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.

欠陥の特徴を用いた自動欠陥修正の評価

横山　晴樹†　肥後　芳樹†　楠本　真二†

† 大阪大学大学院情報科学研究科，吹田市
E-mail: †{y-haruki,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　デバッグは多大な労力を要する作業であり，作業コストの削減が課題である．自動欠陥修正は欠陥を含むプログラムとテスト集合から，修正したプログラムを出力するデバッグ支援手法であり，近年盛んに研究されている．既存の自動欠陥修正に対する評価方法は，修正できた欠陥の個数や，修正時間，修正品質などの評価が主であった．一方，修正結果は欠陥が持つ特徴によって様々であるにもかかわらず，どのような特徴を持つ欠陥を修正できたのかという観点での評価はあまり行われていない．本研究では，欠陥の特徴を欠陥レポートから抽出し，オープンソースの Java プロジェクトにおける 138 個の欠陥に対する自動欠陥修正ツールの修正性能を調査した．調査では，jGenProg と Nopol が高い優先度を持つ欠陥を高い割合で修正できることを明らかにした．また，修正時間や修正行数などの観点から，自動欠陥修正の修正性能を多角的に評価した．

キーワード　デバッグ，自動欠陥修正，欠陥追跡システム，ソフトウェアリポジトリマイニング

1. はじめに

ソフトウェア開発において，デバッグは開発者が多大な労力を要する作業である．デバッグにかかる費用は世界中に年間推定 3000 億ドルにも及び，開発者はコーディング時間の約半分をデバッグに費やすと言われている [1]．そのため，デバッグ費用の削減が課題である．近年，デバッグの支援に向けて自動欠陥修正に関する多数の手法が提案されており，ツールの公開も行われている [2]〜[6]．

自動欠陥修正は，欠陥を含むプログラムとテスト集合を入力として与え，修正に成功すると，与えたテストを全て通過するプログラムを出力する．自動欠陥修正の修正性能は，主に欠陥を含むプログラム集合に対する修正個数によって評価される．また，修正時間 [2]，人間にとって読みやすいか [7]，シンプルなパッチになっているか [8] 等の評価を行うこともある．一方，修正結果は欠陥が持つ特徴によって様々であるにもかかわらず，どのような特徴を持つ欠陥を修正できたのかという観点での評価はあまり行われていない．

そこで，本研究では欠陥の特徴を用いて，自動欠陥修正ツールの評価を行う．欠陥の特徴としては，欠陥追跡システム中の欠陥レポートから得られる以下の 2 つを用いる．

- 優先度
- 再オープンの有無

優先度の高い欠陥は開発者が早急な修正を必要とするものである．再オープンされた欠陥は開発者が修正に失敗した欠陥である．そのため，自動欠陥修正がこれらの欠陥を修正できれば，開発者に対して大きな貢献となる．

我々は，Java プロジェクトのデータセット Defects4J [9] に含まれる 138 個の欠陥と，各欠陥に対する自動欠陥修正ツール jGenProg [5], jKali [5], Nopol [6] の修正記録，各欠陥における優先度と再オープンの有無を調査し，欠陥の特徴と自動欠陥修正の修正性能との関係を分析した．

本研究における主要な貢献は次のとおりである．

- jGenProg と Nopol が優先度の高い欠陥を多く修正することを明らかにした．
- Nopol が再オープンされた欠陥をあまり修正できないことを明らかにした．
- 修正時間や修正行数といった観点から，自動欠陥修正の修正性能を多角的に評価した．

本論文では，2 章で研究背景として欠陥追跡システムや自動欠陥修正を紹介し，研究目的，リサーチクエスチョンを述べる．3 章では調査手法と調査対象を示す．4 章で調査結果を述べ，リサーチクエスチョンに回答する．5 章では調査結果に関して考察を行い，追加で行った調査について述べる．6 章では調査における内的，外的妥当性への脅威について述べる．最後に，7 章では本研究のまとめと今後の研究課題について述べる．

