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

信学技報
TECHNICAL REPORT OF IEICE.

Just-In-Time 欠陥予測支援ツール anko

田中秀太郎† 福島 崇文† 山下 一寛† 亀井 靖高† 鵜林 尚靖†

† 九州大学

あらまし ソフトウェア開発において，効率良くレビューやテストを行うためにソースコードファイルの変更，つまり版管理システムに対するコミットを対象とした欠陥予測（Just-In-Time 欠陥予測）の有用性が示されている．我々の研究グループでは，版管理システム Git により管理されているソフトウェア開発プロジェクトに対して Just-In-Time 欠陥予測を簡単に適用でき，かつ，各機能 (メトリクスの計測やモデルの構築) がカスタマイズ可能であるツールを開発した．本発表では，開発したツールを実際に利用してもらい，利用者から得たフィードバックを基に行ったツールの改善点，及び，利用シナリオについて主に報告する．

キーワード 欠陥予測，変更レベル，Just-In-Time

Just-In-Time Defect Prediction Tool anko

Shutaro TANAKA†, Takafumi FUKUSHIMA†, Kazuhiro YAMASHITA†, Yasutaka KAMEI†, and Naoyasu UBAYASHI†

† Kyushu University

Abstract Some previous studies show that defect prediction at the change-level (i.e., defect prediction for a commit to version control systems) is useful. We refer to this type of defect prediction as Just-In-Time defect prediction. We are developing a prototype tool that developers can easily perform Just-In-Time defect prediction to software development projects and can customize each function such as measuring metrics and building prediction models. In this paper, we describe an overview of this tool and report some improvement points based on feedback from users. And we also report use scenarios.

Key words Defect Prediction, Change Level, Just-In-Time

1. はじめに

ソフトウェア開発において高い品質を確保するためには，できる限り多くのレビュー工数やテスト工数をかけることが望ましい．その一方で，レビューやテストにかけられる人員や費用には限りがあるので，効率良くそれらの品質保証活動を実施することが求められる．この問題を解決する一手段は，欠陥が多く含まれると考えられるモジュールにレビュー工数やテスト工数を重点的に割り当てることである．

そのために，欠陥を含む確率の高いモジュールを予測する手法 (以降，欠陥予測手法) が，これまでに多数提案されてきた [1] [7] [9]．提案，及び，評価されてきた欠陥予測手法では，モジュール (パッケージやファイルなど) が主な予測の単位であり，モジュールのメトリクス (コード行数など計測可能な指標) を説明変数とし，欠陥の有無を目的変数とした数学的モデルを構築する．

これらのモデルは，テスト割り当ての優先付けにおいて一定の有用性が認められるもののいくつかの解決すべき課題を有する．その課題を解決するために，変更レベル (注1) での予測が着目されている．変更レベルでの予測が着目される理由は，1) ファイルに対する予測に比べて予測対象の粒度が小さく，どの部分に欠陥が含まれているかを把握しやすい，2) 予測結果を開発者にすぐにフィードバックできる，という利点があるためである．本稿では，変更毎に欠陥予測を行いソフトウェアの品質向上に役立てることを，Just-In-Time 欠陥予測 (以降，JIT 欠陥予測) と呼ぶ．

JIT 欠陥予測は，開発現場での実用性・有効性が期待できる．しかしながら，一般的な欠陥予測研究も含めて，予測精度を評価する取り組みが積極的に行われている反面，その研究成果を開発プロジェクトに適用するための試みは少なく，欠陥予測研究を開発現場にすぐに適用するための環境も整備されているとは言えない．

(注1)：本稿では，版管理システムに対するコミットを変更の単位とする．

そこで本稿では，先行研究 [17] にて開発を行った JIT 欠陥予測を開発プロジェクトに適用するためのツールを実際に使用してもらい，得られたフィードバックを基にツールの問題の検出，改良を行った．ツールの利用者からのフィードバックにより，1) 処理速度が遅い，2) 予測精度に不安がある，3) 使い方がわかりづらい，という問題が得られた．これらの問題を，実装の変更，新しい手法の追加，利用シナリオの整理，といった方法で改良を行った．

