グラフモデルを用いた OSS バグの影響の可視化手法

今堀 由唯[†1]　広岡 伸之甫[†1]　青山 幹雄[†1]

概要：OSS 開発では，様々な参画者がバグの報告や修正，機能追加や変更を要求する．開発の全貌を把握していない参画者がバグの修正を行う際のコストは一般に高く，プロジェクトへの参画に対する障壁となっている．本研究ではグラフモデルを用いてバグとファイルの関係を可視化し，バグの影響を提示する方法を提案する．バグとファイルの関係を可視化するグラフモデルにはプロパティグラフを用いる．バグとファイルとの間の関係に着目し，作成したグラフから，バグの影響度と，ファイルへの修正の集中度を表示する．その結果影響度の大きいバグ，修正回数の集中しているファイルを容易に知ることができる．提案方法を Chainer，PyTorch Geometric に対して適用した．提案手法により，全貌を把握していない参画者に対する OSS 開発への参画支援の実現が期待できる．

キーワード：グラフデータベース，OSS 開発，バグ影響

Visualizing the Impacts of OSS Bugs Using a Graph Model

YUI IMAHORI[†1]　SHINNOSUKE HIROOKA[†1]　MIKIO AOYAMA[†1]

1. はじめに

OSS 開発では，様々な参画者がバグの報告や修正，機能追加や変更を要求する．一般に 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．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．しかし新規・中途参画者などでは全貌を把握していない参画者にとって，修正のコストは高く，プロジェクトへの積極的な関与や貢献にとって障壁となっている[11]．本研究ではファイルとその時期のバグ報告に着目しグラフモデルを用いてバグとファイルの関係を可視化する方法を，バグの全貌を把握しやすくするために提案する．

2. 研究課題

本研究では，以下の 2 点を研究課題として設定する．

(1)　ファイルと報告されたバグとの関係を可視化するためのグラフモデルの構築

(2)　可視化したグラフ表現から，ファイルとバグの関係を開発期間に対する偏りとして可視化．

†1 南山大学 理工学部 ソフトウェア工学科
Dept. of Software Engineering, Nanzan University

3. 関連研究

3.1 では本研究で使用するグラフモデルについて述べ，3.2 で本研究と同様の目的を持つ既存研究を紹介する．

3.1 グラフモデル[8]

3.1.1 プロパティグラフ

プロパティグラフはノードとエッジ，プロパティから構成される．エンティティをノードでエンティティ間の関係をエッジで表現する．プロパティをグラフのノードおよびエッジに簡単に関連付けることができ，大量のデータセット間の関係に基づく分析，操作が可能になる．

3.1.2 グラフデータベース

Graph DB (Graph Data Base) 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．

本研究には Neo Technology 社によって開発された Neo4j[5]を用いる．

3.2 BugMap[2]

Gong らは，バグの位置情報を地形図を用いて可視化する BugMap を提案した．BugMap はソースコードとバグを入力とし、バグの位置を地形図上に表すことができる。地形図上に等高線を表示することでバグの数の大小を表している．

この結果，ソースコード全体の品質状態を確認することができる．

4. アプローチ

OSS バグの影響を分析するためにはバグと修正されたファイルの関係を含めた大局的な分析が必要となる．バグと修正されたファイルの全体像をプロパティグラフでモデル化する．バグとファイルの関係をグラフモデルで可視化することでファイルに対するバグの偏り，バグに対するファイルの数を確認することが可能となる．また，開発期間ごとにバグのファイル修正をサブグラフで可視化することを可能とする．ここでサブグラフとは期間ごとの可視化結果である．

ソフトウェアの不具合には一部の要因に伴う不具合は 8 割を占めるというパレートの法則と呼ばれる特性があることがわかっている[4]．これはソフトウェアの不具合を修正する際に特定のバグやファイルに着目する 1 つの指標になると考える．パレートの法則に基づき期間ごとの修正されたファイルの総数と 1 つのバグによって修正されたファイルの数を折れ線グラフで可視化する． パレートの法則で示される偏りの基準を折れ線グラフ上に表示することにより，偏り具合が読み取れる．

