一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
MSS2020-32,SS2020-17(2021-01)

テキスト分類による不具合予測システムの実装と企業環境での評価

水野　　修† 　黒田　翔太† 　石原　一宏†† 　山下　大輔†††

† 京都工芸繊維大学
†† バルテス株式会社
††† バルテスモバイルテクノロジー株式会社
E-mail: †o-mizuno@kit.ac.jp, †††ishihara@valtes.co.jp

あらまし　ソフトウェア開発工程において，不具合 (バグ) を早期に発見できれば，修正の労力を少なく抑えることができる．そのため，バグ予測についての研究は数多くなされてきた．我々は，そうした研究の 1 つとしてスパムメールの判別に用いられるベイズ識別を用いたシステムを提案してきた．本論文では，このシステムを用いて，実際の開発に利用できるようなツールに作成することを目的とした．この研究において開発したツール，「CodeLamp」は Eclipse プラグインとして利用することができるもので，ソースファイル，及び Git コミット差分について，テキスト分類に基づいたバグ予測を行うことができる．開発したツールをソフトウェア開発現場での実開発の一部に適用し，評価を行った．評価では，開発の版ごとに予測精度が変化する様が確認された．また，導入後に収集されたアンケート調査によって，さらなる改良点が明確になった．
キーワード　不具合予測，ソースコード，テキスト分類，実環境評価

An Implementation of Text-classification Based Fault-prone Module Detection and Its Application to Industrial Environment

Osamu MIZUNO†, Shouta KURODA†, Kazuhiro ISHIHARA††, and Daisuke YAMASHITA†††

† Kyoto Institute of Technology, Kyoto, Japan
†† VALTES CO.,LTD., Osaka, Japan
††† VALTES Mobile Technology CO.,LTD., Osaka, Japan
E-mail: †o-mizuno@kit.ac.jp, †††ishihara@valtes.co.jp

Abstract　In the software development, early detection of software defects (bugs) contributes to mitigate the development effort and improve the quality of software. For this reason, much research has been done in the prediction of software defects. We proposed an approach for defect prediction using the Bayesian classification. In this study, we implemented a software system of our approach that can be used in the actual software development. We have applied our tool in a part of the actual development project. Based on the feedback from the developers, we confirmed that our tool can be used in the development.
Key words　Defect prediction, Source code, Text classification, Field study

1. ま え が き

ソフトウェア開発におけるバグの混入は開発工程に悪影響を及ぼし，バグの未発見はソフトウェアの質の低下を招く．しかし，開発中の全リソースを人力で精査することは極めて大きな負担となる．このような状況への対処として，開発過程におけるバグの絞り込みと早期発見が必要である．バグがありそうな箇所が絞り込めれば，個別に精査・修正するにしても負担は減る．バグが開発途中に修正できるのであれば，テスト時に発見される不具合も少なくなる．バグがソフトウェアテストの時点で減っているのであれば，修正と再テストに要する回数も少なくなると期待されている．

一般に，これらソフトウェアへのバグの混入を知ることは非常に難しく，出現して初めて認知されることになる．そこで，事前の情報からバグの混入を予測する研究が行われている．そのための手段の一つとしてで研究されているのが「バグ予測」である．これまでに蓄積されてきた開発工程の記録をもとに，バグが発生していそうなモジュールを予測する．これを「Fault-prone モジュール予測」と呼ぶ．

これまでに「Fault-prone モジュール予測」に関する多くの研

究がなされてきた．[1]〜[3] 例えば，2011 年末には，Google がバグ予測アルゴリズム [4] を公開し，それを実装した「bugspots」[5] というツールが登場するなど，様々な研究が行われている．

本研究では，それら「Fault-prone モジュール予測 [6]」の一つ，スパムメールフィルタリングの仕組みで実現する「Fault-prone フィルタリング」という手法を用いる．Fault-prone フィルタリングの研究は数多くなされ [6]〜[10]，多くの検証ツール・スクリプトも作られたが，それらを利用者を選ばずに実用化することは難しかった．本研究はこの「Fault-prone フィルタリング」を「開発現場」で試用できるツールとして実現することを目標としたものである．

本研究はバグ予測に関するものであり，説明や操作において「バグ」の対義語が必要となる．そこで本研究においては，「Bug，バグ，不具合」を「Faulty」なものとまとめ，その対義語「NonBug，バグでない，不具合のない」を「Non-Faulty」なものとして総称する．