以降，2. では，関連研究について述べる．3. では，JIT 欠陥予測をプロジェクトに適用するための環境に求められる要件やその実現に向けた方針，及び，ツールの実装方法について述べる．4. では，ツールの評価として利用者から得たフィードバックを基に行った改良について述べる．5. では，利用シナリオを通したツールの有用性を議論する．そして，6. で本稿のまとめと今後の課題を述べる．

2. 関 連 研 究

2.1 メトリクスと欠陥の関係

これまでに，メトリクスと欠陥の関係を明らかにするために，コードメトリクス (ソースコードを解析することで計測できるメトリクス．ファイルの総行数やクラス，メソッドの個数など) とプロセスメトリクス (開発の履歴を解析することで計測できるメトリクス．ファイルの変更回数や開発者数など) に関する研究が行われてきた [11] [13].

例えば，Moser ら [11] は，欠陥予測に対するこれら 2 種類のメトリクスの効果を評価するために，Eclipse プロジェクトから計測したデータセットを用いて実験を行った．実験により，プロセスメトリクスのほうがコードメトリクスよりも欠陥予測に有用であることを示した．

また，Rahman ら [13] も，同様の研究を行っており，12 のオープンソースプロジェクトの 85 のリリースを用いて欠陥予測モデルを構築した．実験の結果，Moser らと同様，プロセスメトリクスのほうが有用であると示した．コードメトリクスがモデル構築に寄与しなかった理由として，コードメトリクスはリリース間でメトリクスの値に大きな変化が生じないため，リリース間で欠陥の有無が異っているにも関わらずメトリクスの値がほぼ同じという状態が多く発生したためと考察した．

その他に，開発者の経験や開発者数などの属人性に関する研究も行われている [8] [12]．例えば，Matsumoto らは，プロダクトを開発した開発者の特性をメトリクスとして定義し (開発者メトリクス)，欠陥との関係を分析した．Eclipse プロジェクトを分析した結果，開発者の欠陥の混入のさせやすさには少なくとも 5 倍以上の個人差があることを示した．

これらの従来研究ではファイル単位で計測されたメトリクスを用いている一方で，本稿ではソフトウェアの変更から計測できるメトリクスを対象としている．

2.2 ソフトウェア変更からの欠陥検出

版管理システムへのコミットを変更の単位として，Sliwerski ら [15] は，ソフトウェアの変更が欠陥を含むか否かを自動的に検出する方法として，SZZ アルゴリズムを提案した．SZZ アルゴリズムは，欠陥を修正した変更を見つけてから，欠陥を混入させた変更を見つけるという欠陥検出方法である．具体的な手順としては，

（1） 版管理システムに記録されている各変更のコメントの中から，Bug や Fix といった欠陥修正に関係するキーワードを探し，欠陥の修正を目的とした変更を特定する．

（2） その変更によって修正された後のソースコードと，修正される前のソースコードとの差分を調べることで変更箇所 (欠陥) を見つける．

（3） その修正される前のソースコードの該当部分がどの時点で記述されたのかをたどることで，いつ，どのファイルに，誰が欠陥を混入させたかを特定する．

また，Kim らは欠陥の検出精度を上げるために，SZZ アルゴリズムを改良した [4]．Kim らは版管理システム Subversion で管理されるプロジェクトを対象に以下のような実装を行った．

（1） コメント修正の無視．
ソースコードのコメント行の修正はたどらない．

（2） 空白，フォーマット修正の無視．
ソースコードの空白の増減やフォーマット変更の行の修正はたどらない．

（3） 外れ値の無視．
欠陥の修正を目的とした変更の内，一度に変更したファイル数が平均より 5 倍以上多い変更を外れ値として無視する．

SZZ アルゴリズムは従来研究でも多く利用されており，本稿でもツールの実装に用いた．

2.3 ソフトウェア変更に対する欠陥予測

本稿と同様に，ソフトウェアの変更を欠陥予測の対象とした研究が行われきた [3] [14] [15].

