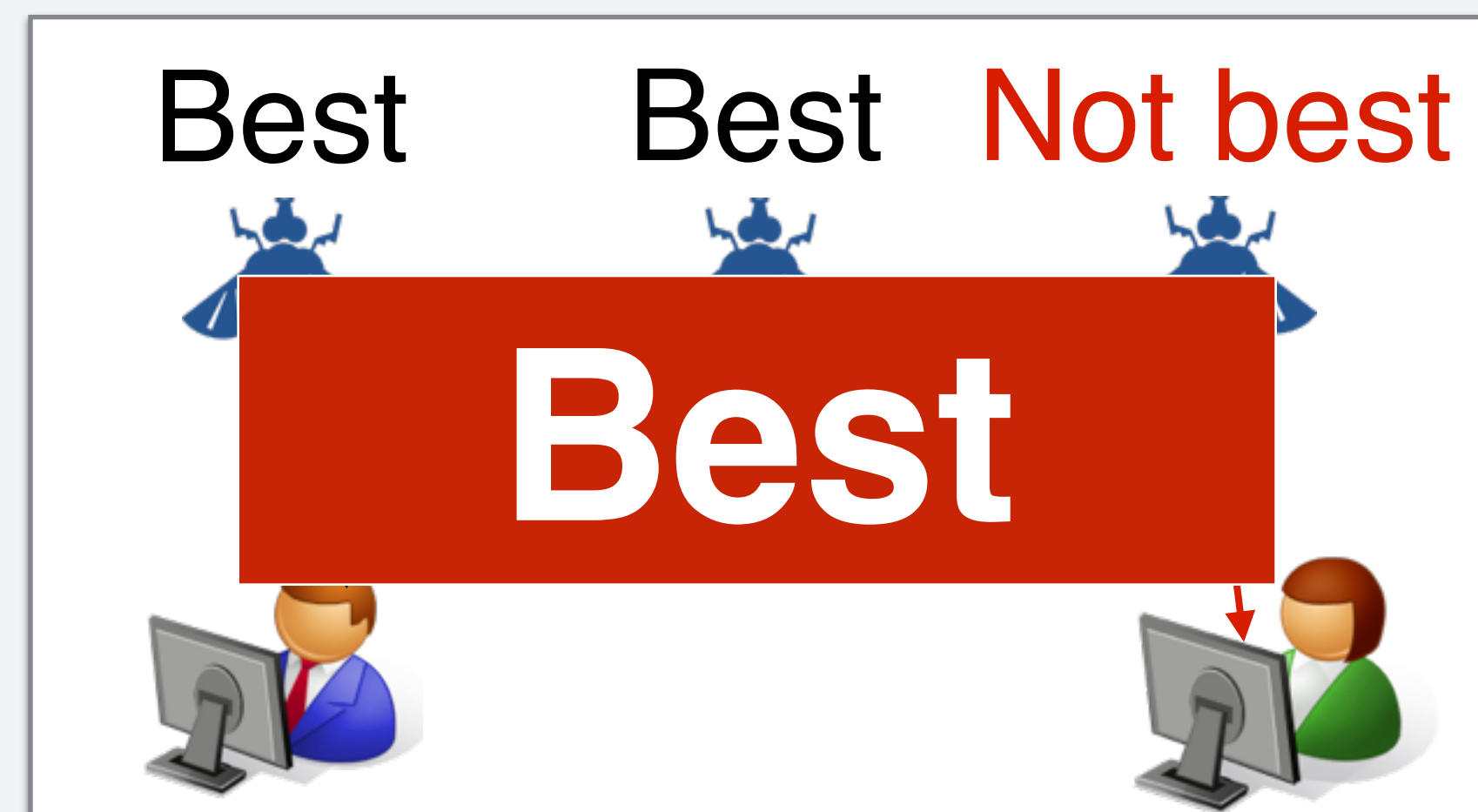
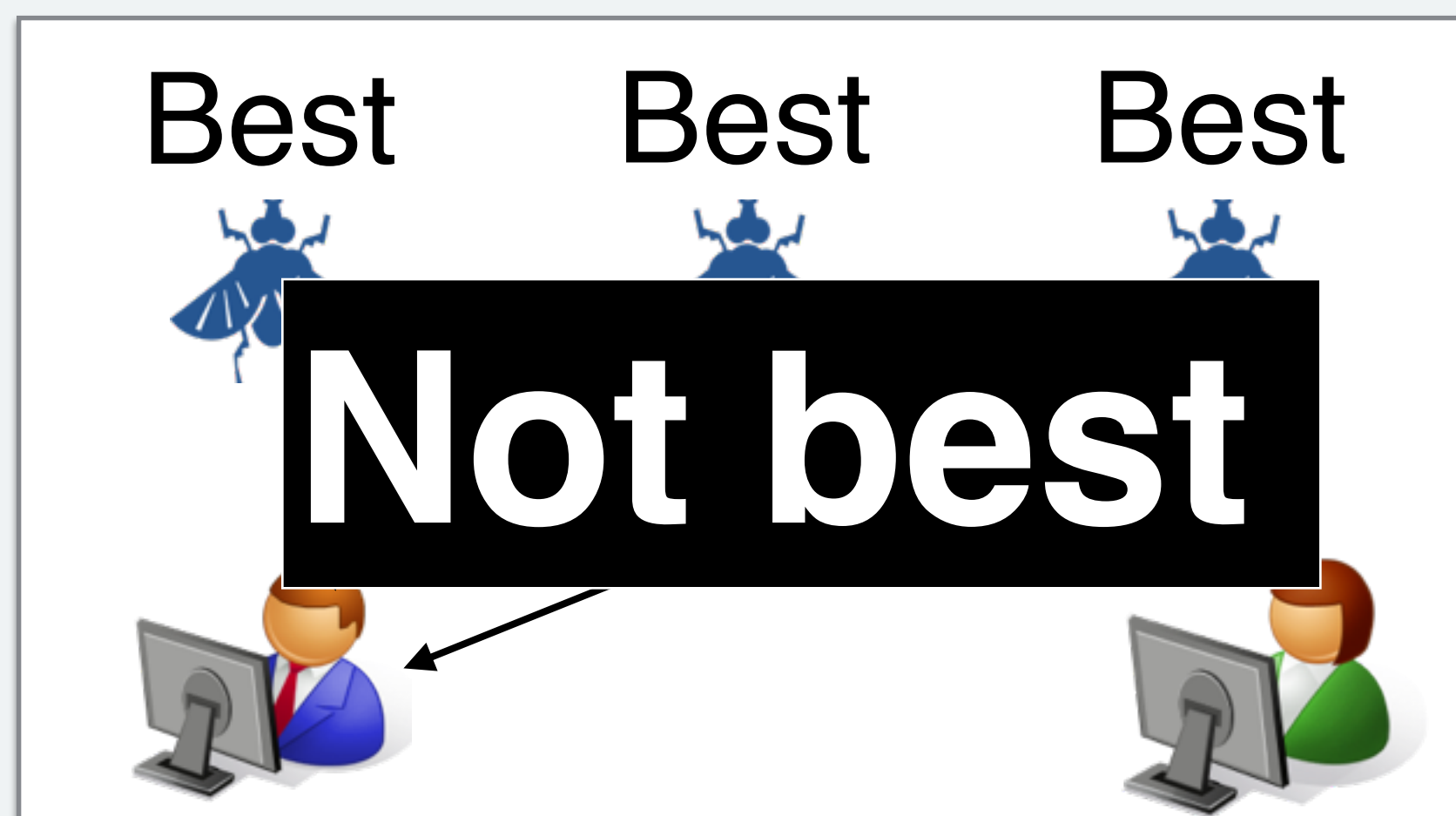
既存手法とその問題点

