スパムフィルタに基づく即時バグ予測ツールの試作

森 啓太
京都工芸繊維大学 大学院工芸科学研究科 情報工学専攻
k-mori@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 大学院工芸科学研究科 情報工学部門
o-mizuno@kit.ac.jp

要旨

ソフトウェア工学において，*Fault-prone* モジュールの予測モデルの研究は数多く行われてきた．これらの研究の最終的な課題は，*Fault-prone* モジュール予測を実際のソフトウェア開発の現場に適用していくことである．

そこで本稿では，その中でも構築が簡単で高い予測性能をもつ「*Fault-prone* フィルタリング」という予測モデルを利用して，*Fault-prone* モジュール予測ツールの試作を行う．本試作ツールでは，ソフトウェアリポジトリを監視して新しい変更があるごとにモジュールがバグを含む確率を予測し，結果をツール使用者に *Web* ページ上で提示する．変更ごとに予測するため，バグを含むファイルを変更した開発者を特定でき，さらにその時のコード片を特定することで，ソフトウェアの品質確保活動にかかる工数の削減とソフトウェア自身の品質の向上が期待される．

また，ツールの性能評価のため，*3* つのソフトウェアリポジトリを対象に評価実験を行ったところ，ある程度の予測性能を伴って予測結果をツールの利用者に *Web* ページで提示することが可能であるとわかった．

1. はじめに

ソフトウェア開発において，品質確保活動であるテストやレビューといったソフトウェアのメンテナンスは重要であり，かつ工数をかけすぎず効率的に行うことが期待されている．そのため，現在のソフトウェア開発ではバージョン管理システム，バグトラッキングシステム，さらに工数見積りツールなどといった様々な開発支援のためのツールが開発され利用されている．

上記で述べた開発支援方法以外に，品質確保活動の工数を削減する手段として Fault-prone モジュール (バグを含むかもしれないモジュール) を予測する手法の研究が数多く行われている [1, 2, 3, 4]．この手法は，バグの含む可能性が高いと予測されたモジュールから重点的に処理できること，さらに早期にバグを発見し除去できることから，品質確保活動の工数を削減する効果や，ソフトウェア自身の品質向上への効果が期待されている．また Kamei らは，モジュールの粒度を変更レベルとした予測モデル [5] を提案し，テストすべき変更を順序付けることにより工数を削減することができたと述べている．さらに Kamei らの予測モデルでは，変更レベルで予測をするため，バグの早期発見やバグの混入者の特定も可能であった．そして，こういった予測モデルを実際に開発支援のためのツールとして現場に適用させていくことが Fault-prone モジュール予測の研究の最終的な課題となっている．

そこで本研究では，構築が簡単で高い予測性能をもつ「Fault-prone フィルタリング」[6] という予測モデルを利用して Fault-prone モジュール予測ツールの試作を行う．この予測モデルは，その他の予測モデルが構築に様々なメトリクスの収集環境を必要とするのに対し，ソースコードのみで構築が可能である．また，試作するバグ予測ツールでは，Kamei らの提案した変更レベルの予測の利点に注目し，ソフトウェアリポジトリからコミットごとに即時的にバグを予測する．本ツールの性能の評価実験は，学習で用いた Git リポジトリの履歴を参照することで，バグが含まれるかどうかがわかっているモジュールと変更情報を関連付け，それらを新しい変更があったと想定してツールに与えることで行う．

2. 目的

本研究の目的は「Fault-prone フィルタリング」という予測モデルを用いて，Git リポジトリから変更ごとに即時的にバグを予測するツールを試作し，「Fault-prone フィルタリング」を用いたバグ予測ツールの有効性を考察することである．

この目的のため，以下の研究設問を設定し，ツールの有効性について考察する．本研究における研究設問は次の通りである．

RQ1: Fault-prone フィルタリングによる即時バグ予測ツールは構築できるか．

RQ2: 提案する即時バグ予測ツールによるバグ予測では，どの程度の精度が得られるのか．

RQ3: 提案する即時バグ予測ツールは開発に有用な情報を提供できるか．

3. 試作の準備

3.1. SZZ アルゴリズム

SZZ アルゴリズム [7] とは，バグトラッキングシステムのバグデータベースとバージョン管理システムのバージョンアーカイブを相互に結びつけて解析することでバグを混入したモジュールを特定するアルゴリズムである．

