ゲーミフィケーションを用いたバグパターンによる欠陥除去の促進

Facilitate Defect Removal Based on Bug Pattern Using Gamification

新井 慧* 坂本 一憲† 鷲崎 弘宜‡ 深澤 良彰§

あらまし プログラムの欠陥を検出する静的解析ツールは，ソフトウェアに作りこまれた欠陥の早期発見に役立つことが知られている．一方，欠陥の誤検出や些細な欠陥の報告が多すぎるといった問題点があり，その有用性にもかかわらず静的解析ツールの普及は進んでいない．上述の問題点を解決するため，より正確な静的解析を実現するための研究が広く進められている．しかし，これらの研究は，検出数の多さから欠陥が無視されがちになる問題を直接的に解決するための研究ではない．

本論文では，静的解析ツールを利用する開発者の欠陥修正に対するモチベーションを向上させ，報告された欠陥がより多く修正されるような仕組みを提案する．静的解析の結果をわかりやすく可視化し，報告される欠陥をより多く修正させるように動機づけることで，開発者が報告される欠陥を無視しがちになる問題を緩和し，ソフトウェアの品質向上を図る．提案する仕組みの有用性を確認するため，提案する仕組みを導入したツールを開発し，被験者実験を行った結果，ツールを利用することによっておよそ 1.5 倍ほど多くの欠陥が修正されることを確認した．

1 はじめに

ソフトウェア開発においてプログラミングを行う実装の工程において，ソースコードの記述に関する誤りによって欠陥が発生することが多い．実装工程における欠陥の混入を防ぐため，ソースコードの静的解析技術を用いてソースコード中で欠陥に繋がる箇所を検出し，ソフトウェアの品質向上を支援するツールが開発されている [1]．静的解析ツールを利用することで，欠陥を早期に発見することが容易となり，開発時の欠陥除去にかかる時間・費用を削減できる．このような静的解析ツールとしては，FindBugs[1] や PMD[2]，Jlint[3] などが有名である．

しかし，実際の開発においては，静的解析ツールはあまり普及していない．その原因として，Johnson らは，欠陥の誤検出と誤検出も含めた検出数の多さを挙げている [2]．現在広く用いられている静的解析ツールにはいずれのツールでもある程度の誤検出が発生する．また，重大な欠陥に直結しない欠陥も検出するため，実際に修正すべき欠陥数に対し，ツールによる検出数は多くなってしまう．膨大な検出数に対し，開発者はそれを修正するモチベーションの維持が難しいため，修正は敬遠されがちである．その他にも Johnson らは，出力結果の理解までに時間がかかる点や，ツールのカスタマイズが難しくチーム間で設定の共有化が難しい点も挙げている．結果的に，静的解析ツールは品質向上に役立つものでありながら，開発者が十分に活用できないため，開発で利用されることが少ない．これらの問題を解決するため，新たな技術に基づき静的解析ツールを開発し，検出の精度を高め，有用性を高める研究が続けられている [3]．しかし，静的解析の技術を向上させる研究は，欠

*Satoshi Arai, 早稲田大学

†Kazunori Sakamoto, 国立情報学研究所

‡Hironori Washizaki, 早稲田大学

§Yoshiaki Fukazawa, 早稲田大学

[1]FindBugs - Find Bugs in Java Programs, http://findbugs.sourceforge.net/.

[2]PMD, http://pmd.sourceforge.net/.

[3]Jlint, http://jlint.sourceforge.net/.

陥が無視されがちになる問題に焦点を当てた研究ではない．

本論文では，誤検出など静的解析ツールの性能上の問題点は考慮せずに，検出された欠陥が無視される問題を緩和する仕組みを提案する．そこで我々は，開発者に報告される欠陥をより多く修正させるため，ゲーミフィケーションを導入した欠陥報告ツールである GBC（Gamebase Bug Catcher）を提案する．ゲーミフィケーションとは，ゲームを構成する要素をゲーム以外のものに応用し，利用者のモチベーション向上を図る手法である [5]．GBC では，バグパターンを修正するごとに利用者に得点を加算する制度（以降，得点制度）を実装し，開発メンバー間の競争を促す．また，一定時間得点が上昇する機能や，特定のバグパターンを修正した場合ボーナスとして得点が追加される機能を加え，競争に幅を持たせる．

