

大規模OSS開発における不具合修正時間の短縮化を目的としたバグトリージ手法

情報処理学会論文誌 Vol.56 No.2 1-12(Feb. 2015)

柏 祐太郎 大平 雅雄 阿萬 裕久 亀井 靖高

背景

◆バグトリージ

- 重要度や優先度を設定し、開発者に修正タスクを割り当てること

◆バグトリージを支援する研究が盛ん

◆既存手法は一部の開発者にタスクが集中する傾向

→開発者のタスク量に制約条件を課し、タスク割り当てを最適化

Research Question

- ◆ RQ1: 既存手法は、特定の開発者への負荷を集中させる傾向があるか？ 提案手法ではその問題を緩和できるか
- ◆ RQ2: 提案手法は、プロジェクト全体の不具合修正時間の短縮化に寄与するか？
- ◆ RQ3: 提案手法で用いる各種設定が、タスク割り当ての最適化にどのように寄与するか？
 - 開発者の適正（プリファレンス）と上限の有無で4つのモデルを構築

提案手法

◆不具合と開発者の組み合わせをナップサック問題と見なす

→0-1整数計画法を用いて解を求める

$$\text{Maximize : } \sum_{i=1}^n v_i x_i \quad (1)$$

$$\text{Subject to : } \sum_{i=1}^n w_i x_i \leq c \quad (2)$$

$$x_i \in \{0, 1\} \quad (i = 1, 2, \dots, n) \quad (3)$$

v_i :利得
 w_i :重み
 x_i :目的変数
 c :最大重量

用語定義

表 1 本論文で用いる用語一覧

Table 1 Terms used in this study

用語	記号	意味
カテゴリ	k	不具合票の「コンポーネント」と「優先度」で分類したもの。
プリファレンス	P_{ij}	修正タスクをどの開発者に優先的に割当てべきかを示す尺度。 P_{ij} とは開発者 D_i がカテゴリ k の不具合修正タスク B_j に対するプリファレンスを示している。 $P_{ij} = \frac{\text{カテゴリ } k \text{ における開発者 } D_i \text{ の修正数}}{\text{カテゴリ } k \text{ における修正タスクの総数}}$
コスト	C_{ij}	開発者 D_i が不具合修正タスク B_j に要する時間。過去に開発者 D_i がカテゴリ k の不具合修正タスクを完了するのに要した修正時間の中央値とした。
上限	L	タスクの集中を防ぐために設定する値
割当可能時間	T_i	一定期間内に修正可能なタスク量（時間）。 T_i は開発者 D_i の担当可能時間を示す。 $T_i = \text{上限 } L - \sum_{j=1}^n C_{ij} * x_{ij}$