4

非効率的

- 軽微な不具合でも優先して有能な開発者に割当てる可能性
- 有能な開発者が修正に使える時間は限られている

タスク過多は再割当の原因



不具合と開発者の1対1で最も効率が良い組合せでなく
プロジェクト全体で最も効率が良い組合せを求めるべき

本研究の目的とアプローチ

目的

- プロジェクト全体で修正効率が高くなる割当て手法を構築
 - 個々の修正タスクに対する開発者の適性を考慮
 - 一定期間に開発者が修正作業に使える時間を考慮

アプローチ

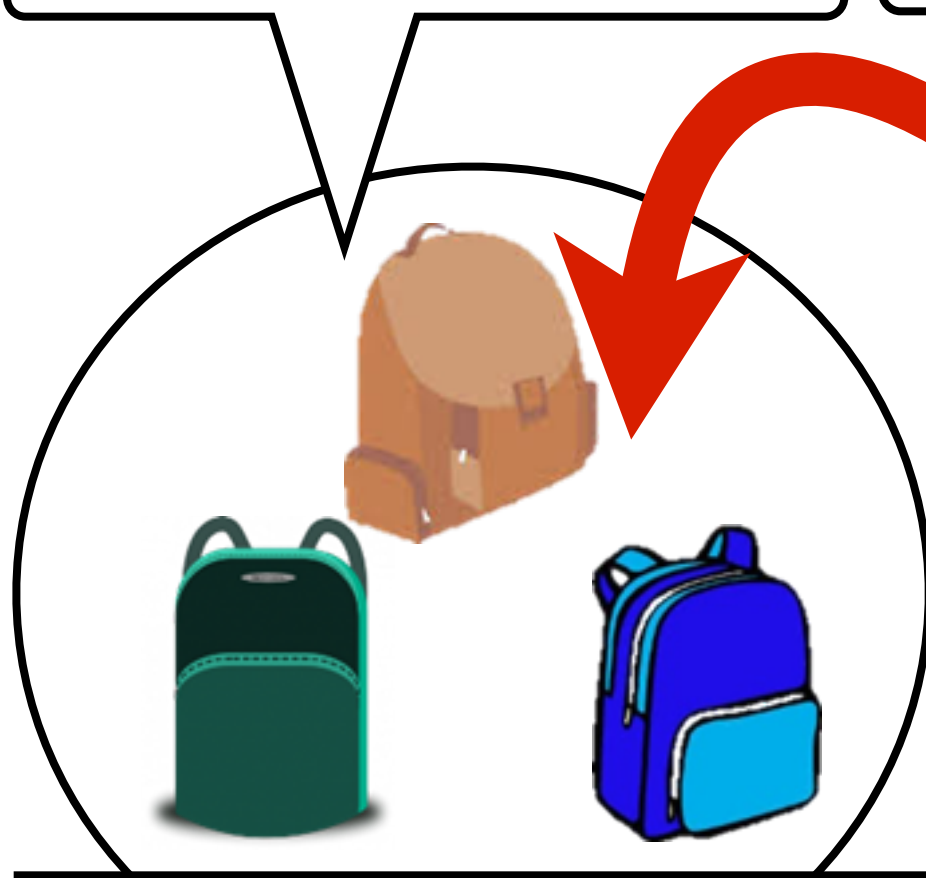
- 不具合と開発者の割当て問題をマルチナップサック問題としてみなして、最適な組合せを求める

