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

2017SE015 広岡 伸之甫　2017SE020 今堀 由唯　2017SE111 吉田 隆佑

指導教員　青山 幹雄

1. 研究背景と課題

1.1 研究背景

OSS(Open Source Software)開発では，参画する人数や規模が大きいことが多い．OSS 開発では，様々な人が変更を加えたり，バグの報告や修正，機能追加や変更を要求したりすることが可能だからである．参画する開発者全員が自由に変更を加えられるのが OSS 開発の大きなメリットである．

OSS 開発の参画者の多くはバグの報告や修正を行っている．中途参画者はバグの修正をする際に規模が大きいとバグの修正を効率よくするのは難しい．開発に初めから携わっている参画者の場合，バグが報告された段階で開発の全貌を把握しているため，迅速に対応できる．しかし中途参画者では全貌を把握していないため，修正に時間がかかると考える．1 つのバグを修正するのに初期参画者に比べて中途参画者のほうがバグ修正へのコストが高いとわかる．

1.2 研究課題

本稿では研究背景を踏まえ以下の 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

2.1 グラフデータベース(Graph DB)

Graph DB はグラフ構造を管理できるデータベース管理システムである．「ノード」，「エッジ」，「プロパティ」の 3 要素によってノード間の関係を表現する「グラフ型のデータモデル」を持つデータベースである．

2.2 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[1].

研究にて開発された予測モデルにて大規模なバグの発見に成功した．分析対象は Java プログラムのため，プログラムに応じてバグの量に偏りがあることを明らかにした．

2.3 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[2].

目的として，開発者に着目し，行動履歴から成長パターンを分類，3層で構成されるコミュニティ進化モデルを明らかにした．

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[3]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).

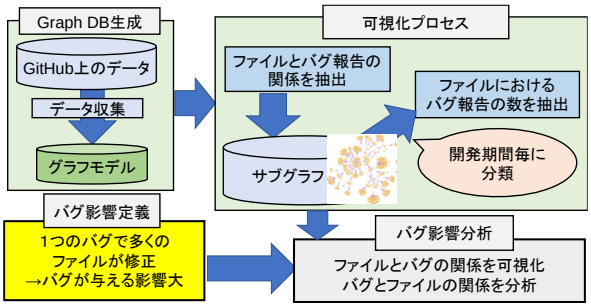


図 1　アプローチ

4. 提案方法

提案方法は次の 6 つのプロセスから成る．(図 2)

(1) グラフモデル定義
ノード，エッジ，それぞれのプロパティを定義する．

(2) 仮説，分析内容設定
設定した仮説に基づいて仮説ごとに明らかにする分析項目を設定する．

(3) Graph DB の実装
Graph DB 実装は (3-a) GitHub からデータ収集，(3-b) Graph DB インスタンス生成の2つのプロセスから成る．

(4) バグ影響分析
バグ影響分析は(4-a)サブグラフ抽出，(4-b)ファイルにおけるバグの数を抽出，(4-c)バグの影響を可視化の 3 つのプロセスから成る．

(5) バグ影響分析の評価
可視化結果を評価する．

(6) 仮説検証
分析で得られた結果から，設定した仮説を検証する．必要に応じ，得られた結果に基づいて，仮説の追加や変更を行い，一連のプロセスを繰り返す．

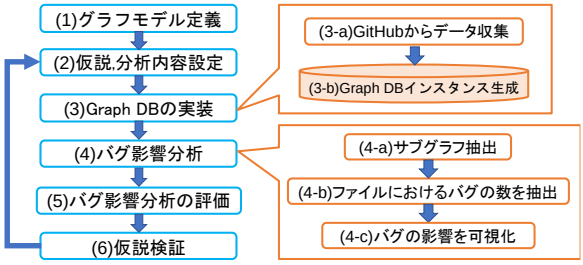


図 2　提案方法

4.1 グラフモデル定義

分析対象は OSS 機械学習フレームワークとする．バグ，ファイル，ディレクトリをグラフモデルとして定義する．

4.1.1 ノード定義

バグ，ファイル，ディレクトリをノードとする．ノードのプロパティ定義を表 1 に示す．

表 1　ノードのプロパティ定義

名称	定義
name	ファイル名 or ディレクトリ名
num	バグの番号
created	バグが報告された日時
closed	バグが解決された日時
finder	バグの報告者

4.1.2 エッジ定義

エッジ定義を以下の表 2 に示す．

表 2　エッジ定義

名称	定義
has	ファイルの所属関係
modified	ファイルの修正関係
added	ファイルの追加関係
removed	ファイルの削除関係
renamed	ファイルの名称変更関係