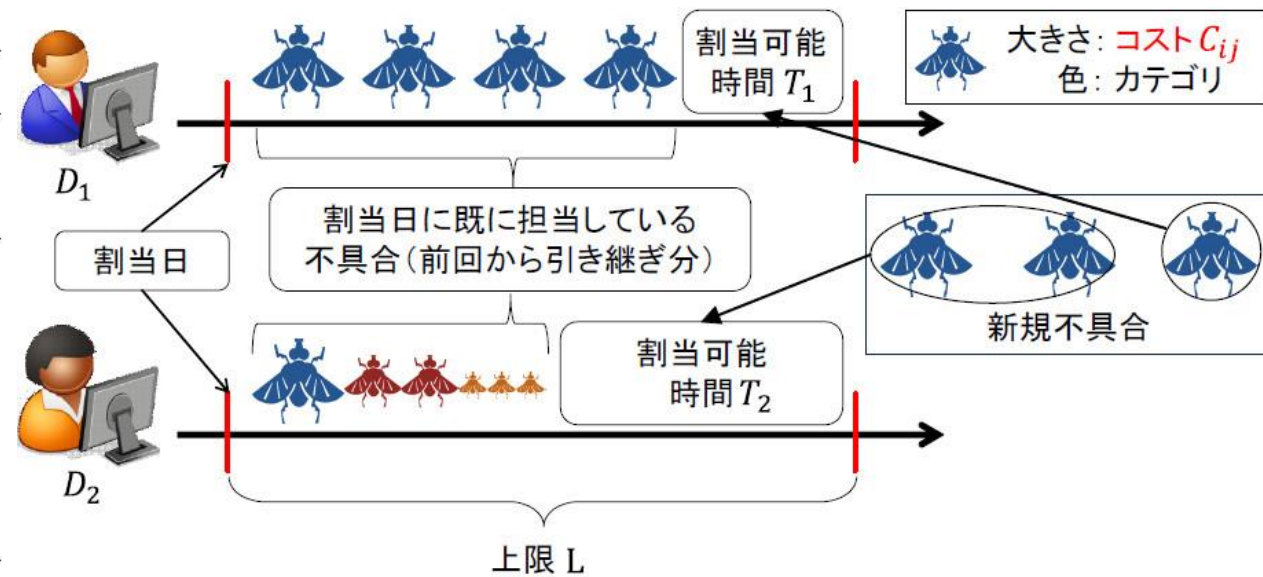


図 1 修正可能なタスク量（時間）の求め方

◆本実験は上限固定

定式化

◆本研究での目的変数、目的関数、制約条件の定義

$$x_{ij} \in \{0, 1\} \quad (5)$$

$$\text{Maximize : } \sum_{i=1}^m \sum_{j=1}^n P_{ij} x_{ij} \quad (6)$$

$$\sum_{j=1}^n C_{ij} x_{ij} \leq T_i \quad (i = 1, 2, \dots, m) \quad (7)$$

$$\sum_{i=1}^m x_{ij} \leq 1 \quad (j = 1, 2, \dots, n) \quad (8)$$

提案手法の適用手順

- ① パラメータの設定
 - ② プリファレンスとコストの算出
 - ③ 割り当て待ち不具合の追加
 - ④ 0-1整数計画法の適用
 - ⑤ T_i の更新
 - ⑥ 次の割当日に進む(T_i を n 日増やす)
- 以下②～⑥の繰り返し

ケーススタディ

表 2 データセット

Table 2 Data sets

プロジェクト	データセット	期間	解決済み 不具合 (件)	対象不具 合数 (件)	コンポー ネント数	カテゴリ リー数
Firefox	学習データ	2002/10/1～2011/9/30	10,165	2,536	28	60
	評価データ	2011/10/1～2012/9/30	1,648	659	31	39
Platform	学習データ	2002/10/1～2011/9/30	21,802	1,910	19	28
	評価データ	2011/10/1～2012/9/30	932	578	17	23

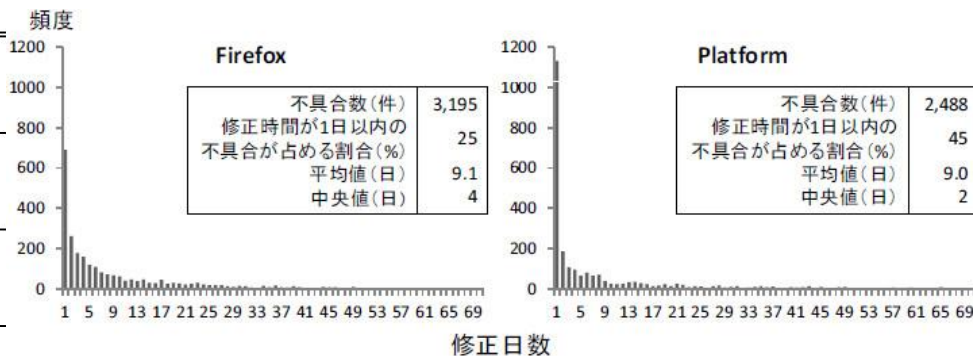


図 2 不具合修正時間のヒストグラムと統計量

表 3 各フィルタリングにおける不具合数

Table 3 Number of bugs in each filtering step

	フィルタリング	Firefox		Platform	
		学習データ	評価データ	学習データ	評価データ
A	各プロジェクトから収集した不具合	105,294	11,765	78,856	3948
B	A のうち修正時間および修正者が特定でき、かつ修正された不具合	10,165	1,648	21,802	932
C	B のうち、不具合の修正時間が外れ値になっていない不具合	8,699	1,496	18,370	871
D	C のうち、アクティブ開発者が修正した不具合	2,536	659	1,910	578