Shihab ら [14] は，企業のプロジェクトから計測したデータを用いて，変更レベルでの欠陥予測の実験を行った．その結果，追加行数，欠陥の修正を行うための変更，その変更と関連付けられた欠陥レポートの数，変更を行う開発者の経験量が，欠陥予測において有用なメトリクスであることを示した．

また，Sliwerski ら [15] は，2 つのオープンソースプロジェクト (Mozilla，Eclipse) を対象として，変更レベルでの欠陥予測における欠陥混入のリスクを上げる要因を調査した．ケーススタディの結果，大きい処理の一部の変更，欠陥の修正を行うための変更，金曜日に行われた変更は欠陥混入のリスクが高いことを示した．

Kamei ら [3] は，6 つのオープンソースプロジェクトと 5 つの企業のプロジェクトのデータを対象に，変更レベルでの欠陥予測の性能を実験的に評価した．評価実験では，版管理システムから 14 種類のメトリクスを計測し，モデル構築手法としてロジスティック回帰分析を用いた．評価実験の結果，全工数の 20%を用いるだけで，欠陥を含む変更の 35%を判別できることを示した．

従来研究では，欠陥予測モデルの精度や欠陥混入のリスクを上げる要因に着目している一方で，本稿では欠陥予測を開発プロジェクトに適用するための支援を行うツールに着目している点が異なる．

表 1　モデル構築手法

手法	略称	R のパッケージ
ランダムフォレスト	rf	randomForest
ロジスティック回帰分析	glm	stats
決定木	rpart	rpart
サポートベクターマシン	ksvm	kernlab
ニューラルネットワーク	nnet	nnet

2.4　欠陥予測の支援ツール

欠陥予測モデルの精度を評価するだけでなく，欠陥予測をプロジェクトに適用することを支援するためのツールの開発も行われている [2] [10] [16].

Hovemeyer ら [2] は FindBugs というツールを開発した．FindBugs は Java で実装され，Eclipse プラグインとして提供されている．FindBugs では，欠陥パターンがあらかじめ 400 以上設定されており，ソースコード (Java が対象) を静的解析することで，引数の型の間違いや，switch 文に重複したコードがあるといったパターンを検出し，欠陥を含む確率が高いディレクトリやファイルの箇所を開発者に報告する．

Mockus ら [10] は開発履歴を用いた欠陥予測を行うツールを開発した．Mockus らは，開発履歴から計測したメトリクス (同時に変更されたファイル数など) を説明変数として用い，ロジスティック回帰分析によって予測モデルを構築した．構築された予測モデルは，5ESS という開発プロジェクトの版管理システムに組み込まれ，ツールとして開発プロジェクトに提供された．当該ツールは，開発者が新たな変更を版管理システムにコミットすると，変更から自動的にメトリクスを計測し，予測結果を開発者に示す．

Zimmermann ら [16] は ROSE というツールを開発した．ROSE は，Java 言語による開発プロジェクトを対象としており，開発履歴からソースコードの編集がどのように行われているかを調べる．Entity(データのまとまり) 同士の関連度を調べてルール化することで，関連度の高い Entity の片方が変更されたときにもう片方も変更されるべきであると推薦する．また，関連度の高い Entity の片方が変更されずに版管理システムにコミットされた場合は，警告を行う．

3.　JIT 欠陥予測ツール概要

先行研究 [17] にて提案及び試作した JIT 欠陥予測ツールの概要について述べる．3.1 では，JIT 欠陥予測を開発プロジェクトで実施する際の課題について述べる．3.2 では，課題を解決するためのツールの設計方針について述べる．3.3 では，課題を解決する機能について述べる．3.4 では，ツールの実装方法について述べる．

