論　文

得点による競争原理を用いた静的解析ツールによる欠陥除去の促進

新井　慧[†]　坂本　一憲[††]　鷲崎　弘宜[†]　深澤　良彰[†]

Facilitate Defect Removal Using Static Analysis Tools by introducing competition

Satoshi ARAI[†], Kazunori SAKAMOTO[††], Hironori WASHIZAKI[†], and Yoshiaki FUKAZAWA[†]

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．静的解析によって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．

キーワード　静的解析ツール，バグパターン，ゲーミフィケーション

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(注1) や PMD(注2)，Jlint(注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ことを目的とした欠陥報告ツールである GBC（Game-based

[†] 早稲田大学，東京都
Waseda University, Tokyo, Japan
[††] 国立情報学研究所，東京都
National Institute of Informatics, Tokyo, Japan
(注1)：FindBugs, http://findbugs.sourceforge.net/.
(注2)：PMD, http://pmd.sourceforge.net/.
(注3)：Jlint, http://jlint.sourceforge.net/.

```
1   public String Method1(String name) {  //Bug
2     int num = 0;
3     String str = null;
4     if (name.equals("SIGSS2014")) {
5       num = 1;
6     } else if (name.equals(null)) {     //Bug
7       num = 2;
8     }
9     switch (num) {                      //Bug
10      case 1 :
11        str = "Bug";                    //Bug
12      case 2 :
13        str = "Pattern";                //Bug
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

Bug Catcher）を提案する．GBC では，バグパターンを修正するごとに開発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．

本論文では，以下の二点を研究課題（RQ）とする．

- RQ1 欠陥を修正する度に得点が加算される機能によって，欠陥はより修正されやすくなるか？
- RQ2 導入した得点制度により，開発者のモチベーションは向上するか？

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，次に静的解析ツール全般につ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．

図 1 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いる．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．

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・導入コストの問題 [4]

（3） 膨大な数の欠陥が報告される問題 [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．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. 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，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．そして，提案した手法を実現するための欠陥報告ツール GBC を開発した．GBC のシステム全体像を図 2 に示す．

- 検出されたバグパターンの情報取得
  - 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

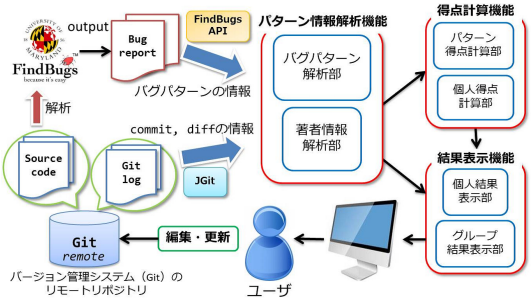

図 2　GBC におけるシステムの全体像

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
  - 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．

3.1　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，機能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(注4) 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．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．

3.2　得点制度に基づく競争機能の実装

開発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，開発者の欠陥修正に対するモチベーション向上を図る．GBC では，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．例えば，獲得できる点数は Rank の高い（重要な）欠陥ほど高く設定する．このように開発者が得点で競うことにより，各メンバーのモチベーション向上を図る [7]．

この制度については，ゲーミフィケーションの概念を参考にしている．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 [5]．例として，ソーシャルネットワークサービスの Foursquare(注5) 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，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，この概念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．

3.3　不正防止機能の実装

得点を用いた機能を実装するにあたり考慮すべき点は，自身で故意に欠陥を埋め込み，それを修正することにより，点数を不正に追加できる点である．これに対して GBC では，開発メンバーがお互いの修正したバグパターンを確認し合える環境を作ることで対策をとっている．ただし，今回行った被験者実験では不正を行った者がいなかったため，効果については考察していない．より厳格な不正防止機能を実装する場合には，修正された各バグパターンごとに追加された時間やコミット番号と，その著者を確認できるようにし，明らかに単純な欠陥を埋め込んだ場合や修正するまでの期間が短い場合などに強調表示されるようにしたい．また，欠陥を埋め込んだ場合に点数が減少する仕組みも考えられるが，単純に減少させるだけだと総合得点が低くなってしまい，競争が捗らない．よって，バグパターンの Rank やカテゴリー，出現頻度などを考慮

(注4)：Git, http://git-scm.com/.

(注5)：Foursquare, https://ja.foursquare.com/.

