Vol.2015-SE-190 No.18
2015/12/16

ユーザ障害情報によるソースコード欠陥箇所予測ツール

小須田 光[1,a)]　亀井 靖高[1,b)]　鵜林 尚靖[1,c)]

概要：ソフトウェアの不具合を解消するため，クラッシュレポートの収集を行う開発プロジェクトが増加している．クラッシュレポートは，ソフトウェアのクラッシュ時に自動作成されるレポートで，クラッシュ以前の実行環境に関する情報をまとめたものであり，これらの情報は不具合修正に利用されている．我々は，既存研究の知見を用いて，ユーザからのクラッシュレポートを収集しているソフトウェア開発プロジェクトに対して，不具合の原因となるソースコード内の欠陥箇所を予測するツールを開発した．本ツールは，ツールとしての汎用性のため，対象となるソフトウェアの言語，構成等に依存せず，クラッシュレポートの情報のみから欠陥箇所を予測する手法をとった．そのため，予測精度は既存研究の結果に及ばないが，より多様なプロジェクトへ適用することが可能である．本稿では，開発したツールの予測手法，予測精度の評価について報告する．

キーワード：欠陥箇所予測，　クラッシュレポート

1. はじめに

ユーザの環境でソフトウェア欠陥による不具合が発生した場合，ユーザに少なくはない時間的／金銭的損害を与えるだけではなく，開発プロジェクトの評判も落とすことになる．そのため，全ての欠陥がリリース前の段階で発見され取り除かれていることが，ユーザと開発者双方にとって望ましい．しかしながら，限られた開発工数と定められた納期の中では，開発者が全ての不具合を事前に発見することは難しい．そこで，一部の開発プロジェクトでは，クラッシュレポートシステムを導入することで，不具合の発生状況を迅速に認識し，不具合を修正する仕組みを構築している [8]．クラッシュレポートシステムは，ユーザ環境下においてソフトウェアが予期せぬ停止をした際にスタックトレースやユーザ環境等，不具合の再現や修正に有用な情報をレポートにまとめ，開発プロジェクトへ送信するシステムである．

大規模なプロジェクトでは，毎日大量のクラッシュレポートが送信されてくるため，全てのレポートを利用するのは困難である．そこで，クラッシュレポートの情報を効率的に欠陥修正に利用するための手法がこれまでに多数提案されてきた [2][3][7][9]．提案，及び，評価されてきた手法では，レポートの送信件数や，レポートを送信したユニークなユーザ数が大きい種類のクラッシュレポートの情報を重要視すべきであるとされている．また，クラッシュレポートに記載されている，スタックトレースの内容を利用することで，ソフトウェアのソースコード内の欠陥箇所を予測する手法も示されており，これらを利用すれば，不具合修正の効率化が期待できる．

しかしながら，これらの手法の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，開発現場で使用するための環境も整備されているとは言い難い．開発プロジェクトでクラッシュレポートを用いた欠陥箇所の予測を実施するには，以下に示す課題がある．

課題 1. クラッシュの原因となった欠陥はレポートの情報から手動で探す必要があり，大量のクラッシュレポートが送信される大規模プロジェクトでは欠陥箇所の発見と修正が間に合わないことがある．

課題 2. クラッシュレポートの種類まではシステムで分類されるが，スタックトレースの詳細については個別のレポートを確認せねばならず，多くのレポートのスタックトレースから欠陥を探すのに時間と手間がかかる．

課題 3. ある種類のクラッシュレポートのスタックトレースで頻出しているメソッドは欠陥の可能性が高いとされているが，他の多くの種類のクラッシュレポートでも同様である場合，単純にソフトウェアの挙動として常に使われるメソッドである可能性が高く，欠陥である可能性が高いか

[1] 九州大学，福岡市
Kyushu University, Japan
[a)] kosuda@posl.ait.kyushu-u.ac.jp
[b)] kamei@ait.kyushu-u.ac.jp
[c)] ubayashi@ait.kyushu-u.ac.jp