Graph DB 生成では，GitHub 上のデータを用いて Graph DB の作成を行う[6][9]．可視化プロセスでは，バグとファイルの関係の可視化を行う．

5. 提案方法

5.1 バグ影響グラフモデル

バグの影響を可視化するために，対象とする機械学習フレームワークをもとにグラフモデルを生成する．本研究では，GitHub から開発データを取得すると仮定している．図 1 にグラフモデルを示す．以下の各節では，各要素の定義について説明する．

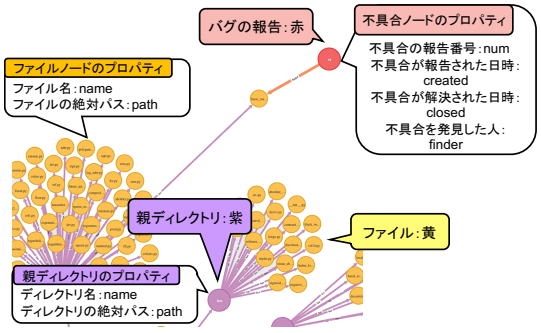


図 1 グラフモデル
Figure 1 Graph Model

5.1.1 ノードの定義

ノードは，対象となるオブジェクトで定義する．バグとディレクトリ，ファイルの関係を分析するために必要となる．本研究ではバグとディレクトリ，ファイルをノードとした．表 1 にノードの定義を示す．

bug ノードはバグと定義し GitHub 上に報告されている issue から取得する．Chainer では issue にある bug タグを bug とする．PyTorch Geometric については bug タグがつけられていなかったため issue 全部を対象とした．tree ノードはディレクトリと定義し blob ノードはファイルと定義し name のプロパティを持つ．

表 1 ノードの定義
Table 1 Definition of Nodes

ノード	定義	プロパティ
bug	バグ	num, created, closed, finder
tree	ディレクトリ	name
blob	ファイル	name

各ノードに対してプロパティを定義する．任意のノードには，1 個以上のプロパティのセットが関連付けられる．プロパティの定義を表 2 に示す．name をファイル/ディレクトリの名前，num をバグの報告番号，created をバグが報告された日，closed をバグが解決された日，finder をバグの報告者と定義し，各ノードに付与する．

表 2 プロパティの定義
Table 2 Definition of Properties

プロパティ	定義
name	ファイル名/ディレクトリ名
num	バグの報告番号
created	バグが報告された日
closed	バグが解決された日
finder	バグの報告者

5.1.2 エッジの定義

エッジは，内在関係で定義する．表 3 にエッジの定義を示す．has はディレクトリとファイルの関係を表す．バグとファイルの関係を表すエッジが modified，added，removed，renamed である．modified はファイル修正，added はファイル追加，removed はファイル削除，renamed はファイルの名称変更の関係と定義する．なお，エッジはプロパティを持たないものとしている．

表 3 エッジの定義
Table 3 Definition of Edges

エッジ	定義
has	ディレクトリとファイルの関係
modified	ファイル修正の関係
added	ファイル追加の関係
removed	ファイル削除の関係
renamed	ファイルの名称変更の関係

5.2 バグの影響度評価方法

開発期間毎のバグに対するファイルの割合を次の式で定義する．

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

縦軸にファイル数と累積比率，横軸にバグ番号をとるパレート図を作成しバグの偏りを観測する．表 4 にバグの偏りの評価方法を示す．偏りの基準はパレートの法則を用いて算出する．

表 4 バグ影響度の偏り分類基準
Table 4 classificatory criteria of bias of impact bugs

基準	定義
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

PyTorch Geometric はバグの取得が難しかったため issue 全体を対象としている．そのため，バグの影響度とは呼ばず issue の影響度と呼ぶこととする．