マルチナップサック問題

- ナップサック集合に入れるアイテムの総利得を最大にするアイテムの組合せを求める問題

ナップサック i
最大重量 (C_i)
: 4kg

アイテム j
利得 (v_j): 10万円
重量 (w_j): 3 Kg



ナップサック集合



目的関数

$$\text{Maximize : } \sum_{i=1}^m \sum_{j=1}^n v_j x_{ij}$$

$$x_{ij} \in \{0, 1\}$$

制約条件

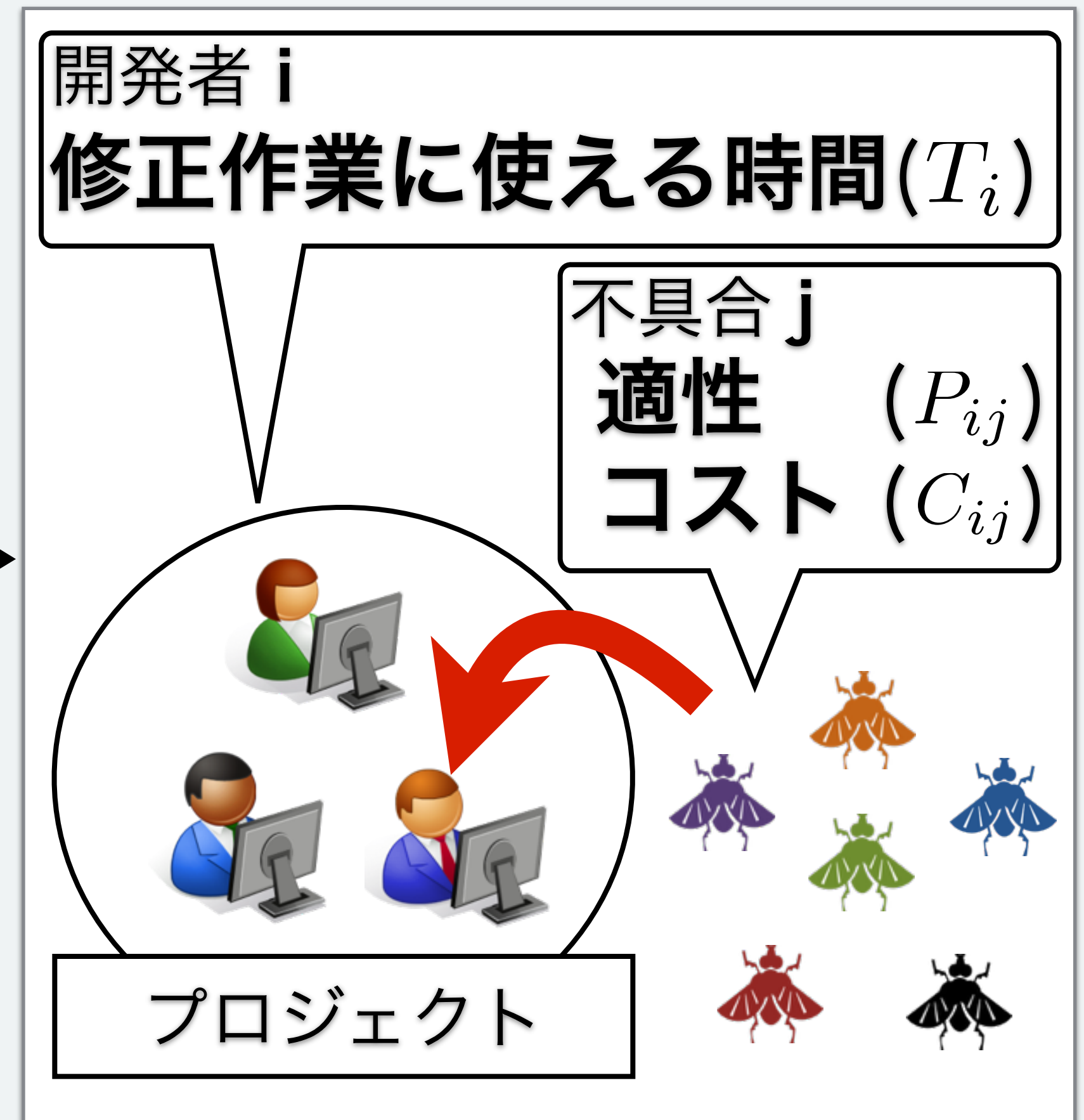
- $$\sum_{j=1}^n w_j x_{ij} \leq C_i$$

$$(i = 1, \dots, m)$$
- $$\sum_{i=1}^m x_{ij} \leq 1$$