2. 研究背景

2.1 欠陥追跡システム

欠陥追跡システム（BTS）とは，ソフトウェア開発中に発生した欠陥を，開発者によって修正されるまで追跡するためのシステムである．BTS は発生した欠陥の一元管理や，修正作業の割り当て，欠陥修正における議論の記録等を可能にする．欠陥は欠陥レポートという形式で BTS に登録される．欠陥レポートには以下のような記録が含まれる．

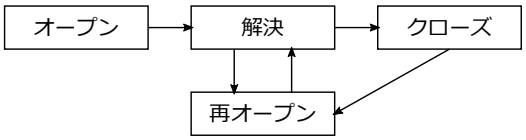

図 1　JIRA における欠陥レポートの簡略化したライフサイクル

- 欠陥の名称
- 欠陥を説明する記述
- 欠陥の報告者名
- 欠陥修正に割り当てられた開発者名
- 欠陥のライフサイクル
- 欠陥の優先度

欠陥レポートが高い優先度を持つことは，開発者が早急に欠陥の修正を行う必要があることを表す．また，欠陥レポートはライフサイクルを持ち，欠陥に対するあらゆるイベントが記録される．例えば，イベントには次のものがある．

- 欠陥レポートのステータスの変更
- 欠陥修正への開発者の割り当て
- 欠陥修正に伴うコミットの記録

図 1 に，BTS の 1 つである JIRA(注1) における，欠陥レポートの簡略化したライフサイクルを示す．各セルは欠陥レポートのステータスを表し，有向辺はステータスの変更を表す．通常，欠陥レポートは欠陥の報告者および開発者によって作成され（オープン），「修正」，「重複」，「修正しない」といった方法で解決され，解決方法が問題なければ閉じられる（クローズ）．しかし，解決方法に問題があったり，閉じられた後に再度問題が発覚したりすると，欠陥レポートは再度開かれる（再オープン）．欠陥が再オープンされると，開発者の作業量増加や，ソフトウェア利用者の信用の低下に繋がる [10]．また，再オープンされた欠陥レポートは，再オープンのない欠陥レポートに比べ，閉じられるまでの過程が長期化するという報告もされている [11]．

2.2　自動欠陥修正

近年，デバッグ支援の研究として，デバッグ自動化を目指した自動欠陥修正が多数提案されている [2]〜[4]．自動欠陥修正とは，欠陥を含むプログラムを入力とし，ソースコードに変更を加え，欠陥が修正されたプログラムを出力する手法の総称である．

これまでに，自動欠陥修正における評価には欠陥の修正個数，修正時間，修正品質といった評価指標が用いられてきた．Martinez らの調査 [12] は，欠陥を含んだ Java プロジェクトのデータセット Defects4J [9] に対し，自動欠陥修正ツール jGenProg [5], jKali [5], Nopol [6] を実行させて，各ツールの修正個数や修正時間を示した．例えば，Defects4J 内の Math(注2) プロジェクトに対する修正個数は表 1 のようになる．修正個数の観点で見れば，Nopol は良い修正性能を持っている．

表 1　Math プロジェクトに対する修正個数

自動欠陥修正ツール	jGenProg	jKali	Nopol
修正個数	18	14	21

(注1)：https://ja.atlassian.com/software/jira

2.3　研 究 目 的

欠陥の中には優先して修正すべき欠陥や，開発者による修正が不完全で，繰り返し修正作業が行われた欠陥などがある．Math プロジェクトは欠陥追跡システム JIRA で管理されており，欠陥毎に優先度(注3) や再オープンの有無(注4) を調査することができる．そこで，予備調査として，jGenProg, jKali, Nopol が修正できた Math プロジェクトの各欠陥に対し，優先度および再オープンの有無の対応付けを行った．