本研究では，研究室において開発・保守されているツール (szz_tools) を利用した．このツールは，バージョンアーカイブとして Git リポジトリ，バグデータベースとしてバグ情報の記述された csv ファイルを用いて SZZ アルゴリズムを適用し，バグが含まれるモジュールとバグの含まれないモジュールを特定してその情報をリポジトリにタグとして添付するものである．このツールでのモジュールの粒度は一つのファイルである．

3.2. Fault-prone フィルタリング

Fault-prone フィルタリングとは，スパムメールの判別を行うスパムフィルタ (CRM114) の技術をソフトウェアのソースコードに適用し，コードのテキスト情報のみから Fault-prone モジュールの予測を行うものである．スパムフィルタは，スパムメールには特定の単語群や頻繁に含まれているという事実に基づき，過去に受信したメール内の単語群を学習してスパムメールかハムメールかを分類するための識別辞書を作成する．そして新しいメールについて，識別辞書とベイズ識別を利用してスパムメールかどうかの分類を行う．これをソフトウェアに関しても，バグが存在するところには特定のコード片が存在するという類推のもと適用する．テキスト情報のみで予測を行うため，複雑なソフトウェアメトリクスを使用した従来の予測モデルよりも容易にプロジェクトに適用することができる．

3.3. 欠陥の変更時の即時予測

従来の Fault-prone モジュールの予測モデルはファイルやパッケージレベルでの予測のものがほとんどであった．それらの欠点として以下のことが挙げられる．

- 予測の粒度が粗い
- 予測が出たとしても，バグを混入した開発者が特定できない
- 予測が遅すぎて，開発サイクルと咬み合わない

これらの欠点のため，変更 (git などにおけるコミット) レベルでの即時予測を行える予測モデルが注目されている．それらの利点として以下のことが上げられる．

- 予測の粒度が細かい
- バグを混入した開発者は，変更を行った開発者であると簡単に特定できる
- 予測が早いので，開発サイクルへのフィードバックが容易

「予測の粒度が細かい」というのは，変更はファイルの差分の集合であると考えられるからである．本研究では，この変更レベルでの即時予測の利点に注目する．

しかし，本研究で使用する Fault-prone フィルタリング自体の予測粒度は第 3.1 節で述べたツールの粒度から，ファイルレベルである．そこで本研究では新しい変更ごとにその変更で修正されたファイルに対して Fault-prone フィルタリングを実行することで変更レベルでの予測とする．これを実装した本ツールに期待される利点は次の通りである．

- 新しい変更に対して即時的にファイルのバグの確率を提示できる．

- コミット情報を保持しているのでバグの確率を上げた開発者及び日付などを特定できる．
- 確率はファイルごとに予測されるが，Git を用いてバグの確率が変動した前後で差分を取ることにより，どういったコード片がバグの確率を上げているのかがわかる．

4. 即時バグ予測ツールの試作実験

本章では，本試作ツールが特定の Git リポジトリを継続的に監視して即時的にバグを予測していく際のツール内部の動作ついて説明する．また，ツールとしてのユーザインタフェース部分である Web ページ部分の表示に関する内部動作，操作方法ついてもこの章で述べる．最後に，予測性能の評価手法について説明する．

4.1. 予測の準備

4.1.1 リポジトリとバグ情報の取得

本実験ではツールでの予測対象として次の 3 つの Java で開発されているオープンソースプロジェクトのリポジトリとそのバグ情報を取得した．

- Apache OpenJPA
- Apache James
- Eclipse Birt

4.1.2 バグ情報との統合

本実験ではリポジトリをバグを予測するための学習のデータセットとして用いるために，これらのリポジトリに対して研究室で開発・保守されている SZZ アルゴリズムを用いたツール (szz_tools) を適用してリポジトリにバグ情報を与える．このツールを用いると，Git の tag コマンドを利用してバグ除去時点のコミットには FIX タグを，バグの混入時点のコミットには BUG タグを添付することができ，さらにバグが含まれるモジュール (ファイル) を特定し，FAULT タグを添付した後，それ以外に INNOCENT タグを添付することができる．

4.2. 予測の手順

本実験のツールでは，前節で用意した Git リポジトリへの新しい変更に対して継続的に予測を行っていく．予測の手順は，以下の項の通りである．

4.2.1 学習

新しい変更に対して予測を行うためには，Fault-prone フィルタリングによる初期学習が必要である．前節で用意したバグ情報を統合した各 Git リポジトリに対して次の手順で学習を実行し，識別辞書を作成する．