$$(j = 1, \dots, n)$$

キーアイデア

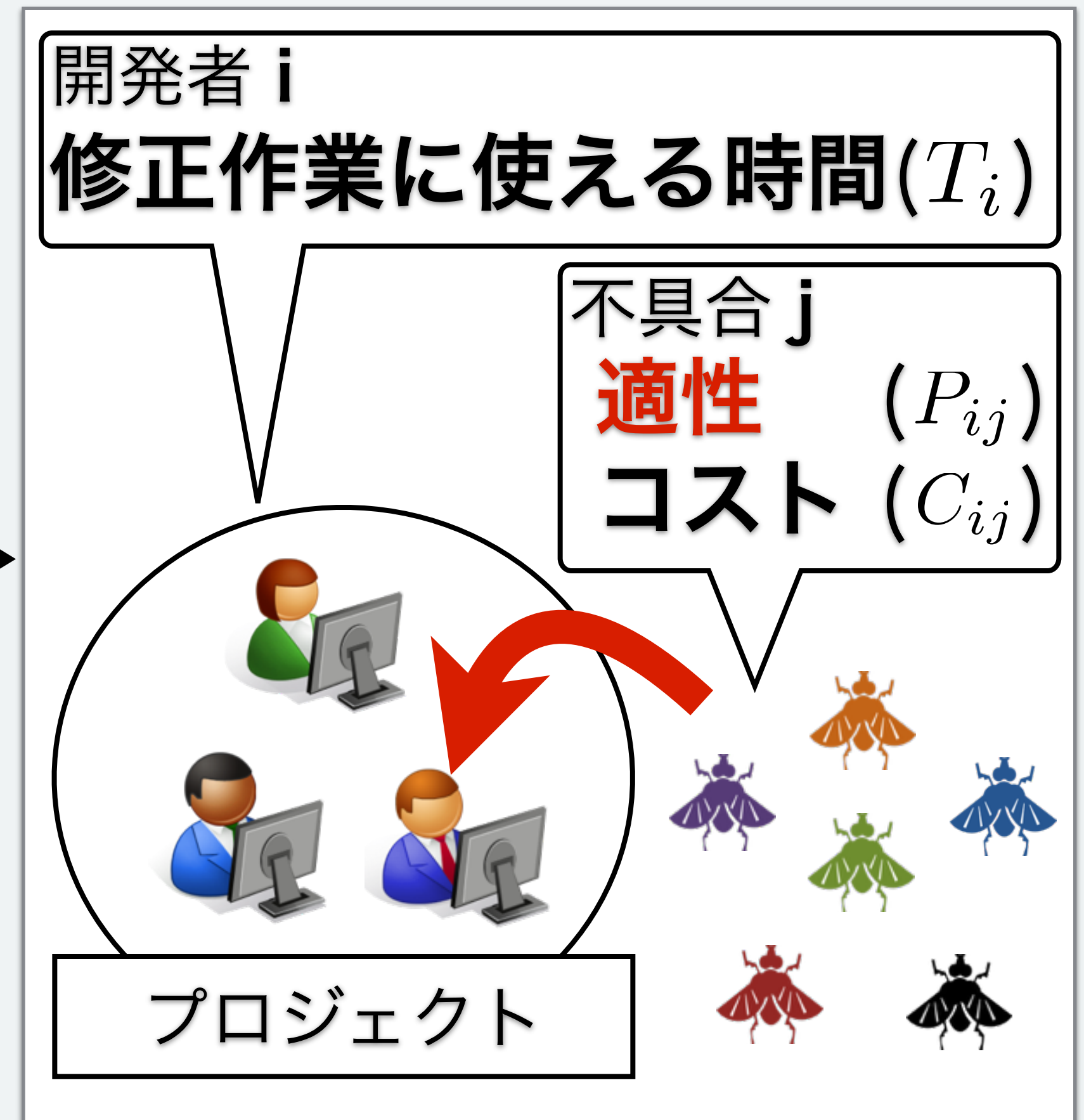
7



プロジェクト全体で修正効率が最も良くなる
不具合と開発者の組合せを求める

キーアイデア

8

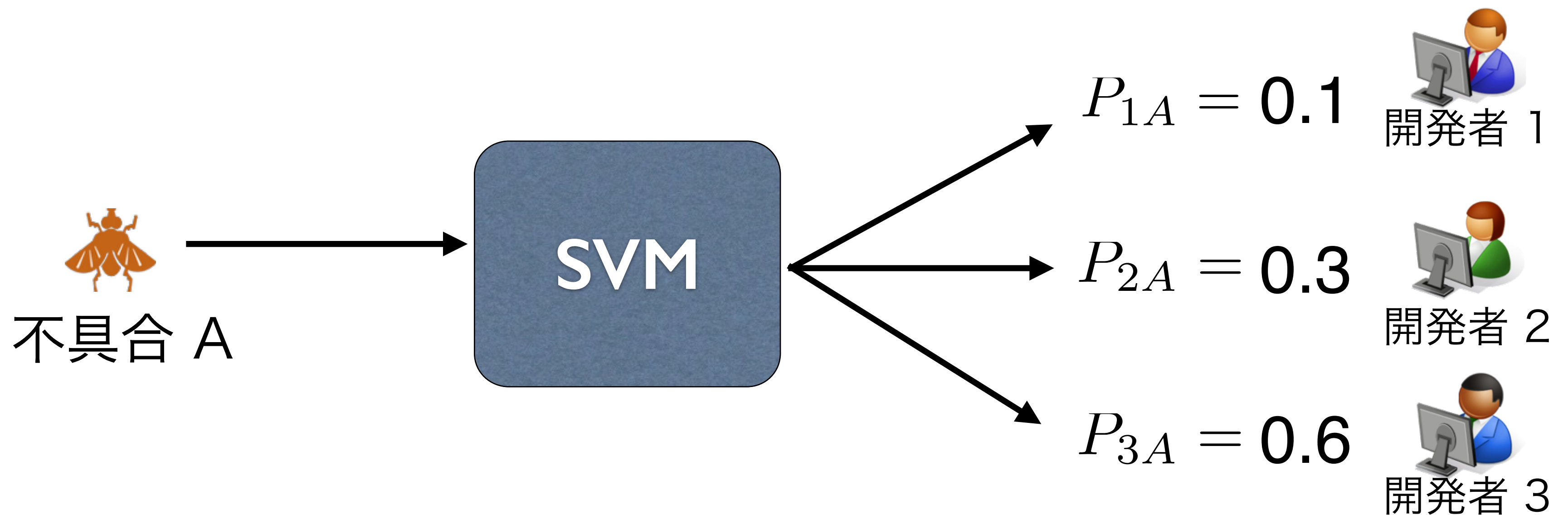


プロジェクト全体で修正効率が最も良くなる
不具合と開発者の組合せを求める

適性 (プリファレンス P_{ij}) の算出方法

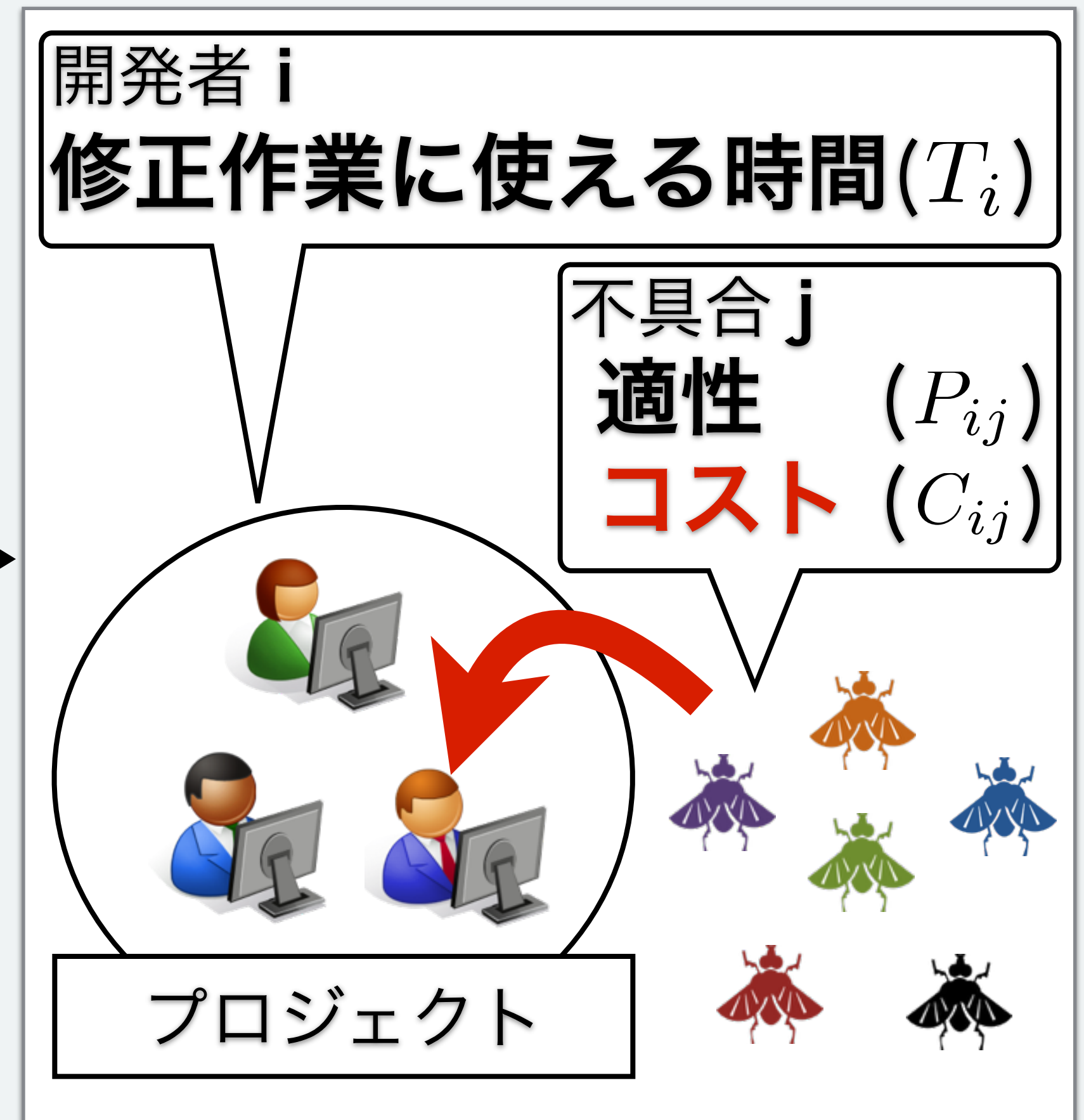
9

- P_{ij} とは開発者 i が不具合 j の修正に適任である確率
 - 既存手法と同じ方法 (SVMを使用)



