4K-02

グラフモデルを用いたOSSバグの影響分析方法の提案と評価

広岡 伸之甫† 今堀 由唯† 吉田 隆佑† 青山 幹雄†

南山大学 理工学部 ソフトウェア工学科†

1. 研究背景と課題

OSS(Open Source Software)開発の参画者の多くはバグの報告，修正を行っている．初期から開発に携わっている参画者は開発の全貌を理解しているためバグが報告されてからの修正を迅速に対応できる．しかし，中途参画者は全貌をすぐに把握することが難しく，対応に時間がかかる問題がある．

以上の研究背景を踏まえ次の 3 点を研究課題とする．

RQ1: バグとファイルの関係を可視化

RQ2: バグ影響の分析方法の提案

RQ3:提案方法の有効性，妥当性評価

2. 関連研究

(1) プログラム内におけるバグと修正の偏り

Hata らは Java プログラム内におけるバグの予測方法を提案している．この中で変更履歴からバグの偏りについても議論している[2].

(2) グラフモデル OSS コミュニティ構造の特徴量分析

KatoらはOSSコミュニティの開発者の行動に関する特徴量を獲得し，その進化構造の分析を提案している[3].

3. アプローチ

OSS バグの影響を分析するためにはバグとそれを修正したファイルとの関係を含む大局的な分析が必要である．バグと修正されたファイルをプロパティグラフ[4]でモデル化し，開発期間ごとにバグのファイル修正への影響を分析することを可能とする(図 1).

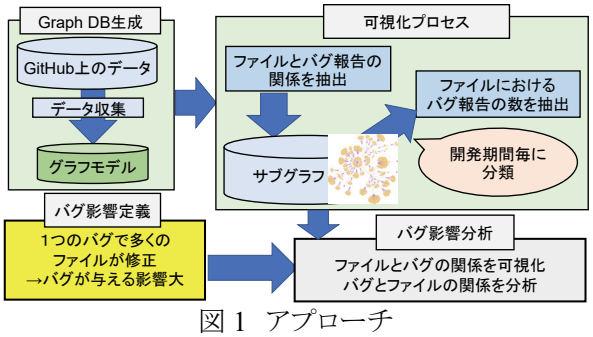

図 1 アプローチ

An Impact Analysis Method of OSS Bugs Using Graph Model and its Evolutions

† Shinnosuke Hirooka, Yui Imahori, Ryusuke Yoshida, Mikio Aoyama

†Department of Software Engineering, Nanzan University

4. 提案方法

提案方法は図 2 に示す 6 プロセスから成る．

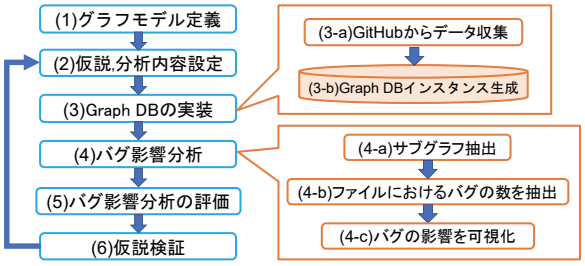

図 2 提案方法

4.1 バグ影響定義

バグの影響をバグ影響度の偏りとファイルの修正集中と定義する．

4.1.1 バグ影響度の評価方法

バグの影響度をバグによって修正されたファイル数の割合をとして，式(1)で定義する．パレートの法則をもとにバグ影響度の偏りを表 1 に示す2つの基準で評価する．

$$\text{バグ N の影響度} = \frac{\text{バグ N によって修正したファイル数}}{\text{開発期間に修正したファイルの総数}} \quad (1)$$

表 1 バグ影響度の偏り分類基準

基準	意味
80/20	8 割以上のファイル数が 2 割未満のバグ数で修正
60/40	6 割以上のファイル数が 4 割未満のバグ数で修正

4.1.2 ファイルの修正集中評価方法

ファイルが修正された回数が偏っていることは特定のモジュールがバグの原因となっている．時系列に沿って開発期間毎のファイルの修正回数を分析する．

5. プロトタイプの実装

本稿の提案方法が GitHub 上の OSS で有効であるか評価するためにプロトタイプを実装する(図 2).

OSS のデータは GitHubAPI[1]を用いて収集する．

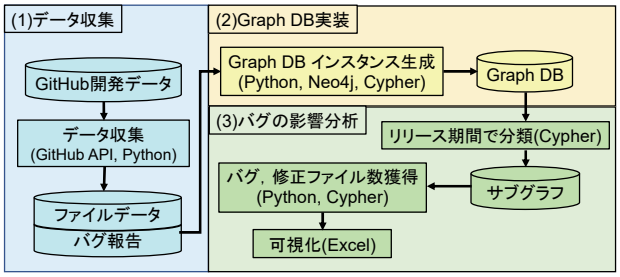

図 2 プロトタイプの構成

6. OSS 機械学習フレームワークへの適用

提案方法を OSS 機械学習フレームワークである Chainer に適用した(図 3). 適用期間を GitHub 上のリリース情報に基づき次の 4 つの期間に分割した. (1)2018/4/17～10/24, (2)2018/10/25～2019/5/15, (3)2019/5/16～12/4,(4)2019/12/5～2020/7/30

期間ごとにサブグラフを生成した(図 4). 多くのファイルを修正しているバグや修正が集中しているファイルを確認することができた. 4 期目は開発が終了しているためバグの報告が他の期間と比較して少なかった.