6. 実装

6.1 実装の目的

本研究の提案方法でバグとディレクトリ，ファイルの関係を可視化し，バグにおけるファイル数，ファイルにおけるバグ数を分析することでバグ影響分析が可能か評価するためにプロトタイプを用いる．

6.2 構成

本研究で提案されている分析方法に基づいて作成したプロトタイプ構成図を図 2 に示す．

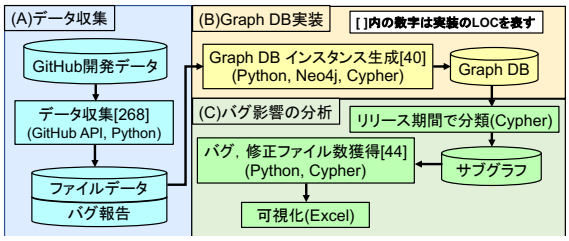


図 2 プロトタイプの構成

Figure 2 Prototype Configuration

(A) データ収集

GitHub から GitHub API を用いてバグとファイルに関するデータを収集する．GitHub に上がっている issue 情報をバグとしデータ収集を行う．Chainer は issue の中に bug タグが存在するので，bug タグのみを取得する．データ収集に用いるプログラムは Python で実装する．

(B) Graph DB 実装

収集したデータを Python で成型する．Neo4j，Cypher を用いて成型したデータから Graph DB インスタンスを生成し DB に格納する．

(C) バグ影響分析

サブグラフからバグ，修正ファイル数を獲得し Excel を用いてバグ影響度偏り分布を可視化する．

表 5 にプロトタイプを実現するためのデータ収集，可視化の実装環境を示す．

表 5 実装環境

Table 5 Implementation Environment

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

7. OSS プロジェクトへの適用と評価

本研究の提案方法が実際の OSS プロジェクトに対して有効であるかを検証するために機械学習フレームワークの OSS プロジェクトを対象に分析を行う．対象とする OSS プロジェクトは Chainer と PyTorch Geometric とした．Chainer は近年の OSS プロジェクトの中で盛んに開発が行われ，プロジェクトが終了している機械学習フレームワークとして選択した．PyTorch Geometric は Chainer の移行先である PyTorch の機械学習の拡張機能として選択した．

分析対象と分析期間を表 6，表 7 に示す．分析期間は GitHub 上で公開されているリリースノートに基づき作成する．

表 6 分析対象データ(Chainer)

Table 6 Analysis Target Data (Chainer)

分析対象	リリース	Chainer
期間 1	v4.0.0~v4.5.0	2018/04/17～2018/10/24
期間 2	v5.0.0~v5.4.0	2018/10/25～2019/05/15
期間 3	v6.0.0~v6.6.0	2019/05/16～2019/12/04
期間 4	v7.0.0~v7.7.0	2019/12/05～2020/07/30

表 7 分析対象データ(PyTorch Geometric)

Table 7 Analysis Target Data (PyTorch Geometric)

分析対象	リリース	PyTorch Geometric
期間 1	v0.1.1	2017/10/06~2018/05/24
期間 2	v0.1.1~v0.3.0	2018/05/25~2018/08/13
期間 3	v0.3.0~v1.0.0	2018/08/13~2018/12/18
期間 4	v1.0.0~v1.1.0	2018/12/18~2019/04/01
期間 5	v1.1.0~v1.2.0	2019/04/01~2019/04/29
期間 6	v1.2.0~v1.3.0	2019/04/29~2019/06/29
期間 7	v1.3.0~v1.4.1	2019/06/29~2020/02/04
期間 8	v1.4.1~v1.5.0	2020/02/04~2020/05/25
期間 9	v1.5.0~v1.6.0	2020/05/25~2020/07/07
期間 10	v1.6.0~v1.6.3	2020/07/07~2020/12/02

8. 適用結果

8.1 プロパティグラフへの適用

8.1.1 Chainer

図 3 に Chainer の Graph DB の作成結果を示す．赤色のノードがバグを，紫色のノードがディレクトリを，黄色のノードがファイルを示す．赤色の枠で囲んだファイルと関係のないバグが確認された．また中心に赤色のノードが集中していることからバグが集中している箇所が確認できる．