3.1　JIT 欠陥予測を実施するための課題

課題 1.　ソフトウェア開発プロジェクトの初期段階では，開発履歴が少ないため，予測モデルを構築することができない．

課題 2.　欠陥予測の実施は，開発者は開発履歴からデータの収集，加工を行った上で，メトリクスの計測や欠陥の検出を行わなければならないため，手間が掛かる．

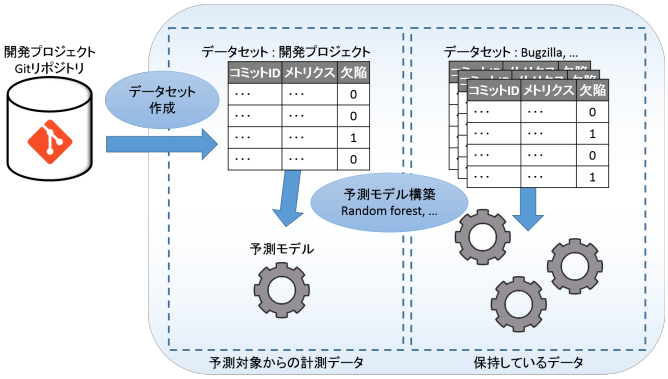

図 1　欠陥予測モデル構築の流れ

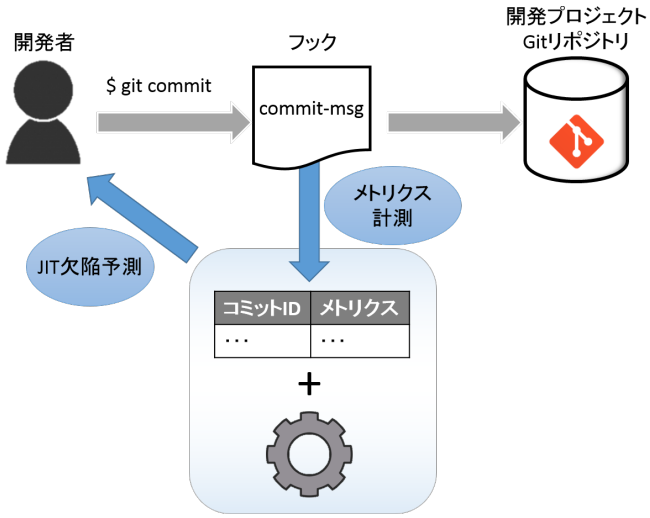

図 2　JIT 欠陥予測の流れ

課題 3.　予測モデルを構築するためには，統計手法や機械学習手法をデータセット (各変更のメトリクスと欠陥の有無) に適用するが，モデル構築手法には様々な種類がある．どのモデル構築手法が良いかはプロジェクトによって異なるため，複数の手法を比較したいが，各手法の構築方法の調査に手間が掛かる．

3.2　方　　針

本ツールは，3.1 に示す課題を解決することで，開発プロジェクトにおける JIT 欠陥予測の実施の支援を目的としている．Git リポジトリを実装対象の版管理システムとして，メトリクスの計測機能，欠陥混入変更の検出機能，予測モデルの構築，分析機能を実装した．Git を対象とした理由は，近年，多くのオープンソースプロジェクトで採用され，かつ，ソフトウェアリポジトリマイニング分野でも評価に頻繁に利用されているためである．

3.3　課題を解決するための機能

本ツールの 3 つの機能を以下に示す．機能 1，2，3 は，それぞれ 3.1 の課題 1，2，3 と対応している．

機能 1.　新規開発 (もしくは開発が始まって間もない) プロジェクト，および，データ収集を行っていないプロジェクトにおいても，欠陥予測モデルを利用することができる．Bugzilla，