図 3 Chainer の GraphDB 生成結果

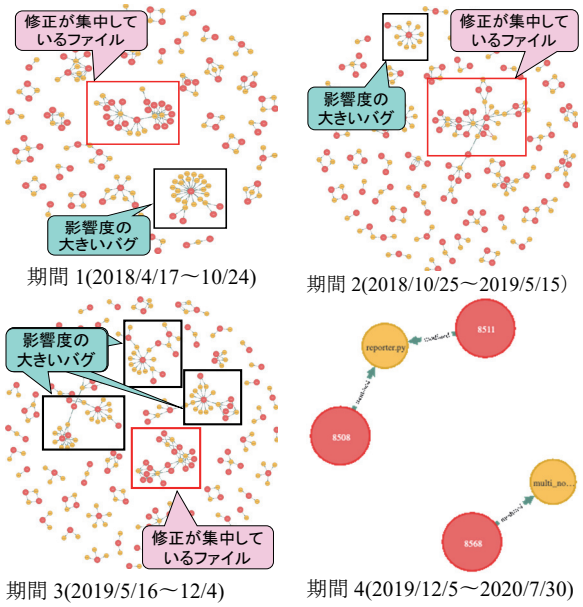

図 4 Chainer のバグと修正ファイルの関係

バグ影響分析結果を示す(図 5, 図 6). 期間 4 はデータが少ないため対象外とした.

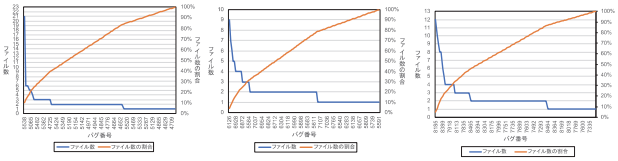

図 5 バグ影響度の偏り分布

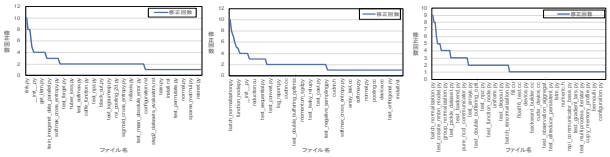

図 6 ファイル修正回数の分布

7. 評価

全ての期間でバグ影響度は 80/20 の基準を満たすことはなかったが 60/40 の基準は全ての期間が満たしていた. また, batch_normalization.py のファイルに修正が集中していることが明らかとなり, 期間 2 と期間 3 では修正回数が共に最多であった. Chainer では機能テスト用のファイルがあり, テスト対象ファイルと同頻度で修正されていることも明らかとなった. このことからバグ修正が特定のファイルに偏っていることが明らかとなった.

8. 考察

本稿の提案方法が有効であるか考察する.

(1) バグ影響分析方法

グラフモデルを用いたことによりバグとファイルの関係をグラフとして可視化可能となった. さらに, 特定の期間でスライスしたサブグラフを作成することで関係の時間的変化の分析が可能となった. 分析結果よりバグ影響度に偏りがあることや修正が特定のファイルに集中していることが明らかとなった. このことから, 特定のファイルに着目してバグ修正の効率化が期待できる.

(2) グラフを用いたバグ分析

バグ分析にグラフモデルを用いたことによりファイル構造やバグと修正ファイルの関係の分析が可能となった. また, サブグラフを作成することにより局所的な特徴を分析することが可能である.

(3) 関連研究[2]との比較

関連研究[2]ではプログラム内のバグと修正の偏りを明らかにした. 本稿はシステム全体でのバグと修正ファイルの偏りを明らかとした点で意義がある.

9. まとめ

本稿ではバグとファイルの関係をグラフモデルを用いてバグの影響を分析する方法を提案した. バグとファイルの関係を可視化, 分析することによりバグの影響度や修正するファイルの偏りが明らかとなった.

10. 参考文献

[1] C. Dawson, et al., Building Tools with GitHub, O'Reilly, 2016.

[2] H. Hata, et al., Bug Prediction Based on Fine-Grained Module Histories, Proc. of ICSE 2012, IEEE Computer Society, Jun. 2012, pp. 200-210.

[3] S. Kato, et al., A Structural Analysis Method of OSS Development Community Evolution Based on A Semantic Graph Model, Proc. of COMPSAC 2018, IEEE Computer Society, Jul. 2018, pp. 292-297.

[4] I. Robinson, et al., Graph Databases: New Opportunities for Connected Data, 2nd ed., O'Reilly, 2015.