
When Do Changes Induce Fixes?

~ SZZ Algorithm ~

.....

ACM SIGSOFT Software Engineering Notes Volume 30 Issue 4 pp 1–5

背景

| 現在のソフトウェアの状態の理解

- ソフトウェアの履歴（バージョンアーカイブ）を調査される
- 過去のすべての変更が有益とは限らない

CVS

誰が/何を/いつ/なぜ/どのように
変更したかの情報

BUGZILLA

バグの状態とその対応の情報

研究目的

問題となった変更（バグ混入コミット）を特定する

CVSとバグデータの紐付け

| 修正コミットを見つける

- 構文的信頼度/意味的信頼度の計算

| CVSのログメッセージからトークンを取得

- バグ番号が以下の正規表現にマッチする場合
 - `bug[ # ¥t ] * [ 0 - 9 ] +`
 - `pr[ # ¥t ] * [ 0 - 9 ] +`
 - `show¥_bug¥.cgi¥? id = [ 0 - 9 ] +`
 - `¥[[ 0 - 9 ] + ¥]`
- キーワードが以下の正規表現にマッチする場合
 - `fix(e[ds])?|bugs?|defects?|patch`
 - 単語が英数字の文字列

CVSとバグデータの紐付け | 構文解析

| 構文的信頼度の計算

- 構文的信頼度の初期値は0で、最大値は2（0～2の整数）
- ログメッセージ内のバグへの潜在的なリンクの信頼度を計算
 - バグ番号の正規表現にマッチする
 - ログメッセージにキーワードが含まれている

Fixed bug 53784: .class file missing from jar file export

- “*bug 53784*” → バグ番号の正規表現にマッチ
 - “*Fixed*” → キーワードの正規表現にマッチ
- } 信頼度 2

52264,51529

- バグ番号の正規表現にマッチする（ $[[0-9]^+]$ ）
- } 信頼度 1

CVSとバグデータの紐付け | セマンティック分析

| 意味的信頼度の計算

- 意味的信頼度の初期値は0で，最大値は4（0～4の整数）
- ログメッセージ内のバグへの潜在的なリンクの信頼度を計算
 - バグ報告bは少なくとも一度はFIXEDとして解決されている
 - バグ報告bの短い説明は，変更tのログメッセージに含まれている
 - 変更tの作成者がバグ報告bに割り当てられた
 - 変更の影響を受けるファイルの1つ以上がバグ報告bに添付されている

Fixed bug 53784: .class file missing from jar file export.

- バグ53784はクローズされておらず，LATERとして解決
 - バグ報告tに割り当てられた変更者が作成者と異なる
- } 信頼度 0

紐付けの結果

| ECLIPSE

– 25,317のリンクを確認

- 修正されたバグ報告の47%と変更の29%が結びついた

syn / sem	0	1	2	3	4	total
0	270 (1%)	1,287 (5%)	2,057 (8%)	1,439 (6%)	2 (0%)	5,055 (20%)
1	324 (1%)	4,152 (16%)	9,265 (37%)	1,581 (6%)	5 (0%)	15,327 (61%)
2	110 (0%)	1,922 (8%)	2,421 (10%)	482 (2%)	0 (0%)	4,935 (19%)
total	704 (3%)	7,361 (29%)	13,743 (54%)	3,502 (14%)	7 (0%)	25,317 (100%)