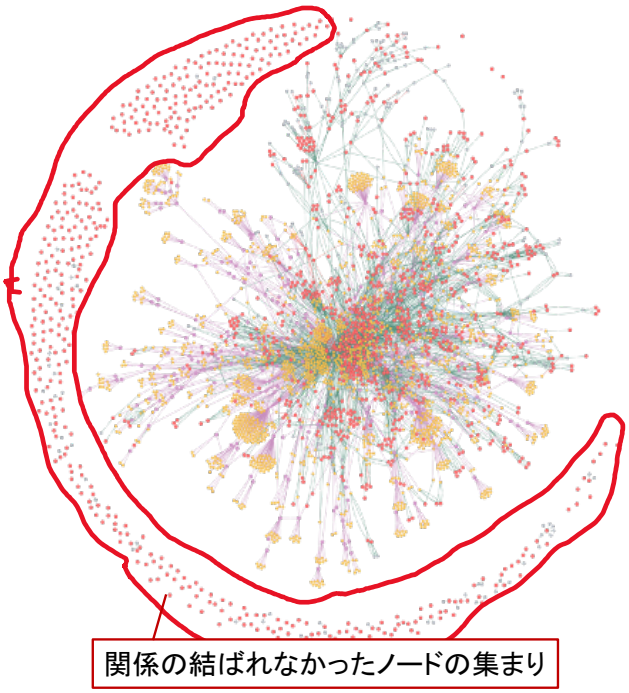
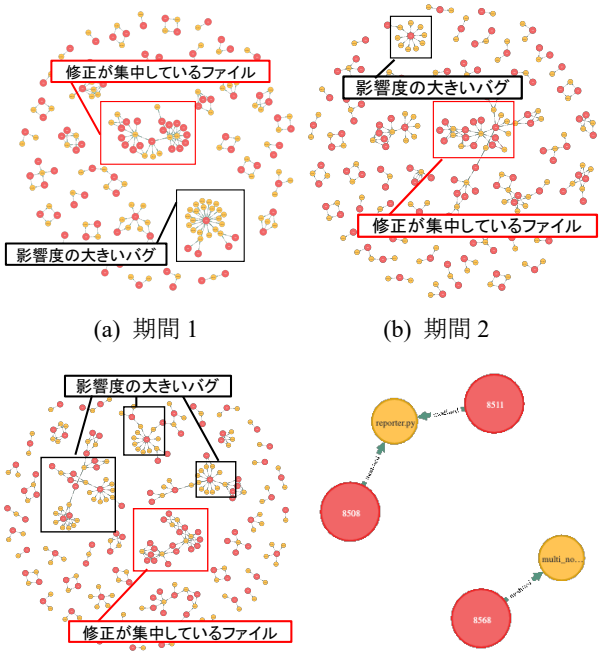


図 3 Graph DB 作成結果(Chainer)
Figure 3 Graph DB(Chainer)

期間 1 から期間 4 のサブグラフ可視化結果を図 4 に示す．期間 1 から期間 3 では黒色の四角で示す複数のファイルと関係をもつバグの存在を確認することができた．また赤色の四角で示す複数のバグと関係をもつファイルの存在を確認することができた．

期間 4 では Chainer の開発が終了したため，バグの報告が他の期間より少なかった．



(a) 期間 1　(b) 期間 2

(c) 期間 3　(d) 期間 4

図 4 サブグラフ(Chainer)
Figure 4 Sub Graph (Chainer)

8.1.2 PyTorch Geometric

図 5，図 6 にグラフデータベースの作成結果を示す．図 5 は 2021 年 1 月 5 日までの issue とディレクトリ，ファイルの全てのノードとエッジを示す．図中の赤色の枠の中が関係の結ばれなかった issue のノードを表す．この可視化結果から issue として報告されているが修正されていないものが多くあることがわかる．

