Vol.2021-SE-209 No.2
2021/11/25

表現学習を用いたソフトウェア安定性の分析

可知 敬朗[1]　青山 幹雄[2]　野呂 昌満[2]　沢田 篤史[2]

概要：本研究は，モジュール依存に基づくグラフ表現によってソフトウェアを抽象化し，表現学習を用いてその特性を抽出，可視化することでソフトウェアの安定性を分析する方法の提案を目的とする．近年，OSS 開発は活発である一方，複数ユーザの参画により開発実態の把握が困難であるという問題がある．要因の一つとして，モジュールの変更時，依存関係にある他のモジュールに対する影響が明確でないことが挙げられる．本研究では，モジュールとその依存関係を基にソフトウェアをグラフとして抽象化し，グラフの構造変化の観点からソフトウェアを分析する．グラフに表現学習を適用し，ソフトウェアの特性を特徴ベクトルとして獲得，クラスタリングにより特徴ベクトルを分析し，計測対象の代表ベクトルを特定，クラスタと代表ベクトルの時系列変化を観測することにより，ソフトウェアの安定性の議論を行う．グラフデータベース Neo4j，グラフ表現学習 graph2vec を用いて提案方法を支援するプロトタイプを実装し，実際の OSS に対して提案方法を適用する．

キーワード：表現学習，安定性，OSS，グラフモデル，グラフデータベース

Analyzing Stability of Software Systems using Representation Learning

1. はじめに

OSS（Open Source Software）は，通常複数のユーザによって頻繁に変更が加えられ，その結果，ソフトウェアの構造が複雑化し，変更による影響の理解が困難になる．これを解決する一方法として，利用関係の観点からソフトウェア全体の構造を明らかにし，変更による影響を安定性として分析する研究がある [3], [4], [10].

ソフトウェアの依存関係は，呼出し（Call）関係や利用（Use）関係に着目したグラフモデルで表現できる．ソフトウェアを抽象化したグラフに対し，グラフ分析の方法を適用することで，グラフの構造変化を予測する観点から，ソフトウェアを分析することが可能である [2], [6]．これらの研究で用いられる方法としてはグラフの局所的な探索が多く，構造の大域的な特性の時間的推移を分析することは困難であった．

ソフトウェアの安定性は，ソフトウェア全体の時間的変化に伴って分析される [3], [4], [10]．これまでの局所的なグラフ分析方法では，大域的に，ソフトウェア全体を対象とする安定性の分析をすることが困難であった．

本稿では，モジュールとその依存関係の観点からソフトウェアをグラフとして抽象化し，大域的な特性を分析する新たな方法を提案する．グラフの特性を定量的な表現として獲得するため，グラフ表現学習による特徴ベクトルの獲得を行う．獲得されたベクトルをクラスタリングし，計測対象ソフトウェアの代表ベクトルを特定する．代表ベクトル，クラスタの時間的変化を観測することで，ソフトウェアの安定性の議論を行うことが可能になると考える．

この着想の妥当性を確認するために，本研究では次の 2 点を明らかにすることを研究課題とする．

RQ1: モジュールの利用関係を表す依存関係グラフモデルはどのように定義できるか

RQ2: 依存関係グラフの表現学習によりソフトウェアの安定性はどのように分析できるか

ソフトウェアのグラフへの抽象化について，ソフトウェアのモジュールとその依存関係をそれぞれノード，エッジとしてグラフモデルを定義する．分析対象ソフトウェアの各バージョンのグラフインスタンスを，グラフ管理データベース Neo4j[9] に格納する．グラフの特徴ベクトル獲得に

[1] 南山大学大学院理工学研究科ソフトウェア工学専攻
Graduate Program of Software Engineering, Nanzan University
[2] 南山大学理工学部ソフトウェア工学科
Dept. of Software Engineering, Nanzan University

は，グラフ表現学習 graph2vec[8] を用いる．ベクトルの分析については k-means 法を用いたクラスタリングを行う．各グラフインスタンスに対して特徴ベクトルの獲得と分析を行い，明らかとなったクラスタとその重心の時間的変化から安定性の議論を行う．詳細については 3 節，4 節にて述べる．本稿では，プロトタイプ実装のフレームワークの定義まで実施している．

2. ソフトウェアの安定性とその分析に関する課題

2.1 ソフトウェア安定性

ソフトウェアの安定性に関する研究は継続的に行われてきた [4], [10].