本論文では，以下の二点を研究課題（RQ）とする．

- RQ1 既存の静的解析ツールに対して，ゲーミフィケーションを導入することができるか？
- RQ2 検出された欠陥を修正することにより，得点が加算される機能を追加することで，欠陥はより修正されやすくなるか？

本論文の構成は以下の通りである．2 節では，欠陥検出に関する既存の研究とその問題点について説明し，3 節ではそれらの問題を緩和するために GBC で利用する技術について述べる．次の 4 節において，提案手法に基づいて実装した機能の詳細について述べ，5 節に本論文の実験手順および結果と考察を述べる．6 節では妥当性への脅威，7 節に本手法の制限について述べる．そして，8 節にて関連研究を紹介し，最後の 9 節でまとめと今後の展望を述べる．

なお，本論文では，静的解析ツールによって示された「欠陥を引き起こす可能性のある箇所」についても同様に「欠陥」と呼び，FindBugs によって検出された「欠陥」を「バグパターン」と呼ぶことにする．

2　既存の静的解析ツールとその問題点

本節では，まず静的解析ツールの一つである FindBugs について紹介し，次に FindBugs も含めた静的解析ツールにおいて指摘されている問題点について述べる．

2.1　FindBugs による静的解析

FindBugs はバグパターンに基づいて欠陥を検出する静的解析ツールであり，Eclipse や Jenkins などでプラグインとしても提供されている．静的解析ツールのベンチマークを測定する研究ではよく引き合いに出されるなど [3]，Java 言語向けの静的解析としては最も有名なツールの一つである．

FindBugs において用いられているバグパターンにはそれぞれ名称があり，その欠陥の種類や発生理由，影響度などでカテゴライズされている．例として，図 1 にバグパターンの埋め込まれているソースコードを示す．また図中では，コメントアウトされた部分にその行で検出されたバグパターンの名称を追記している．

図 1 は中のソースコードには，FindBugs によればこの中に 5 つのバグパターンが存在しており，図 2 で示すバグパターンは，それらから一部を抜粋したものである．3 行に 1 つの割合でバグパターンが存在しており，またそれらは修正方法が異なるバグパターンであるため，これらを修正するには労力がかかる．

図中にある Confidence, Rank, Pattern, Type, Category についてそれぞれ説明する．Confidence は検出結果の信頼性であり，これが高ければ正しい検出結果である可能性が高い．Rank は，バグパターンの重要性を示しており，0 から 20 まで 21 段階のレベルで表現される．この数値が低いほど，修正する優先度の高いバグパターンである．Pattern と Type はバグパターンの名称や属性を示す．Category には主に Correctness, Style(Dodgy Code), Bad Practice などがあり，そのバグパターン

```
1   public String Method1(String name) {  //  NM_METHOD_NAMING_CONVENTION
2     int num = 0;
3     String str = null;
4     if (name.equals("FOSE2013")) {
5       num = 1;
6     } else if (name.equals(null)) {  // EC_NULL_ARG
7       num = 2;
8     }
9     switch (num) {          //  SF_SWITCH_NO_DEFAULT
10      case 1 :
11        str = "Bug";        //  SF_SWITCH_FALLTHROUGH
12      case 2 :
13        str = "Pattern";      //  SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH
14    }
15    return str;
16  }
```

図 1　5 つの欠陥が検出される Java サンプルコード

Line 9

Confidence: Normal, Rank: Scary (9)
Pattern: EC_NULL_ARG
Type: EC, Category: CORRECTNESS (Correctness)

Line 16

Confidence: High, Rank: Scariest (1)
Pattern: SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH
Type: SF, Category: CORRECTNESS (Correctness)

図 2　サンプルコードから検出されるバグパターンの一部

が引き起こし得る欠陥の種類で分類している．例えば Correctness は，正確性の問題と訳され，開発者が意図しない動作をする可能性が高いコード片を指す．

2.2　主な静的解析ツールの問題点

静的解析ツールの有用性についてはこれまでの研究で明らかになっている [1–4]．しかし，それらのツールを利用する上で主に以下の 3 つの問題が指摘されている．

1. 静的解析ツールの誤検出の問題 [3]
2. 静的解析ツールの選定・導入コスト [4]
3. 開発者が処理しきれないほどの検出数 [4]