否かの判断が困難である．

しかし，これらの課題は既存研究による知見から解決が可能である．そこで本稿では，既存研究による知見を利用し，クラッシュレポートの情報を用いて行う欠陥箇所予測を，ソフトウェア開発に適用するためのツールを開発した．本ツールは，より多様な開発プロジェクトに適用するため，ソフトウェアの言語，構成等に依存せず，クラッシュレポートのスタックトレースの情報のみを用いてソースコードの欠陥箇所を予測する．スタックトレースはユーザ環境でのメソッドの呼び出し順序を記録したものである．クラッシュレポート群から，このスタックトレースを解析することで欠陥を含む可能性の高い箇所をファイル単位で予測する．

以降，2 章では，クラッシュレポートシステムの詳細と，開発者による現行の利用方法について紹介し，3 章では，ソフトウェア開発支援やクラッシュレポートに関する関連研究について述べる．4 章では，本ツールの方針と特徴，構成について述べる．5 章では，ツールの実装方法と，本ツールの有用性を議論する．そして，6 章で本稿のまとめと今後の課題について述べる．

2. クラッシュレポートシステムの仕組み

2.1 クラッシュレポートシステム

ユーザからのフィードバックを得るために，一部のソフトウェアシステムにはクラッシュレポートシステムが組み込まれている．ユーザがソフトウェアを利用している最中に，予期せぬ停止（クラッシュ）が発生すると，クラッシュレポートシステムが実行環境に関する情報をクラッシュレポートにまとめ，開発プロジェクトのクラッシュサーバへ送る（図 1（a））．

クラッシュレポートには，メソッドの呼び出しの順序を記録したスタックトレースが含まれている．各スタックフレームに，順番，モジュール，メソッドシグニチャ，対応するソースコードへのリンクが記載されている．メソッドシグニチャは，単にメソッドの名前のみを表しているのではなく，メソッドの名前とそのメソッドの引数の組み合わせを表す．

クラッシュレポートを保管するサーバでは，類似するクラッシュレポートを同一種類のレポートとしてまとめる．これは，クラッシュレポートシステムによって集められるクラッシュレポートの数が膨大であり，開発者がどういったクラッシュが頻繁に発生しているかを把握できないためである．同一種類のクラッシュレポートをまとめることは，頻出するクラッシュレポートの種類の把握を容易にし，開発者が優先的に取り組むクラッシュを決定する際に役立つ．Firefox プロジェクトの Socorro の場合，クラッシュレポートはスタックトレースのトップメソッドシグニチャに基づいて自動でグループ分けされる（図 1（b））．しかしながら，同じクラッシュタイプ内でもスタックトレースでのその後に続くフレームについてはクラッシュレポート毎に様々である [7]．

2.2 クラッシュレポートの報告からソースコードの修正までのプロセス

本節では，開発者による，クラッシュレポートの情報を用いたソースコードの修正のプロセスを紹介する．まず，先にも述べたように，ユーザがソフトウェアを利用している最中に，クラッシュが発生すると，クラッシュレポートが開発プロジェクトのサーバへ送られる（図 1（a））．次に，クラッシュサーバは，大量のクラッシュレポートを類似するレポート毎にグループ化する（図 1（b））．

レポータと呼ばれる開発者が，クラッシュレポートを調査しており，もし，クラッシュレポートの原因である不具合が，不具合管理システム（Bugzilla）に登録されていない場合，レポータは，その不具合を不具合管理システムに新たな不具合として登録する（図 1（c））．また，開発者は，クラッシュタイプと不具合票を関連付ける．クラッシュタイプと不具合は，多対多の関係で関連付けられ，一つのクラッシュタイプが複数の不具合と関連付けられる場合や，一つの不具合が複数のクラッシュタイプと関連付けられる場合もある（図 1（d））．開発者は議論を重ね，優先的に修正する不具合を選び，修正対象の不具合を決定後，その不具合に修正担当者を割り当てる [12]．開発者が不具合の修正を行う際は，関連付けられたクラッシュタイプのレポートを参照し，不具合の発生環境等の情報を得ることができる [5]．開発者は不具合を修正するためのソースコード（修正パッチ）を作成すると，不具合管理システムへ提出する（図 1（e））．パッチの内容に誤りが無い場合，パッチは開発プロジェクトのソースコードへ統合される．