2. 研 究 背 景

過去の「Fault-prone フィルタリング」に関する研究では，様々な入力対象に対して，Fault-prone フィルタリングを検証してきた．それらの研究ごとに測定ツール・スクリプトが利用されてきたが，それらは検証の意味合いが強く，それらを直接，開発現場に導入することは難しかった．

そこで，本研究は共同研究の一環として，Fault-prone フィルタリングを実装した，「現実的な」開発ツールの補助アプリケーションを作成する．

本研究は，補助アプリケーションの実装環境案として，フトウェア開発で広く使われている統合開発環境「Eclipse」[11] において利用する Eclipse プラグインとして Fault-prone フィルタリングを実装を目指したものである．

2.1 ツールとしての要件

現実的な開発ツールの補助アプリケーションとして実装するために，不特定多数の開発者の利用を想定し次の要件を挙げる．

- Eclipse 本体および Java 実行環境を除く外部ツールの利用，特にインストールの必要な外部ツールの利用を避ける．
- Eclipse はマルチプラットフォームアプリケーションであるので，同様に，なるべくマルチプラットフォームで利用できるツールとする．

また，Eclipse プラグインとして作成するにあたり，次の点を考慮する．

- シンプルである．つまり，使い方が過剰に複雑化しない．
- 使用者によってツールの利用方法は一定ではないことに配慮したインターフェースを作る．

一方で，Fault-prone フィルタリングに用いるデータベースを，初めて作成するときには，膨大なソースコードから学習すると想定されるために，非常に時間がかかることが想定されるため，Eclipse プラグインとは独立したツールとして用意する．

3. 準　　備

3.1 Eclipse プラグイン

Eclipse [11] は Java で実装された統合開発環境であり，オープンソースソフトウェア (OSS) として提供されている．数ある統合開発環境の中でも特にユーザ数が多いと考えられるため，今回の開発における基盤として選定した．

「Eclipse」において，「Eclipse プラグイン」が機能拡張を果たす．基本となる Eclipse Platform 以外の機能，例えば，Java も含め各開発言語への対応も Eclipse プラグインによるものである．つまり，プラグインを実装することで，Eclipse は様々な機能を追加できる大きな汎用性を持っている．

Eclipse プラグインの開発もまた，Eclipse プラグインである PluginDevelopment Environment(PDE) という専用のツールを使用する．Eclipse プラグインで UI を作成するためには，Eclipse の GUI 作成に使われている SWT や JFace を利用することができる．SWT も JFace も，Swing・AWT や JavaFX といった GUI ライブラリと同様に，GUI を持つ Java アプリケーションの作成に利用でき，Eclipse プラグイン専用のライブラリというわけではない．PDE を利用して，Eclipse の UI ライブラリ (org.eclipse.ui.*) 上に SWT・JFace を用いて GUI を構成する．

本研究では SWT で UI を構成した Java アプリケーションを作成し，その上でプラグイン化する．

3.2 JGit [12]

Eclipse では，Git バージョン管理として EGit による UI が提供されているが，その EGit が Git 操作に使用している API が JGit である．

JGit とは，Git リポジトリに対して Git バージョン管理を Java から行える機能を実現する JavaAPI である．Java プログラマが Git 操作を行う機能を実装するためには，ProcessBuilder クラスを用いてコマンドプロンプトやシェルなどの外部プロセス経由で Git コマンドを使用することもできるが，そのためは実行環境に Git がインストールされている必要があり，万全ではない．一方で，JGit を使えば，Git がインストールされていない実行環境においても Java コードによる Git 操作が可能となる．

3.3 Git

開発などで広く使われる「分散型バージョン管理システム」なるもので，ファイルの変更履歴を保存・追跡するための仕組み．Git を適用した Git リポジトリと呼ばれるディレクトリは「.git」フォルダを持ち，その下にファイル変更時に関する記録 (変更者，時間，変更理由) や，ハッシュで分類されて変更差分などの様々な記録が保存されている．