1 の問題は，静的解析ツールの信頼性に関わる問題の中で最も重要であり，全ての静的解析ツールはこの問題を可能な限り解決することが求められる．2 の問題は，実際の開発においてどの程度ツールが使いやすいかに関する問題である．ツールの選定や導入，設定にかかるコストが高いと，いくら性能が高くとも利用されづらくなるため，ツールを広く利用してもらうにはこの点を考慮する必要がある．

3 の問題は，誤検出や修正する優先度の低い欠陥も含め，すべてを報告すると開発者の手におえなくなる問題である．一方で，無暗に優先度の高い欠陥のみを検出するように設定して検出数を減らすと，ツール自体を利用する必要性が薄れてしまう問題もある．この 2 点はトレードオフの関係にあり，開発者ごとに求める検出精度は変化するため，最適なバランスを定義することは難しい．

1 や 2 の問題を緩和するために，静的解析の精度をより向上させる手法や，複数の静的解析ツールを組み合わせることで欠陥の誤検出や導入コストを減らす手法が提案されている．しかし，静的解析の精度向上に関しては広く議論されており，多くの静的解析ツールが提供されているが，誤りのない欠陥検出が可能なツールはまだ存在しない．また，3 の問題に関しては，誤検出を減らし，合計の検出数も減らすことで解決を図っている．この問題に対し我々は，たとえ検出数が多くとも，開発者にそれらを修正するだけの意欲があれば，問題の重大さも小さくなると考える．

3　開発者のモチベーション向上による問題の緩和

本論文では，検出数が多く処理しきれない問題を緩和することを目的とする．そこで我々は，検出される欠陥の信頼性や優先度を問わず，検出される多くの欠陥に対して修正するモチベーション（意欲）を保つ仕組みを提案する．検出精度の向上によって問題を緩和する従来の手法とは別の視点からのアプローチである．

3.1　ゲーミフィケーションの導入

ゲーミフィケーションとは，ゲーム要素をゲーム以外の分野に応用することで，利用者のモチベーション向上を図る手法である．例として，ソーシャルネットワークサービスの Foursquare[4] を挙げる．Foursquare は，携帯端末などの位置情報を利用し，ある場所へチェックインすることでその情報を友人達と共有できる．また，チェックインした場所やその数に応じて利用者に得点などの報酬が与えられ，友人たちと競争して楽しむことができる．ゲーミフィケーションは，ソーシャルネットワークサービス以外にも，新たな教育手法として着目されており [7] [8]，ビジネスや教育の分野を問わず広く導入されている．本論文は，欠陥修正に対するモチベーションの向上を目指しているため，この手法が有効であると判断し，どのように導入すべきかを検討した．

3.2　ゲーム化に向けたゲームメカニクスの利用

ゲームメカニクスとは，ゲームの構成するいくつものルールを総称したものである．例えば，敵を倒す，経験値を得る，レベルアップする，といったルールはゲームメカニクスとして分類される．ゲームメカニクスの一つ一つには，プレイヤーにゲームを楽しんでもらうための工夫が凝らされている．本論文で提案する GBC では，得点制度を導入することで，モチベーションの向上を図る．得点制度はゲームメカニクスとしては一般的であり，利用者のモチベーションを向上させるための手法として用いられている [10]．さらに，利用者や他のメンバーが獲得した点数を表示することで，メンバーの間で競争を生み出す．競争は，ゲームの概念としても一般的であるほか，学習する意欲を向上させる手段としても有効である [9]．

また，ゲームはプレイヤーが飽きると遊ばれなくなるため，それを回避するためにゲームに何らかの変化を加える．そこで，得点制度の利用に合わせて，一定の条件下で得点を上昇させるボーナス制度を追加する．このボーナス制度は，得られる報酬を上昇させることでさらにモチベーションを向上させる狙いがある．

4　欠陥修正のモチベーションを向上させるツールの実装

本論文では，ゲーミフィケーションを導入した欠陥報告ツール GBC を開発し，そのシステムの全体像を図 3 に示す．また，提案手法を実現するために満たすべき要求と，それに対して GBC で実装した機能は，以下の通りである．

- 検出されたバグパターンの情報取得
  - GBC で利用する得点をバグパターンの情報から算出するため，バグパター

[4] Foursquare, https://ja.foursquare.com/.