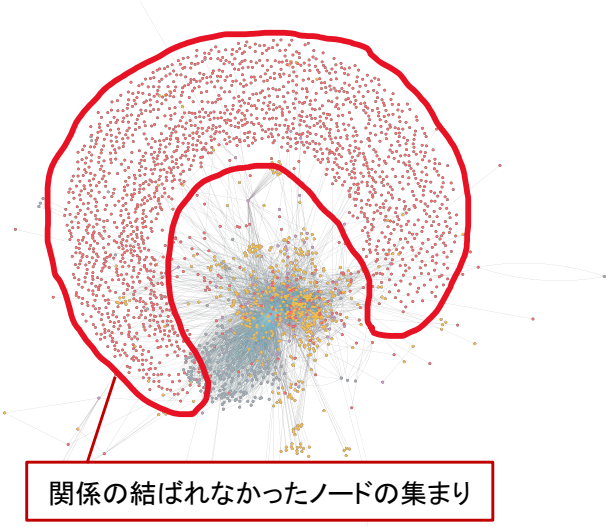
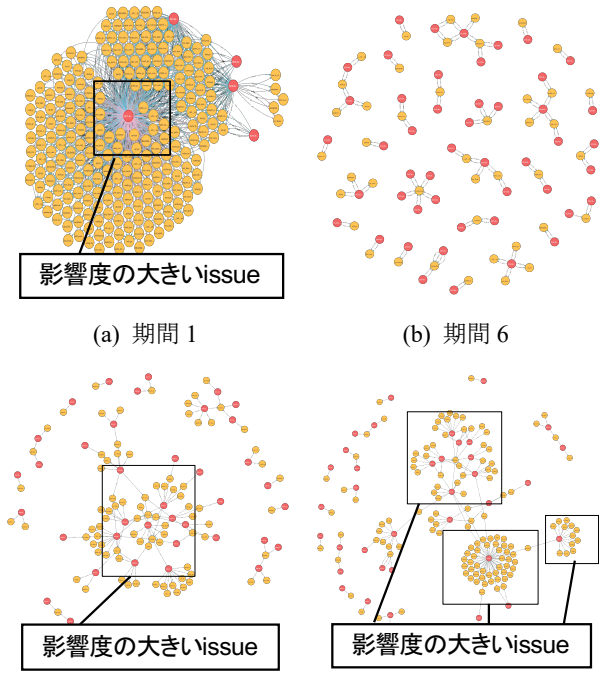


図 5 Graph DB 作成結果(PyTorch Geometric)
Figure 5 Graph DB (PyTorch Geometric)

期間 1 と期間 6，期間 7，8 のサブグラフ可視化結果を図 6 に示す．期間 1 と期間 7，期間 8 では複数のファイルと関係をもつ issue の存在を確認することができた．



(a) 期間 1　(b) 期間 6

(c) 期間 7　(d) 期間 8

図 6 サブグラフ(PyTorch Geometric)
Figure 6 Sub Graph (PyTorch Geometric)

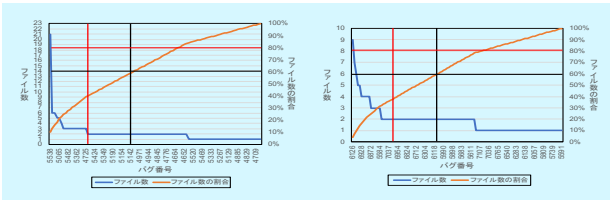
8.2 バグ影響度の偏り

バグ影響度の偏りをパレートの法則に基づきグラフを作成する．折れ線グラフの青色の線が修正されたファイルの数を表し，オレンジ色の線が累積比率を表す．グラフ中に補助線として横線でファイル数の割合，縦線でバグの割合を示す．赤線の横がファイル全体の 8 割，縦がバグ全体の 2 割，黒線がファイル全体の 6 割，縦がバグ全体の 2 割で補助線をひいた．