| MOZILLA

– 53,574のリンクを確認

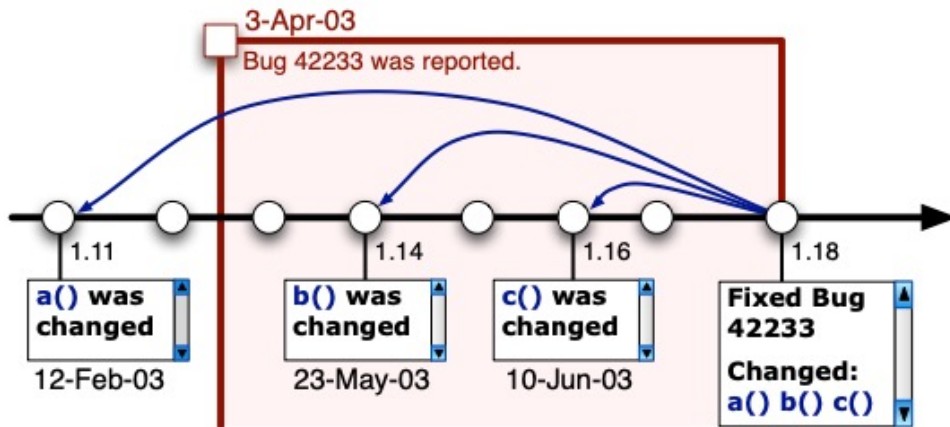
- 修正されたバグ報告の55.30%と変更の43.91%が結びついた

syn / sem	0	1	2	3	4	total
0	560 (1%)	2,899 (5%)	4,281 (8%)	639 (1%)	8 (0%)	8,387 (16%)
1	1,211 (2%)	9,059 (17%)	16,336 (30%)	2,241 (4%)	22 (0%)	28,669 (54%)
2	478 (1%)	5,250 (10%)	9,133 (17%)	1,645 (3%)	12 (0%)	16,518 (31%)
total	2,249 (4%)	17,208 (32%)	29,750 (55%)	4,525 (8%)	42 (0%)	53,574 (100%)

修正を誘発する変更の特定

| 修正の原因となった変更を自動的に見つける

e.g.) リビジョン $r1 = 1.17$ を $r2 = 1.18$ に変更



```
$ cvs annotate -r 1.17 Foo.java
```

```
...
19: 1.11 (john 12-Feb-03): public int a() {
20: 1.11 (john 12-Feb-03):     return i/0;
...
39: 1.10 (mary 12-Jan-03): public int b() {
40: 1.14 (kate 23-May-03):     return 42;
...
59: 1.10 (mary 17-Jan-03): public void c() {
60: 1.16 (mary 10-Jun-03):     int i=0;
...
```

CVSのdiffコマンドで
変更箇所を取得
(この例では20, 40, 60行目)

$r1$ に対してCVS annotateコマンド
を呼び出し、行レベルで変更された
リビジョンを取得

取得した行レベルの変更から、
バグ報告bより後にコミットされた
変更tを除外

Commit Guru: Analytics and Risk Prediction of Software Commits

ESEC/FSE 2015: Proceedings of the 2015 10th
Joint Meeting on Foundations of Software
Engineering August 2015 Pages 966–969