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して定義する．

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 3 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 3　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

5.1 実装環境

プロトタイプの実装環境を以下の表 4 に示す．

表 4　実装環境

コンポーネント	名前	バージョン
OS	macOS Catalina	10.15.7
Graph DBMS	Neo4j Desktop	1.3.4
クエリ	Cypher	-
データ取得，変換	Python	3.9.1
利用する外部 API	GitHub API	V3
データ可視化ツール	Excel	15.33

5.1 プロトタイプの構成

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)．

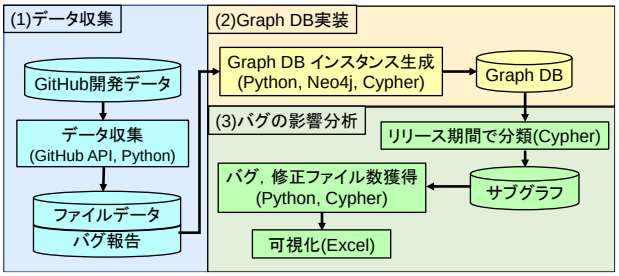


図 2　プロトタイプの構成

(1) データ収集
GitHub から GitHubAPI を用いてバグとファイルに関するデータを収集する．データ収集に用いるプログラムは Python で実装する．

(2) Graph DB 実装
収集したデータを Python で成型する．成型したデータから Graph DB インスタンスを生成し，Graph DB に格納する．

(3) バグの影響分析
サブグラフからバグ，修正ファイル数を獲得し Excel を用いて可視化，分析をする．

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 と PyTorch Geometric に適用した(図 3，図 4,図 5)．各グラフの構成要素を表 5 に示す．

表 5　グラフの構成要素

適用対象	Chainer	PyTorch Geometric
ノード数(blob)	1,752	557
ノード数(tree)	233	56
ノード数(bug)	1,305	1,973
エッジ数	3,981	3,175

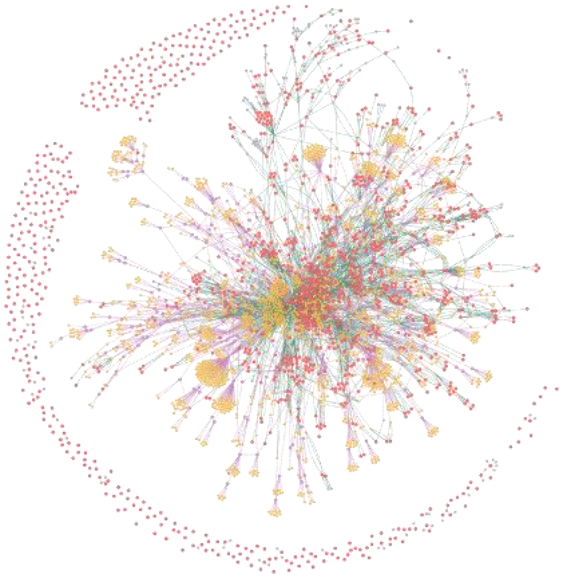


図 3　Chiner GraphDB 作成結果

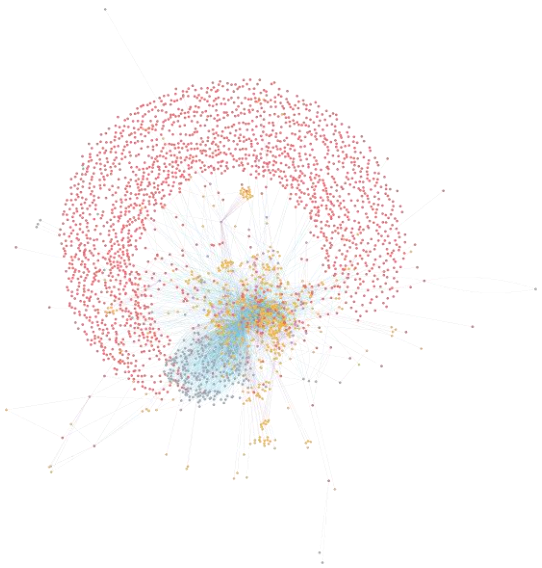
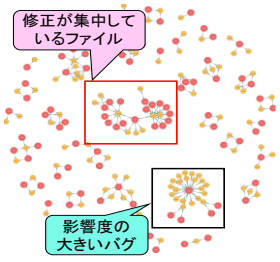