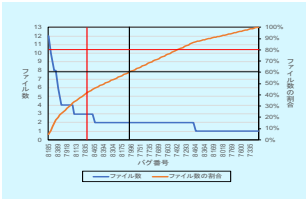
オレンジ色の折れ線グラフが赤色の補助線の交点か交点より右上にある場合 80/20 の基準を満たすことを示し，黒色の補助線の交点か交点より右上にある場合 60/40 の基準を満たすことを示す．

8.2.1 Chainer

Chainer のバグ影響度の偏り評価を図 7 に示す．赤色の線が 80/20 の基準を満たす線，黒色の線が 60/40 の基準を満たす線となっている．80/20 の基準を満たした期間をピンク色，60/40 の基準を満たした期間を青色，基準を満たさなかった期間を白色で示す．すべ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．Chainer は提案方法を適用した全ての期間でバグ影響度の偏りを確認することができた．また，全ての期間で影響度は類似した曲線を描いていた．



(a) 期間 1　(b) 期間 2



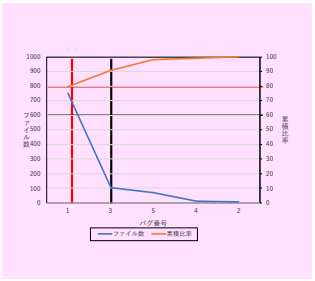
(c) 期間 3

図 7 バグ影響度の偏り (Chainer)
Figure 7 Bugs Impact Bias (Chainer)

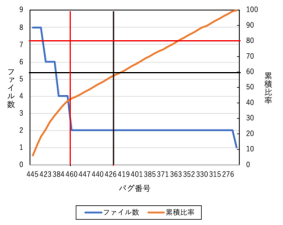
8.2.2 PyTorch Geometric

PyTorch Geometric はバグのみを取得することができなかったためバグ影響度ではなく，issue の影響度とよぶこととする．PyTorch Geometric の issue の影響度の偏り評価を図 8 に示す．赤色の線が 80/20 の基準を満たす線，黒色の線が 60/40 の基準を満たす線となっている．80/20 の基準を満たした期間をピンク色，60/40 の基準を満たした期間を青色，基準を満たさなかった期間を白色で示す．issue の影響の偏り分布はバグ 1 つからファイルに影響を与えていたため作成されなかった．期間 1，2 では issue の影響度 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．

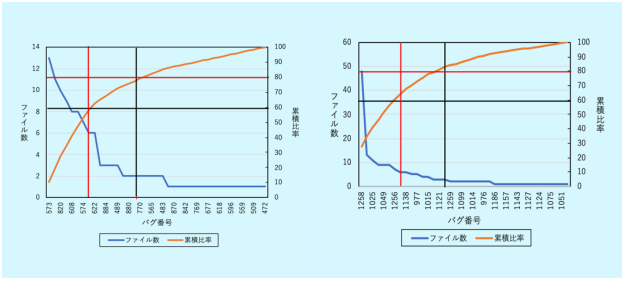
本稿では(a)に 80/20 の基準を満たした期間 1 のサブグラフ，(b)に基準を満たさなかった期間 6 のサググラフ，(c)，(d)に 60/40 の基準を満たした期間 7, 8 のサブグラフを提示する．



(a) 期間 1



(b) 期間 6



(c) 期間 7　(d) 期間 8

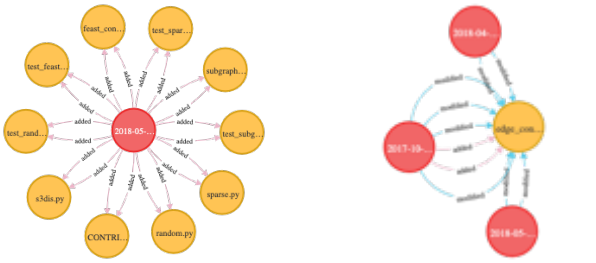
図 8 バグ影響度の偏り (PyTorch Geometric)
Figure 8 Bugs Impact Bias (PyTorch Geometric)