キーアイデア

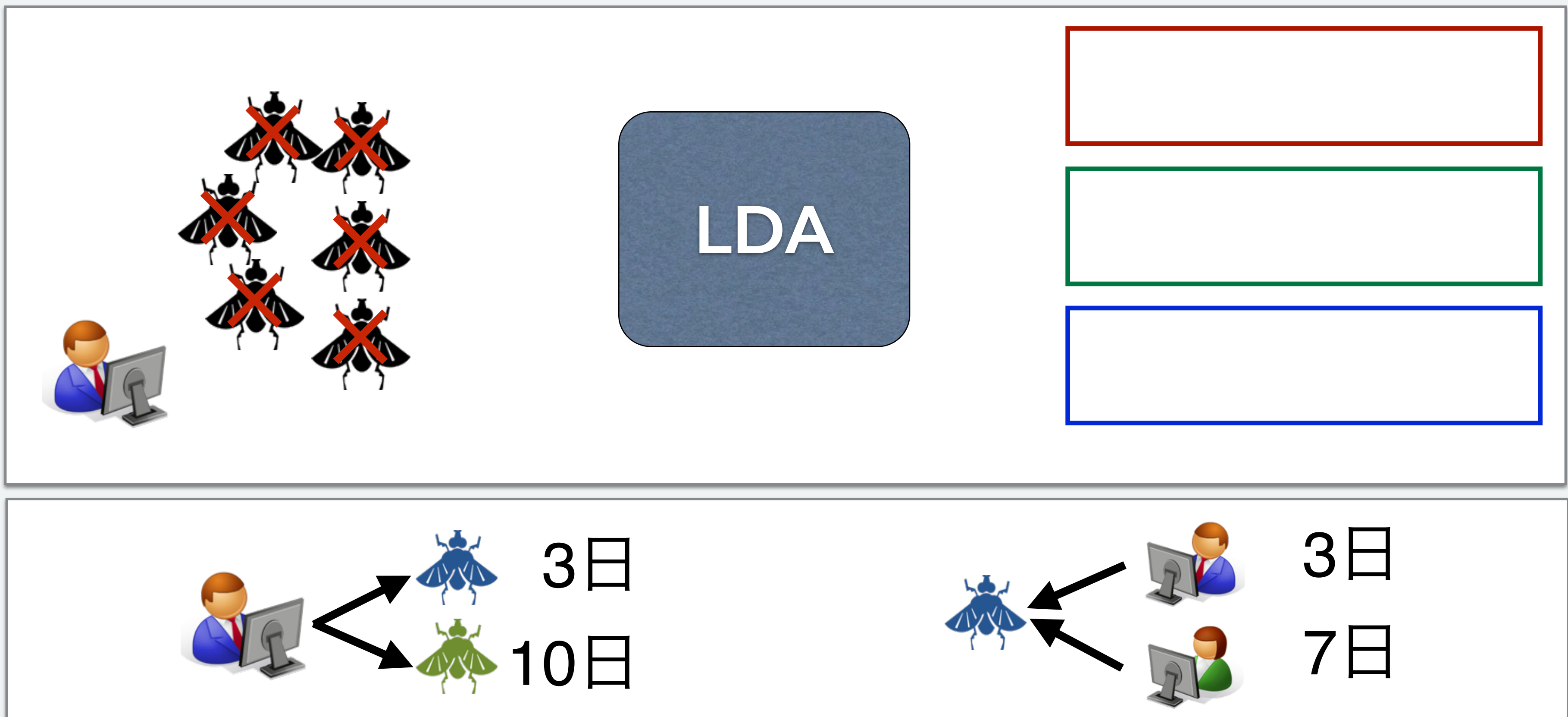
10



プロジェクト全体で修正効率が最も良くなる
不具合と開発者の組合せを求める

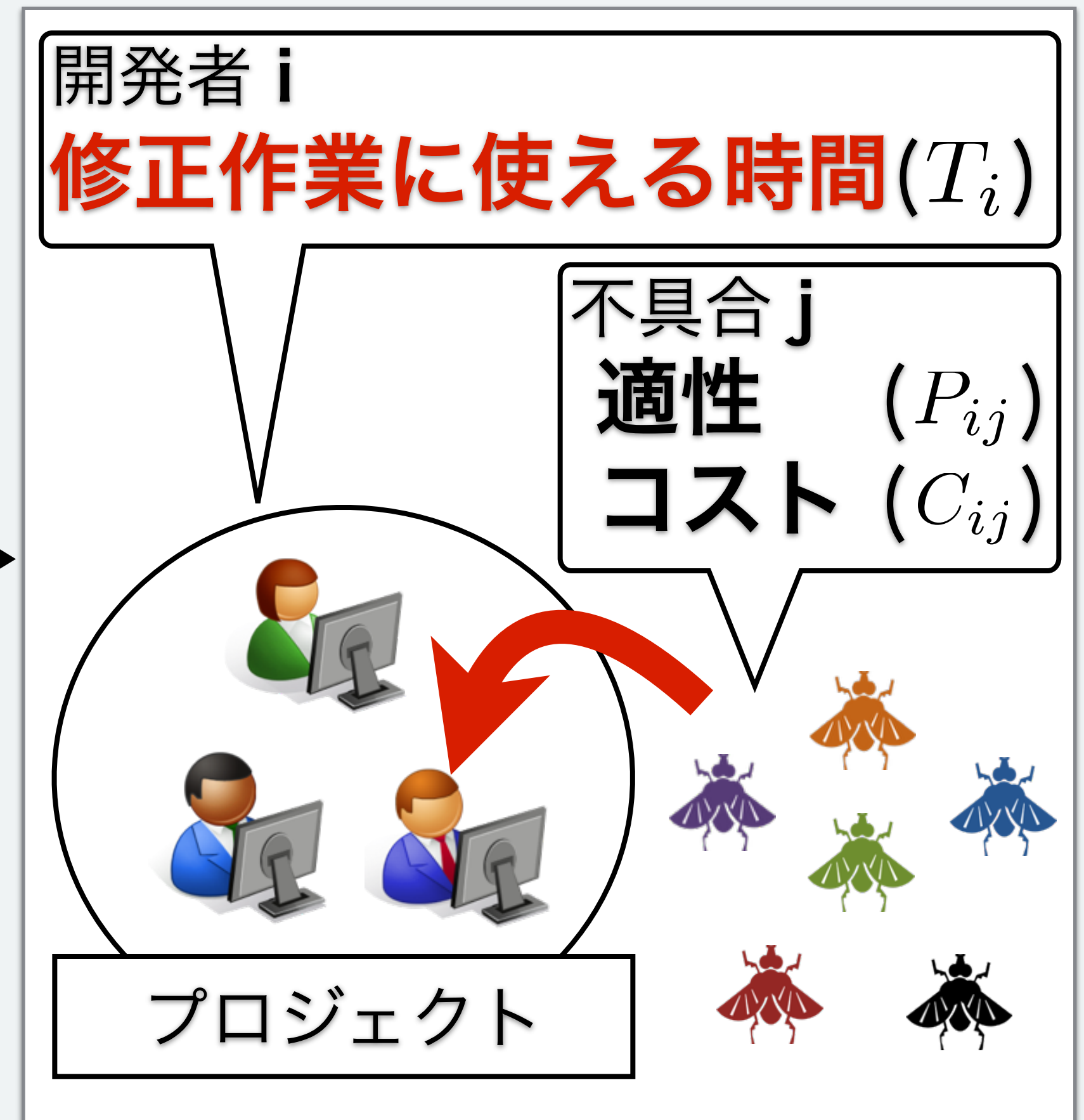
コストの算出方法

- C_{ij} とは開発者 i が不具合 j の修正に必要な時間
 - Latent Dirichlet Allocationを用いて不具合を分類
 - 分類ごとの不具合修正時間の中央値をコストとする



キーアイデア

12



プロジェクト全体で修正効率が最も良くなる
不具合と開発者の組合せを求める

修正作業に使える時間 (Ti)

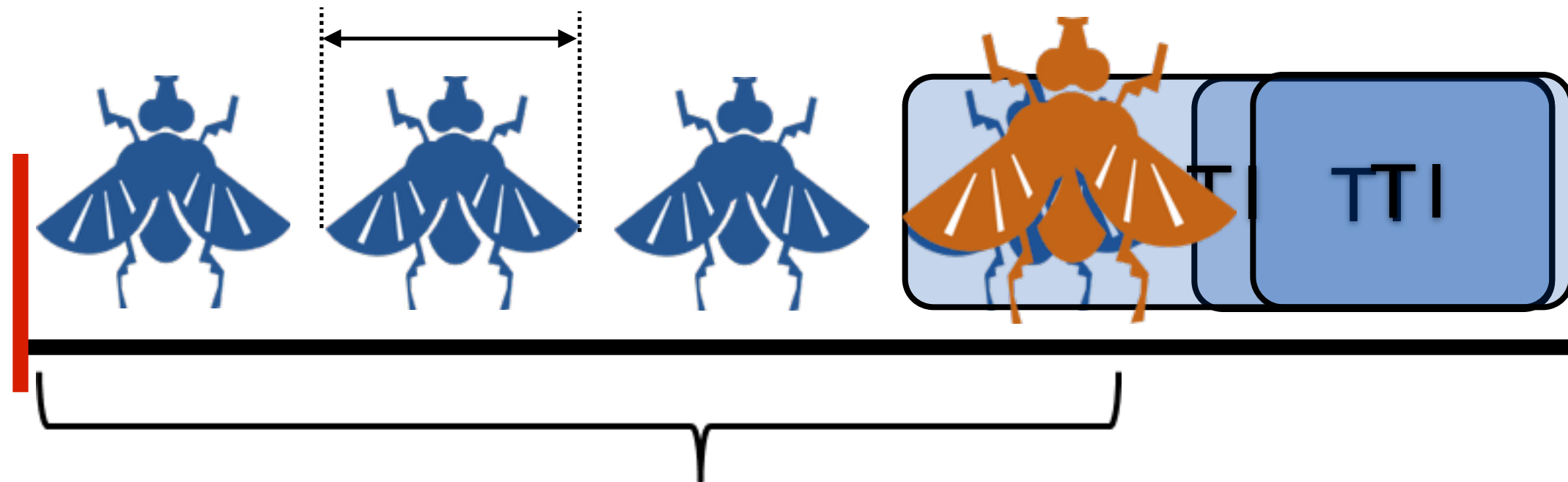
13

$$T_i = \text{上限 } L - \sum_{j=1}^n C_{ij} x_{ij} \quad (i = 1, \dots, m)$$



開発者1

コスト Cij



割当てを行う日に既に担当している
不具合のコストの合計

上限 L (日)