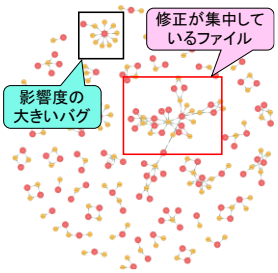
図4 PyTorch Geometric GraphDB 作成結果

図 5 PyTorch Geometric GraphDB 作成結果
（関係のあるノードのみ）

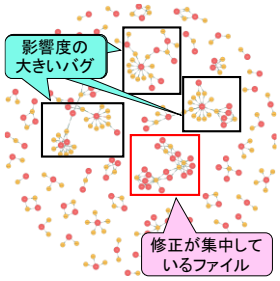
期間ごとに Chainer のサブグラフを生成した(図 6). 多くのファイルを修正しているバグや修正が集中しているファイルを確認することができた. 4 期目は開発が終了しているためバグの報告が他の期間と比較して少なかった.



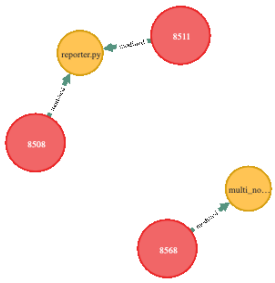
期間 1(2018/4/17～10/24)



期間 2(2018/10/25～2019/5/15)



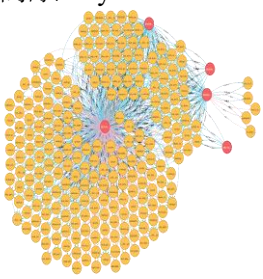
期間 3(2019/5/16～12/4)



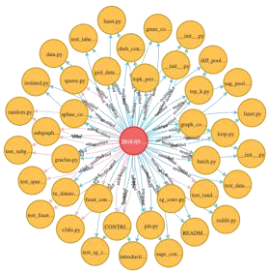
期間 4(2019/12/5～2020/7/30)

図 6 Chainer のバグと修正ファイルの関係

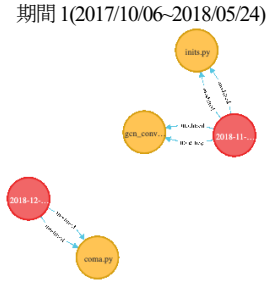
期間毎に PyTorch Geometric のサブグラフを作成した(図 7).



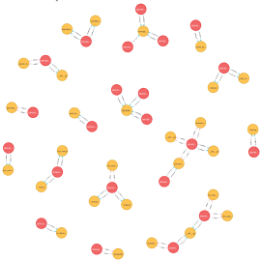
期間 1(2017/10/06~2018/05/24)



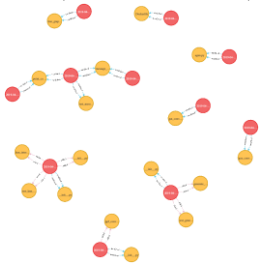
期間 2(2018/05/25~2018/08/13)



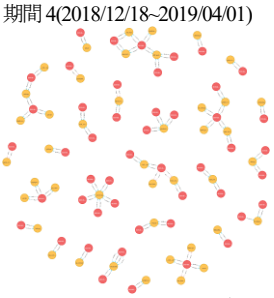
期間 3(2018/08/13~2018/12/18)



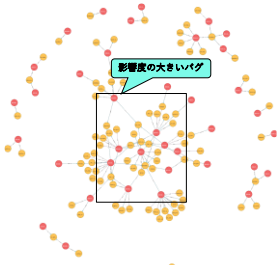
期間 4(2018/12/18~2019/04/01)



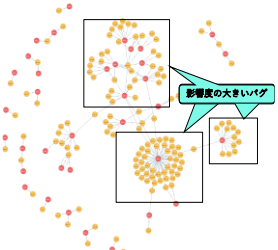
期間 5(2019/04/01~2019/04/29)



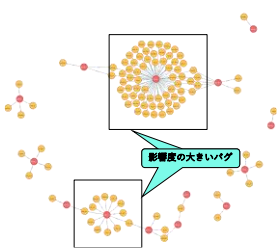
期間 6(2019/04/29~2019/06/29)



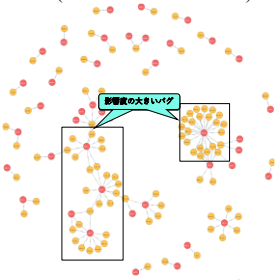
期間 7(2019/06/29~2020/02/04)



期間 8(2020/02/04~2020/05/25)



期間 9(2020/05/25~2020/07/07)



期間 10(2020/07/07~2020/12/02)

図 7 Chainer のバグと修正ファイルの関係

Chainer のバグ影響分析結果を示す(図 8, 図 9). 期間 4 はデータが少ないため対象外とした.