Eclipse などのオープンソースソフトウェア (OSS) プロジェクトや，開発者自身の他のプロジェクトから計測されたデータセットを用いて欠陥予測モデルを構築する．現時点で，4 つの OSS プロジェクトのデータセット (表 2) をツール内に保存しており，予測対象プロジェクトの開発履歴がなくても欠陥予測を行うことができる．

機能 2. モデル構築に必要なメトリクス，欠陥を自動的に計測し，データセットを作成する (図 1)．また，開発者によってソースコードが変更されコミットされると，当該変更からもメトリクスを自動的に計測する (図 2)．

機能 3. 欠陥予測モデルを構築する手法を柔軟に選択できる仕組みを提供する．本ツールでは，モデル構築手法として 5 つの手法を用意した (表 1)．これにより開発者は少しでも精度の高い予測モデルを選択することができる．また，この 5 つに限らず，開発者が他に利用したいモデル構築手法を開発者自身が追加できる仕組みを提供する．

3.4 実　　装

Git リポジトリから得られるログファイルなどを解析し，メトリクスを計測するプログラムや，各変更に対して欠陥を検出するプログラムは，Ruby を用いて実装した．また，モデル構築やその後の欠陥予測に関するプログラムについては，統計パッケージが豊富に用意されている R を用いて実装した (表 1)．これらのプログラムを自動的に実行するために，Git のフック機能 (注2) を用いた．

3.4.1 メトリクス計測機能の実装

ソフトウェアの変更 (コミット) に対して欠陥予測を行うために，従来研究 [3] で用いられたメトリクスの中から，5 つのカテゴリに分類される 14 種類のメトリクスを計測する機能を実装した．これらのメトリクスは開発プロジェクトのプログラミング言語に依存しないため，どの開発プロジェクトからでも計測できる．

3.4.2 欠陥検出機能の実装

各コミットが欠陥を含むか否かを検出する方法として，関連研究に示す，Sliwerski ら [15] によって提案された SZZ アルゴリズムを実装した．

3.4.3 欠陥予測機能の実装

メトリクスと欠陥の有無に関する情報をまとめたデータセットから予測モデルを構築する機能を実装した．本ツールでは，予測モデルを構築する手法として，ファイル単位での欠陥予測においてモデル間の性能比較を行った Lessmann ら [5] の従来研究を参考に，5 つのモデルのカテゴリ (Statistical classifiers や Decision tree approaches など) からそれぞれ 1 つのモデル構築手法を選んだ (表 1)．このモデルに新しいコミットから計測したメトリクスを入力することで欠陥予測を行う．

3.4.4 コミット時のプログラム呼び出し機能の実装

Git のフック機能を用いて，開発者がコミットする (ソフトウェアの変更を行う) 毎に欠陥予測の結果を開発者に示す．その実現のために，コミット完了前に実行されるフックである commit-msg ファイル (注3) を変更した．これによりコミットが行われることで，メトリクス計測 (3.4.1)，および，欠陥予測 (3.4.3) を行うプログラムが実行される．プログラムが呼び出されると，欠陥予測の結果が出力される．開発者は欠陥予測の結果を確認し，その結果次第でコミットの中止または完了を選択することができる．

(注2)：コミットをきっかけに自動的に実行されるプログラム．

4. フィードバックによる改良

本ツールを Git リポジトリを用いて開発を行う個人の開発者に利用してもらい，利用者から得たフィードバックを基に以下の改良を行った．

4.1 処理速度の改良

ツールの実行に時間がかかりすぎるとのフィードバックを得たため，以下の改良を行い，処理速度を上げた．

4.1.1 メトリクス計測の高速化

メトリクス計測の処理に時間がかかりすぎるとの意見を得たため，メトリクス計測の高速化に取り組んだ．具体的には，あるコミットの経験量に関するメトリクスの計測を行う際に，それ以前のコミット全てを参照していたが，メトリクスの値を蓄積しながら行うように改良した．これにより以前のコミット参照にかかる時間がなくなり，各コミットの計測にかかる時間が短くなった．