表 2 は，Math プロジェクトに対する修正個数を優先度および再オープンの有無の観点で分離したものである．優先度に注目すると，jGenProg や jKali は最も高い優先度 Blocker を持つ欠陥の修正に成功している．また，再オープンに注目すると，jGenProg や jKali は再オープンされた欠陥の修正に成功している．このように，優先度や再オープンの有無といった欠陥の特徴を用いることで，既存の修正個数などの観点とは異なる視点で自動欠陥修正を評価できる可能性がある．しかし，上記の予備調査は少数のデータに対する分析であり，妥当な評価を行うためには，多数のデータを用いた調査が必要である．

本研究の目的は，既存の自動欠陥修正を欠陥の特徴を用いて評価することである．欠陥の特徴を得る手段として，BTS に登録される欠陥レポートを利用する．本研究では特に，以下の 2 つを欠陥の特徴として用いる．

- 優先度
- 再オープンの有無

自動欠陥修正が早急な修正を必要とする優先度の高い欠陥を修正できるならば，開発者への貢献が大きくなる．また，自動欠陥修正が再オープンされた欠陥を修正できるならば，開発者がデバッグに費やす時間を削減することができる．

2.4　リサーチクエスチョン

本研究では，以下の 2 つのリサーチクエスチョンに回答することで，欠陥の特徴を用いた自動欠陥修正の評価を行う．

RQ-1　自動欠陥修正は優先度の高い欠陥を修正できるか．

表 2　優先度別および再オープンの有無別の修正個数

自動欠陥修正ツール	jGenProg	jKali	Nopol
Blocker	1	1	0
Critical	2	0	2
Major	11	11	15
Minor	4	2	4
再オープンあり	1	1	0
再オープンなし	17	13	21

(注2)：http://commons.apache.org/proper/commons-math

(注3)：優先度の高さは Blocker>Critical>Major>Minor>Trivial である．なお，Defects4J 内の Math プロジェクトには優先度 Trivial を持つ欠陥は含まれていない．

(注4)：欠陥レポートのステータスが一度でも再オープンとなれば，「再オープンあり」となる．

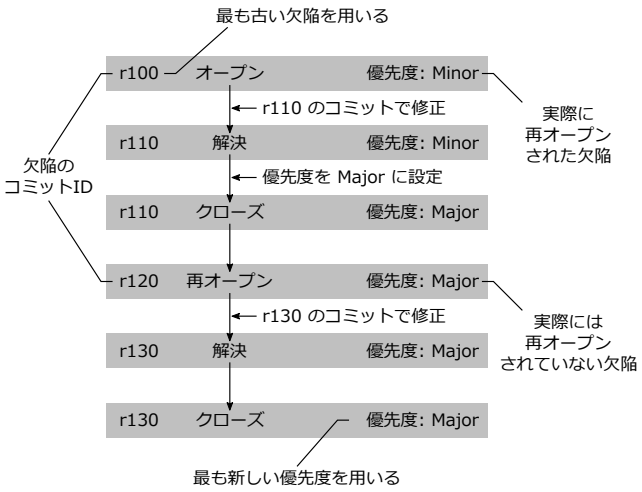

図 3　1 つの欠陥レポートに対して複数の欠陥が対応する例

RQ-2　自動欠陥修正は再オープンされた欠陥を修正できるか．

RQ-1 では，自動欠陥修正が優先度の高い欠陥を優先度の低い欠陥に比べ高い割合で修正できるかを調査する．ここで，優先度の高い欠陥とは，JIRA において優先度 Critical 以上の欠陥である．自動欠陥修正が高い優先度の欠陥を修正できれば開発者への大きな貢献となる．

RQ-2 では，自動欠陥修正が再オープンされた欠陥を再オープンのない欠陥に比べ高い割合で修正できるかを調査する．ここで，再オープンされた欠陥とは，欠陥レポートの状態が一度でも再オープンとなったことのある欠陥である．自動欠陥修正が再オープンされた欠陥を修正できるならば，開発者がデバッグに費やす時間を削減することができる．