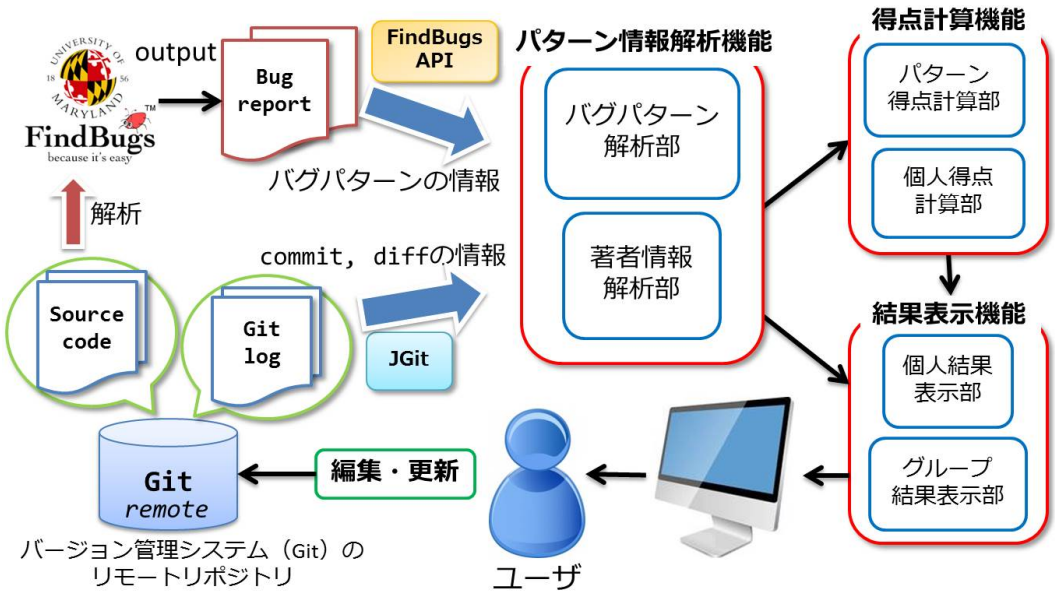

図 3　GBC におけるシステムの全体像

ン解析部で情報を取得する．

- 検出されたバグパターンの著者・修正者の特定
  - 得点を開発者ごとに算出するため，著者情報解析部でバグパターンごとに著者・修正者を特定する．
- 開発者が修正したバグパターン修正履歴の保持
  - 開発者ごとに総合得点を計算し表示するため，パターン情報解析機能により修正履歴を保持する．
- 得点制度およびボーナス制度の実装
  - 開発者により多くのバグパターンを修正させるため，得点計算機能により得点を加算する．
- 修正結果や獲得点数の表示
  - 得点を開発者に対して表示し，競争を生み出すため，結果表示機能により個人とグループ全体の成績を表示する．

4.1　FindBugs と Git を併用したバグパターンの解析

GBC を開発するにあたり，欠陥検出のための静的解析ツールとして，FindBugs を採用した．バグパターンおよび FindBugs を対象とした理由は，FindBugs では単に欠陥の箇所を提示するだけではなく，欠陥の種類や重要度など様々な情報を提示するため，ゲームの仕組みを構築する際に利用可能な要素が十分にあり，提案手法に適していると判断したためである．また，ソースコードのバージョン履歴を利用するため，Git[5] を併用する．Git とは，バージョン管理システムの一つであり，対象フォルダに commit（ファイルの編集・変更を決定し更新すること．以降，コミット）による log（変更履歴）や diff（差分）の情報が保存される．Git によって管理された，あるリビジョンのソースコードとその更新元であるリビジョンのソースコー

[5] Git, http://git-scm.com/.