4.1.2 JIT 欠陥予測の高速化

コミットしてから欠陥予測結果を出力するまでの時間が長いとの意見を得たため，JIT 欠陥予測の高速化に取り組んだ．具体的には，予測モデルの構築を以前はコミット時に毎回行っていたが，データセットの作成の際に同時に構築し，保存しておくように改良した．これにより欠陥予測結果を出力するまでの時間が短くなった．

4.2 欠陥予測に用いるアルゴリズムの追加

コミット単位以外の欠陥予測結果との比較も行いたいとのフィードバックを得たため，Lewis ら [6] の研究で採用されている，コミットを基にファイル単位での予測を行うアルゴリズムを実装した．このアルゴリズムはコミットに対するコメントを基に欠陥が含まれていそうなファイルを見つける．これによって，JIT 欠陥予測の際にコミットに対する予測と変更したファイルに対する予測の両方を行うことができる．

4.3 利用シナリオの改良

ツールで何を行えるかがわかりづらいとのフィードバックを得たため，ツールの用途を理解しやすくするために，利用シナリオの整理を行い，5. にまとめた．

4.4 そ の 他

4.4.1 欠陥検出精度の向上

欠陥検出の精度が低いとの意見を得たため，欠陥検出精度の向上に取り組んだ．具体的には，Kim ら [4] の論文を参考に SZZ アルゴリズムの改良を行った．

(注3)：フックの一種．コミットに対するコメントを記述した後，コミットの処理が完了する前に，このファイルが実行される．

表 2　計測済みの OSS データセット

OSS プロジェクト名	コミット数	欠陥コミット数	計測期間	GitHub URL
Bugzilla	9,156	6,525	1998-08-26 - 2014-09-29	https://github.com/bugzilla/bugzilla.git
Eclipse JDT	21,259	7,518	2001-06-05 - 2014-10-14	https://github.com/eclipse/eclipse.jdt.core.git
Eclipse Platform	7,930	2,811	2001-05-03 - 2014-10-08	https://github.com/eclipse/eclipse.platform.git
PostgreSQL	37,270	12,655	1996-07-09 - 2014-10-12	https://github.com/postgres/postgres.git

4.4.2　データセットの再作成

4.4.1 にて欠陥検出精度を改良したため，それに伴い，表 2 のデータセットを新たに作成しなおした．これにより，データセットの欠陥の有無の部分の値が更新された．

5.　ツールの利用シナリオ

ツールの利用者としては，欠陥予測を行いたい開発者だけでなく，リポジトリから得られるデータセットを分析したい研究者も考えられる．以下では，ツールの利用シナリオについて述べる．

5.1　欠陥予測を行いたい場合

品質管理を行うマネージャが，Git で管理される開発プロジェクトに本ツールを導入し，開発者が本ツールを用いる場合の利用シナリオを考える．

5.1.1　新規のプロジェクトに対して欠陥予測を行う場合

新規開発プロジェクトの場合，欠陥予測を行うための学習データである開発履歴がない，あるいは少ない．開発履歴が少ない場合，十分な性能の予測モデルを構築することができない．そのため，既存のオープンソースプロジェクトの開発履歴をデータセットとして用いることでこの問題を解決する．

（1）　導入

マネージャがインストールの一環として新規プロジェクトの Git リポジトリをツールに登録する．

（a）　初期化

マネージャは初期化コマンド (init) を，プロジェクトの Git リポジトリへのパスを引数に実行する．

（b）　パスの設定 (ツールによる自動処理)

ツールはプロジェクトへのパスを登録する．

（c）　フックの作成 (ツールによる自動処理)

ツールはプロジェクトの Git リポジトリ内にフック (3.4) を作成する．

この時点でツール内に保存している OSS のデータセットを用いた欠陥予測を行えるようになる．予測に用いるデータセットとして初期設定では，Bugzilla が採用される．この操作は最初の 1 回だけ行う．