変更を記録することは「コミット」と呼ばれ，このコミット単位ごとに履歴が管理・追跡される．

Git は一本道に変更履歴を保存・追跡するだけではなく，「ブランチ」として履歴を分岐して記録することもできる．「ブランチ」を分ける目的は，他のブランチの影響を受けない環境で，開発・修正して他のブランチに合流したり，別の案・プロジェクトとして独立するために利用される．また，作業中の最新のコミットは「HEAD」として保存されており，ここから作業中

のブランチも判別する．

本研究で主に利用する相当する Git コマンドは `git log` と `git diff` である．`git log` は Git の更新情報を確認するためのコマンドであり，`git diff` は 2 つのコミット間の差分を出力する．

3.4 Fault-prone フィルタリング法

モジュール (例えばソースコード) 中にバグが含まれていそうなことを予測することを「Fault-prone モジュール予測」といい，Fault-prone フィルタリング法とは，Fault-prone モジュール予測の手法の一つ．Fault-prone モジュール予測の例では，「bugspots」というツールが利用している Google のバグ予測アルゴリズムなども知られている．

Fault-prone フィルタリングとは，Orthogonal Sparse Bigram (OSB) を用いてベイズ推定を行うという，スパムメールフィルタリングに用いられる手法をソースコードに対して応用するという，水野らの論文 [6] に基づく手法である．本研究では，この Fault-prone フィルタリングをバグ予測に利用する．

3.5 Orthogonal Sparse Bigram (OSB)

OSB とは，ある単語に対して 5 文字目までの各単語との組み合わせで分割した単位をもとにする方法である．これを用いたベイジアンフィルタ，OSBF [13] を本研究で使用する．

例えば，“WORA is a slogan of the Java language . . . .” という文章を OSB で解釈する分類すると，先頭の「WORA」に対して次のような組み合わせが成り立つ．

表 1　単語対生成例

	WORA	is	a	slogan	of	the	Java	language
単語対 1.1	WORA	is						
単語対 1.2	WORA		a					
単語対 1.3	WORA			slogan				
単語対 1.4	WORA				of			
単語対 1.5	WORA					the		
単語対 2.1		is	a					
単語対 2.2		is		slogan				
⋮								

3.6 ベイジアンフィルタ (ベイズ推定)

ベイジアンフィルタは，対象に対しベイズの定理 (Bayesian theorem) に基づく条件付き確率の推定を行うもので，スパムメールフィルタなどで用いられている．

スパムメールフィルタを参考に，条件付き確率の推定のための計算を整理すると，カテゴリ (Cat) をスパム (spam)，スパムでない (ham) とし，対象となる入力 (Doc) は単語 (word) で形成されている ($Doc = \Sigma_i word_i$) とすると，

$$P(spam|Doc) = P(Doc|spam) \times \frac{P(spam)}{P(doc)}$$
$$= \Pi_i P(word_i|spam) \times \frac{P(spam)}{P(doc)}$$

のようにして，各単語がスパムカテゴリに含まれているの積算に対して，係数 $\frac{1}{P(Doc)}$ とカテゴリの事前確率 $P(spam)$ が乗算されて，条件付き確率が推定される．

3.7 CRM114 Descriminator [14]

OSB を実現するツールとして CRM114 がある．本研究で使用しているのは次のパッケージである．

- CRM114crm114-20081111-BlameBarack-Ger-4560 [15]

OSB 自体は SQL のようなデータベースでも実装できるが，仮に SQL データベースで実装した場合，単純に重複なしと考えた場合，OSB の仕組み上，約 5 倍のテキスト情報が必要となってリソースを要し，また，処理時間も大きくなる．

一方，CRM を用いれば，データベースのリソースは抑制できる．CRM ではバケットとみなしたバイナリをデータベースとして利用するので，データベース作成時のリソースを上回ることがない．また，C 言語ベースで実装されており動作も軽快である．

CRM114 が利用する OSBF というフィルタ方式では，独自の係数を条件付き確率に乗じて予測値として算出している．[13]

4. CodeLamp

本研究にあたり作成した「Fault-prone フィルタリング」ツールを「CodeLamp」と呼ぶ．

本ツールでは，ソースコードに対する「Fault-prone フィルタリング」を用いた判別・学習を行う．

開発当初は Java のみで Git コミット差分を取得することが困難であると見込まれたため，ソースファイルに対する自動判別機能の実装にとどまり，「ぼやっとした (ソースファイルという広い) 範囲で」バグの可能性を判断できる程度の予定であったためこのような名称となった．