ドを対象にそれぞれ FindBugs を実行し，検出される欠陥の差分から追加または修正された欠陥を特定し，リビジョン間の更新者から著者および修正者を判断する．

4.2 得点制度に基づく競争機能の実装

利用者がそれぞれ修正したバグパターンの数や情報をもとに得点制度を実装し，メンバーの欠陥修正に対するモチベーション向上を図る．GBC では，バグパターンの Rank，Category の情報に基づいて，得点制度を実装する．パターン情報解析機能で特定された修正者に対し，修正したバグパターンの Rank に応じた得点を与える．獲得できる点数は（21 - Rank）点と定め，重要な欠陥ほど得点を高く設定し，グループメンバーに点数で競わせることで，各メンバーのモチベーション向上を図る．加えて，ボーナス制度に該当する機能を実装する．GBC では，一定時間無条件で得点を倍にする機能と，修正したバグパターンの Category に応じて得点が追加されるボーナス機能を実装した．

5 GBC を利用した被験者実験と考察

GBC を用いて，実際に被験者実験を行った．今回対象とした被験者は，Java 言語の経験がある情報系の大学生および大学院生計 6 名である．

5.1 実験手順

本実験は，Java 言語で開発されたオープンソースソフトウェア（以降，OSS）を対象とし，FindBugs によって検出されるバグパターンを修正することを目的とする．対象とする OSS は，Minecraft のサーバーを拡張するソフトウェアである bukkit[6] と，Twitter の機能を Java プログラム上から利用するためのライブラリである twitter4j[7] である．ただし，twitter4j は検出されるバグパターンの個数を考慮し，機能全体のコア部である twitter4j-core のみを対象とした．なお，bukkit の LOC は 26917 行であり，twitter4j-core は 18123 行であった．

実験手順は以下の通りである．

1. 3 名ずつのグループを 2 つ作る．以降，それぞれを G_A，G_B とする．
2. 2 種類の OSS（以降，bukkit を S_A，twitter4j-core を S_B）を我々の Git リポジトリ上に用意し，各グループはそれらを各自のマシンにコピーする．
3. まず，30 分間 S_A を対象に実験を行う．G_A は FindBugs と GBC を利用した状態でバグパターン修正を行い，G_B は FindBugs のみを利用して修正を行う．
4. GBC を用いる場合は，各メンバーの編集がリモートリポジトリに反映されるごとに修正結果を表示する．修正結果の表示例を，図 4 および図 5 に示す．
5. 開始から 30 分後，修正対象を S_B に変更し，同様に 30 分間，G_A は FindBugs のみでバグパターン修正を行い，G_B は GBC を併用して修正を行う．
6. 同様に，Git リポジトリが更新されるごとに修正結果を表示する．

5.2 研究課題 1 への回答

ゲーミフィケーションを導入したツールを開発し，実際に評価実験を行うことに成功したため，この時点で研究課題 1 について回答する．

- RQ1 既存の静的解析ツールに対して，ゲーミフィケーションを導入することができるか？

[6] https://github.com/Bukkit/Bukkit

[7] https://github.com/yusuke/twitter4j

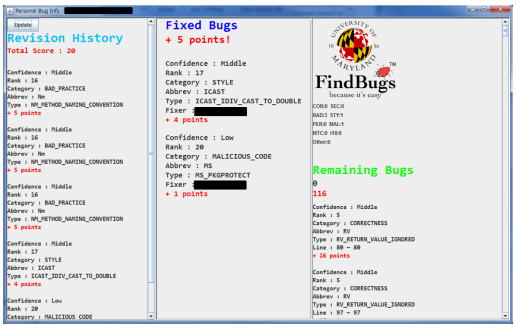

図 4　個人が修正したバグパターンの一覧表示例

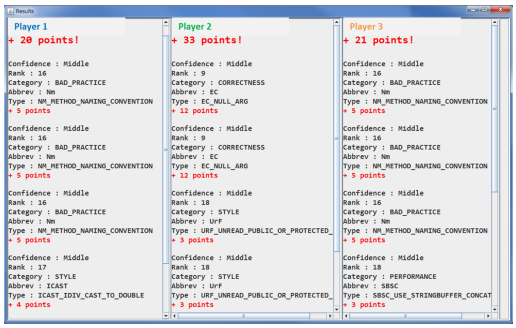

図 5　グループメンバーが修正したバグパターンの一覧表示例

Git リポジトリの履歴から別途ファイルの更新者を取得し，修正者を特定することで，グループの情報を保持できる環境を構築した．ゲーム化できる要素を増やすため，バグパターンの情報から Rank や Category を利用し，ゲームメカニクスに基づき得点制度を導入したことで，ゲーミフィケーションに成功していると言える．したがって，静的解析ツールによる欠陥修正を行う際に，バグパターンを用いたゲーミフィケーションを導入することができる．