3. 関連研究

3.1 欠陥箇所予測の支援ツール

ソースコードの欠陥箇所予測の研究が行われており [1][6]，それを支援するツールの開発も行われている [14][4][13]．Ferdian ら [4] は，Bugzilla を利用しているプロジェクトを支援するツールを開発した．Bugzilla では，開発者がプロジェクトの不具合の報告や修正履歴を 不具合票に記録することができる．Ferdian らのツールは，不具合票に出現する単語とソースコードのファイルとの類似度から，不具合の原因となった欠陥のある確率の高いファイルを開発者に報告する．

本稿では，既存の不具合報告の情報がない状態でも，ユーザによるクラッシュレポートの情報から，ソースコードの欠陥箇所を予測することのできるツールを開発する．

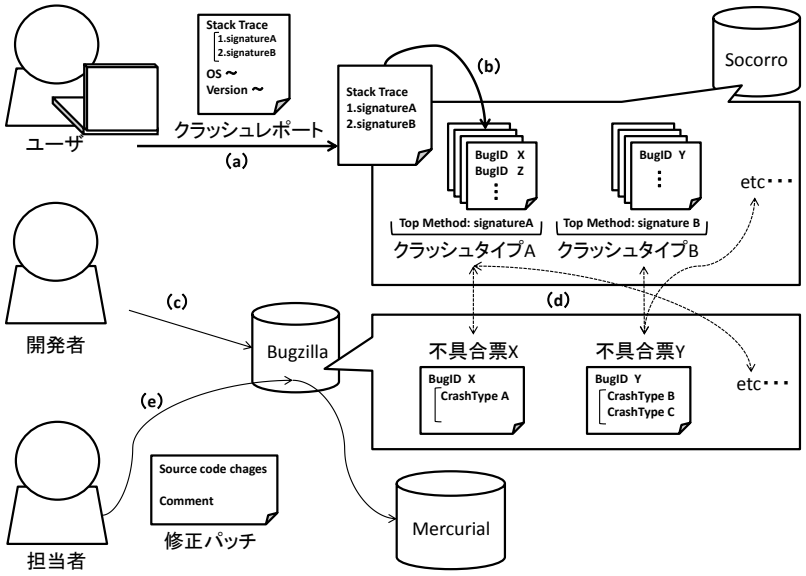

図 1　クラッシュレポートシステムの流れ [10]

3.2　優先的に扱うべきクラッシュレポート

大規模なプロジェクトでは，ソフトウェアのユーザは世界中に存在し，送信されるクラッシュレポートも膨大な量になる．レポートの収集，分類までは自動で行われるものの，認識している不具合とクラッシュレポートの関連付けは手動で行っているため，全てのレポート情報を利用できないので，優先順位を設けて利用していく必要がある．

Kim ら [9] は，クラッシュレポート全体のうち大半を占めている少数のクラッシュタイプのレポート (トップクラッシュ) が存在することに着目し，トップクラッシュをリリース後の短期間で送られてくるレポートから判定し対処する手法を提案した．また，Khomh ら [7] は，修正すべき不具合の優先度判定の改善のため，クラッシュレポートを送信したユニークなユーザ数を調査した．これにより，より多くのユーザに影響を与えているために優先的に修正される不具合を，クラッシュレポートの情報から判別できることを示した．