1. FAULT タグと INNOCENT タグをリポジトリから取得する．
2. これらのタグを用いて，リポジトリからモジュールのソースコードを読み出し，FAULT であるか INNOCENT であるかの情報と共に記録していく．
3. それらすべてをトレーニングセットとしてテキストフィルタを用いて学習させ，FAULT と INNOCENT の識別辞書をそれぞれ作成する．

本実験のツールは試作であるため，スパムフィルタリングで用いられる手法である「誤判定時のみ学習」による学習の強化を実装していないので，この学習の処理は予測を開始する前の 1 回のみ行われる．

4.2.2 変更の取得

変更の取得と解析の手順は次の通りである．

1. git fetch コマンドを使用して差分を取得する．
   このコマンドにより，Git はリモートリポジトリからローカルリポジトリとの差分を取得する．このとき，フェッチしてきたコミットの中で最新のものは Git 内で FETCH_HEAD という名前で参照できる．
2. git log HEAD..FETCH_HEAD –name-status コマンドを使用して差分に対するログを取得する．
   「HEAD..FETCH_HEAD」という引数を指定をすることにより，HEAD(ローカルリポジトリの最新のコミット) から FETCH_HEAD(リモートリポジトリの最新のコミット) までの範囲を指定して Git からログを取り出す．さらに「–name-status」とい

う引数を指定することにより，各コミットで変更されたファイルのファイル名とその変更情報を加えて取得することが出来る．

3. ログを解析する．
   得られたログから各コミットごとにそのコミットの情報と，予測する対象のモジュールとして追加，変更，コピー，または修正のあったファイルの名前を記録していく．

4. git merge FETCH_HEAD コマンドを使用してローカルリポジトリを更新する．
   このコマンドにより，差分のログの解析が終わったローカルリポジトリをリモートリポジトリと同じ状態にする．つまり HEAD が FETCH_HEAD の指すものとなる．これにより次回の変更の取得に備えることができる．

4.2.3 分類

前節でログを解析して得られた各モジュールをスパムフィルタによる分類にかけ，得られた確率，ファイル名，及びコミットの情報をデータベースに格納していく．

4.3. リポジトリの監視

本実験のツールで，リポジトリの監視をしていくためには以下の操作を行う．

- 登録用プログラムによるツールへのリポジトリの登録
  前節第 4.2.1 項で述べたように，予測をする前には学習が必要である．しかしこの学習には時間が大きくかかるため，リポジトリ登録用プログラムをツールの一部として作成した．このプログラムは，学習が終わった時にツールへとリポジトリの情報の登録を行う．新しいリポジトリを監視したい時は，この登録用プログラムの引数として新しいリポジトリを指定することによってツールへ監視対象として登録ができる．

- cron によるリポジトリの監視の設定
  本実験ではリポジトリの動きを監視するために，Unix 系 OS においてコマンドの定時実行のスケジュール管理を行うために用いられる cron というプログラムを利用する．cron に前節の第 4.2.2 項から第 4.2.3 項までの処理を行うプログラムを指定し，任意の時間間隔で自動的に実行させる．これにより，定期的に変更があったかどうかを確認し，即時的な Fault-prone モジュールの予測を行うことが出来る．

4.4. Web ページへの表示

それぞれのページについての出力や操作方法について以下に説明する．

- main.cgi
  本ツールのメインページであり，図 1 で示すように予測結果の検索フォームによる検索と，cron での実行のログの確認ができる．検索フォームで入力できるものは次の通りである．
  - Repository's name (セレクトボックス)
    結果を表示したいリポジトリの名前を指定する．
  - Omit commits which have no java files (ラジオボタン)
    Java ファイルを持っていないコミットを表示するかどうかを指定する．
  - Order of commit's date (ラジオボタン)
    コミットの日付が新しい順番で表示するか古い順番で表示するかを指定する．
  - Date (日付入力欄)
    コミットの期間を指定する．
  - Commit ID (テキストボックス)
    コミットの ID(40 桁の SHA-1 ハッシュ) を指定する．
  - Author (テキストボックス)
    コミットした開発者名を指定する．
  - Commit message (テキストボックス)
    コミットメッセージを指定する．
  - File name (テキストボックス)
    ファイル名を指定する．
  - show (ボタン)
    上記の条件で検索を開始する．検索結果は show_result.cgi で表示する．