3. 調　　査

3.1 調 査 手 法

本研究における調査手法の概要を図 2 に示す．調査は以下の 2 ステップで構成される．

ステップ 1　欠陥と欠陥レポートの対応付け

ステップ 2　優先度と再オープンの有無の抽出

ステップ 1 では，自動欠陥修正の修正対象となる各欠陥と，その欠陥を含んだプロジェクトと連携している BTS から，各欠陥と欠陥レポートの ID を 1 対 1 に対応させる．ここで，各欠陥はバージョン管理システムから抽出されており，バージョン管理システムにおけるコミット履歴は BTS の欠陥レポートから参照できると仮定する．各欠陥がバージョン管理システムから抽出されているならば，その欠陥のコミット ID を得ることができる．そして，そのコミット ID を記録した欠陥レポートを検索することで，欠陥と欠陥レポートの対応付けを得ることができる．ただし，図 3 のように，一般に欠陥レポートとコミット履歴は 1 対多の関係にある．それは，欠陥の修正が複数のコミットに渡ったり，再オープンによって修正コミットが複数回になったりするからである．そこで，1 つの欠陥レポートに対して複数の欠陥が対応する場合，本調査では最も古い欠陥を調査に用いる．

図 3 のように欠陥レポートが 1 度再オープンされている場合，最も古い欠陥 r100 に注目すると，この欠陥は r110 のコミットで修正され，閉じられ，r120 の時に再び開かれている．そのため，r100 は実際に再オープンされた欠陥である．一方，r120 の欠陥に注目すると，この欠陥は r130 のコミットで修正され，閉じられ，その後再び開かれていない．そのため，r120 は実際には再オープンされていない欠陥であり，この欠陥を再オープンされた欠陥として用いるのは適切ではない．

ステップ 2 では，各欠陥レポートから，優先度と再オープンの有無を抽出する．優先度は欠陥レポートに記述されている．ただし，図 3 のように開発者は欠陥レポートの優先度を変更することがある．そこで，我々は最新の優先度が開発者の判断する優先度を最も反映すると判断し，本調査では，欠陥レポートに記述されている最も新しい優先度を用いる．また，再オープンの有無も欠陥レポート中のライフサイクル記録から判断することができる．JIRA のように欠陥レポートのステータスとして再オープンが用意されていれば，再オープンの有無の判断は容易である．一方，Github(注5) のように，あらかじめ再オープンが用意されていない BTS の場合，本調査では，クローズが複数回存在すれば，「再オープンあり」と判断する．

3.2 調 査 対 象

調査対象は，Java のオープンソースプロジェクトから欠陥を抽出したデータセットの Defects4J である．そのうち，Martinez らの調査 [12] において実行されたプロジェクト（Lang(注6), Math, Time(注7)）を用いる．そのため，本研究に置ける調査対象は表 3 の通りとなる．ここで，表 3 の優先度および再オープンの列はそれぞれ，優先度および再オープンの有無を調査できるか否かを表している．Time では，Github の欠陥レポートに優先度の項目が存在しないため，優先度を調査できない．

4. 調 査 結 果

各ツールに対する修正個数を優先度および再オープンの有無で分類した結果を表 4 に示す．さらに，各ツールと優先度および再オープンの有無との組み合わせにおける修正成功率を図 4 に示す．我々は，特定の優先度に対する修正成功率が全体に対する修正成功率より高いとき，自動欠陥修正がその優先度を多く修正できると判断する．また，再オープンされた欠陥に対する修正成功率が再オープンされていない欠陥に対する修正成功率より高いとき，自動欠陥修正が再オープン

表 3　調査対象のプロジェクト

プロジェクト	欠陥の数	BTS	優先度	再オープンの有無
Lang	65	Jira	利用可	利用可
Math	106	Jira	利用可	利用可
Time	12	Github	利用不可	利用可