8.2.3 エッジのプロパティごとのサブグラフ

エッジのプロパティごとのサブグラフを図 9 に示す．(a)から(c)に PyTorch Geometric のエッジのプロパティごとのサブグラフの抽出結果を提示する．(a)はエッジのプロパティを指定せずに出力したサブグラフ，(b)はエッジのプロパティを modified で定義しているファイルの修正のみを出力したサブグラフ，(c)は added で定義しているファイルの追加のみで出力したサブグラフとなっている．(d)はファイルを基準にどのようなバグの影響があるかを出力しているサブグラフである．

(a) プロパティなし

(b) modified



(c) added　　(d) ファイル基準のバグ

図 9 エッジのサブグラフ

Figure 9 Sub Graph (Edge Properties)

プロパティなしの可視化結果と比較し，どのバグからどのファイルにどのような変更が行われているのかを確認することができた．また，変更の内容ごとに可視化することにより，変更回数の差の出力が可能になった．

ファイルを基準に変更を与えたバグの可視化を行うことでどのバグからどのような変更が行われたか，同じバグから複数の変更を加えていることが確認できた．

9. 評価

9.1 プロパティグラフへの適用

8.1 の可視化結果から全体のバグとファイルの関係を可視化すること，期間ごとのバグとファイルの関係をサブグラフで可視化すること，影響度の大きいバグや修正が集中しているファイルを確認することが可能となった．サブグラフで可視化することにより着目すべきバグやファイルの確認を容易にすることが可能となった．このことから，バグの全貌の把握や着目すべきバグやファイルの確認に有用である．

9.2 バグ影響度の偏り

8.2 の可視化結果から基準を満たした期間と満たさなかった期間を比較することでグラフに特徴があることが確認できる．基準を満たした期間のグラフは似たような形を取ることが確認でき，影響度の大きいバグが存在する類似したグラフを出力することが推測できる．

9.3 エッジのプロパティごとのサブグラフ

8.3 の可視化結果から期間ごとの全体のサブグラフだけでなくどのバグの影響でどのファイルにどのような変更を加えたのかを可視化することが可能になる．また，1 つバグに着目しどのようなファイルに影響を与えたかを確認すること，1 つのファイルに着目しどのようなバグから変更が行われたかを確認することが可能になり，この可視化方法は有用である．

10. 考察

10.1 バグ影響分析方法

図 3 から図 6 のように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モデルとして可視化することができる．さらに，図 5，6 のよう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る．可視化結果より多くの期間でバグとファイルの関係には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．特定のバグ，ファイルに着目することでバグ修正の効率化が期待できる．このことから，全貌を把握していない参画者が着目すべき特定のファイルやバグを見つけることに有用である．

10.2 プロパティグラフを用いたバグ分析方法

図 9 のように可視化にプロパティグラフ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る．また，サブグラフを作成することにより変更されたファイルや変更内等等局所的な特徴を可視化することが可能となる．

10.3 関連研究との比較

Hata らの研究では詳細なプログラム履歴に基づいたバグ予測を提案し，プログラム内のバグと修正の偏りを明らかにしている[3]． 分析対象として Java で書かれている 8 つの OSS を選出，既存の成果に比べて，多数，大規模，広範なバグの発見が可能であることを示した．

[3]の著者らは，この結果から，メソッドレベルのバグ予測がパッケージおよびファイルレベルのバグ予測より優れていることを示したのに対し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．

10.4 BugMap との比較

BugMap とグラフモデルを用いた可視化結果の比較を表 8 に示す．今回の比較ではバグの数，変更の内容，ファイルを基準としてみたバグの 3 項目で行う．

表 8 比較結果

Table 7 Compare the Results

	バグの数	変更の内容	ファイル基準
本研究	△	○	○
BugMap	○	△	—

バグの数は本研究では見た目や数える等で確認することは可能だが，BugMap では等高線や色を用いることで表現しているので一目で確認することが可能である．そのため，本研究よりも優れていると考える．変更の内容は本研究ではエッジに付与したプロパティによって出力が可能である，BugMap では詳細情報を確認することで変更の内容がわかるので本研究の方が優れていると考える．またグラフモデルによる可視化を行うことでファイルを基準としどの程度バグが影響しているかを確認することが可能である．その結果，本研究ではファイルを基準としたファイルに関連のあるバグのみを確認することが可能だが，BugMap では確認することができない．