- show_result.cgi
  main.cgi の検索フォームからの入力を受け取り，そ

れらを条件としてデータベースから予測結果を検索し，Web ページにコミットごとに予測結果をまとめて図 2 のように表示する．

- about.html
  本ツールの使い方について記述されているページ．
- setting.cgi
  監視しているリポジトリに対して，データの削除などの設定を行う．
- setting_apply.cgi
  setting.cgi からの入力を受け取り，実際にデータベースやデータファイルの内容を変更し，設定結果の旨を表示する．

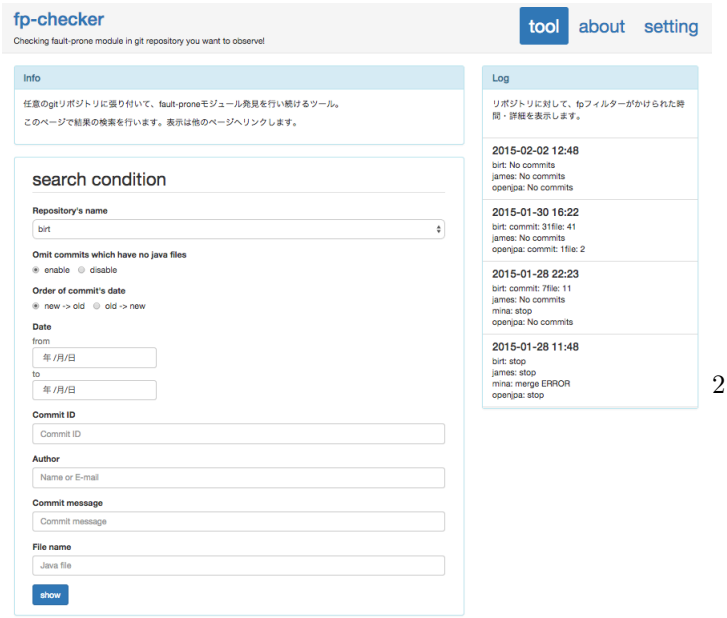


図 1. ツールのメインページ

4.5. 予測性能の評価実験

評価実験の概要

最後に，本研究において本ツールの性能・信頼性の評価の実験は，研究設問に回答し，ツールの有効性を考察するために必要不可欠なものである．

この実験では，実際のツールの使用を想定して，学習時に用いた第 4.1.2 節で FAULT，または INNOCENT タグを添付されたモジュール，つまりバグかどうか予めわかっているものを，Git リポジトリの履歴を参照し変更と関連付け，それらの変更を新しい変更があったと想定してツールに与えることで評価する．また，ただデータを集計するだけでなくツールとして Web ページ上で，どの程度予測の精度があるのかを確認できるようにする．

以下の手順で評価実験を行う．

1. テストデータの準備
   本ツールで行う予測では，テストのために利用するデータはコミットごとにそのコミットの情報と関連付けられていているモジュールでなければならない．しかし，モジュールに添付された FAULT タグと INNOCENT タグにはコミットの情報が入っていないため，リポジトリの履歴を参照し，タグを逆算することによりコミットごとのモジュールを取得する．取得するコミットは，バグを含むモジュールと含まないモジュールの数のバランスを考え，BUG タグがついているコミットを選択する．SZZ アルゴリズムの特性上，これをしなければバグを含まないモジュールの数が多くなってしまうためである．取得したコミットとモジュールを関連付けたものをテストデータとする．
2. 分類テスト
   前述で準備したテストデータを用いて，分類テストを以下の手順で行う．
   - 各モジュールに対してバグである確率を予測して，モジュールが実際にバグであるかの情報と共に予測した確率をデータベースに登録する．
   - データベースから前述で登録した情報を取り出し，閾値を 0.5 として分類して結果を集計する．

5. 予測の結果

5.1. 対象とした Git リポジトリの学習時のデータ

前章で述べたように 3 つの実験対象としたリポジトリに対して，SZZ アルゴリズムを利用したタグの添付によりモジュールがバグを含むかどうかを特定した．その結果の集計を表 1 に示す．