(注5)：https://help.github.com/articles/about-issues

(注6)：http://commons.apache.org/proper/commons-lang

(注7)：http://www.joda.org/joda-time

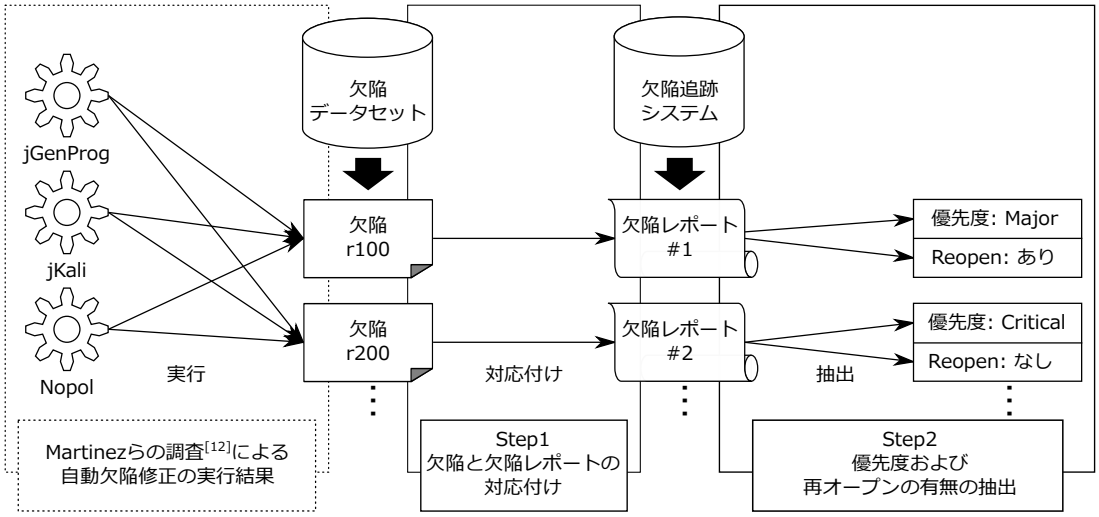

図 2　調査手法の概要

表 4　優先度別および再オープンの有無別の修正個数

	Blocker	Critical	Major	Minor	Trivial	優先度なし	再オープンあり	再オープンなし	計
jGenProg	1	2	11	4	0	2	1	19	20
jKali	1	0	11	2	0	2	1	15	16
Nopol	1	2	21	4	0	1	0	29	29
欠陥の総数	3	8	101	44	1	12	11	172	183