して定める必要があると考える．

4. 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．

4.1 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(注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(注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 を 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 bukkit 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 3 および図 4 に示す．

（5） 開始から 30 分後，修正対象を twitter4j に変更し，同様に 30 分間，G_A は FindBugs のみで修正を行い，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．

4.2 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時間ごとの遷移をそれぞれ図 5 と図 6 に示す．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．

(注6)：https://github.com/Bukkit/Bukkit
(注7)：https://github.com/yusuke/twitter4j

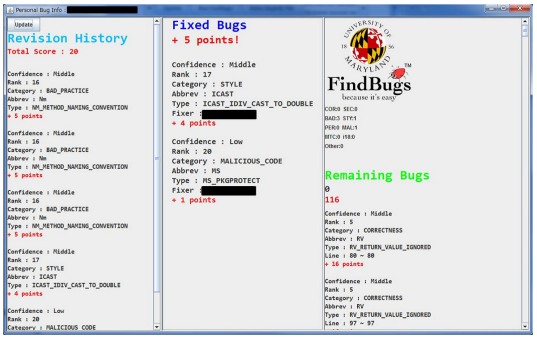

図 3 個人が修正したバグパターンの一覧表示例

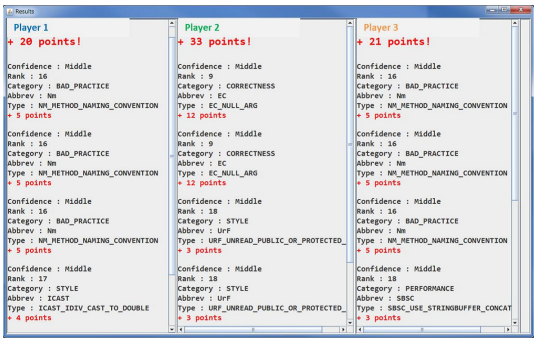

図 4 グループメンバーが修正したバグパターンの一覧表示例

表 1 バグパターン修正数と減少率

グループ	対象 OSS	GBC	初期検出数	修正数	減少率
G_A	bukkit	あり	168	26	15.48%
	twitter4j	なし	150	13	8.67%
G_B	bukkit	なし	168	41	24.40%
	twitter4j	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．

実験結果を見ると，両グループとも GBC を併用した場合では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 倍程度増加している．よって，得点制度を導入することで，検出された欠陥の修正数は上昇することが期待できる．

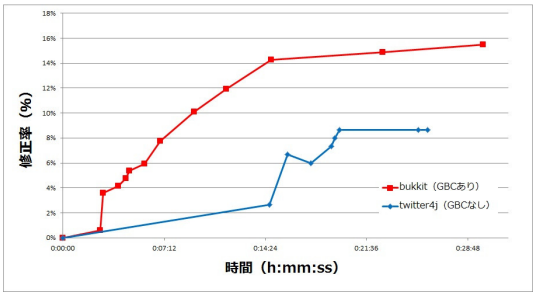

図 5　G_A の修正結果

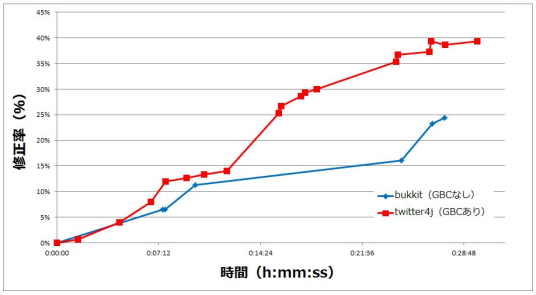

図 6　G_B の修正結果

4.3　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,「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．

4.4　考　　察

実験結果をもとに，RQ1 と RQ2 について回答する．

- RQ1 欠陥を修正する度に得点が加算される機能によって，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 倍程度増加しており，得点制度によって検出されたバグパターンの修正数は増加した．よって，得点制度の機能は欠陥修正数の向上に有効である．

- RQ2 導入した得点制度により，開発者のモチベーションは向上するか？

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．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．したがって，GBC はモチベーションを向上させ，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．

5.　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．

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．

6.　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ため，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．Git の操作や FindBugs による修正手順に慣れ，編集を進めやすくなったため，コミット数が上がったとも考えられる．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，GBC が持つ機能を提供する方が，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．

7. 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，一般的な誤検出とは異なる．彼らは，3 種類の OSS を対象に実験を行い，FindBugs と PMD は False Negatives を防ぐために比較的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．

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．ソフトウェア開発行程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．

8. 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，得点制度を用いた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，機能の実装に利用できる情報を集約した上で，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，欠陥を修正する度に得点が加算される機能によって，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．
- **RQ2) モチベーションの向上**：被験者に向けて行った意識調査の結果から，GBC の機能はモチベーションを向上させる効果を期待できる．

今後の展望としては，現在完全には自動化できていない GBC の機能を自動化し，利便性の向上を図りたい．また，新たなゲーミフィケーションに関する手法をどのように取り入れるべきかを考え，欠陥除去工程との相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．

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 and 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 and 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 and L. Nacke, "From Game Design Elements to Gamefulness: Defining "Gamification"", *15th International Academic MindTrek Conference*, pp. 9-15, 2011.

[6] L. Singer and 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] L. von Ahn and L. Dabbish, "Designing games with a purpose", *Communications of the ACM*, pp. 58-67, 2008.