4.1 ツールの構成と動作環境

CodeLamp は次の 3 つのツールから構成されている．

- CodeLamp-Creator

CodeLamp を利用するための準備を行う．

- CodeLamp-E

ソースファイル，もしくは Git コミットを対象として，バグ予測を行う本体の Eclipse プラグイン版．

- CodeLamp-J

バグ予測を行う本体の Java アプリケーション版．

上記の各ツールについて，利用方法を**??**にてまとめる．

動作条件として，CodeLamp-E は Eclipse，CodeLamp-Creator・CodeLamp-J については Java RuntimeEnvironment [16] の導入が必要となる．

Windows 環境下での動作を想定しているが，UNIX(OSX 及び Linux) で動作させるためには，別途ツールをインストールする必要がある．

4.2 CodeLamp-Creator の動作

「Faulty」「Non-Faulty」に区分されたコミットはリストとして保持され，GUI とは別のスレッドで随時コミット差分を出力・学習する．別のスレッドを用意したのは，学習には時間がかかるため，アプリケーションが止まったと誤解されないようにす

るため．

コミットメッセージをもとに自動的にラベリングしているが，単純にメッセージに「bug」と含まれていれば Faulty なコミット，「fix」と含まれていれば Non-Faulty なコミットとしてみなしているため，厳密な判別はしていない．ラベリングは手動で変更できる．

コミット差分は生成するフィルタ用データベースと同じディレクトリを作業場としてに生成される．このコミット差分はコミット時刻がひとつ前のコミットと比較して出力されている．コミット差分の出力時に，対象言語以外のソースコードやドキュメントの差分は，学習におけるノイズになりかねないので取得しない．

なお，JGit は diff 経由で差分を表示する必要があるので，初回のコミットについては出力することができない．

4.3 CodeLamp-E 及び CodeLamp-J の動作

4.3.1 ソースファイルの自動判別，及び手動判別・学習

別スレッドにて判別キュー，学習キューの監視を行っている．各キューが要素を持っていた時はそのソースファイルを判別，もしくは学習する．

また別のスレッドにて対象プロジェクトを監視し，ソースファイルの変更・追加・削除を監視している．なお，この監視は，リソースを少しでも少なく抑えるために，フィルタに関連付けられた言語の拡張子以外のファイルを無視している．

自動判別は，ソースファイルが待機状態 (No order)・除外指定 (exclude) ではないときに，最終判別時間よりも最終更新時間が遅いときに，用判別とみなして学習キューに追加する．

4.3.2 Git コミットごとの自動判別，及び手動判別・学習について

別スレッドにて，該当 Git リポジトリの HEAD の変更を監視している．HEAD が変更された場合，Git コミット一覧の再取得と，新しい HEAD に関して判別キューに追加する．

ソースファイルの方と異なり，同じスレッドで判別・学習キューの監視も行っている．これは，コミットの方がファイル更新よりも頻度が低いため，干渉する可能性は低いと考えられるため．こちらもデータベース作成時と同じく，対象言語以外のソースコードやドキュメントの差分は，学習におけるノイズになりかねないので取得しない．

5. 動 作 実 験

本ツールで，期待される一連の動作が行えることを確認し，本実装における要件を満たしているか確認した．

5.1 実 験 準 備

動作確認実験として次の 2 種類を行った．

（1） 実験 1 Java で実装されたあるプロジェクトについて，過去のコミットをもとに学習したデータベースから，特定の 5 コミットを対象として判別する．

- CodeLamp-Creator を使用して自動ラベリングに従って学習したデータベースを使って，CodeLamp-J にて対象コミットについて手動で判別する．

（2） 実験 2 複数の GIt リポジトリのコミットから学習データベースを作成し，それをもとに実験 1 と同じ対象について判別する．

- CodeLamp-Creator を用いて自動ラベリングをもとに学習・生成したデータベースを用いて，CodeLamp-E にて当プロジェクトを判別する．

5.2 実 験 対 象

（1） 実験 1

- バグデータベースと対応して，すでに Faulty/Non-Faulty 箇所を該当コミットにタグ付けしている Git リポジトリを対象とする．
  - mina [17]
  - openjpa [18]
  - james [19]
- コミットメッセージ中に含まれる「bug」「fix」の単語に従った自動分類を学習内容とし，バグタグ付け済みコミットについて判別結果を取得する．