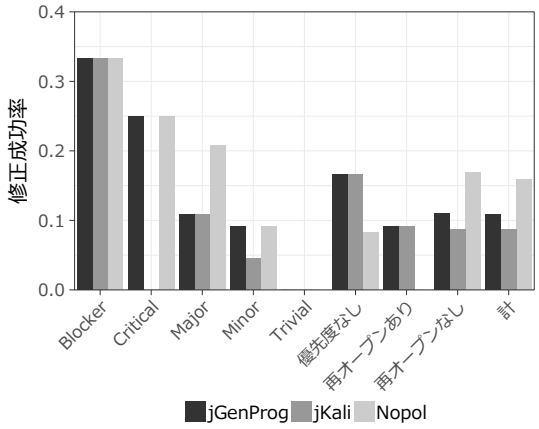

図 4　優先度別および再オープンの有無別の修正成功率

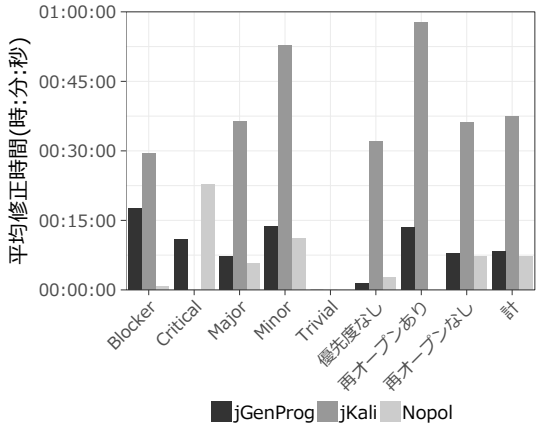

図 5　優先度別および再オープンの有無別の平均修正時間

された欠陥を多く修正できると判断する．

我々が注目する点は，早急な修正を必要とする Blocker や Critical の欠陥に対する修正成功率が，全体に対する修正成功率に比べ高くなっていることである．ただし，jKali は Critical の欠陥を全く修正できていない．そのため，RQ-1 への回答は，jGenProg と Nopol に関しては Yes と言える．一方，jKali に関しては No となる．また，Trivial の欠陥はどのツールでも修正できなかった．

一方，再オープンの有無では修正成功率はあまり変わらなかった．また，Nopol は再オープンされた欠陥を全く修正できなかった．そのため，RQ-2 への回答は，全てのツールに対して No となる．

上記の調査に加え，我々は各ツールが欠陥の修正に成功した際の修正時間を調査した．図 5 は，優先度別および再オープンの有無別の平均修正時間を示している．ここで，平均修正時間とは表 4 のセル毎に算出した修正時間の平均である．例えば，Nopol と再オープンなしの組み合わせでは，29 個の欠陥から平均修正時間を算出している．この結果から，全てのツールにおいて，Minor に対する平均修正時間が全体平均よりも長いことがわかる．再オープンされた欠陥に対しては，jGenProg と jKali の平均修正時間が長くなっている(注8)．

また，我々は優先度や再オープンの有無と修正成功率との関係を分析するため，自動欠陥修正の修正行数に注目した．自動欠陥修正において，欠陥の修正に成功した場合，修正部分をソースコードの差分として得ることができる．ソースコードの差分は追加部分と削除部分で構成される．そこで，追加部

(注8)：Nopol は再オープンされた欠陥を全く修正できていない．

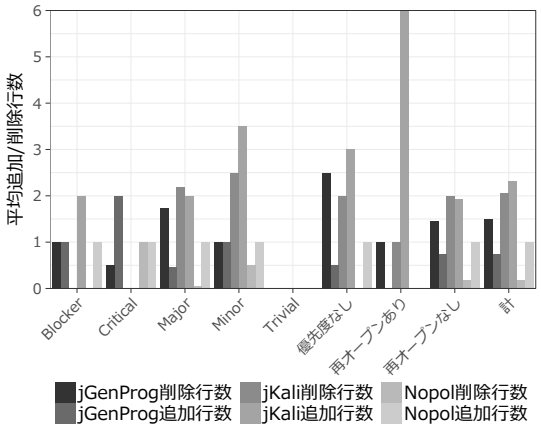

図 6　優先度別および再オープンの有無別の平均追加/削除行数

分の行数と削除部分の行数を別々に集計した．各ツールに対する平均追加/削除行数を図 6 に示す．ここで，平均追加/削除行数とは，表 4 のセル毎に算出した追加行数および削除行数の平均である．図 6 から，Minor に対する平均追加/削除行数が全体の平均追加/削除行数よりも多くなっていることがわかる．また，再オープンされた欠陥に対しては，jKali の追加行数が非常に多くなっている．

5. 考　　察

調査の結果，我々は jGenProg と Nopol が優先度の高い欠陥を多く修正することを明らかにした．また，全てのツールが再オープンされた欠陥をあまり修正できなかったことも明らかにした．特に，Nopol は再オープンされた欠陥を全く修正できなかった．修正時間の観点では，全てのツールが Minor の欠陥や再オープンされた欠陥の修正に多くの時間を掛けていた．修正行数の観点では，全てのツールの Minor の欠陥に対する修正行数が多くなっており，再オープンされた欠陥に対しては，jKali の追加行数が非常に多くなっていた．この結果から，修正行数の増加が修正時間の増加に繋がる可能性があると言える．