ソフトウェア品質

| ソフトウェア品質の重要性

- システムの重要性と複雑性が増している
- 対処が極めて困難な問題
- 将来的に欠陥が発生し得る箇所の予測の研究が盛ん

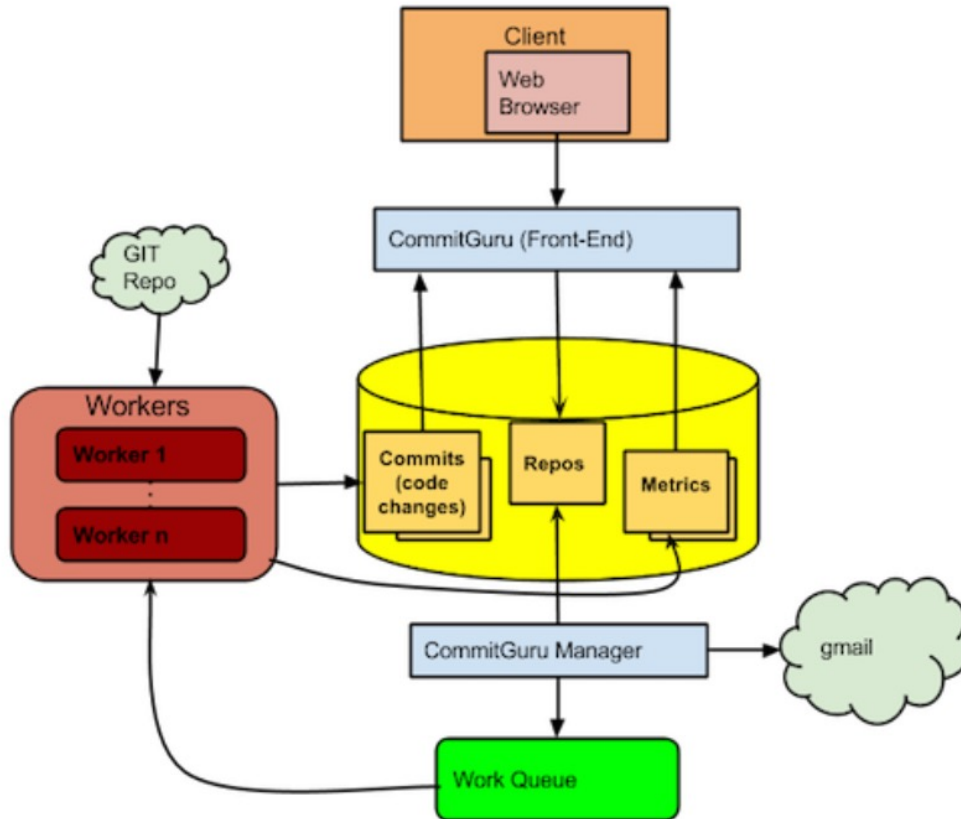
| 欠陥予測の限界

- 最先端の分析と予測技術を組み込んだツールの不足が原因で、予測に関する研究を実務に採用することは未だ困難

—— ツール開発の目的 ——

最先端の分析と予測を技術を基にして、
開発者と管理者にコミットのリスクレベルを提供

Commit Guruの概要



バックエンド

- コミットの全ての分析/評価を担当

フロントエンド

- 分析されたデータの可視化を担当

フロントエンド | 入力

Commit Guru Home Repos About Sign In

It's 8pm, do you know how risky your commits are?

Ask the Commit Guru

View automatically generated reports of your repository's commits. Anywhere.

Analyze a Repository

Repository URL*

Email

☒ List publicly? (Always will be accessible directly)

[Start Now >](#)

Create an account

Email:

Password

[Sign Up](#)

Recent Repositories

laravel	4,427	15%	
ReactiveCocoa	5,575	10%	
linux	517,502	3%	
elasticsearch	10,705	30%	
JDeodorant	782	48%	

[View all >](#)

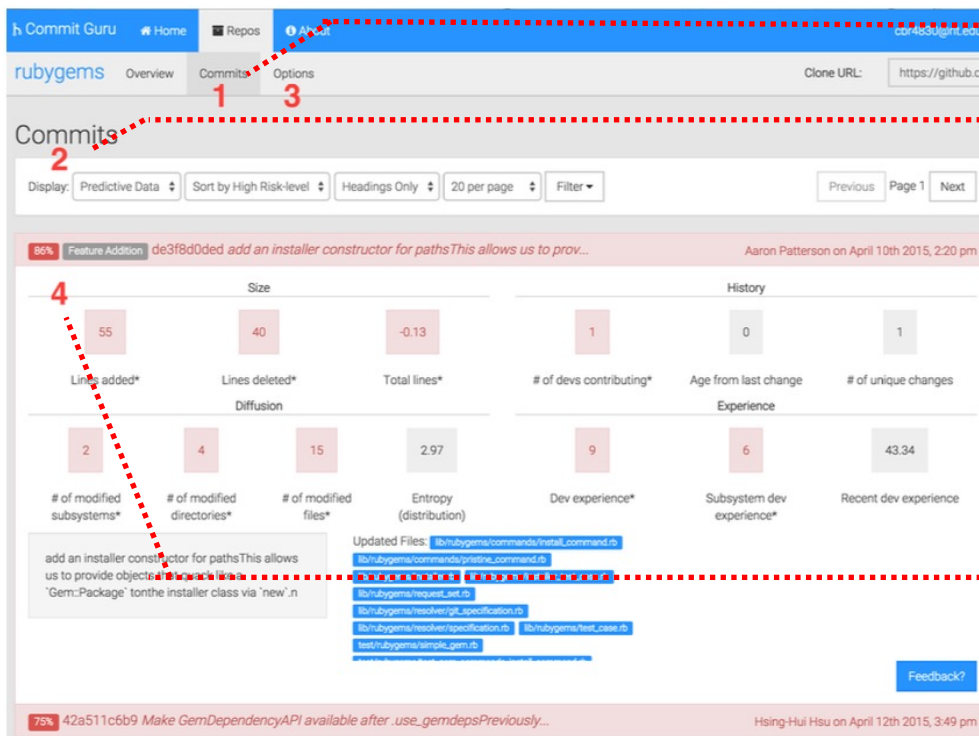
- 分析/予測したい gitリポジトリのURL
- ユーザのメールアドレス (任意)