表 4 データセットに含まれるアクティブな開発者

Table 4 The number of active developers in the data sets

プロジェクト	全開発者	アクティブ開発者
Firefox	734	21
Platform	301	23

Firefox、Platformを対象に提案手法と既存手法を比較

実験の手順

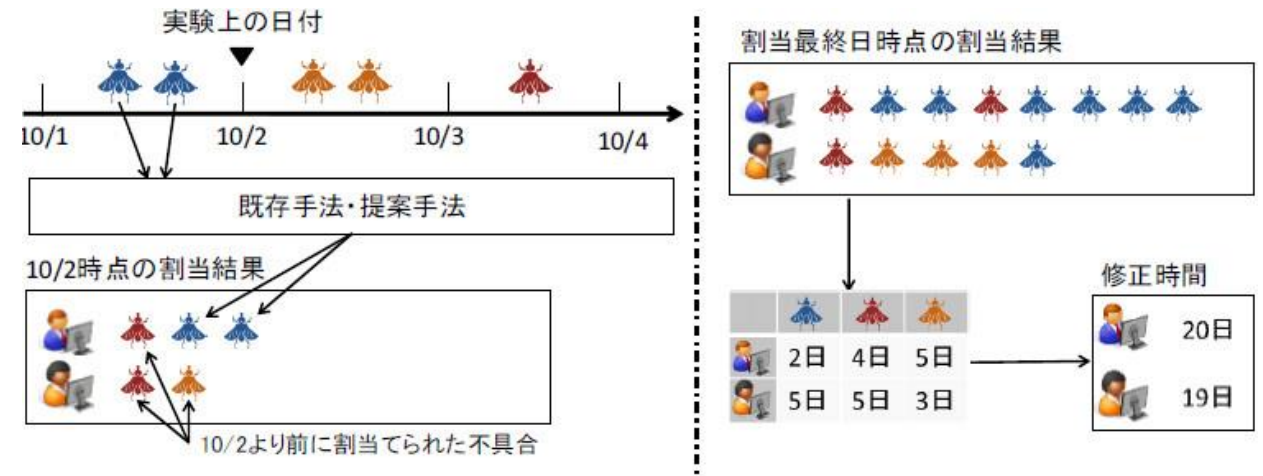


図 3 実験の概要

- ① 実験上の日付を用意し、不具合報告を再現
 - 本実験では365日分の割り当てを行う

- ② 修正時間の算出

◆ Firefox: L=10、Platform: L=9、n=1 日

実験結果: RQ1

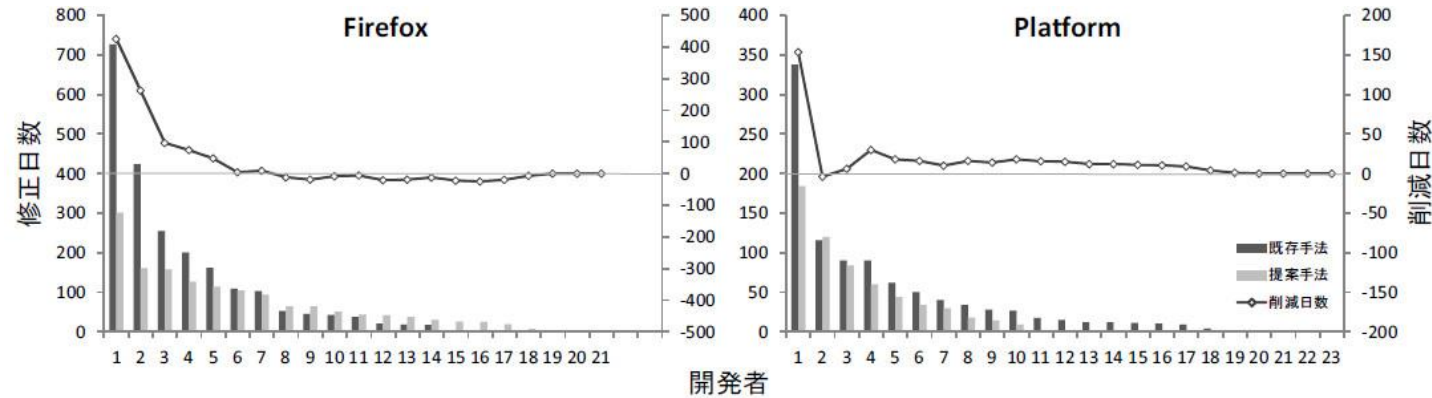


図 4 全期間における各開発者の修正日数（既存手法 vs. 提案手法）

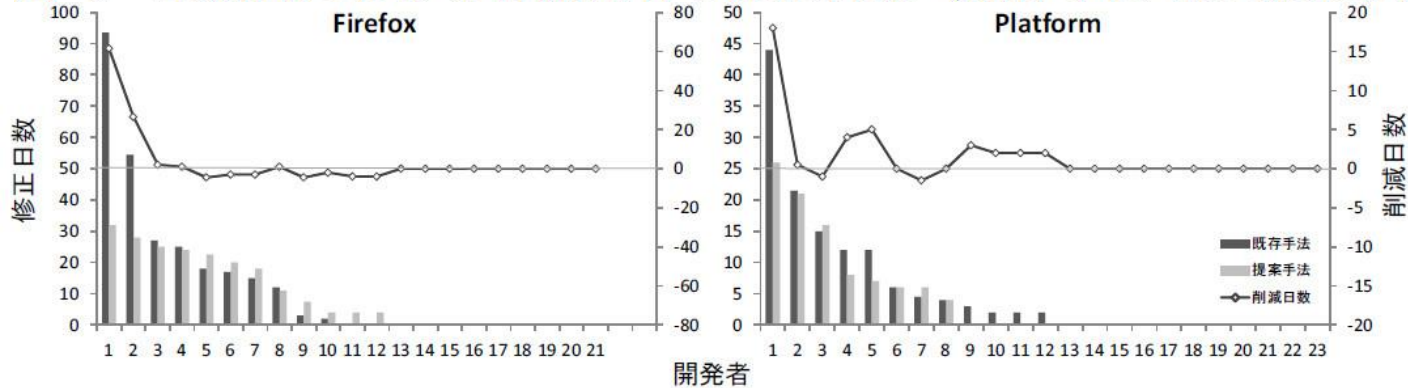


図 5 繁忙期における各開発者の修正日数（既存手法 vs. 提案手法）

◆一部の開発者へタスクが集中するのを緩和できるといえる

実験結果: RQ2

表 5 現状の割当方法, 既存手法および提案手法によるタスク割当結果の比較

Table 5 Comparison of task assignments between the three methods

プロジェクト		Firefox			Platform		
手法		現状の割当方法	既存手法	提案手法	現状の割当方法	既存手法	提案手法
割当てた不具合 (件)	共通化前	659	439	440	578	386	572
	共通化後	377			385		
割当てた開発者 (人)	共通化前	16	17	18	20	19	12
	共通化後	16	15	18	20	19	11
合計修正日数 (日)	共通化前	5,816	3,091	1,685	4,893	995	901
	共通化後	2,920	2,220	1,471	3,433	967	599

◆ Firefoxは現状の割当に比べ50%、既存手法に比べ34%削減

◆ Platformは既存手法に比べ38%削減

修正時間算出方法の妥当性

表 8 コスト算出方法の妥当性の検証

Table 8 The total costs (actual vs. simulated data)

	現状の割当方法	実験	差
Firefox	4,451	4,660	-209
Platform	4,787	1,982	2,806