■ 目的関数

- プロジェクト全体で、プリファレンスの総和が最大になる様に開発者にタスクを割当てる

$$\text{Maximize : } \sum_{i=1}^m \sum_{j=1}^n P_{ij} x_{ij}$$

$$x_{ij} \in \{0, 1\}$$

■ 制約条件

- 各開発者が修正作業に使える時間 (T) は有限

$$\sum_{j=1}^n C_{ij} x_{ij} \leq T_i \quad (i = 1, \dots, m)$$

- 不具合の担当は1人未満

$$\sum_{i=1}^m x_{ij} \leq 1 \quad (j = 1, \dots, n)$$



実験の概要と目的

■ 目的

- 提案手法がタスクの集中を軽減し、プロジェクト全体で修正活動を効率化できるかを確認する
 1. 既存手法と比べ、タスク集中をどれだけ軽減できるか
 2. 現実の割当方法より修正時間を短くすることが出来るか

■ 概要

- 不具合に記載されている報告日時に従って、割当てを再現する
 - 現実の割当手法および既存手法と比較する

- pureCBR(Content-Based Recommendation) [Anvik'06]
 - SVMを使用し，開発者の適性を重視した手法
- CosTriage [Park'11]
 - pureCBRをベースとし，適性が高く，修正時間が短い開発者を割当てる手法

		実験2	実験1
	適性	修正時間	開発者の状況
pureCBR	○	×	?
CosTriage	△	○	?
提案手法	?	?	?

発表はここまでの予定

実験の設定

18

■ データセット



Mozilla Firefox

学習データ：2,536件

評価データ：659件



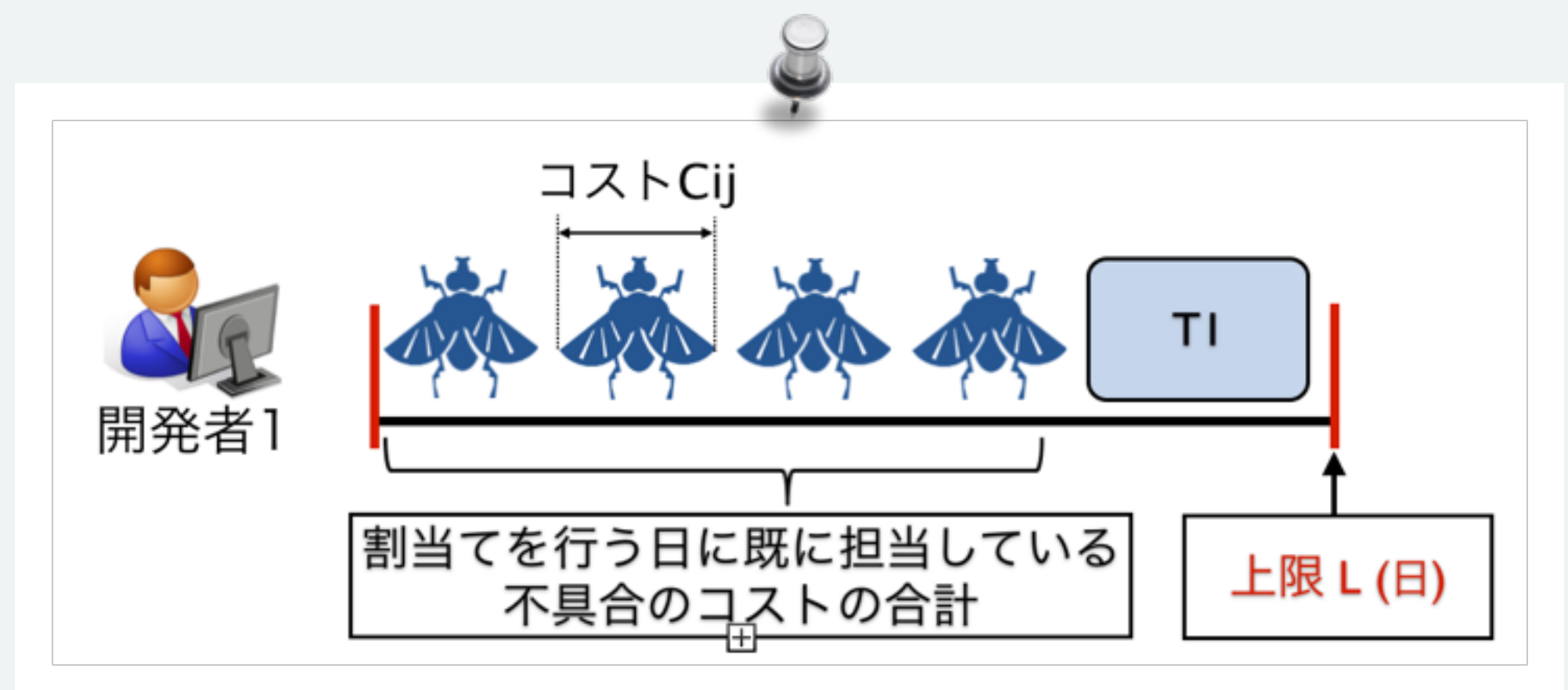
Eclipse Platform

学習データ：1,910件

評価データ：578件

■ パラメータ (上限L)