（2）　コミット

開発者は，自身が行った変更を Git リポジトリにコミットする．

（3）　JIT 欠陥予測

ツールは，コミットに対して JIT 欠陥予測を行う．

（a）　コミットに対するメトリクス計測 (ツールによる自動処理)

コミットをきっかけに手順 (1c) で作成された Git のフックが呼び出され，ツールは今回のコミット内容からメトリクスを計測する．

（b）　JIT 欠陥予測の実行 (ツールによる自動処理)

ツールは，手順 (3a) で計測したメトリクスをあらかじめ保存してある OSS のデータセット (表 2) から構築した予測モデルに入力し，欠陥予測を実行する．

予測に用いるモデルとして初期設定ではランダムフォレストとロジスティック回帰分析の 2 つが採用される．このモデル構築手法及び OSS のデータセットはいつでも変更することができる (5.3，5.4)．

（c）　欠陥予測結果の出力 (ツールによる自動処理)

欠陥予測の結果として，コミットから計測したメトリクスを基に算出した欠陥混入の確率を画面に出力する．

（4）　コミット完了または中止の選択

開発者はその結果を確認した後でコミットを中止するかどうかの選択を行う．

5.1.2　既存のプロジェクトに対して欠陥予測を行う場合

開発履歴が十分にあるような既存プロジェクトはそれを基にデータセットを作成し，欠陥予測を行うことができる．

（1）　導入

マネージャがインストールの一環として既存プロジェクトの Git リポジトリをツールに登録する．

（a）　初期化

マネージャは初期化コマンド (init) をプロジェクトの Git リポジトリへのパスを引数に実行する．

（b）　パスの設定 (ツールによる自動処理)

ツールはプロジェクトへのパスを登録する．

（c）　フックの作成 (ツールによる自動処理)

ツールはプロジェクトの Git リポジトリ内にフックを作成する．

（d）　データセットの作成

ツールは 5.2 の手順でプロジェクトの開発履歴からデータセットを作成し，予測モデルを構築する．

この時点で既存プロジェクトのデータセットを用いた欠陥予測を行えるようになる．この操作は最初の 1 回だけ行う．

以降の手順は 5.1.1 に従う．

5.2　データセットを作成したい場合

プロジェクトのリポジトリから作成したデータセットは予測モデルの構築だけでなく，開発履歴の分析に用いることもできると考えられる．ここでのツールの利用者は，開発者や研究者を考える．

（1）　リポジトリの用意

利用者はデータセットを作成したいプロジェクトの Git リポ

ジトリを用意する．

（2） コマンドの実行

利用者はデータセット作成コマンド (create_dateset) を手順 (1) の Git リポジトリへのパスを引数に実行する．

（3） データセットの作成 (ツールによる自動処理)

ツールは Git リポジトリのコミットのログから自動的にメトリクス，欠陥を計測し，データセットを作成する．

（4） 予測モデルの構築 (ツールによる自動処理)

ツールは手順 (3) で作成したデータセットを基に予測モデルを構築する．

5.3 他のモデルで欠陥予測をしたい場合

5.1 で欠陥予測を行う際に，複数のモデルを比較したり，予測精度の良いモデルに変更したりする場合が考えられる．

（1） コマンドの実行

開発者は予測モデル変更コマンド (set –model) を表 1 のモデル構築手法名を引数に実行することで，欠陥予測に用いるモデルを変更することができる．欠陥予測は複数のモデルを同時に用いて行うこともできる．これにより，予測を行う際に用いるモデルが変更される．

5.4 他のデータセットで欠陥予測をしたい場合

新規プロジェクトの開発が進み，開発履歴が十分に蓄積された時に，他のプロジェクトのデータセットではなく，自身のプロジェクトのデータセットを用いた欠陥予測を行いたい場合など，予測に用いるデータセットを現在使用しているものから他のものへ変更したい場合が考えられる．