5.3　実験結果

GBC を利用した場合とそうでない場合で，修正されるバグパターンの数を比較し，その結果を表 1 に示す．また，各グループごとのバグパターン修正数とコミットごとの遷移をそれぞれ図 6 と図 7 に示す．グループ G_A と G_B では Java 言語を編集する知識や能力に差があったため，グループを跨いで比較することはできない．そのため，各グループごとに GBC を利用した場合とそうでない場合で修正数を比較し，GBC を評価する．

表 1　バグパターン修正数と減少率

グループ	対象 OSS	GBC あり/なし	初期検出数	修正数	減少率
G_A	S_A	あり	168	26	15.48%
	S_B	なし	150	13	8.67%
G_B	S_A	なし	168	41	24.40%
	S_B	あり	150	59	39.33%

表 1 における初期検出数とは，本実験で修正作業を行う前に，対象 OSS から検出されたバグパターンの個数である．また，修正数は本実験において各グループがそれぞれの OSS で修正したバグパターンの個数であり，減少率は初期検出数からどの程度バグパターンが減少したかを表している．なお，ボーナス機能 2 種類を残り 10 分の段階で動作させたが，修正数が増加するなど実験結果に反映させることが出来なかった．これは，被験者が修正作業に飽きるまで実験時間を確保できなかったことに起因すると思われるため，さらに規模の大きい実験を行うことで，修正数の増加効果が表れると考える．

実験結果を見ると，G_A，G_B ともにゲーミフィケーションを導入した場合でバグパターンの修正数は上昇している．同時に，ソフトウェア上のバグパターンがどれほど減少したかを示す減少率は，どちらも 1.5 倍以上増加している．よって，得点制度に基づいたゲーミフィケーションを導入することで，検出された欠陥の修正数

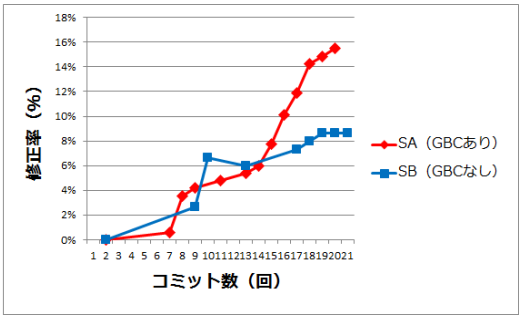

図 6　G_A の修正結果

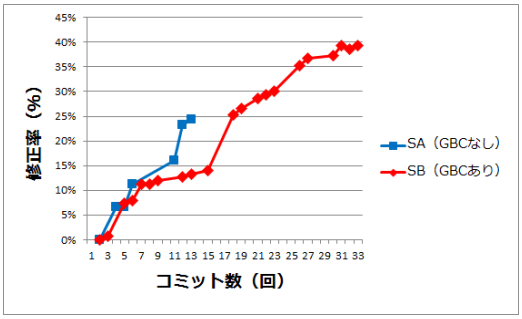

図 7　G_B の修正結果

は上昇することが期待できる．

5.4　評価実験に対する被験者の感想

被験者に本実験の感想・意見を聞き，実際にモチベーションの向上はあったのか調査した．被験者 6 名のうち，GBC を利用した場合とそうでない場合ではどちらの方が良いか，という質問に対し，5 名はある方が良いと答えたため，概ね実装した機能はバグパターン修正のモチベーション向上に対して有効だと言える．

GBC を利用した場合,「バグパターンを修正するモチベーションが上がり，積極的に修正しようと思うようになった」という肯定的な意見が聞かれた．一方で,「時間制限があることで安易な修正のまま更新してしまい，正しい修正を怠ったように感じた」という，否定的な意見も聞かれた．GBC を利用しない場合は,「他のメンバーと競争する必要がないため，修正作業に集中できる」という，我々の提案手法に対しては否定的な意見が聞かれたが,「単純な作業になるために退屈になり，修正するモチベーションが上がらない」といった提案手法が緩和すべき意見も聞かれた．