Fayad ら [4] は，変更に伴うソフトウェアの安定性について議論を行っている．変更によるソフトウェア全体の再設計を避けるため，ソフトウェアの変化の少ない部分にプロジェクト自体を合わせることで，安定したソフトウェアの設計を行うことが可能だと主張している．

Ramirez ら [10] は，アーキテクチャの観点からソフトウェアの安定性の定義を議論するために，複数の関連研究の調査を行った．その結果，ソフトウェアの安定性は様々なレベルや観点から測定されるものであり，リリースや進化再利用など，時間的変化を含む特性から明らかになる可能性があると述べている．

2.2 外部安定性と内部安定性

我々は，安定性が変更に起因するものであると捉える．この観点は，先行研究等でも広く了解されているものであり，その対象は，コードレベルからアーキテクチャ，要求レベルまで異なる抽象度にわたる．ここで，主要な課題の一つは，アーキテクチャに関連したソフトウェアの安定性である．Constantinou ら [3] は進化再利用の観点から，パッケージ利用関係の変化に基づき，アーキテクチャに関連する安定性について次の 2 つの尺度を定義した．

- **外部安定性 (ES: External Stability)**：
  ソフトウェアのある連続したバージョン間で，削除されたパッケージの個数と再利用されたパッケージの個数により算出される値であり，以下の数式で表現される．

$$ES(i, i+1) = ESR(i, i+1) * ESC(i, i+1)$$

  - $ES(i, i+1)$：ソフトウェアの $i, i+1$ バージョン間における外部安定性指標
  - $ESR(i, i+1)$：$i, i+1$ バージョン間におけるモジュール削除に関する尺度
  - $ESC(i, i+1)$：$i, i+1$ バージョン間におけるモジュール変更に関する尺度

  ES の取り得る値の範囲は $[0, 1]$ であり，0 の場合は全てのパッケージが削除された場合，1 の場合は全てのパッケージが再利用された場合を意味する．

- **内部安定性 (IS: Internal Stability)**：
  ソフトウェアのある連続したバージョン間で，パッケージ間の利用関係の削除と追加の個数により算出される値であり，以下の数式で表現される．

$$IS(i, i+1)$$

$$= \frac{\Sigma_{REL_i(P_j,P_k)\neq 0} \frac{PSA(i,i+1)+PSR(i,i+1)}{2}}{|REL_i(P_j, P_k), REL_i + 1(P_j, P_k) \neq 0|}$$

  - $IS(i, i+1)$：ソフトウェアの $i, i+1$ バージョン間における内部安定性指標
  - $PSA(i, i+1)$：$i, i+1$ バージョン間における利用関係の追加に関する尺度
  - $PSR(i, i+1)$：$i, i+1$ バージョン間における利用関係の削除に関する尺度
  - $REL_i(P_j, P_k)$：バージョン i におけるモジュール P_j，モジュール P_k 間の利用関係

  IS の取り得る値の範囲は $[0, 1]$ であり，0 の場合は前のバージョンの全ての利用関係が削除され新しい利用関係だけが次のバージョンに残っている場合を意味し，1 の場合は全ての利用関係が変更されていない場合を意味する．

これら 2 つの尺度はソフトウェアの安定性を定量的に示すために有効であり，Constantinou らは，外部安定性と内部安定性からソフトウェア全体の安定性が定義できるものとしている．これらの指標をもとに，ソフトウェアの進化構造の定量的評価についても言及している．

2.3 グラフモデルに基づくソフトウェアの依存性と変更影響の分析

ソフトウェアを構成するプログラムの依存関係は依存グラフ（Dependence Graph）でモデル化され，ソフトウェアの理解支援や静的解析の基礎として研究されてきた [2], [6].

Bohner[2] は，複雑な依存関係を持つ大規模ソフトウェアに対し，依存関係の可視化と変更影響分析（Change Impact Analysis）の技術を用いることで解析を行った．この解析を基に，相互運用性を組み込んだ依存関係の分析，また，分析からソフトウェアの変化のパターンを特定し，開発支援を実現することにも言及している．

Horwitz ら [6] は，コードをノード，制御をエッジとしてプログラムをプログラム依存グラフとして抽象化し，グラフ解析をすることで，言語の熟練度に依らないプログラムの理解支援ツールの実現について述べている．