また，我々は全てのツールが Trivial の欠陥を修正できないことを明らかにした．Trivial の定義は，機能やデータに影響しない欠陥（例えばスペルミス）である [11]．しかし，実際の欠陥レポート MATH-393(注9) に含まれるコミットを目視確認すると，ソースコードの変更と共に，テストコードの変更が行われていた．テストコードの変更部分は図 7 の通りである．テストにおける期待値がごく僅かに変更されているが，1.0×10^{-13} の精度が要求されるため，`getResult()` および `FunctionValue()` メソッドの機能面での修正が必要である．そのため，開発者が Trivial と判断したこの欠陥レポートは，実際には Trivial の定義に反した欠陥レポートであることがわかった．

本調査は，自動欠陥修正の評価方法に，欠陥の特徴という新たな視点を与えた．既存の評価方法としては，修正個数の他，修正時間 [2]，人間にとって読みやすいか [7]，シンプルなパッチになっているか [8] など，ツールの出力を分析するものが主であった．一方で，ツールへの入力となる欠陥が持つ特徴から，ツールの性能を分析する研究はあまり見られなかった．今後，研究者が新たな自動欠陥修正を提案し，評価を行なう際には，修正個数や修正時間だけでなく，欠陥の特徴を評価に取り入れることを推薦する．欠陥の特徴を評価に取り入れることで，多角的に評価を行うことができる．我々の今後の課題は，調査対象を拡大したり，より多くの欠陥の特徴を用いたりすることで，本研究を一般化することである．

6. 妥当性への脅威

6.1 内的妥当性への脅威

本研究では，修正時間を評価する際にツールの実行時間を用いているが，修正に失敗している場合のツールの実行時間を考慮していない．ただし，修正に失敗している場合のツールの実行時間は，ツールを強制停止させるタイムアウト時間の影響を受けてしまう．また，タイムアウト時間を設けない場合，場合によって現実的な時間内に動作を停止しないツールもある [2], [5]．そのため，ツールの真の実行時間を考慮できないことは承知の上で，修正に成功している場合のみを採用している．

6.2 外的妥当性への脅威

本研究の調査対象においては，55%以上が優先度 Major であった．一方，優先度 Critical 以上の欠陥は 6%程度であり，優先度 Critical 以上の欠陥に対する調査としては，調査対象の数が不十分であったと言える．そのため，優先度 Critical 以上の欠陥を追加した調査が必要である．また，再オープンされた欠陥は 6%程度であった．これは，Mi らの Bugzilla(注10) に対する調査 [11] において再オープンされた欠陥の割合が 6〜10%であるという調査結果を支持する結果となった．しかし，再オープンされていない欠陥との修正成功率や修正時間の差異を比較するには不十分であるため，再オープンされた欠陥を追加した調査が必要である．

本研究の調査対象は，Java の欠陥データセットと，Java と対象とした自動欠陥修正である．そのため，他の言語において同様の調査を行った場合，調査結果が異なる可能性がある．

また，本研究で用いた欠陥の特徴は欠陥追跡システムから得られる優先度と再オープンの有無のみである．欠陥の特徴を追加することで，欠陥の特徴と自動欠陥修正の修正性能の関係を新たな視点から明らかにできる可能性がある．

7. お わ り に

本研究では，オープンソースの Java プロジェクトにおける 138 個の欠陥を収集したデータセットに対し，欠陥追跡システムの欠陥レポートから得られる欠陥の特徴を結びつけ，既存の自動欠陥修正の修正性能を評価した．調査の結果，jGenProg と Nopol が優先度の高い欠陥を多く修正できること，Nopol が再オープンされた欠陥をあまり修正できないことを明らかに