5.5　研究課題 2 への回答

5.3 と 5.4 の結果より，研究課題 2 について回答する．

- RQ2 検出された欠陥を修正することにより，得点が加算される機能を追加することで，欠陥はより修正されやすくなるか？

バグパターンの減少率は，どちらのグループも 1.5 倍以上増加しているため，得点制度によって検出されたバグパターンの修正数は上昇することが確認できた．さらに，被験者の意見調査結果では，欠陥修正に対するモチベーションが上がるため，GBC を利用した方が良いと考える意見が多く聞かれた．逆に，時間制限に圧迫され，修正の正確性が落ちる可能性あることを理由に，GBC を利用しない方が良いという意見もある．ただし，時間制限は実験において便宜的に導入したものであるため，圧迫する理由は GBC の仕様に限られたものではない．総合して，GBC は欠陥修正数の向上に有効な機能を持っていることが分かった．

GBC を利用しない場合の意見から，検出数が多く処理しきれない問題が実際にあることが確認できたため，既存の技術のみではこの問題を緩和することができないことが分かる．一方，GBC を利用した場合では同様の意見が出なかったため，この問題の原因を解消している．したがって，GBC は検出数の多さから欠陥が無視されがちになる問題を緩和するための一助となっており，欠陥をより多く修正するためのしくみとして有効である．

6　提案手法の制限

GBC は，実行するためにはコマンドライン版の FindBugs や，ant によるビルド環境，実行結果の保存ディレクトリを別途用意する必要があるため，被験者に直接配布することが出来なかった．そのため本実験では，GBC を実行できる環境を持つマシンを 1 台用意し，実行結果を被験者に向けて表示するという方法を取った．これは静的解析ツールが導入にコストがかかるため利用されないという問題 [4] に反するため，GBC はこの制限を取り除く必要がある．ただし，将来的に GBC を Web アプリケーション化することで，導入のコストを減らすことができると考えている．

本論文はソフトウェアの品質を高めるためのアプローチを提案したが，FindBugs によるバグパターン修正によってどの程度品質が向上するかを測定していない．しかし，静的解析ツールは品質向上に役立つと述べている文献 [1–4] を踏まえて，バグパターンを修正することで品質が向上するという仮説に基づいている．

7　妥当性への脅威

FindBugs や Git を用いて実際に実験を行ったが，これらの取り扱いには経験や慣れが影響する．本来であれば，2 回目に行った実験結果の方がより良い結果をもたらすことが想定される．実際に，S_A を対象にした 2 回目の実験では，両グループともにコミット数は上がっている．これは，Git の操作や FindBugs による修正手順に慣れ，コミット可能な状態まで編集を進めやすくなったため，コミット数が上がったと考えられる．ただ，1 回目に GBC を利用した実験を行っているグループも，GBC 利用時の方がバグパターン修正数・修正率は上昇している．そのため，ある一定期間での慣れによる修正技術の向上よりも，ゲーミフィケーションに基づいたシステムを提供する方が，モチベーションの向上が見込めると判断できる．

本実験では既存のソフトウェアから検出されるバグパターンを修正する作業のみを行った．しかし，静的解析ツールはソフトウェアの開発を進めながら，定期的に欠陥を除去するために利用されるのが一般的である．また，自身が開発に全く関わっていないソフトウェア上の欠陥を除去する機会は実開発では稀である．今後は一般的な開発プロセスを調査し，その開発形態に合わせて GBC を改良する予定である．

8　関連研究

Thung らは，FindBugs, PMD, Jlint の 3 種類の静的解析ツールを対象に，False Negatives についての評価を行っている [3]．False Negatives とは，本来欠陥であるはずだが検出されないものを指し，欠陥ではないが欠陥として検出されるものである False Positives とは異なる．彼らは，3 種類の OSS を用いて実験を行い，FindBugs と PMD は比較した静的解析ツール中では False Negatives を防ぐために有用であると述べている．ただし，いずれのツールの評価でも誤検出を防ぐことはできず，静的解析ツールに挙げられている問題は依然として存在している．

Nanda らは，選定コストや欠陥検出数といった問題を緩和するため，複数の静的解析ツールを実行できるオンラインポータルを開発した [4]．論文では，ツール上の Usability Problems と Usefulness Problems を解決するため，ユーザからのフィードバックや検出結果の比較機能を実装し，検出精度の向上を図っている．ユーザからのフィードバックを受け取ることにより検出結果が誤検出かどうか判断することで，欠陥の報告数を抑える手法をとっており，本論文とはアプローチが異なる．