Result Acc: 0.7868 Pre: 0.8044 Rec: 0.8381 F1: 0.8209

author	commit	message	date
Richard G. Curtis <curtisr7@apache.org>	e48cd3e59d82ce352d161109987f283ecc369b66	OPENJPA-2508 : Account for JOIN FETCH statements when loading from the Query Cache. git-svn-id: https://svn.apache.org/repos/asf/openjpa/trunk@1600757 13f79535-47bb-0310-9956-ffa450edef68	2014-06-05 20:49:01
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department_.java			45%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/TestJoinFetchWithQueryDataCache.java			100%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee_.java			92%
openjpa-kernel/src/main/java/org/apache/openjpa/datacache/QueryCacheStoreQuery.java			100%
Richard G. Curtis <curtisr7@apache.org>	b95e0d1c6c2e2cc8f6f9d422058dfe232f6091d5	OPENJPA-2502 : Update accessPath metas in CriteriaExpressionBuilder. Merged changes from 2.2.1.x. Patch contributed by Albert Lee. git-svn-id: https://svn.apache.org/repos/asf/openjpa/trunk@1600682 13f79535-47bb-0310-9956-ffa450edef68	2014-06-05 15:48:54
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/TestJoinFetchWithQueryDataCache.java			100%
openjpa-persistence/src/main/java/org/apache/openjpa/persistence/criteria/CriteriaExpressionBuilder.java			100%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Department_.java			34%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee.java			0%
openjpa-persistence-jdbc/src/test/java/org/apache/openjpa/persistence/jpql/joins/Employee_.java			89%

図 2. 予測結果表示ページ (対象は Apache OpenJPA)

これらのモジュールは，ツールでの即時予測と評価実験を行うための学習でデータセットとして用いた．

表 1. 各 Git リポジトリの学習時のモジュール数

	Apache OpenJPA	Apache James	Eclipse Birt
コミット数	4383	11866	29501
バグを含まないモジュール	9273	21600	41794
バグを含むモジュール	12518	3794	35398
合計	21791	25394	77192

5.2. 評価実験

5.2.1 評価指標

試作したツールの評価実験で得られた結果の評価指標として精度 (Accuracy)，再現率 (Recall)，適合率 (Precision)，F_1 値を用いる．分類結果の凡例を表 2 に示す．

表 2. 分類結果の凡例

実測 \ 予測	バグを含まない	バグを含む
バグを含まない	TN	FP
バグを含む	FN	TP

- TP (True Positive): バグと分類したもので，実際にバグであったモジュールの数．
- FP (False Positeve): バグと分類したもので，実際はバグでなかったモジュールの数．
- TN (True Negative): バグでないと分類したもので，実際にバグでなかったモジュールの数．
- FN (False Negative): バグでないと分類したもので，実際はバグであったモジュールの数．

精度 (Accuracy)

精度は，分類結果全体の中で分類成功であった割合を示す．よって以下のように定義される．

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

全体的な傾向を把握することができるがデータの隔たりなどに大きく影響を受けるので，再現率と適合率を併用する．

再現率 (Recall)

再現率は，実際にバグであったモジュールの中でバ

グであると分類したものの割合，つまり網羅率を示す． よって以下のように定義される．

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

適合率 (Precison)

適合率は，バグであると分類したモジュールの中で実際にバグであったものの割合，つまり無駄なモジュールをどれだけ調べる必要があるかというテストのコストを示す．よって以下のように定義される．

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

F_1 値

F_1 値は，再現率と適合率がトレードオフの関係にあるため，両者を総合的に判断するための指標であり，再現率と適合率の調和平均によって示される．よって以下のように定義される．

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{4}$$

5.2.2 評価結果

第 4.5 節の手順の通りに結果を示す．

1. 用意したテストデータ
   用意したテストデータの集計を表 3 に示す．この表で示すデータには，本研究の変更ごとの即時バグ予測を行うという特性上から，複数のコミットが同じ状態のモジュールを指すことがあるため，テストセットとして同じモジュールがいくらか含まれている．そのため独立したモジュールの数も集計した．

表 3. 評価実験で用いたテストデータ

	Apache OpenJPA	Apache James	Eclipse Birt
コミット数	1828	1513	7098
バグを含まないモジュール	6375	9481	18732
独立したモジュール	6160	5812	18489
バグを含むモジュール	8909	4947	24253
独立したモジュール	8811	2347	24049
モジュールの合計	15284	14428	42985