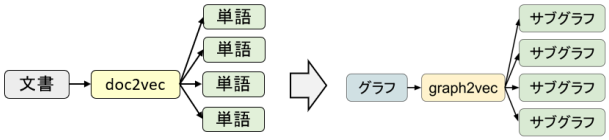

図 1 doc2vec と graph2vec の比較
Fig. 1 Comparison doc2vec and graph2vec

2.4 グラフ／ネットワーク表現学習

グラフ／ネットワーク表現学習（Graph/Network Representation Learning）[1], [5] とは，グラフ構造の特性を分散表現（特徴ベクトル）として獲得する方法である．代表的アルゴリズムとして graph2vec[8] がある．

graph2vec は，ニューラルネットワークを用いたグラフサンプリングを実現するためのシステムであり，word2vec や doc2vec と同様の設計思想を持つ．word2vec，doc2vec はそれぞれ，自然言語における単語と文書を特徴ベクトルとして獲得するシステムである．**図 1** に示すように，獲得されたベクトルはベクトル空間において，意味が近い単語や文書が近い位置に配置されるように生成される．graph2vec に関して，doc2vec における文書に相当するものがグラフ全体であり，文書中の出現単語に相当するものがサブグラフである．graph2vec は，サブグラフにラベルを割り当て，（グラフ）コーパスを生成することにより，word2vec や doc2vec と同様に Skip-gram モデルを適用する特徴を持つ．これらの設計思想により，意味的に類似したサブグラフはベクトル空間の近い位置に特徴ベクトルとして生成される．

Narayan ら [8] は，graph2vec で獲得された特徴ベクトルに対し，k-means 法に代表されるクラスタリングを適用する方法にも言及している．獲得された特徴ベクトルは，graph2vec の設計思想ゆえに，近い意味を持つもの同士がベクトル空間上でも近い位置に存在する．従って，クラスタリングによるクラスタの分離を行いやすいことから，有効な分類タスクが出来ることを示している．

2.5 グラフ表現学習による OSS コミュニティ分析

加藤ら [7] は，OSS 開発コミュニティにおける開発者，及びその活動を SCGM（Software Community Graph Model）として定義し，グラフ表現学習を用いて開発者の活動に関する特徴ベクトルを獲得している．

グラフに関して，開発者をノード，Commit や Merge に代表される開発者の活動をエッジとして定義し，複数期間のグラフインスタンスを生成している．開発者を開発に寄与するレベルによって中核的か非中核的か区別し，開発者のレベルとその活動をラベルとして学習を実施している．**図 2** に示すように，特徴ベクトルをクラスタリングした結果と，グラフの変遷を比較することにより，OSS 開発コミュニティにおける開発者の成長パターンとその進化構造を明らかにした．

加藤らは，開発者の活動の観点から OSS コミュニティの構造とその時間的変化について分析した．しかし，安定性を含むソフトウェアの特性，及びソフトウェアの構造の時間的変化については言及していない．

2.6 分析の問題点と技術課題

Constantinou らの定義によれば，ソフトウェアの安定性は外部安定性と内部安定性を分析することで明らかとなるとされており，外部安定性／内部安定性はそれぞれ ES 値／ IS 値という指標によって評価されるものとしている．ES 値／ IS 値は共に $[0, 1]$ として定量的に表現されるが，ここで，ソフトウェアの安定性を $[0, 1]$ の範囲で説明することが可能であるのかという疑問が生じる．

本稿では，ニューラルネットワークを用いてソフトウェアの時間的変化に関する特徴ベクトルを獲得することにより，Constantinou らの先行研究における $[0, 1]$ の範囲よりも，詳細な安定性の定量表現を定義する．特徴ベクトルの獲得について，ソフトウェアのグラフへの抽象化と，グラフ／ネットワーク表現学習を用いる．分析の結果を先行研究と比較することで，本稿における安定性が，$[0, 1]$ よりも詳細に表現されていることを示す．

以上より，本稿の研究課題を次のように設定する．

RQ1: モジュールの利用関係を表す依存関係グラフモデルはどのように定義できるか

RQ2: 依存関係グラフの表現学習によりソフトウェアの安定性はどのように分析できるか

RQ1 について，ソフトウェアのモジュールをノード，依存関係をエッジとしてグラフに抽象化し，各バージョン毎にグラフインスタンスをグラフ管理データベースに格納することで，バージョン間の変更も補完されることを目的として実装を行う．

RQ2 について，生成したグラフインスタンスに表現学習を適用し，特徴ベクトルを獲得，獲得したベクトルにクラスタリングを行うことでその特性の分析を行う．各バージョンのグラフインスタンスに対してこの処理を行い，特性の時間的変化を観測することでソフトウェアの安定性の議論を行う．

3. グラフモデルの表現学習を用いた安定性分析

本研究では，表現学習による特徴ベクトルの獲得，クラスタリングによる分析を経ることにより，ソフトウェア安定性の定量分析を行う．この方法により，Constantinou らの定義した外部安定性と内部安定性を，$[0, 1]$ の範囲よりも詳細に定量化することを目指す．ここで詳細に定量化するとは，1 次元ではなく多次元において数値の変動が観測で