Singer らは，バージョン管理システムを利用したチーム開発にゲーミフィケーションを導入し，コミットする頻度を増やす手法を提案している [6]．チームメンバーのコミット数を管理し，ニュースフィードで逐一報告することで，チーム間の競争を生み出し，多くのコミットを行うよう促している．導入実験を行った結果，より多くのコミットを心がけるようになるなど，競争を促すことに好意的な意見が聞かれ

たが，この仕組みに対して否定的な意見も聞かれている．バージョン管理システムを利用したソフトウェア開発にゲーミフィケーションを導入する研究であるが，欠陥修正にゲーミフィケーションを導入する研究はまだ報告されていない．

9 おわりに

本論文では，開発者により多くの欠陥を修正するように促すため，ゲーミフィケーションを用いたバグパターンによる欠陥検出手法を提案した．FindBugs と Git からバグパターンの情報，ソースコードの変更履歴などを抽出し，ゲーミフィケーションに利用できる情報を集約した上で，ゲームメカニクスに基づく欠陥報告ツール GBC を開発した．GBC を用いて実際に評価実験を行い，バグパターンの修正数にどのような変化が現れるかを測定した．その結果，ゲーミフィケーションによってバグパターン修正数および修正率が向上することを確認した．

本論文で定めた研究課題に対する回答は，以下の通りである．

- **RQ1) 欠陥修正へのゲーミフィケーションの導入**：バージョン管理システムの変更履歴とバグパターンの情報を用いることで，ゲーミフィケーションを導入したツールを開発することが可能である．
- **RQ2) ゲーミフィケーションによる欠陥修正数の向上**：修正情報を提示し，得点制度による競争を促すことで，より多くの欠陥を修正する効果を確認した．

今後の展望としては，環境に依存せず GBC の機能を広く利用できるように，Web アプリケーション版を開発したい．加えて，現在完全には自動化できていない FindBugs の実行やボーナス機能の処理を自動化し，利便性の向上を図りたい．また，その他のゲームメカニクスを取り入れることができるか，どのように取り入れるべきかを考え，欠陥除去工程とゲーミフィケーションの関係性について議論を深めたい．

謝辞 本論文の一部は，公益財団法人中山隼雄科学技術文化財団の助成による．

参考文献

[1] D. Hovemeyer and W. Pugh, "Finding bugs is easy", *19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 92-106, 2004.
[2] B. Johnson, Y. Song, and E. Murphy-Hill, "Why Don't Software Developers Use Static Analysis Tools to FindBugs?", *2013 International Conference on Software Engineering*, pp. 672-681, 2013.
[3] F. Thung, Lucia, D. Lo, L. Jiang, F. Rahman, P. T. Devanbu, "To What Extent Could We Detect Field Defects? An Empirical Study of False Negatives in Static Bug Finding Tools", *27th International Conference on Automated Software Engineering*, pp. 50-59, 2012.
[4] M. G. Nanda, M. Gupta, S. Sinha, S. Chandra, D. Schmidt, P. Balachandran, "Making Defect-Finding Tools Work for You", *32nd International Conference on Software Engineering*, pp. 99-108, 2010.
[5] S. Deterding, D. Dixon, R. Khaled, L. Nacke, "From Game Design Elements to Gamefulness: Defining "Gamification"", *15th International Academic MindTrek Conference*, pp. 9-15, 2011.
[6] L. Singer, K. Schneider, "It Was a Bit of a Race: Gamification of Version Control", *The 2nd International Workshop on Games and Software Engineering*, pp. 5-8, 2012.
[7] J. Pieper, "Learning Software Engineering Process through Playing Games", *The 2nd International Workshop on Games and Software Engineering*, pp. 1-4, 2012.
[8] N. Tillman, J. Halleux, T. Xie, "Pex4Fun: Teaching and Learning Computer Science via Social Gaming", *24th Software Engineering Education and Training*, pp. 546 - 548, 2011.
[9] G. Gabrysiak, R. Hebig, L. Pirl, H. Giese, "Cooperating with a Non-governmental Organization to Teach Gathering and Implementation of Requirements", *26th International Conference on Software Engineering Education and Training*, pp. 11-20, 2012.
[10] L. von Ahn and L. Dabbish, "Designing games with a purpose", *Communications of the ACM*, pp. 58-67, 2008.