本稿で紹介するツールにおいても，同じクラッシュタイプのレポートの件数による重みづけを行っている．また一方で，3.3 節から，同一の欠陥可能性箇所を含む，異なるクラッシュタイプの種類数にも重みをつけている．

3.3　クラッシュレポートを用いた欠陥予測

Rongxin ら [11] は，クラッシュレポートのスタックトレースから，欠陥箇所を関数単位で予測する手法を提案した．この手法の評価の中で，特定のクラッシュタイプのレポートに頻出し，他のクラッシュタイプのレポートでは出現しない関数は欠陥を含む可能性が高いとしている．

本ツールの欠陥箇所予測はこの手法を参考に行っている．ただし，Rongxin らがソースコードを解析し，利用することで関数単位で欠陥箇所を予測したのに対し，本ツールではソースコードへの依存を避けるため，クラッシュレポートの情報のみから予測を行っているため，ファイル単位での予測になる．さらに，同手法の評価実験に用いられたデータセットに比べ，大きな件数のクラッシュレポートを用いているため，5 章で行う予測精度評価では，より大規模なプロジェクトの開発に適用することを想定した実験が可能となる．

4.　ユーザ障害情報による欠陥箇所予測支援ツール

4.1　方針と特徴

本ツールは，ソフトウェア開発プロジェクトにおいて，クラッシュレポートの情報を欠陥箇所の修正に利用するまでに時間や手間がかかるという問題を解決することで，開発プロジェクトにおけるクラッシュレポート利用を支援することを目的としている．Mozilla のクラッシュレポートシステム，及び，版管理システムを利用しているプロジェクトを対象に，クラッシュレポートの情報からソースコードの欠陥を含む可能性が高いファイルを予測し，結果を返す．

本ツールの特徴を以下に示す．特徴 1，2，3 はそれぞれ前述（1 章）の課題 1，2，3 と対応している．

特徴 1. 手動で欠陥箇所を探すのが困難な量のクラッシュレポート群からでも，クラッシュレポートの内容を自動で解析することで，欠陥を含む可能性が高い箇所を迅速に示すことができる．これにより，開発者は従来では発見，修正できなかった欠陥にも対応することができる．