（1） コマンドの実行

開発者はデータセット変更コマンド (set –dataset) を表 2 や 5.2 で作成したプロジェクト名を引数に実行することで，欠陥予測に用いるデータセットを変更することができる．欠陥予測は複数のデータセットを用いて行うこともできる．

これにより，5.1.1 で予測を行う際に用いるデータセットが変更される．

6. おわりに

本稿では，JIT 欠陥予測を開発プロジェクトで適用する際の課題 (予測モデル構築のための開発履歴のデータ量不足やメトリクス計測に掛かる手間など) を解決するために，JIT 欠陥予測を開発プロジェクトに適用するためのツールの設計，及び，実装を行った．また，利用者からのフィードバックを基にメトリクス計測や欠陥予測の高速化，利用シナリオの整理，及び，欠陥検出精度を向上させるための実装などを行った．

今後の課題としては，ツールをオープンソースとして公開することで，多くの人々が利用できるような形にする．そして，更なるフィードバックをもらうことでツールの改善を行う予定である．ツールを公開するための準備として，現在は GitHub(注4) にプライベートリポジトリを用意した．

謝辞 本研究の一部は，日本学術振興会科学研究費補助金 (若手 A：課題番号 24680003) による助成を受けた．

(注4)：https://github.com/

文　献

[1] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.

[2] David Hovemeyer and William Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106, December 2004.

[3] Yasutaka Kamei, Emad Shihab, Bram Adams, A Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans. Softw. Eng.*, 39(6):757–773, 2013.

[4] Sunghun Kim, Thomas Zimmermann, Kai Pan, and E James Whitehead. Automatic identification of bug-introducing changes. *Proc. Int'l Conf. on Automated Softw. Eng. (ASE'06)*, pages 81–90, 2006.

[5] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Trans. Softw. Eng.*, 34(4):485–496, 2008.

[6] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E James Whitehead. Does bug prediction support human developers? findings from a google case study. In *Int'l Conf. on Softw. Eng. (ICSE'13)*, pages 372–381. IEEE, 2013.

[7] Paul Luo Li, James Herbsleb, Mary Shaw, and Brian Robinson. Experiences and results from initiating field defect prediction and product test prioritization efforts at abb inc. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'06)*, pages 413–422. ACM, 2006.

[8] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, and Ken-ichi Matsumoto. An analysis of developer metrics for fault prediction. In *Proc. Int'l Conf. on Predictive Models in Softw. Eng. (PROMISE'10)*, pages 18:1–18:9, 2010.

[9] Osamu Mizuno, Shiro Ikami, Shuya Nakaichi, and Tohru Kikuno. Fault-prone filtering: Detection of fault-prone modules using spam filtering technique. In *Proc. Int'l Symposium on Empirical Softw. Eng. and Measurement (ESEM'07)*, pages 374–383. IEEE, 2007.

[10] Audris Mockus and David M Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[11] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'08)*, pages 181–190. IEEE, 2008.

[12] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Programmer-based fault prediction. In *Proc. Int'l Conf. on Predictor Models in Softw. Eng. (PROMISE'10)*, pages 19:1–19:10, 2010.

[13] Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'13)*, pages 432–441. IEEE Press, 2013.

[14] Emad Shihab, Ahmed E Hassan, Bram Adams, and Zhen Ming Jiang. An industrial study on the risk of software changes. In *Proc. SIGSOFT Int'l Symposium on the Foundations of Softw. Eng. (FSE'12)*, pages 62:1–62:11. ACM, 2012.

[15] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proc. SIGSOFT Softw. Eng. notes*, volume 30, pages 1–5. ACM, 2005.

[16] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. *IEEE Trans. Softw. Eng.*, 31(6):429–445, 2005.

[17] 田中 秀太郎, 山下 一寛, 亀井 靖高, and 鵜林 尚靖. 変更レベルに着目したバグ予測支援ツールの設計と実装. In **電子情報通信学会 技術研究報告**, volume 113, pages 113–118, 2014.