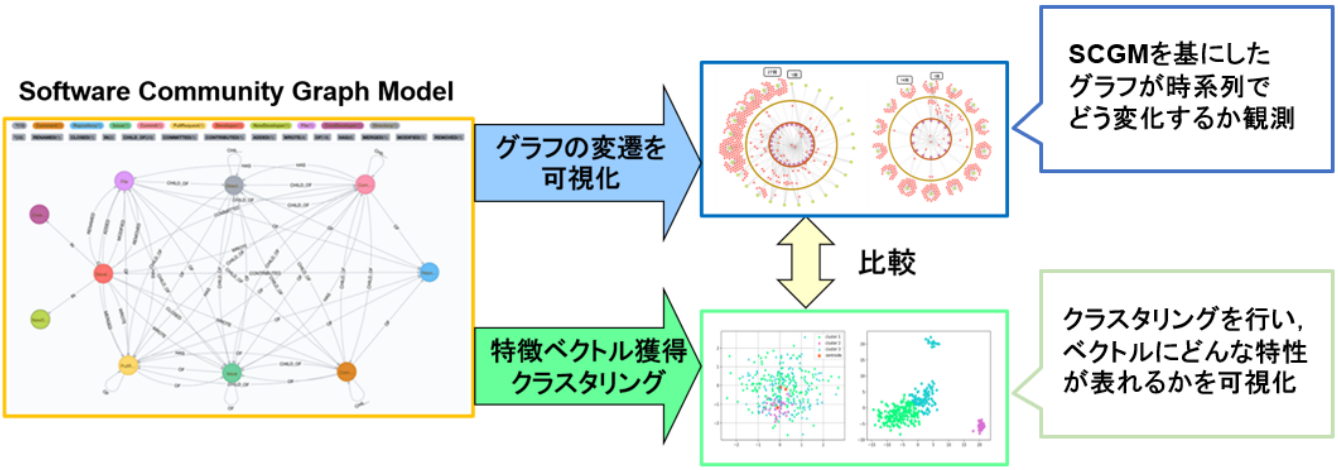

図 2 OSS コミュニティ分析
Fig. 2 Analysis of OSS community

きることである．

フレームワークは以下の通りである．

(1) 依存関係グラフモデル定義
ソフトウェアの変更の影響を安定性として分析するために，モジュールの利用関係に基づくグラフへの抽象化を行う [3], [4]．実際のグラフの生成に関して，グラフ管理データベース（GDBMS）である Neo4j[9] を利用する．Neo4j を利用する理由としては，
- プロパティグラフを管理できること
- クエリによるグラフの探索が容易であること

が挙げられる．実際のグラフインスタンスの生成に関しては 4.2 節で記述する．

(2) 特徴ベクトル獲得
依存関係グラフによって表現される安定性を，定量的なメトリクスとして抽出するために，グラフ表現学習によってソフトウェア全体の構造変化に関する特徴ベクトル（Feature）を獲得する．graph2vec[8] を学習アルゴリズムとして利用し，連続するバージョンにおけるモジュールの削除／追加の個数，モジュール間の利用関係の削除／追加の個数に基づいて学習のラベリングを行うことで，外部安定性と内部安定性それぞれにおいてモジュールの変更と利用関係の変更に関する分類タスクを実行する．この操作により特徴ベクトルを獲得する．ベクトル空間において，意味的に類似したベクトル同士が近くにマッピングされていることが期待される．

(3) 特徴ベクトル分析
(2) において獲得された特徴ベクトルをクラスタリングし，その構造を分析する．特徴ベクトルがどの点に集合しているかを調べるため，k-means 法を用いることで特徴ベクトルの重心を求める．また，この操作により分析対象となったモジュールを意味的に類似するクラスタに分離できる．この処理を (2) における外部安定性，内部安定性のそれぞれの特徴ベクトルに対し実施する．

(4) ソフトウェア安定性分析
ソフトウェア安定性の変化を観測するため，(3) で得られた特徴ベクトルの重心とクラスタの時間的変化を分析する．具体的な分析方法は 4.3 節で説明する．

4. 分析プロセスとプロトタイプシステムの実装

次に示すフレームワークに従うことで，Constantinou らの先行研究よりも詳細に安定性の定量的分析，安定性の変化の視覚的な理解を行うことが可能であると考える．