（2） 実験 2

- 学習用プロジェクト
  - mina
  - openjpa
  - james
  - eclipse [20]
  - junit [21]
  - tomcat [22]
  - jetty [23]
- 実験 1 と同じ判別対象のコミット差分について，判別結果を取得する

5.3 確認する要件

ツールに関して実験時に確認すべき点として，特に次の要件を挙げる．

- 要件 1 Eclipse 本体および Java 実行環境を除く外部ツールの利用，特にインストールの必要な外部ツールの利用を避ける．
- 要件 2 Eclipse はマルチプラットフォームアプリケーションであるので，同様に，なるべくマルチプラットフォームで利用できるツールとする．
- 要件 3 シンプルである．つまり，使い方が過剰に複雑化しない．
- 要件 4 使用者によってツールの利用方法は一定ではないことに配慮したインターフェースを作る．
- 要件 5 安定した動作をする．

5.4 実験の結果

対象となるコミットは，コミットメッセージから自動判別して学習したところ，表 2 の量の学習量が得られた．

対象となったコミット数が多くても，内容 (ソースコード) の文量や，学習済み単語数との重複で一回の学習量は異なる．作成したデータベースのバケットは最大 3396996 個で，比較的学習量が多かった james で全てのバケットが使用されたが，各バケットはそれぞれ異なる値を保有するため，データベースの最大まで学習した訳ではない．

表 2　フィルタ用データベースごとの学習量

学習対象	分類	使用バケット数
mina	Faulty	45,395
	NonFaulty	2,027,588
openjpa	Faulty	111,894
	NonFaulty	524,288
james	Faulty	3,396,996
	NonFaulty	3,396,996
実験 2	Faulty	3,396,996
	NonFaulty	3,396,996

表 3　対象のバグコミットについての判別実験 1,2 の結果

	Commit		実験 1		実験 2	
Name	Hash	Time	Prob.	Result	Prob.	Result
mina	af37d31	2012/2/15 15:22:50	0	NF	0	NF
	cc20296	2011/9/23 17:29:04	0.5	F	0	NF
	7c581a8	2011/8/26 20:21:42	0	NF	0	NF
	e33bd1c	2010/9/10 23:03:56	0	NF	0.5	F
	10e9280	2010/9/10 11:20:27	0	NF	0	NF
openjpa	7063dd5	2012/3/17 1:43:34	0	NF	0	NF
	b68cf1d	2012/3/11 6:26:31	0	NF	0	NF
	ecf492a	2012/3/8 0:20:04	0.5	F	0	NF
	ce44b7d	2012/2/25 5:35:02	0	NF	0	NF
	c275da6	2012/2/25 1:09:31	0	NF	0	NF
james	34a875d	2012/2/21 17:30:43	0.5	F	0	NF
	8c512fb	2012/2/14 20:23:19	0.5	F	0.5	F
	d7ecf44	2012/2/5 20:12:58	0.5	F	0.5	F
	f968934	2012/1/27 2:12:06	0	NF	0.5	F
	73cec66	2012/1/20 19:57:41	0	NF	0	NF

これらのデータベースを用いて行った実験結果表 3 のように求まった．

予測結果である probably 値 (Classified val) が 0 であるものというのは，「Faulty」データベースに予測対象の単語対があまり学習されていなかったため，「Faulty」予測が非常に小さく出てしまったため，probably 値が 0 で「Faulty」と判別されてしまっている．実験 2 ではより学習量の多いデータベースを使用したにも関わらず，結果からはその影響を見ることが出来なかった．

5.5　考　　察

これら実験 1，2 はツールの動作確認が目的だったが，利用時における課題として，次のことが確認できた．

- 判別精度

– 学習量の少ないリポジトリでは特に判別できなかった．一方で，学習量が実験対象の中では比較的多くても，今回の実験で行った程度の学習量では不足しているように見えた．