グラフモデルとしてバグとファイルの関係を可視化する

ことにより着目すべきバグやファイルを確認できる点，局所的な特徴を可視化できる点で本研究は意義がある．

11. 今後の課題

今後の課題は以下の 2 点である．

(1) グラフモデルの改善

本研究ではグラフモデルを作成する際，バグとファイルの関係を分析するためバグのノードのプロパティのみに日付を付与している．エッジのプロパティへ日付を付与することでバグとファイルの修正履歴を可視化できると想定している．期間毎のファイルへのバグの影響可視化をする際はそれぞれのノード，エッジが持つプロパティを再度検討する必要がある．

(2) 消されたファイルを含めたバグ影響分析

本研究では現在のリポジトリに存在するディレクトリとファイル，バグの関係を可視化しバグ影響の可視化を行った．OSS 開発が進むと新しくファイルを作り以前まで使っていたファイルを消すことがあることがわかっている．期間毎の分析をする際，現在のディレクトリ，ファイルのみで分析するとバグ影響の可視化結果に偏りが生じるため消されたファイルを含めた可視化を検討する必要がある．

(3) issue からバグ修正のみを取り出す手法の適用

issue への報告は必ずしもバグ修正だけの内容ではない．PyTorch Geometric は bug のみの取得が難しかったため issue 全体に対して 5.2 章のバグ影響度を適用した．SZZ アルゴリズム[10]などを用いて，issue からバグに関する報告のみを抽出し，バグとファイルとの関係を求める手法を検討する必要がある．

12. まとめ

本研究ではバグとファイルの関係をグラフモデルを用いて可視化する方法を提案した．バグと修正したファイルの関係をグラフデータベース上で可視化することによりサブグラフを用いた特定の期間の分析が可能となった．分析結果よりバグの影響度の偏りとファイル修正の集中が明らかとなった．この分析により，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．

参考文献

[1] Chainer，https://github.com/chainer/chainer/.
[2] J.Gang and H.Zhang, BugMap: a topographic map of bugs, ESEC/FSE 2013，pp.647-650, 2013.
[3] H. Hata, et al., Bug Prediction Based on Fine-Grained Module Histories, Proc. of ICSE 2012, IEEE Computer Society, Jun. 2012, pp. 200-210.
[4] 河村 透 他，ソフトウェアのテスト管理システム，東芝レビュー，Vol.66,No.1,2011, pp. 32-36.
[5] Neo4j, https://neo4j.com/.
[6] 尾上 紗野 他, GitHub 上の活動履歴分析による開発者分類, 情報処理学会論文誌, Vol.56,No.2,Feb.2015, pp. 715-719.
[7] PyTorch Geometric，https://github.com/rusty1s/pytorch_geometric
[8] I. Robinson, et al., Graph Databases: New Opportunities for Connected Data, 2nd ed., O’Reilly, 2015 [木下 哲也(監訳), グラフデータベース，オライリー・ジャパン, 2015].
[9] 柴藤 大介， 矢谷 浩司， GitHub のプルリクエストを用いたプログラミング課題自動生成システムの実現可能性に関する検討，情報処理学会第 80 回全国大会, 2018 年 3 月, pp. 319-320.
[10] J. Sliwerski, et al., When do changes induce fixes? In Proceedings of the 2005 International Workshop on Mining Software Repositories (MSR2005), pp.24-28, 2005
[11] 吉行勇人 他，修正者のタスク優先順位付けが不具合修正時間に与える影響,ウィンターワークシップ 2014 年・イン・大洗論文集,情報処理学会,2014 年 1 月, pp.99-100.