4.1 分析プロセス

分析プロセスは**図 3** に示す 5 つのステップから構成される．

(1) 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べき分析項目を設定する．

(2) Graph Database（GraphDB）実装
本ステップは，3 節における「依存関係グラフモデル定義」に対応し，以下の 2 つのサブステップから成る．
- Git からのデータ収集
- GraphDB インスタンス生成

(3) フィーチャ分析
本ステップは，3 節における「特徴ベクトル獲得」,「特徴ベクトル分析」に対応し，以下の 2 つのサブステッ

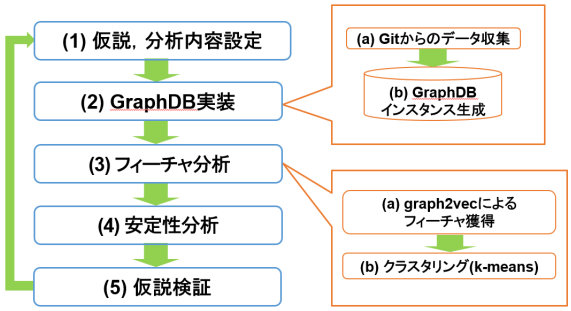

図 3 本研究の分析プロセス
Fig. 3 Analysis process of this study

プから成る．

- graph2vec によるフィーチャ獲得
- k-means 法によるクラスタリング

(4) 安定性分析
本ステップは，3 節における「ソフトウェア安定性分析」に対応する．分析結果から，ソフトウェアの外部安定性，内部安定性を求め，ソフトウェア全体としての安定性を分析する．また，ソフトウェアの安定性をバージョンごとに時系列で視覚化することにより，多期間にわたるソフトウェアの安定性の変遷を観測する．

(5) 仮説検証
分析結果から，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．

これらの 5 ステップに従うことにより，本研究における成果物が明確になり，安定性分析に関する議論が可能になると考える．

4.2 GraphDB 実装

3 節における依存関係グラフモデル定義に基づき，Neo4j にてグラフインスタンスを生成する．依存関係グラフはノードとエッジに属性集合を定義できるプロパティグラフ [11] を用いる．ノードのプロパティ定義とエッジの型定義を**表 1** に示す．

分析対象は，

- 利用関係の明確化（他モジュールの利用）
- 分析対象リポジトリの豊富さ

の理由から，Python のプログラム及び，Python のプログラムが含まれるディレクトリとする．自モジュールで同ソフトウェア内の他モジュールを「import」することを本研究では「USE」エッジとして表現し，分析における利用関係に該当するものとする．「CONTAIN」エッジは利用関係ではなく，ディレクトリの階層構造を表現するため，視覚化したグラフの全貌の理解を容易にするために導入している．

4.3 フィーチャ分析と安定性分析

Constantinou らの安定性の定義に基づき，外部安定性，内部安定性について分析する．

4.3.1 外部安定性分析

Constantinou らの外部安定性の定義に基づき，モジュールの変更を元にソフトウェアの安定性を分析する．ソフトウェアの連続するバージョンにおいて，モジュールの削除，再利用をラベルとし，生成したグラフインスタンスに表現学習を適用する．グラフインスタンスにおけるノードが元のソフトウェアにおけるモジュールと対応する．学習の入力は，あるバージョンのモジュール依存関係グラフの分割されたサブグラフであり，出力は特徴ベクトルである．特徴ベクトルはサブグラフが変換されたものであり，モジュール変更に関して，意味的に類似するサブグラフがベクトル空間上の近い位置にベクトルとしてマッピングされていることを期待する．

図 4 に，本研究のクラスタリングによる分析と先行研究の ES 値の比較について示す．獲得した特徴ベクトルに対して k-means 法を適用することで重心ベクトルを特定する．重心の特定により，外部安定性に関する特徴ベクトルを，「モジュールの削除に関するクラスタ」，「モジュールの再利用に関するクラスタ」に分離する．graph2vec により，これらの 2 クラスタには意味の近いベクトル同士が属していることを期待する．ここで安定性の変化に関して，クラスタの分散の変化，重心ベクトルの移動に着目する．

クラスタの分散の変化を分析することで，先行研究における ES 値との比較を行う．モジュールの削除に関連するクラスタの分散が増加した場合，元のソフトウェアではバージョン移行の際に削除されたモジュールが増えた事を意味するので，ES 値としては 0 に近づくと考えられる．反対に，再利用に関連するクラスタの分散が増加した場合，バージョン移行で再利用されたモジュールが増えた事を意味するので，ES 値としては 1 に近づくと考えられる．