– とは言え，この実験程度の学習量でも，データベース作成には非常に時間がかかったため，個人で多くのソースコードを学習したデータベースを作成するのは負担が大きい．

– 学習データにも問題は多く，コミットメッセージから「bug」か「fix」かで分類しただけなので必ずしも適切なコミットを取得できていないという懸念はあった．

– 学習対象として，バグの発生が確認されたとバグデータベースからタグ付けられたコミットを対象としたが，それぞれタグ付けされた時点の 1 コミットについて判別しただけなので，必ずしもバグコードが含まれている部分ではなかった (それ以前のコミット時にバグが含まれたが発見されていなかったものなど) 可能性もある．

6.　試験利用による評価

6.1　適用前における評価

製作中の本ツールを現場の技術者に提示し，要件についていくつかの改善を得た．その際に，「Git のコミット差分についても判別できるようにしてほしい」との意見が得られた．

開発初期において，「コミット差分の出力を行う」ことが困難であると認識したため，当時の予定では「ソースコードについて判別・学習する」，「ソースコードが更新された場合，そのソースコードについて判別する」という機能を目的として製作していた．その上で，

「開発過程では Git で管理していることが多く，コミットの差分について判別することで，『どの編集部分でバグが生じた可能性があるか』分かる方が都合がよい」

との率直な意見を頂き，再検討した結果現完成版において「Git のコミットについても判別・学習する」機能を追加するに至った．

6.2　実環境における試行

試験利用における目的は，本手法が実際の開発現場において有効に活用できるかを確認することである．そのため，我々は共同研究実施企業の開発者に対して，本ツールの仕様と操作マニュアルを提示して，開発環境における導入を促した．また，同時に本手法の動作アルゴリズムについての説明も実施している．

適用に当たっては，まず，対象企業の過去のプロジェクトから取得した不具合関連データを用いて不具合・非不具合辞書を作成した．この初期コーパスを組み込んだシステムを開発環境において適用し，いくつかのコミットにおいて標準的な手順での不具合判定を実施した．適用後に開発者からのフィードバックを収集して分析を実施した．

6.3　開発者からのフィードバック

本研究において，開発者は新規の開発リポジトリを準備し，対象となるソースコードのファイルをインポートした．その上で，開発リポジトリ上でいくつかの更新を実装するタスクを行った．この際に，開発したツールを利用したため，自動的に不具合の推定が実施されている．

コミットごとにツールは不具合傾向の予測結果を表示した．また，もし予測結果が誤っていると思われた場合には，開発者が手動で判定を変更し，ツールに正しい予測結果（不具合の有無）を学習させるように促している．このプロセスを一連のタスクが終了するまで繰り返した．

タスクが終了した後，利用した時に気づいた点などをまとめ

て，フィードバックとした．このフィードバックより，次のようなことが明らかになった．

- **開発者は，本ツールを静的コード解析ツールのように利用した．**

本ツールはコーディング中の即時バグ検出ツールとして利用するつもりでの開発であった．開発者が実際に利用してみると，各コミット時点で静的解析ツールのように利用する方が，開発者のニーズに合致することが分かった．

- ***1* コミットのサイズが想定よりも大きい．**

開発者がツールを利用する頻度が想定よりも低かったため，1回の分析に対して渡されるコード差分の量が多くなってしまった．その結果，ツールが頻繁にフリーズしたような症状を呈した．（実際には計算に時間が掛かっていた．）その結果として，利用性に関する難点としてレポートが上がってきた．

- **ツールのインターフェイスが単純すぎる．**

開発者の作業を邪魔しないように本ツールのインターフェイスは可能な限りシンプルにしていたが，その結果として，静的解析ツールのように利用する開発者に対しては何をすれば良いのかが分からなくなる状況になってしまった．

- **不具合予測の精度があまり高くない．**

事前に学習済みのコーパスを利用したが，最初の実行では殆どが間違った予測結果を返した．過去の研究 [24] でも同様の報告がなされているが，現時点で有効な方法は見つかっておらず，今後の課題となる．

- **不具合の位置を正確に予測するべき．**

開発者からは，どこに不具合があるのかをより詳細に教えて欲しいという要望が上がった．現在のツールはファイルそのものか，ファイルの差分かに対して不具合の存在可能性を指摘する．ファイルや差分のどの部分かを指摘できれば，さらに利便性が向上するが，精度を保ったままで実現することは難しい．