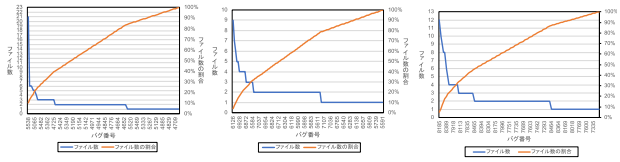


図 8　Chainer のバグ影響度の偏り分布

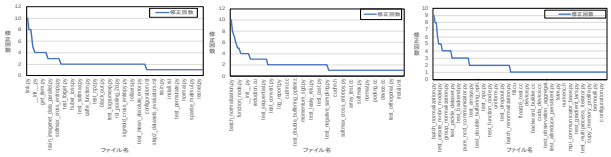


図 9　Chainer のファイル修正回数の分布

PyTorch Geometric のバグ影響分析結果を示す(図 10, 図 11).

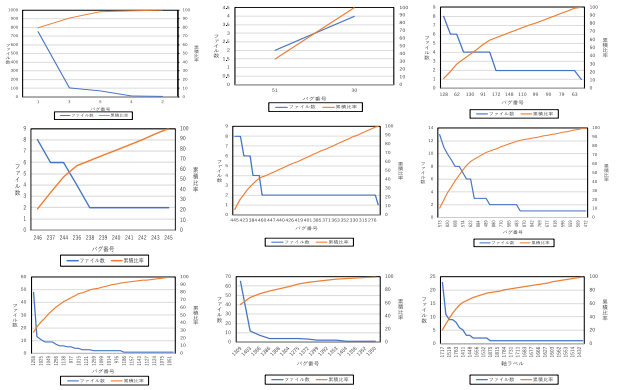


図 10　PyTorch Geometric のバグ影響度の偏り分布

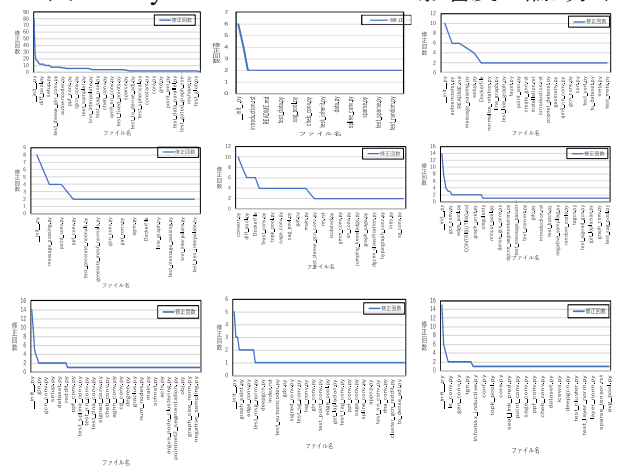


図 11　PyTorch Geometric のファイル修正回数の分布

7. 評価

7.1 Chainer

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. 全ての期間で影響度は類似した曲線を描いている. このことから修正したファイル数が同じバグの分布が各期間を通して類似していることが推測できる. 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. ファイル名からバッチ正規化の機能であると思われる. したがって期間 2 からバッチ正規化に関する修正が多く加えられていることが推測できる. また 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.

7.2 PyTorch Geometric

期間 1, 2 ではバグ影響度 80/20 の基準を満たした.期間 4, 5, 7, 8, 9, 10 では 80/20 の基準は満たさなかったが 60/40 の基準を満たした. 期間 3, 6 は基準を満たさなかった.多くの期間でファイルに与える影響が大きいバグが存在することが確認できた.

期間ごとのバグ番号を確認すると初期から中期ではバグ番号が連続していないことが確認できる. またバグに繋がっているファイルの数も少ない. このことからグラフデータベース作成結果のときに確認できたグレーのノードである消されたファイルと繋がっていたバグが多数あることが推測できる.

全ての期間でファイルの修正集中の偏りを確認することはできなかった.

ファイル修正回数の分布図では__init_.py のファイルの変更回数が最多であるという結果が確認できた. 異なるディレクトリに同じ名前のファイルが存在する. そのため, サブグラフではファイルの修正集中の偏りは確認できなかったがファイル修正回数の分布図では修正が集中しているファイルが存在することが確認できる結果となった.

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

[2]. 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.

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