特徴 2. 各クラッシュタイプ毎に，原因となる欠陥を含む可能性が高いと予測した箇所を示す．情報量が多く，ク

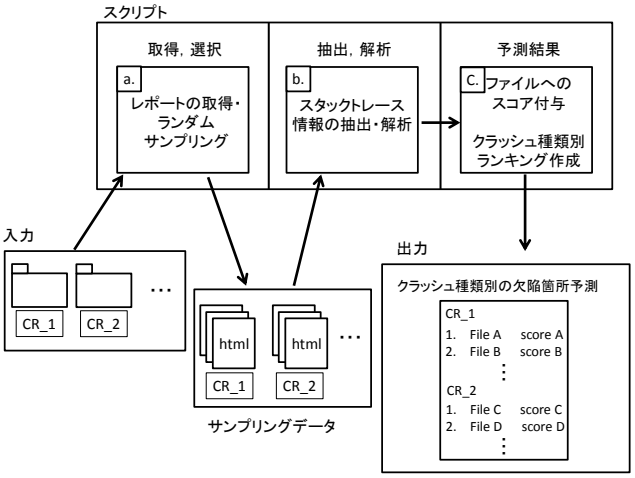

図 2　ツールの構成

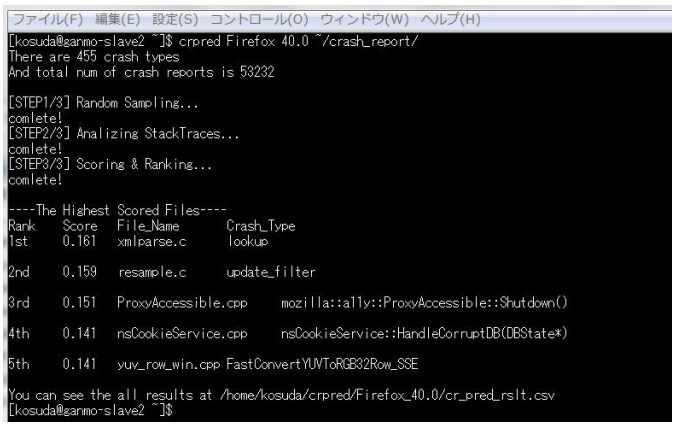

図 3　実行結果画面

ラッシュタイプの分類にも用いられるスタックトレースの情報を用いて欠陥予測を行っており，個別のスタックトレースを確認する大きな手間を省略できる．

特徴 3. 複数のクラッシュタイプのレポートのスタックトレースで頻出する箇所については，該当箇所が出現したクラッシュタイプの種類数に応じて欠陥可能性が低いと予測する．これにより，開発者は常時頻出する箇所ではなく，各クラッシュタイプ固有の頻出箇所に注目できる．

4.2 本ツールのアーキテクチャ

ツールのアーキテクチャの概要を図に示す．主にスクリプト a，b，c で欠陥箇所予測に関する処理を行う．

a．では，入力として受け取ったクラッシュレポートの取得，分類を行った後，予測のために用いるレポートを選択するため，クラッシュタイプ別にランダムサンプリングを行う．各クラッシュレポートは Mozilla クラッシュレポートシステムのレポート形式を想定している．ユーザにより指定されたプロダクトのクラッシュレポートであるか，どのクラッシュタイプのレポートであるかを自動で判断し，サンプリングまでを行う．

b．では，a．で取得，選択したクラッシュレポートのスタックトレースに対し，解析を行う．各クラッシュタイプのレポートの，メソッドの呼び出し順序，当該メソッドが存在するファイルについて解析，記録する．

c．では，b．で解析したスタックトレースの情報から，各クラッシュタイプのレポートに対し，呼び出されたメソッドが存在するファイルについて，欠陥を含む可能性の高さを表すスコアを付与する．そのスコアが高い順にランキングを作成し，出力する．

c．で作成されたランキングは csv ファイルに出力されるが，それとは別にスコアが高い順に 5 つのファイルをクラッシュタイプとともにコマンドライン上に出力する（図 3）．

5. 実装と評価

5.1 概要

Mozilla クラッシュレポートシステム，を実装対象のクラッシュレポートシステムとし，スタックトレースの解析機能，ソースコードのメトリクス計測機能，欠陥箇所予測機能を実装した．Mozilla クラッシュレポートシステムを対象とした理由は，本稿を執筆している現在公開されているクラッシュレポートシステムの中で最も大規模であり，クラッシュレポートの情報としてスタックトレースを含んでいるためである．

本ツールにおける，クラッシュレポートのスタックトレースの解析をするプログラムや，Mercurial リポジトリから出力されるログファイル等を解析しメトリクスを計測するプログラム，及び，欠陥箇所予測を行うプログラムは，Ruby によって実装した．また，本ツールの評価のため，予測結果の解答となる過去の欠陥箇所をリポジトリマイニングにより取得した．その際，Mozilla プロジェクトで多く用いられている Mercurial を利用した．ただし，この解析は Mercurial リポジトリに強く依存するものではなく，他の版管理システムにも応用が可能である．

5.2 実装

以下に 4.1 で示した本ツールの 3 つの特徴をどのように実現しているかを示す．本ツールでは，大量のクラッシュレポートを用いた予測を効率的に行うため，100 件を超える同種のクラッシュレポートはランダムに 100 件を選択し利用し，100 件を超えないクラッシュレポートは全てを利用した [3]．これは，同じくクラッシュレポートのスタックトレースの情報を用いて欠陥箇所の予測を行った，Rongxin ら [11] も採用している手法である．以下に示したスコアを付与する際の重みづけに関しても，Rongxin らの手法を参考にしている．