- Firefox：10日
- Platform：9日



■ 不具合報告の日付に従って不具合の割当てを再現



既存手法・提案手法

割当て結果

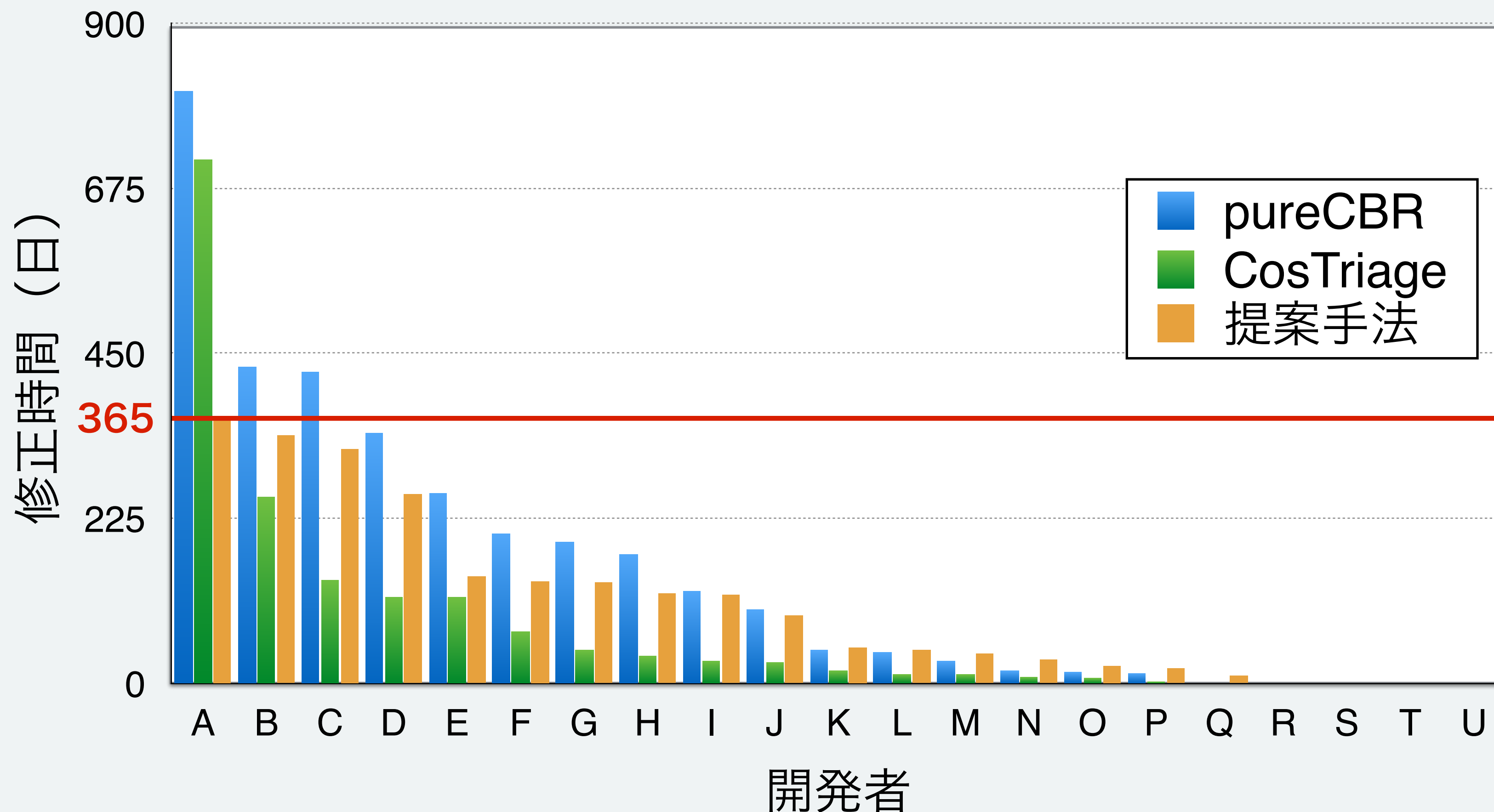


実験1：タスクの集中

20

- 各開発者の修正時間が、テストデータの期間より、超えていないかを確認（期間：365日）

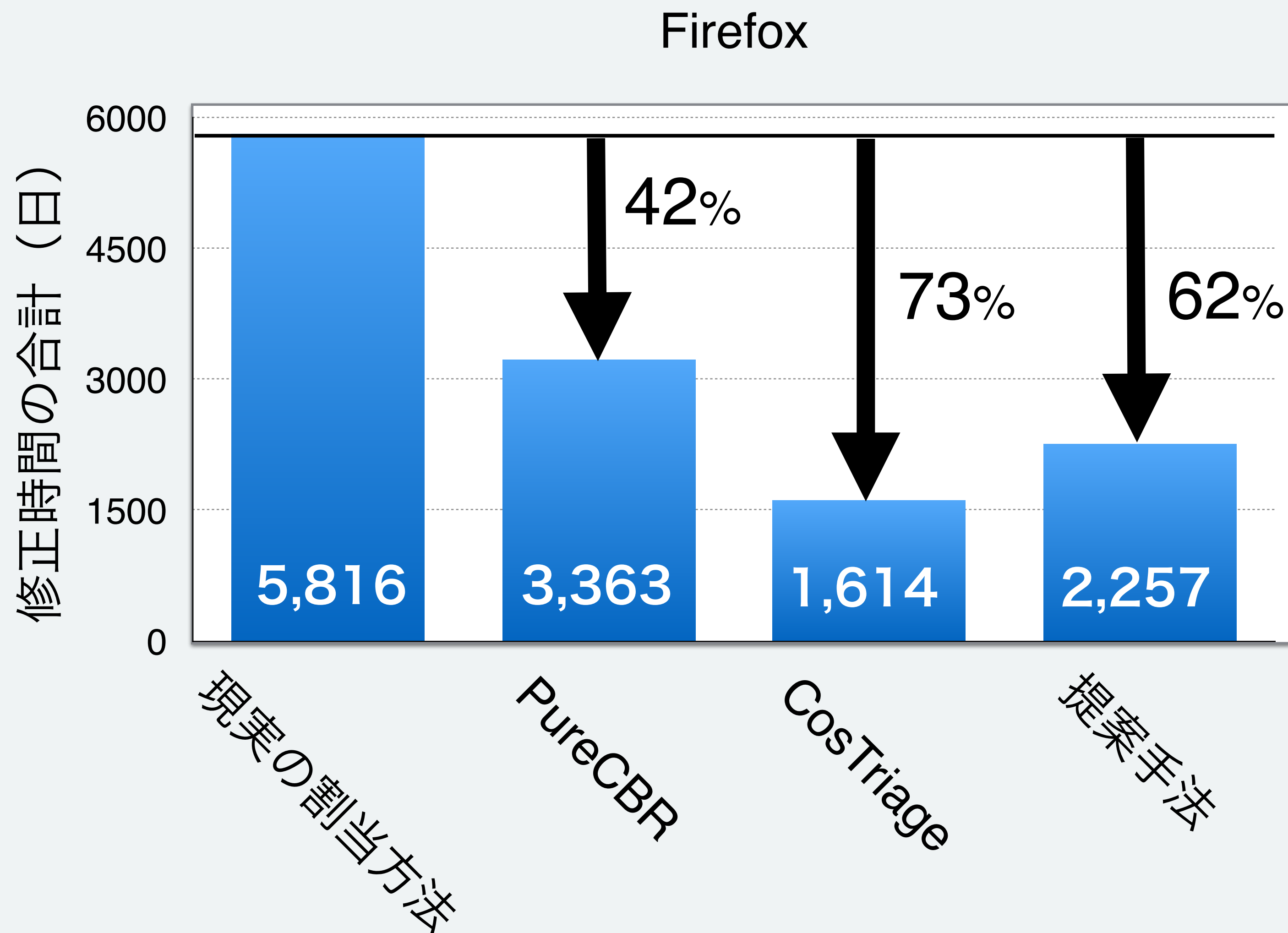
Firefox



実験2：修正時間の削減

21

■ プロジェクト全体での修正時間を比較



- pureCBR(Content-Based Recommendation) [Anvik'06]
 - SVMを使用した手法
- CosTriage [Park'11]
 - Anvikの手法をベースとして修正時間を考慮した手法