(注9)：https://issues.apache.org/jira/browse/MATH-393

(注10)：https://www.bugzilla.org

```
--- .../MultiStartUnivariateRealOptimizerTest.java (r799874)
+++ .../MultiStartUnivariateRealOptimizerTest.java (r979032)
@@ -83,8 +83,8 @@
   assertEquals(-0.27195612846834, minimizer.optimize(f, GoalType.MINIMIZE, -0.3, -0.2), 1.0e-13);
-  assertEquals(-0.27194301946870, minimizer.getResult(), 1.0e-13);
-  assertEquals(-0.04433426940878, minimizer.getFunctionValue(), 1.0e-13);
+  assertEquals(-0.27195612846834, minimizer.getResult(), 1.0e-13);
+  assertEquals(-0.04433426954946, minimizer.getFunctionValue(), 1.0e-13);
   double[] optima = minimizer.getOptima();
   double[] optimaValues = minimizer.getOptimaValues();
```

図 7　MATH-393 におけるテストコードの変更部分

した．また，優先度別，再オープンの有無別に，修正時間や修正差分の行数を調査し，自動欠陥修正が全く修正できなかった優先度 Trivial の欠陥に対する目視分析を行った．

我々は，調査で得た知見から，実際のソフトウェア開発における以下の場面での自動欠陥修正の活用を提案する．

- 優先度の高い欠陥に対して jGenProg や Nopol を実行することで，高い割合で修正することができる．
- デバッグの後，バージョン管理システムへコミットする前に自動欠陥修正を実行することで，開発者による差分と，自動欠陥修正による差分の違いから，再オープンに繋がるミスを防ぐ機会を得ることができる．

今後の研究課題としては，優先度の高い欠陥および再オープンされた欠陥を調査対象に追加し統計的な評価を行うことや，欠陥の特徴を増やした評価，対象言語およびツールの種類を増やした評価等が挙げられる．

謝辞　本研究は JSPS 科研費 JP25220003 の助成を受けたものです．

文　　献

[1] T. Britton, L. Jeng, G. Carver, P. Cheak, and T. Katzenellenbogen, “Reversible debugging softwareǍİ,” Technical report, University of Cambridge, Judge Business School, 2013.

[2] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, “A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each,” Proceedings of the 34th International Conference on Software Engineering, pp.3–13, 2012.

[3] F. Long and M. Rinard, “Staged program repair with condition synthesis,” Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp.166–178, 2015.

[4] S. Mechtaev, J. Yi, and A. Roychoudhury, “Angelix: Scalable multiline program patch synthesis via symbolic analysis,” Proceedings of the 38th International Conference on Software Engineering, pp.691–701, 2016.

[5] M. Martinez and M. Monperrus, “Astor: A program repair library for java,” Proceedings of the 25th International Symposium on Software Testing and Analysis, Demonstration Track, pp.441–444, 2016.

[6] J. Xuan, M. Martinez, F. DeMarco, M. Clement, S.L. Marcote, T. Durieux, D.L. Berre, and M. Monperrus, “Nopol: Automatic repair of conditional statement bugs in java programs,” IEEE Transactions on Software Engineering, pp.1–25, 2016.

[7] D. Kim, J. Nam, J. Song, and S. Kim, “Automatic patch generation learned from human-written patches,” Proceedings of the 35th International Conference on Software Engineering, pp.802–811, 2013.

[8] S. Mechtaev, J. Yi, and A. Roychoudhury, “Directfix: Looking for simple program repairs,” Proceedings of the 37th International Conference on Software Engineering, pp.448–458, 2015.

[9] R. Just, D. Jalali, and M.D. Ernst, “Defects4j: A database of existing faults to enable controlled testing studies for java programs,” Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp.437–440, 2014.

[10] E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, and K. i.Matsumoto, “Predicting re-opened bugs: A case study on the eclipse project,” 2010 17th Working Conference on Reverse Engineering, pp.249–258, 2010.

[11] Q. Mi and J. Keung, “An empirical analysis of reopened bugs based on open source projects,” Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, pp.37:1–37:10, 2016.

[12] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, “Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset,” Springer Empirical Software Engineering, pp.1–29, 2016.