(1) 本ツールでは，入力されたクラッシュレポートのスタックトレースを自動で解析し，呼び出されたメソッ

ドが存在するファイルを抽出する．そして，(2)，(3) に示す手法により，各クラッシュタイプのクラッシュの原因である可能性に応じて各ファイルにスコアを付け，その順位により予測を行う．

(2) 各クラッシュタイプ毎に，欠陥を含む可能性が高いファイルのランキングが出力される．これにより，全てのクラッシュタイプに関して，欠陥発見の手間と時間が大幅に短縮され，対応可能なクラッシュタイプ数が増加することが期待できる．

(3) スコアの計測に際して，3 つの要素を設定した．以下，スタックトレースにあるファイルに含まれるメソッドが呼び出されることを，ファイルが出現する，と表現する．1 つ目は $FF(FileFrequency)$ である．これは，あるクラッシュタイプ（B）のレポートのスタックトレースで，ファイル（f）が出現する頻度の高さを表す．

$$FF(f,B) = \frac{N_{f,B}}{N_B} \quad (1)$$

$N_{f,B}$ は，あるクラッシュタイプ（B）のレポートのスタックトレースのうち，ファイル（f）が出現したものの数であり，N_B は，ファイル（f）が出現しなかったものの数である．2 つ目は $IBF(InverseBucketFrequency)$ である．これは，ファイル（f）の異なるクラッシュタイプへの出現範囲の広さを表す．

$$IBF(f) = \log(\frac{\#B}{\#B_f} + 1) \quad (2)$$

$\#B$ は全クラッシュタイプの種類数，$\#B_f$ はファイル（f）が出現したクラッシュタイプの種類数である．3 つ目は $IAD(InverseAverageDistancetoCrashPoint)$ である．

$$IAD(f,B) = \frac{N_{f,B}}{1 + \sum_{i=1}^{n} dis_i(f)} \quad (3)$$

n はあるクラッシュタイプ（B）のレポートのうち，スタックトレースにファイル（f）を含むもの，$dis_i(f)$ は，ファイル（f）が呼び出されたスタックの，上から数えた番号である．

これらの要素からスコア（$Score(f)$）を算出する．スコアの算出式を以下に示す．

$$Score(f) = FF(f,B) * IBF(f) * IAD(f,B) \quad (4)$$

すなわち，あるクラッシュタイプでは頻出するが，他のクラッシュでは出現せず，スタックトレースの上部（クラッシュの直前）で出現しているファイルには高いスコアが付与される．

5.3 欠陥箇所予測の評価

開発において，実際に本ツールを用いて欠陥箇所予測を行うことを想定し，実在するソフトウェアに対し，予測を行った．対象としたソフトウェアとクラッシュレポート数を表 1 に示す．クラッシュレポートは Socorro サーバに保管されている Mozilla クラッシュレポートシステムのレポートを対象とした．実際の開発期間に対応するため，各ソフトウェアの各バージョンに対し，リリースからサポート終了までの期間に送信されたクラッシュレポートを収集した．また，欠陥箇所は，対象ソフトウェアに対しリポジトリマイニングを行い，クラッシュレポートに関連付けられた不具合票の不具合番号が記載されたコミットにおける変更ファイルとした．

また，予測結果の精度について表 2 に示す．実施した予測の評価にあたり，以下の 2 つの指標を用いた．予測の評価には Recall と Precision，及び，F-measure が用いられることが多いが，本実験では通常と異なり，予測箇所数と欠陥箇所数が一致しないため，これらの指標を用いることができない．そこで，手法の参考とした Ronxing ら [11] と同様に，以下の指標を用いた．

(1) Recall@N：各クラッシュタイプのレポートに対し，欠陥を含む可能性が高い上位 N 位のファイルの中に，実際に欠陥が含まれている割合を表す指標である．これが大きいほど（最大で 1），欠陥箇所を少なくとも 1 つ的中させているクラッシュタイプの割合が大きい．