フロントエンド | 出力

| プロジェクトのコミット統計

– コミット総数とリスクのある(バグを誘発する)コミットの割合

| リスクのあるコミットとリスクのないコミットの中央値



- プロジェクトの全てのコミット
- ビューの切り替え
 - リポジトリの全コミットを表示し、リスクの高いコミットは赤字でハイライト
 - 過去3ヶ月間に行われたコミットのリスクを回帰モデルで評価
- 個々のコミットの特定のメトリック値を表示可能

バックエンド | 1. 入力

	名前	URL	ユーザのE-mail	作成日
1	hogehoge	https://~~	~~~@gmail.com	10/4/2015
⋮	⋮	⋮	⋮	⋮
n	piyopiyo	https://~~	~~~@gmail.com	20/5/2017

Category	Associated Words	Explanation
Corrective	bug, fix, wrong, error, fail, problem, patch	Processing failure
Feature Addition	new, add, requirement, initial, create	Implementing a new requirement
Merge	merge	Merging new commits
Non Functional	doc	Requirements not dealing with implementations
Perfective	clean, better	Improving performance
Preventive	test, junit, coverage, asset	Testing for defects

リビジョンログの各コミットを解析し、コミットログに分類のためのキーワードが含まれている場合、各コミットを分類

不具合を修正しているコミット

2. 分析

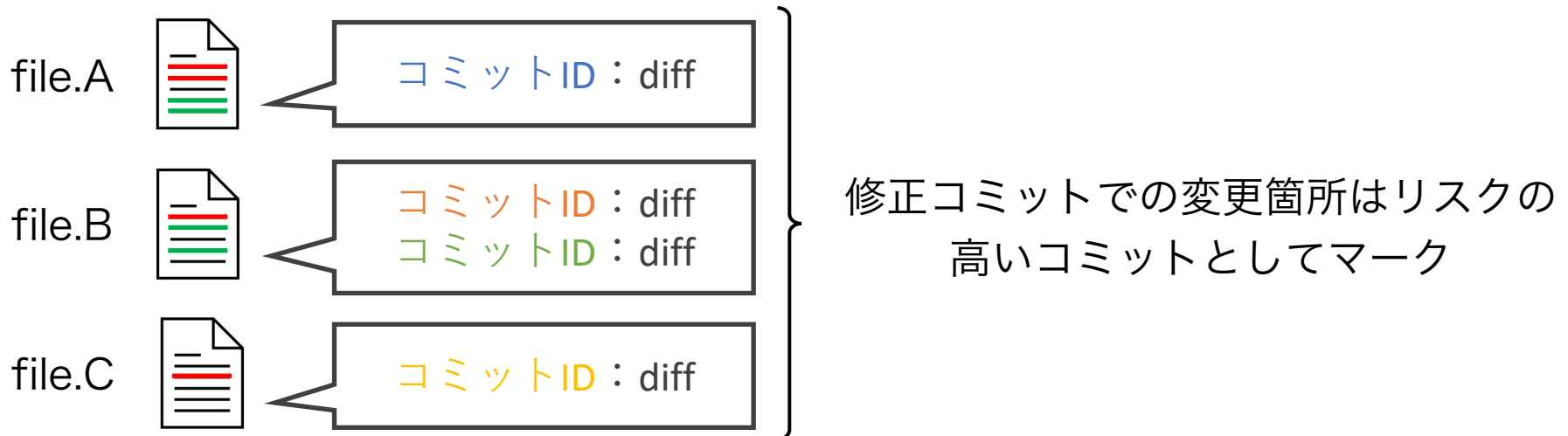
バックエンド | 2. 分析

| リスクの高いコミットと修正コミットの紐付け

－ 修正コミットからリスクのあるコミットを特定

- diffコマンドで修正行を特定し，annotate/blameコマンドで修正行がどのコミットで変更されたかを取得
- ソースコードファイル/スクリプトファイルのみを収集

修正コミット



バックエンド | 2. 分析

| メトリクスの計算

- Kameiらが導出した13のコミットレベルメトリクスを使用
- 1. リスクのある/ないコミットの2つの集合への中央値を計算
 - Wilcoxon testで中央値の差が統計的に有意であることを確認
- 2. 中央値を使用してメトリックごとにリスク度を出力
 - リスクのあるコミット/ないコミットの集合の中央値を比較

LOCの場合

リスクのあるコミット：100行
リスクのないコミット：20行



あるコミットの行数が10行の場合
そのコミットはほぼ安全だろう

補足資料

亀井先生のメトリクス

Dim.	Name	Definition	Rationale	Related Work
Diffusion	NS	Number of modified subsystems	Changes modifying many subsystems are more likely to be defect-prone.	The defect probability of a change increases with the number of modified subsystems [41].
	ND	Number of modified directories	Changes that modify many directories are more likely to be defect-prone.	The higher the number of modified directories, the higher the chance that a change will induce a defect [41].
	NF	Number of modified files	Changes touching many files are more likely to be defect-prone.	The number of classes in a module is a good feature of post-release defects of a module [47]
	Entropy	Distribution of modified code across each file	Changes with high entropy are more likely to be defect-prone, because a developer will have to recall and track large numbers of scattered changes across each file.	Scattered changes are more likely to introduce defects [10], [19].