2. 分類テストの結果
   各リポジトリの分類結果を表 4 から表 6 に示す．次に前節で定義した評価指標を用いた分類テストの評価を表 7 に示す．

またツールとして，Web ページでは，図 3 に示すようにファイル名の右側に実際にバグを含む ([B]) か含まないか ([N]) という情報を加えて表示することで，どの程度予測が正しいかツール上でもおおまかに確認することが出来た．また，計算した評価指標を Web ページ上でも示すことによって，ツールの予測性能の状態を把握できるようにした．

表 4. 分類結果 (Apache OpenJPA)

	予測	
実測	バグを含まない	バグを含む
バグを含まない	4559	1816
バグを含む	1442	7467

表 5. 分類結果 (Apache James)

	予測	
実測	バグを含まない	バグを含む
バグを含まない	3657	5824
バグを含む	233	4714

表 6. 分類結果 (Eclipse Birt)

	予測	
実測	バグを含まない	バグを含む
バグを含まない	6498	12234
バグを含む	2519	21734

6. 考察

6.1. 本ツールの有効性

表 4 から表 7 に示した評価実験の結果を踏まえて，第 2 章で定めた研究設問に答えることで本ツールの有効性について考察する．

RQ1: Fault-prone フィルタリングによる即時バグ予測ツールは構築できるか．

本予測モデルの構築に必要となるのはソースコードのみであるので，ツールとしての実装は容易である．この点に関しては，様々なメトリクスを要求する予測モデルを用いたツールよりも優れていると考えられる．しかし，FindBugs[8] のような静的コード解析ツールと比較すると，機械学習のコストが構築の際にかかってしまう．

Result Acc: 0.5802 Pre: 0.4473 Rec: 0.9529 F1: 0.6088

author	commit	message	date
Stephen K. Brewin <sbrewin@apache.org>	d425aebb99aa790ec245ab5824f041ec27e4c26d	JAMES-1355 Switched to using the FileSystem service to obtain the 'sieve' sub-directory of the root directory of the application git-svn-id: https://svn.apache.org/repos/asf/james/server/trunk@1222684 13f79535-47bb-0310-9956-ffa450edef68	2011-12-23 14:09:28
/mailets/src/main/java/org/apache/james/transport/mailets/ResourceLocatorImpl.java [N]			0%
/mailets/src/main/java/org/apache/james/transport/mailets/LocalDelivery.java [N]			100%
/mailets/src/main/java/org/apache/james/transport/mailets/SieveMailet.java [N]			0%
Norman Maurer <norman@apache.org>	00f75520dea733f81b8f9f470e1a4f3174711d41	Add protocols-lmtp as dependency as we factor out the lmtpserver code to protocols. See PROTOCOLS-1 git-svn-id: https://svn.apache.org/repos/asf/james/server/trunk@1176509 13f79535-47bb-0310-9956-ffa450edef68	2011-09-27 18:40:48
/lmtpserver/src/main/java/org/apache/james/lmtpserver/CoreCmdHandlerLoader.java [B]			100%
/lmtpserver/src/main/java/org/apache/james/lmtpserver/hook/MailboxDeliverToRecipientHandler.java [N]			0%
/lmtpserver/src/main/java/org/apache/james/lmtpserver/DataLineLMTPHandler.java [N]			0%
Norman Maurer <norman@apache.org>	528d39591513fbc86d5675609c1f185514ebd668	Add file based queue implementation. See JAMS-1316 git-svn-id: https://svn.apache.org/repos/asf/james/server/trunk@1171244 13f79535-47bb-0310-9956-ffa450edef68	2011-09-15 19:52:51
/queue-file/src/main/java/org/apache/james/queue/file/FileMailQueue.java [N]			100%
/queue-file/src/main/java/org/apache/james/queue/file/FileMailQueueFactory.java [N]			0%

図 3. ツール上での評価結果の表示 (対象は Apache James)

表 7. 分類テストの評価

	精度	再現率	適合率	F_1 値
Apache OpenJPA	0.7868	0.8381	0.8044	0.8209
Apache James	0.5802	0.9529	0.4473	0.6088
Eclipse Birt	0.6568	0.8961	0.6398	0.7466