(2) MRR：欠陥を含む可能性が高い順位のファイルが実際に欠陥箇所であるかを示す指標である．あるクラッシュレポート（i）に関して，原因となった欠陥箇所のうち，最初のファイルを的中できた際の，ランキング内での該当ファイルの順位を $rank_i$ とし，$rank_i$ の全クラッシュタイプ分の調和級数をとる．その級数を全クラッシュタイプの種類数で除したものである．これが大きいほど（最大で 1），高い順位が付けられているファイルが実際に欠陥箇所である割合が大きい．

$$MRR = \frac{1}{Q}\sum_{i=1}^{n}\frac{1}{rank_i} \quad (5)$$

本ツールの予測精度は，Rongxin ら [11] の手法の予測精度には及ばないが，これは Rongxin らの手法と比較して，本ツールはソースコードの解析結果を利用していないことに起因すると考えられる．しかし，これにより，本ツールはソフトウェアの言語，構成等に依存することなく，開発プロジェクトへの適用が可能である．

Recall@1 の平均が約 30%であることから，本ツールで予測した結果，各クラッシュタイプのに対する予測結果の上位 1 位のファイルを調査することで，約 3 割の確率で欠陥ファイルを発見することができる．また，上位 5 位までのファイルを調査することで，約半数のクラッシュタイプのクラッシュの欠陥ファイルを発見することができる．上位 5 位から 10 位についても表示はするが，Recall@5 と

表 1　予測実験対象

プロダクト	クラッシュレポート数	欠陥数
Firefox 38.0	34,226	1,017
Firefox 39.0	577,695	1,562
Firefox 40.0	53,232	1,000
FennecAndroid 37.0	54,232	783
FennecAndroid 38.0b10	5,416	572

表 2　予測結果

プロダクト	Recall@1	Recall@5	Recall@10	MRR
Firefox 38.0	31.0%	65.5%	65.5%	0.455
Firefox 39.0	31.8%	50.2%	51.0%	0.396
Firefox 40.0	24.6%	47.4%	49.6%	0.340
Fennec Android 37.0	33.2%	45.4%	53.1%	0.411
Fennec Android 38.0b10	30.3%	47.6%	52.7%	0.402
Average	30.2%	51.2%	54.4%	0.401

Recall@10 の平均値にあまり差がないことから，調査するファイルは上位 5 位までのものにするとよいと考えられる．

また，MRR の平均値は約 0.4 である．MRR は，各クラッシュタイプの予測結果のうち，最初に正解の欠陥箇所を予測できるのが上位何位であるかの逆数の平均である．すなわち，本ツールでは，各クラッシュタイプの予測結果上位 2〜3 位までのファイルを調査すると欠陥を発見できる割合が高いと考えられる．

よって，本ツールを実際に開発で使用する場合，通常は上位 2〜3 位までのファイルを調査し，特に重点的に調査したいクラッシュについては，上位 5 位までのファイルを調査すると，効率よく欠陥ファイルを発見できると考えられる．

6. おわりに

本稿では，クラッシュレポートの情報を開発プロジェクトに活用する際の問題点（大量のクラッシュレポートの調査にかかる時間的，人員的コスト等）を解決するために，既存研究の知見を用いた欠陥箇所予測を開発プロジェクトで適用するためのツールの設計，及び，実装を行った．このツールは，多くの人々が自由に利用できるようにするため，GitHub に公開している [*1]．また，実際のツールの使用を想定し，実在するプロジェクト対し欠陥箇所予測と評価を行った．具体的に，本ツールは，クラッシュレポートのスタックトレースに記録されたメソッドの呼び出し履歴から，欠陥を含む可能性の高いファイルを順位付きで提示することができる．