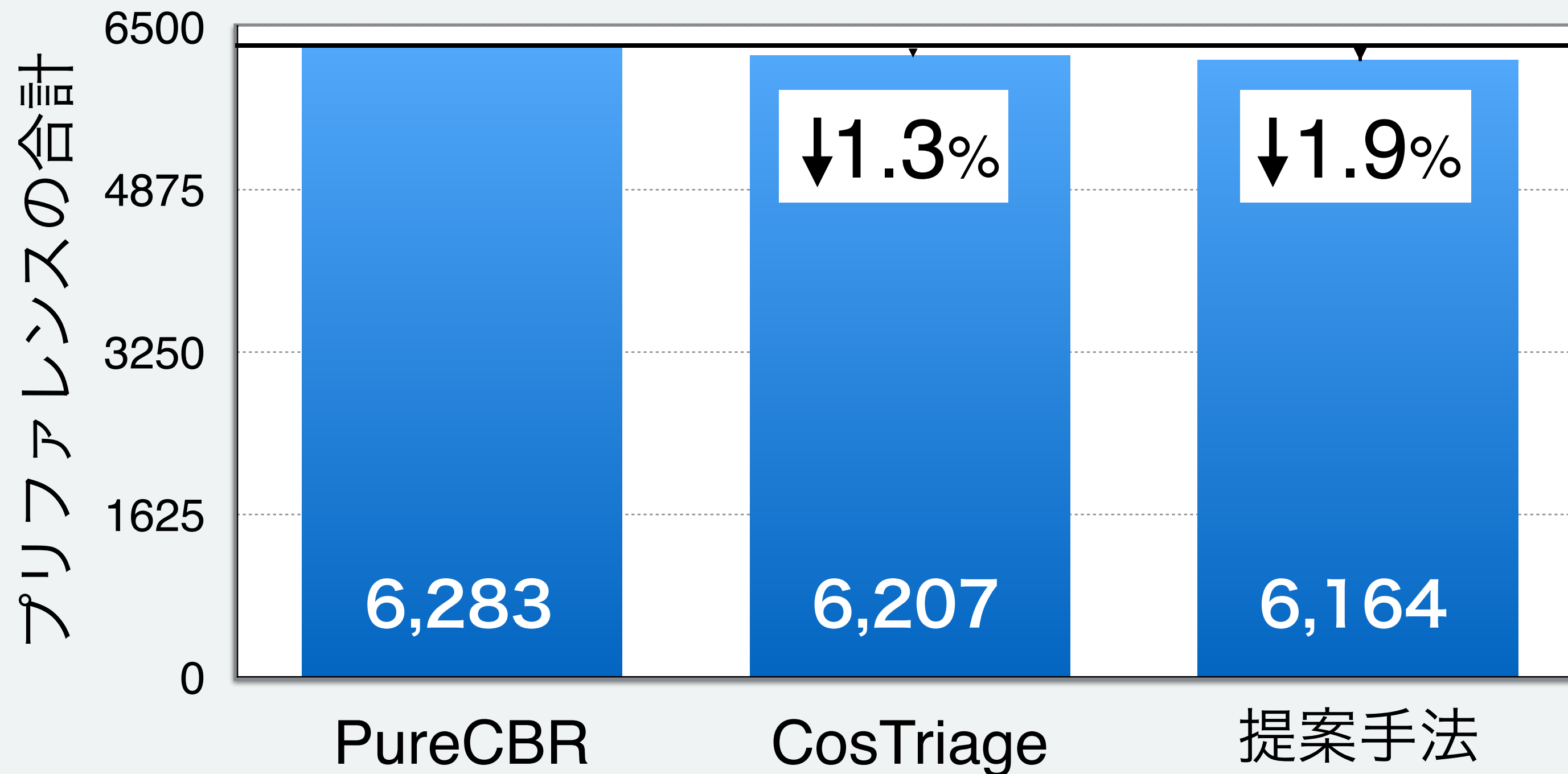
	実験2		実験1
	適性	修正時間	開発者の状況
pureCBR	○	×	×
CosTriage	△	○	×
提案手法	?	△	○

考察：割当ての正確さ

23

- 再割当ての可能性を確認するために追加分析
 - プリファレンスの大きさを比較

Firefox



プリファレンスの1.9%がどの程度
再割当てに影響を与えるかを評価する必要

現状のタスク割当て手法と問題点

- 管理者が不具合報告の内容を精査し、担当の開発者を決定



不具合の担当者変更（再割当）の頻発により修正時間が長期化

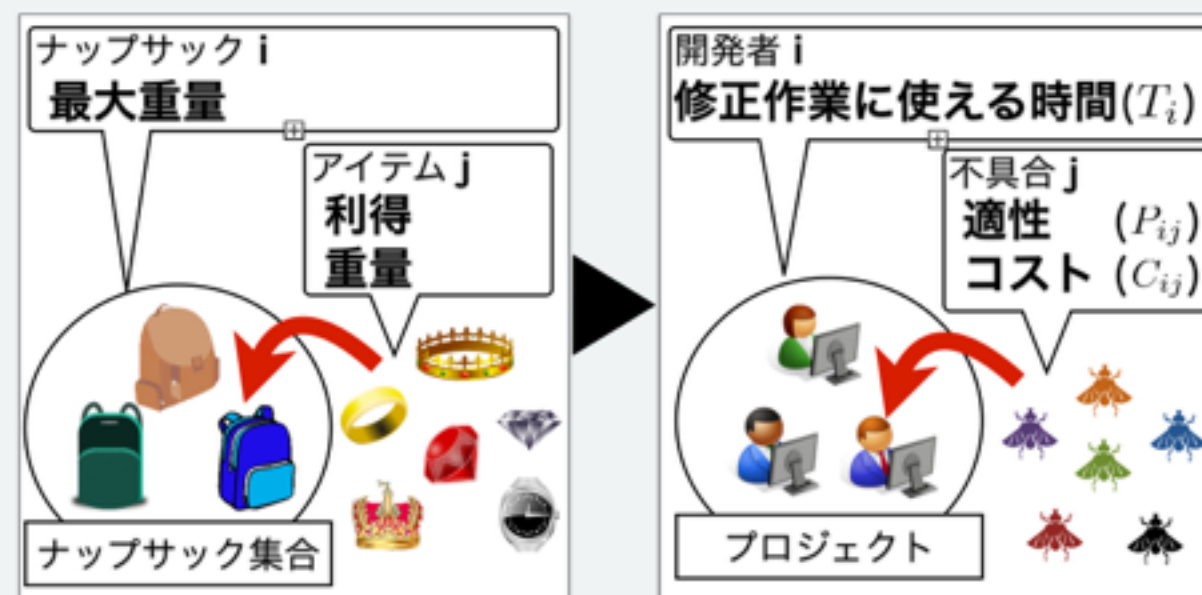
既存手法の問題点

- 機械学習に基づく不具合の割当 [Anvik'06]



一部の有能な開発者にタスクが集中する傾向

キーアイデア



プロジェクト全体で修正効率が最も良くなる
不具合と開発者の組合せを求める

実験のまとめ

- pureCBR (Content-Based Recommendation) [Anvik'06]
 - SVMを使用した手法
- CosTriage [Park'11]
 - Anvikの手法をベースとして修正時間を考慮した手法

	正確さ	修正時間	開発者の状況
pureCBR	○	×	×
CosTriage	△	○	×
提案手法	△	△	○