分散の変化に加え，バージョン移行の際の重心ベクトルの移動距離にも着目する．重心の移動距離が小さい場合，クラスタに属するベクトル群に差異はほとんど無いと考えられるため，ES 値の大きな変動はないと推測できる．重心の移動距離が大きい場合，クラスタに属するベクトル群について，前バージョンとは大きく構成が変化したと考えられるため，ES 値も大きく変動すると推測できる．

以上の様に，モジュールとその変更を，グラフのノードに対する変更として置き換え，表現学習によって特性を抽出することにより，外部安定性の議論を行うことが可能となる．

4.3.2 内部安定性分析

外部安定性分析に対し内部安定性分析では，Constantinou らの内部安定性の定義に基づき，モジュール利用関係の変更を元にソフトウェアの安定性を分析する．ソフト

表 1　ノードのプロパティとエッジの型の定義
Table 1　Definition of node properties and edge types

	属性/型	記述項目
ノード	name	ディレクトリ名 or モジュール名
	def	ディレクトリ形式 or ファイル形式
	path	分析対象ルートディレクトリからのパス
エッジ	CONTAIN	あるディレクトリ下に他のディレクトリ，プログラムが存在
	USE	あるプログラムが他のプログラムを import

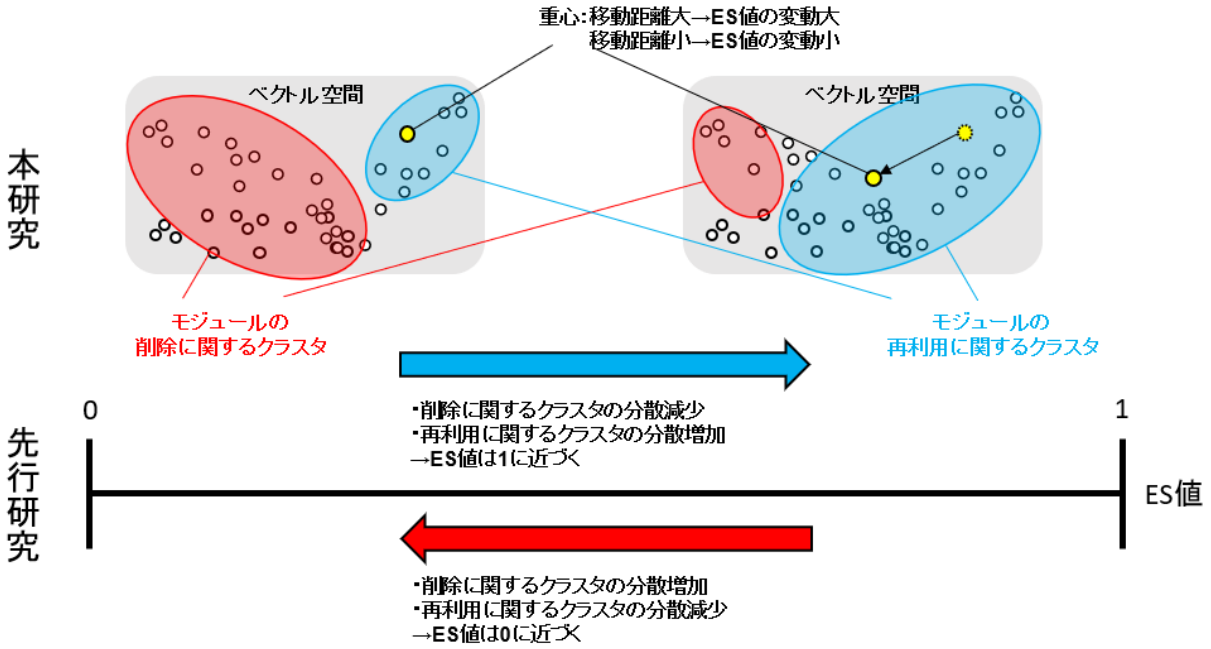

図 4　クラスタと重心の変化に基づく ES 値の変化
Fig. 4　Changes of ES with clusters and their centroid points

ウェアの連続するバージョンにおいて，モジュール利用関係の削除，再利用をラベルとし，生成したグラフインスタンスに表現学習を適用する．グラフインスタンスにおけるエッジが元のソフトウェアにおけるモジュール利用関係と対応する．学習の入力は，あるバージョンのモジュール依存関係グラフの分割されたサブグラフであり，出力は特徴ベクトルである．特徴ベクトルはサブグラフが変換されたものであり，モジュール利用関係の変更に関して，意味的に類似するサブグラフがベクトル空間上の近い位置にベクトルとしてマッピングされていることを期待する．