◆Firefoxは約4%の誤差

◆Platformは誤差が大きい

➤RQ2で現状の割当方法との比較は計測不能

実験結果: RQ3

表 7 モデルごとのタスク割当ての結果

Table 7 Comparison of task assignments between the four models

プロジェクト		Firefox				Platform			
モデル		A	B	C	D	A	B	C	D
適性		無	無	有	有	無	無	有	有
上限		無	有	無	有	無	有	無	有
割当てた不具合 (件)	共通化前	659	659	559	440	578	578	575	572
	共通化後	440	440	440	440	572	572	572	572
割当てた開発者 (人)	共通化前	1	15	10	18	1	14	12	12
	共通化後	1	15	9	18	1	14	12	12
合計修正日数 (日)	共通化前	4,284	3,066	4,897	1,685	5,780	1,186	997	901
	共通化後	2,860	2,022	2,115	1,685	5,720	1,174	910	901

表 6 条件の有無による 4 つのモデル

Table 6 Four models with and without P and L

	L : 無	L : 有
P : 無	モデル A: プリファレンスと上限を設定せず, 修正時間の短い開発者に不具合を優先的に割当てて	モデル B: 上限を設定した条件下で, 修正時間の短い開発者に不具合を優先的に割当てて
P : 有	モデル C: プリファレンスのみ設定し, カテゴリごとのプリファレンスが高い開発者にタスクを割当てて	モデル D: 上限を設定した条件下で, プリファレンスの合計が最大となるように不具合を割当てて (提案手法)

- ◆ AとB: タスク割り当てを分散させる効果がある
- ◆ AとC: タスク割り当てを分散させる効果がある
- ◆ BとD: 上限が先に設定している場合、必ずしも有効ではない
- ◆ CとD: プリファレンスを設定した上で上限を設定することは有効な可能性がある

まとめ

◆提案手法について以下の3つの効果が確認できた

- 特定の開発者へタスクが集中する問題を緩和できる
- 現状のタスク割当方法に比べFirefoxでは50%(Platformでは誤差が大きすぎるため計測不能)、既存手法に比べFirefoxでは34%、Platformでは38%不具合修正時間を削減できる
- プリファレンスと上限が、タスクの分散効果に同程度寄与する