7. ま と め

本報告では Fault-prone フィルタリングの実用化を目指し，開発環境 Eclipse 用のプラグインとしたツール「CodeLamp」の開発を行った．事前に定めた仕様を満たす実装を行い，また，現場からの意見による仕様改訂を通じて，実践的なツールとして整備した．

今後の課題として，以下が挙げられる．まず，概ね動作を確認することはできたが，Git リポジトリの構造をわかりやすくビューに反映させる必要がある．実験 1 の結果からは，1 プロジェクトほどの大きさでは学習量が圧倒的に少なく，判別精度が低いことがわかる．しかし，個々の所持するプロジェクトの量はそれほど多くないので，実験 2 のように，まったく別のプロジェクトを利用して結果を得られることが望ましい．まず，学習段階において大量のノイズが混入しているので，それをいかに取り除くかという問題が認識されている．一方で，使用者がノイズなく学習させることができるような仕組みの開発も必要である．一方で，一連のツール群として CodeLamp を整備してため，系統的に評価実験を行うことが可能となった．また，開発補助ツールとして開発現場での使用の道を拓くこともできるため，今後の共同研究における活用が期待できる．

文 献

[1] P. Bellini, I. Bruno, P. Nesi, and D.Rogai, “Comparing fault-proneness estimation models,” Proc. of 10th IEEE International Conference of Engineering of Complex Computer Systems, pp.205–214, 2005.

[2] L.C. Briand, W.L. Melo, and J.Wust, “Assessing the applicability of fault-proneness models across object-oriented software projects,” IEEE Trans. on Software Engineering, vol.28, no.7, pp.706–720, 2002.

[3] L.Guo, B.Cukie, and H.Singh, “Predicting fault prone modules by the dempster-shafer belief networks,” Proc. of 18th International Conference on Automated Software Engineering, pp.249–252, 2003.

[4] Google.com, “Bug prediction at google”. http://google-engtools.blogspot.jp/2011/12/bug-prediction-at-google.html

[5] I. Grigorik, “github igrigorik/bugspots (mit license)”. https://github.com/igrigorik/bugspots

[6] O. Mizuno and T. Kikuno, “Training on errors experiment to detect fault-prone software modules by spam filter,” Proc. of 5th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundation of software engineering, pp.405–414, 2007.

[7] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, “Fault-prone filtering: Detection of fault-prone modules using spam filtering technique,” Proc. of 1st International Symposium on Empirical Software Engineering and Measurement, pp.374–383, 2007.

[8] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, “Spam filter based approach for finding fault-prone software modules,” Proc. of Fourth International Workshop on Mining Software Repositories (MSR07), p.4, May 2007.

[9] 森 啓太, 水野 修, “スパムフィルタに基づく即時バグ予測ツールの試作,” ソフトウェア・シンポジウム 2015，pp.37–46，June 2015.

[10] 藤原剛史，水野 修，“バイトコードを用いたテキスト分類による不具合予測,” ソフトウェア・シンポジウム 2015，pp.80–88，June 2015.

[11] eclipse.org, “Eclipse”. https://eclipse.org/

[12] eclipse.org, “Jgit (javaapi)”. http://www.eclipse.org/jgit/

[13] osbflua.luaforge.net, “Exponential differential document count”. http://osbf-lua.luaforge.net/papers/yosbf-eddc_slides.pdf

[14] crm114.source.forge.net, “Crm114 discriminator”. http://crm114.sourceforge.net/

[15] hebbut.net, “Crm114 for win32/unix”. http://hebbut.net/Public.Offerings/crm114.html

[16] Oracle.com, “Java runtime environment”. https://java.com

[17] apache.org, “github apache/mina”. https://github.com/apache/mina

[18] apache.org, “github /openjpa”. https://github.com/apache/openjpa

[19] apache.org, “github /james”. https://github.com/apache/james

[20] eclipse.org, “github eclipse/eclipse”. https://github.com/eclipse/eclipse

[21] apache.org, “github /junit”. https://github.com/junit-team/junit

[22] apache.org, “github apache/tomcat”. https://github.com/apache/tomcat

[23] eclipse.org, “github eclipse/jetty”. https://github.com/eclipse/jetty

[24] O. Mizuno and T. Kikuno, “Training on errors experiment to detect fault-prone software modules by spam filter,” ESEC-FSE ’07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp.405–414, ACM, New York, NY, USA, 2007. Dubrovnik, Croatia.