RQ2: 提案する即時バグ予測ツールによるバグ予測では，どの程度の精度が得られるのか．

評価実験により，表 7 に示すような精度が得られた．この結果に対して以下のことが考察できる．

- このツールでのバグの網羅性
  得られた再現率は 8 割から 9 割であり，これは本ツールでチェックしたコミットに属するファイルのうち，実際にバグであるもののほとんどを網羅できていることを示す．
- このツールでの確率の提示による品質確保活動のコスト
  得られた適合率は，4 割から 8 割とかなりばらつきがあった．4 割であった場合，1 つのバグを含むモジュールを発見するためには，1 つか 2 つの実際はバグを含まないモジュールを調べなければならないことになる．しかし，このツールではバグの確率を変更ごとに予測することができるため，Git を用いてバグの確率が変動した前後で差分を取り，その差分のみをレビューするなどの工夫をすることで，工数を削減できる．また，予測のタイミングが早く，開発者の特定も可能なこともコードのメンテナンスにかかる工数を減らす要因となるだろう．

RQ3: 提案する即時バグ予測ツールは開発に有用な情報を提供できるか．

本ツールは，変更ごとにモジュールがバグを含む確率の情報を与えることでコストをかけるべきファイルを選択することが出来る．評価実験では，バグであると判断する閾値を 0.5 としたが，この値を変更することで品質確保活動のコストの調整がある程度可能となるだろう．例えば，本実験結果では再現率が十分に高いので，閾値を上げることでバグの判断を厳しくし，レビューするモジュールを減らすことができる．

また，開発者へ開発のフィードバックを与えること

が出来る．本ツールで実際にリポジトリを監視していたところ，図 4 のように「Refactor code.」というメッセージを持つ変更の前後でファイルに対するバグである確率の変動が見られた．この図のように確率が下がった場合は，達成度を開発者に与えられるだろう．また，このようなときに Git を用いて差分を取ればどういったコード片がバグの可能性を高めているのかを知ることが出来る．

6.2. 妥当性の検証

本研究の妥当性について以下に示す．

- オープンソースプロジェクトに対する実験
  実験で用いたのは，全てオープンソースのプロジェクトである．このため商業のプロジェクトに今回の実験と同じ手法を適用しても，同様の結果が得られるとは限らない．

- SZZ アルゴリズム
  本研究でモジュールにバグを含むかどうかのタグを添付するのに用いた SZZ アルゴリズムは，完全にバージョン管理システムとバグ情報を結び付けられるものではない．

  また，最近のコミットで変更されたファイルについては，バグ情報が不足しているのでほとんどのものに INNOCENT タグがつけられてしまう．

- 評価実験におけるテストデータの偏り
  本実験では，上述した SZZ アルゴリズムの特性から，INNOCENT タグが添付されたモジュールが多くならないように，BUG タグのついているコミットのみを実験時のツールの入力として使用した．しかし，BUG タグが付けられているコミットの期間，つまりバグ情報が存在している期間を抽出し，その期間での BUG タグがついていないコミットもテストデータとして含めた方が実験における評価の妥当性が上がる可能性がある．

  また，結果の表 3 に示すように Apahce James においては，独立したモジュールの数が全体のモジュールに対して半分ほどであったため，本実験の評価の妥当性は低い可能性がある．

6.3. 今後の課題

本研究で試作したツールにおける今後の課題は以下の通りである．

- 継続的な学習の実装
  本ツールを実際の現場へと適用するためには，予測した結果が正しかったかどうかの情報を後から入力することにより，誤判定ならば学習するという「誤判定時のみ学習」による状況に合わせた学習の強化は必須である．また，そのたびに Web ページ上で提示する精度などの評価指標も更新すると良いと思われる．

- 予測結果の Web ページへの表示速度の改善
  予測結果の絞込をする際，データベースの検索に時間が掛かる場合があった．このためデータベースのスキーマやプログラムを改善する必要がある．