図 5 に，本研究のクラスタリングによる分析と先行研究の IS 値の比較について示す．外部安定性分析と同様に，獲得した特徴ベクトルに対して k-means 法を適用することで重心ベクトルを特定し，内部安定性に関する特徴ベクトルを，「モジュール利用関係の削除に関するクラスタ」，「モジュール利用関係の再利用に関するクラスタ」に分離する．また，ES 値と同様に，クラスタの分散と重心の移動距離に着目して本研究と先行研究における IS 値との比較を行う (図 5)．

クラスタの分散に関して，モジュール利用関係の削除に関連するクラスタの分散が増加した場合，元のソフトウェアではバージョン移行の際に削除された利用関係が増えたことを意味するので，IS 値は 0 に近づくと考えられる．対して，モジュール利用関係の再利用に関するクラスタの分散が増加した場合は，バージョン移行で再利用された利用関係が増えたことを意味するので，IS 値としては 1 に近づくと考えられる．

バージョン間の重心の移動距離に関して，重心の移動距離が小さい場合，クラスタに属するベクトル群に差異はほとんど無いと考えられるため，IS 値の大きな変動はないと推測できる．対して，重心の移動距離が大きい場合，クラスタに属するベクトル群について，前バージョンとは大きく構成が変化したと考えられるため，IS 値も大きく変動すると推測できる．

以上の様に，モジュール利用関係とその変更を，グラフのエッジに対する変更として置き換え，表現学習によって

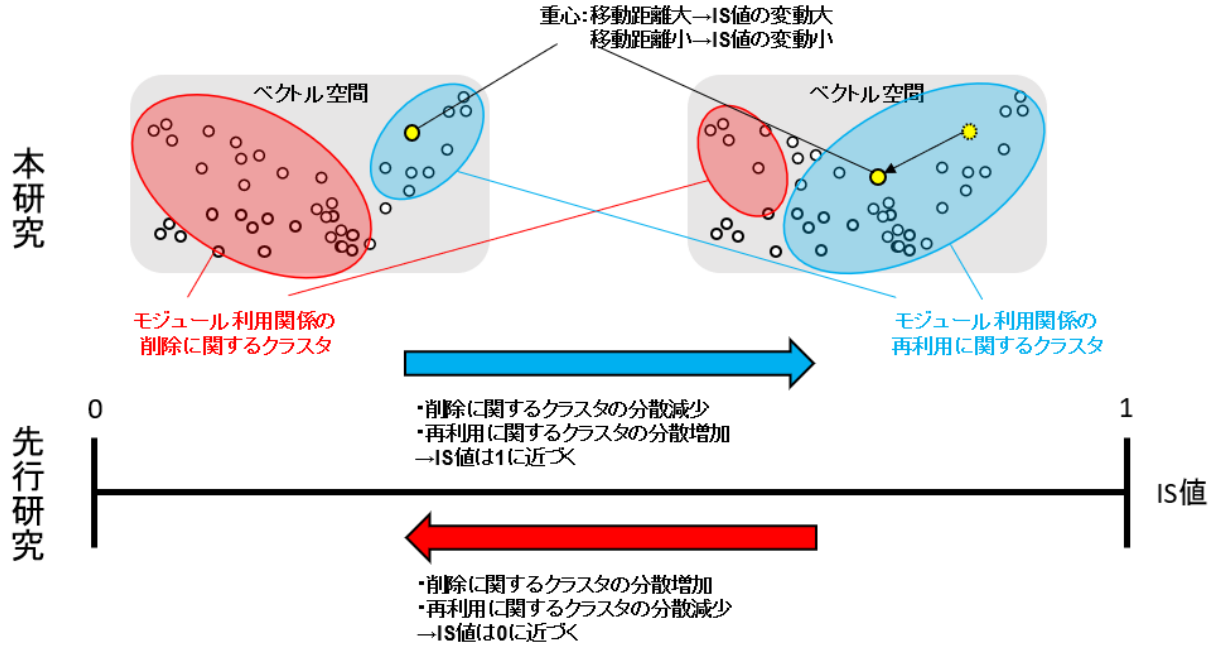

図 5 クラスタと重心の変化に基づく IS 値の変化
Fig. 5 Changes of IS with clusters and their centroid points

特性を抽出することにより，内部安定性の議論を行うことが可能となる．

5. 考察

RQ1 で述べたグラフモデル定義について，本稿では，OSS を対象に，モジュールをノード，その利用関係をエッジとしてグラフへの抽象化を行い，各バージョン毎にグラフインスタンスとしてグラフ管理データベース Neo4j に格納することで，時間的変化を含むソフトウェアの特性を，グラフの変化として定義した．この結果，ソフトウェアの時間的変化をグラフの変遷として可視化することができ，視覚的なソフトウェアの変化の理解が可能となる点がメリットとして挙げられる．