Size	LA	Lines of code added	The more lines of code added, the more likely a defect is introduced.	Relative code churn measures are good indicators of defect modules [43], [46].
	LD	Lines of code deleted	The more lines of code deleted, the higher the chance of a defect.	
	LT	Lines of code in a file before the change	The larger a file, the more likely a change might introduce a defect.	Larger modules contribute more defects [27].
Purpose	FIX	Whether or not the change is a defect fix	Fixing a defect means that an error was made in an earlier implementation, therefore it may indicate an area where errors are more likely.	Changes that fix defects are more likely to introduce defects than changes that implement new functionality [16][52].
History	NDEV	The number of developers that changed the modified files	The larger the NDEV, the more likely a defect is introduced, because files revised by many developers often contain different design thoughts and coding styles.	Files previously touched by more developers contain more defects [32].
	AGE	The average time interval between the last and the current change	The lower the AGE (i.e., the more recent the last change), the more likely a defect will be introduced.	More recent changes contribute more defects than older changes [15].
	NUC	The number of unique changes to the modified files	The larger the NUC, the more likely a defect is introduced, because a developer will have to recall and track many previous changes.	The larger the spread of modified files, the higher the complexity [10], [19].
Experience	EXP	Developer experience	More experienced developers are less likely to introduce a defect.	Programmer experience significantly decreases the defect probability [41].
	REXP	Recent developer experience	A developer that has often modified the files in recent months is less likely to introduce a defect, because she will be more familiar with the recent developments in the system.	
	SEXP	Developer experience on a subsystem	Developers that are familiar with the subsystems modified by a change are less likely to introduce a defect.	