今後の課題としては，予測のために利用できる情報を増やすため，クラッシュレポートのスタックトレース以外の情報を用いることが考えられる．クラッシュレポートには，スタックトレースの他にもユーザの環境等の多くの情報が記録されているため，これらの情報を利用することができれば，予測精度を向上することができると期待される．

[*1] https://github.com/ulksd/DefectPredictor.git

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，挑戦的萌芽：課題番号 25540026）による助成を受けた．

参考文献

[1] Bangcharoensap, P., Ihara, A., Kamei, Y. and Matsumoto, K.: Locating Source Code to Be Fixed Based on Initial Bug Reports - A Case Study on the Eclipse Project -, Proc. Int'l Workshop on Empirical Software Engineering in Practice (IWESEP 2012), pp. 10-15 (2012).

[2] Dang, Y., Wu, R., Zhang, H., Zhang, D. and Nobel, P.: ReBucket: a method for clustering duplicate crash reports based on call stack similarity, Proc. Int'l Conf. on Softw. Eng. (ICSE'12), pp. 1084-1093 (2012).

[3] Dhaliwal, T., Khomh, F. and Zou, Y.: Classifying field crash reports for fixing bugs: A case study of Mozilla Firefox, Proc. Int'l Conf. on Software Maintenance (ICSM'11), pp. 333-342 (2011).

[4] Ferdian, T., Tien-Duy, L., Pavneet, K. and David, L.: BugLocalizer: Integrated Tool Support for Bug Localization (FSE'14), pp. 767-770 (2014).

[5] Iftekhar, Ahmed., Nitin, Mohan., and Carlos, Jensen.: The Impact of Automatic Crash Reports on Bug Triaging and Development in Mozilla, The International Symposium on Open Collaboration (OpenSym'14), pp. 1-8 (2014).

[6] Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K., Adams, B. and Hassan, A. E.: Revisiting Common Bug Prediction Findings Using Effort Aware Models, Proc. Int'l Conf. on Software Maintenance (ICSM'10), pp. 1-10 (2010).

[7] Khomh, F., Chan, B., Zou, Y. and Hassan, A. E.: An Entropy Evaluation Approach for Triaging Field Crashes: A Case Study of Mozilla Firefox, Proc. Working Conf. on Reverse Engineering (WCRE'11), pp. 261-270 (2011).

[8] Khomh, F., Dhaliwal, T., Zou, Y. and Adams, B.: Do faster releases improve software quality? An empirical case study of Mozilla Firefox, Proc. Int'l Conf. on Mining Software Repositories (MSR'2012), pp. 179-188 (2012).

[9] Kim, D., Wang, X., Kim, S., Zeller, A., Cheung, S. C. and Park, S.: Which Crashes Should I Fix First?: Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts, IEEE Trans. Softw. Eng., Vol. 37, No. 3, pp. 430-447 (2011).

[10] 長本貴光，亀井靖高，伊原彰紀，鵜林尚靖：クラッシュログを用いたソースコード不具合箇所の特定に向けた分析，情報処理学会研究報告, ソフトウェア工学研究会，pp. 1-6 (2013).

[11] Rongxin, W., Hongyu, Z., Shing-Chi, C. and Sunghun, K.: CrashLocator: Locating Crashing Faults Based on Crash Stacks (ISSTA2014), pp. 204-214 (2014).

[12] Śliwerski, J., Zimmermann, T. and Zeller, A.: When Do Changes Induce Fixes?, Proc. Int'l Conf. on Mining Software Repositories (MSR'05), pp. 1-5 (2005).

[13] Servant, F. and Jones, J. A.: WhoseFault: automatic developer-to-fault assignment through fault localization, Proc. Int'l Conf. on Softw. Eng. (ICSE'12), pp. 36-46 (2012).

[14] Zhou, J., Zhang, H. and Lo, D.: Where should the bugs be fixed? - more accurate information retrieval-based

bug localization based on bug reports, Proc. Int'l Conf. on Softw. Eng. (ICSE'12), pp. 14-24 (2012).