RQ2 について，各グラフインスタンスに対してグラフ表現学習 graph2vec を適用することで特徴ベクトルを獲得，クラスタリング手法 k-means 法により分析することで，ソフトウェアの代表点（重心）を明らかにした．重心とクラスタの時間的変化に着目することで，本研究では ES 値／IS 値の変化を先行研究より詳細化した．この結果，外部安定性と内部安定性のより詳細な定量表現が可能となる点，及びクラスタリング結果を可視化することにより，ES 値／IS 値の変化の視覚的な理解が可能となる点がメリットとして挙げられる．

Constantinou らはソフトウェアの安定性を，モジュールとモジュール間の利用関係の変更に基づく数理モデルを定義して分析している．対して本研究では，外部安定性と内部安定性を，依存関係グラフモデルとグラフ表現学習を用いることで，先行研究における $[0, 1]$ の定量表現をニューラルネットワークによって再定義した．本研究における分析により，外部安定性についてはグラフのノード，内部安定性についてはグラフのエッジの変化に着目することで，その推移が観測できることを明らかにした．本提案によって，安定性の推移のより詳細な観測を行うことが可能である．

6. おわりに

本研究では，ソフトウェア構造の複雑化に対し，変更による影響を安定性として分析する既存研究に則り，表現学習を用いてその特性を抽出，可視化して安定性を分析する方法の提案を行った．ソフトウェアを抽象化したグラフと表現学習 graph2vec による特徴ベクトル獲得，k-means 法による代表点の特定，および代表点の変遷の観測により，安定性の分析が可能となる．成果として，ソフトウェアの時間的変化をグラフの変遷として可視化できる点，Constantinou らの定義した ES 値／IS 値の詳細化，またその変化を視覚的に理解することが可能となる点が挙げられる．

今後の課題として，提案方法を支援するプロトタイプの実装，OSS を対象とした実験，先行研究と比較した安定性分析の妥当性，有効性の評価が挙げられる．

謝辞 本研究の一部は，科研費（基盤研究（C）20K11759），および 2021 年度南山大学パッヘ奨励金 I-A の助成による．

参考文献

[1] 浅谷 公威, 大知 正直, 森 純一郎, 坂田 一郎：ネットワーク表現学習によるネットワークの成長可視化, 情報処理学会第 186 回知能システム研究会, Vol. 2017-ICS-186, No. 6, 情報処理学会, Mar. 2017, pp. 1–6.
[2] Bohner, S. A. : Software Change Impacts: An Evolving Perspective, Proc. of ICSM 2002, IEEE Computer Society, Oct. 2002, pp. 263–271.
[3] Constantinou, E., and Stamelos, I. : Architecture Stability and Evolution Measure for Software Reuse, Proc.of SAC 2015, ACM, Apr. 2015, pp. 1580–1585.
[4] Fayad, M. E., and Altman, A. : An Introduction to Software Stability, Vol. 44, No. 9, CACM, Sep. 2001, pp. 95–98.
[5] Hamilton, W. L., Ying, R., and Leskovec, Jure : Representation Learning on Graphs: Methods and Applications, Apr. 2018, pp. arXiv:1709.05584v3.
[6] Horwitz, S., and Reps, Thomas : The Use of Program Dependence Graphs in Software Engineering, Proc. of ICSE 1992, ACM, Jun. 1992, pp. 392–411.
[7] 加藤 聖也, 青山 幹雄：機械学習によるグラフモデル OSS 開発コミュニティ構造の特徴量分析方法の提案と評価, No. 6N-03, 情報処理学会, Mar. 2019, pp. 281–282.
[8] Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S : graph2vec: Learning Distributed Representations of Graphs, Proc. of MLG 2017, Mining and Learning with Graphs, Jul. 2017, https://arxiv.org/abs/1707.05005/.
[9] Neo4j.Inc : Neo4j. https://neo4j.com.
[10] Ramirez, S. M., Cortes, K., Ocharan-Hernandez, J. O., Sanchez Garcia, A. J. : Software Stability : A Systematic Literature Review, Proc. of CONISOFT 2018, IEEE Computer Society, Oct. 2018, pp. 109–115.
[11] Robinson, I., Webber, J., and Eifrem, E. : Graph Databases, 2nd Edi-tion, O’Reilly, 2015.