7. 結言

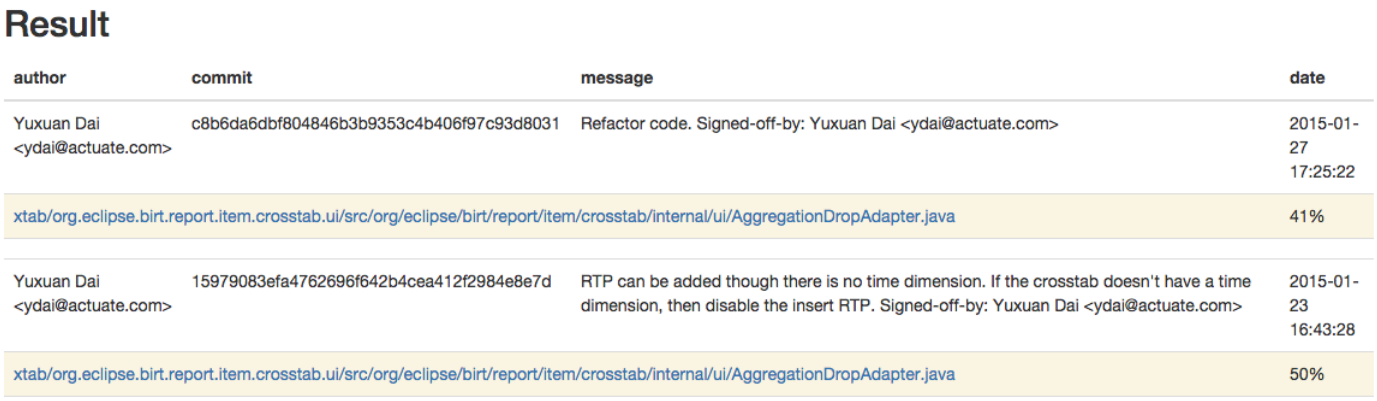
本稿では,「Fault-prone フィルタリング」という予測モデルを利用して Fault-prone モジュール予測ツールの試作を行い，このツールの理論，内部動作，操作方法，評価手法およびその結果，考察を述べてきた．3 つのソフトウェアリポジトリを対象にした評価実験の結果をもとに考察を行った結果,「Fault-prone フィルタリング」を用いたバグ予測ツールは構築可能であり，本ツールはある程度の予測性能を伴って予測結果をツールの利用者に Web ページで提示することが可能であるとわかった．

参考文献

[1] 畑　秀明, 水野　修, 菊野　亨, “不具合予測に関するメトリクスについての研究論文の系統的レビュー “, コンピュータソフトウェア, vol.29, no.1, pp.106-117, Feb. 2012.

[2] P. Bellini, I. Bruno, P. Nesi, and D. Rogai, “Comparing Fault-Proneness Estimation Models“, *Proc. of 10th IEEE International Conference on Engineering of Complex Computer Systems*, pp.205-214, 2005.

[3] Tim Menzies, Jeremy Greenwald, and Art Frank, “Data Mining Static Code Attributes to Learn Defect Predictors“, *IEEE Trans. on Software Engineering*, vol.33, no.1, pp.2-13, Jan. 2007.



Result

author	commit	message	date
Yuxuan Dai <ydai@actuate.com>	c8b6da6dbf804846b3b9353c4b406f97c93d8031	Refactor code. Signed-off-by: Yuxuan Dai <ydai@actuate.com>	2015-01-27 17:25:22
xtab/org.eclipse.birt.report.item.crosstab.ui/src/org/eclipse/birt/report/item/crosstab/internal/ui/AggregationDropAdapter.java			41%
Yuxuan Dai <ydai@actuate.com>	15979083efa4762696f642b4cea412f2984e8e7d	RTP can be added though there is no time dimension. If the crosstab doesn't have a time dimension, then disable the insert RTP. Signed-off-by: Yuxuan Dai <ydai@actuate.com>	2015-01-23 16:43:28
xtab/org.eclipse.birt.report.item.crosstab.ui/src/org/eclipse/birt/report/item/crosstab/internal/ui/AggregationDropAdapter.java			50%

図 4. バグである確率の変動 (対象は Eclipse Birt)

[4] Naeem Seliya, Taghi M. Khoshgoftaar, and Shi Zhong, “Analyzing Software Quality with Limited Fault-Proneness Defect Data“, *Proc. of 9th IEEE International Symposium on High-Assurance Systems Engineering*, pp.89-98, 2005.

[5] Yasutaka Kamei, Emad Shihab and Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi, “A Large-Scale Empirical Study of Just-in-Time Quality Assurance“, *IEEE Trans. Softw. Eng*, vol.39, no.6, pp.757-770, June 2013.

[6] O. Mizuno and T. Kikuno, “Training on Errors Experiment to Detect Fault-Prone Software Modules by Spam Filter”, *The 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2007)*, pp.405-414, 2007.

[7] J. Śliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?”, *Proceedings of the 2005 international workshop on Mining software repositories*, pp.1-5, 2005.

[8] (2015, Mar.) Findbugs™- find bugs in java programs. [Online]. Available: `http://findbugs.sourceforge.net/`