ソフトウェア進化履歴を利用して APIの後方互換性の欠如を予測する手法に関する研究

M2020SE001 大道裕矢
指導教員：名倉正剛

1 はじめに

ソフトウェアの開発プロセスの進行とともに，ライブラリに規定される API の外部プログラムからの呼び出しに対して互換性を欠如するような変更を実施すると，そのライブラリを利用する外部プログラムの動作に影響を与える．しかし一般的にライブラリの API に対する後方互換性の欠如は外部プログラムの実行にしか不具合を発生させないので，後方互換性が欠如する原因を引き起こしたライブラリの開発者は不具合を意識することがない．そのため，ライブラリの開発者が意識せずに変更を実施し変更実施後の互換性（後方互換性）が保てなくなることは，ソフトウェアの不具合の原因の把握を困難にする．本研究では，開発プロセスの進行から，開発プロジェクトにおいてそのような後方互換性を保てなくなるような変更が生じる可能性があることを予測するための分析プロセスを定義した．

2 API の後方互換性とその検査手法

2.1 API の変更と後方互換性

ソフトウェア部品の再利用性向上のために，API を規定してライブラリを作成することがある．そして規定された API は，一般的に外部のプログラムから呼び出すことができるように公開される．またソフトウェア開発プロセスの進行につれて機能が変更されたり機能が削除されることにより，API が変更されることがある．そして外部プログラムから利用されていることをライブラリのソフトウェア開発者が意識しないことにより，変更実施後に API の後方互換性が保てなくなる場合がある [1]．

後方互換性が保てない変更例を，図 1 に示す．この図では，上半分がライブラリのクラス X のバージョン進化を，下半分がそれを利用する外部プログラムを表している．なお，クラス X のリストの左上に示した丸囲み数字のようにバージョン進化を行うとする．そして，外部プログラムでは，*main* メソッドで，クラス X のインスタンスを生成し，そのインスタンスを利用してメソッド *foo* を呼び出しているとする．この例では，バージョンが ① から ② に変化するときに，ライブラリ X ではメソッド *foo* に行が追加されている．この場合には外部プログラムからのメソッド呼び出しの方法が変更されていないので，外部プログラムからはバージョン ① と同様にメソッド *foo* を呼び出すことができる．一方，② から ③ に変化する場合には，可視性が *public* から *protected* に変更されている．これにより外部プログラムから呼び出すことができず，呼び出しに失敗する．このように，ライブラリが外部に公開する API に関する仕様（クラス名，メソッド名，シグネチャ，戻り値の型，修飾子，継承元の基底クラス，例外など）を変更する場合，変更実施後の互換性を失うことになる．① から ② のような変更を [1] では後方互換性（backwards compatibility）を保持すると定義しており，本研究では後方互換性を保てない変更が行われることを，後方互換性の欠如と呼ぶ．

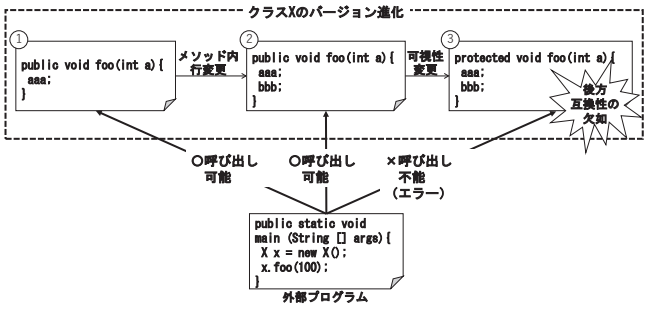

図 1 後方互換性が保てない変更例

一般的にライブラリの API に対する後方互換性の欠如は外部プログラムの実行にしか不具合を発生させない．後方互換性が欠如する原因を引き起こしたライブラリの開発者は，自身の開発部分に不具合が発生していないので，不具合を意識するきっかけがないことが多い．また，ライブラリの開発者とそれを利用する外部プログラムの開発者の開発コミュニティ（組織など）が同一であるという保証もない．したがって，不具合の発生を外部プログラムの開発者がライブラリの開発者に伝え，原因を特定してもらうことが困難な場合がある．このように，API に対する後方互換性の欠如は，ソフトウェアの不具合の原因の把握を困難にする．

2.2 後方互換性を検査する手法

Java ライブラリを対象に API の後方互換性が欠如する状況を調査した研究 [2] が報告されており，そのような変更の存在を検査する手法として APIDiff が提案されている [3]．これは，Java プログラムの 2 つのバージョンを入力として受け取り，変更箇所が後方互換性を変更するかを検査する手法である．具体的には各バージョンのコードをトークンに分解し，クラス，メソッド，フィールドを抽出する．そして，バージョン間でのコードの類似度の計算により類似コードを検出する．そして検出した類似コード片がリファクタリング実施されたものとみなし，リファクタリング前後のコード片に対して，バージョン間で API をマッピングする．これにより API の変更を検出し，変更種類によって後方互換性を欠如させる変更かどうかを判別する．

この APIDiff はコードの類似度に基づいて API の変更を検出するが，この際にコードの類似度の閾値を適切

に設定することが難しいことが指摘されている．そのため精度を高めるためにリファクタリング操作を抽象構文木による文単位のマッチングによって検出する手法 [4] が提案されている．

3　対象とする課題と本研究の目的

前述のように API の後方互換性の欠如はソフトウェアの不具合の原因の把握を困難にする．そこで [3] や [4] を利用することで，ソフトウェア変更時に変更されたバージョンのコミットが後方互換性を欠如させるものかどうかを検査することができる．

しかし, 変更したソフトウェアをコミットする時点で，これらの手法によって後方互換性が欠如されるような変更を発見できたとしても，その API を利用し互換性の欠如が影響する外部プログラムが多くの場所にわたる可能性がある．場合によっては後方互換性を欠如した API を，ライブラリ自体で内部的に利用していることもあり得る．そのような場合にはコミット時点で後方互換性の欠如を検出しても解消するための修正コストが高くなってしまう．

一方で，前述のようにソフトウェア開発プロセスの進行の過程で API が変更されることを考えると，後方互換性を欠如させるような変更が生じる場合に特徴的なプロセスの進行が存在するのではと考える．例えば一度の変更で普段よりも多く削除をしている場合，後方互換性を精査せずに削除をしていたならばその結果として外部から呼び出されているメソッドが削除され，後方互換性を欠如させることにつながる．このように考えると，ソフトウェア開発プロセスの進行を分析することによって，実際に後方互換性を欠如させる変更コミットが発生する前に後方互換性の欠如を予測できる可能性がある．

そこで本研究では，後方互換性を欠如させるような変更が生じる一つの要因として，開発プロジェクトでの開発プロセスの進行に着目する．そして，ソフトウェア進化履歴を保存するためのリポジトリを利用し，ソフトウェア進化履歴を観察した．その結果，ソフトウェア進化履歴から取得できるメトリクスによって，後方互換性を失うような変更を予測できる可能性があることが分かった．この結果を利用して，後方互換性を失うような変更が発生する可能性があることを予測するための分析プロセスを検討した．

4　後方互換性の欠如の状況の調査

4.1　後方互換性欠如の検出方法

あるコミットに着目した際に，そのコミットが後方互換性を欠如させるものかどうかを，先行研究 [3] で実装された APIDiff [1] を利用した [2]．APIDiff では後方互換性の検出手順として，対象プロジェクトの master ブランチの全コミットについてまとめて後方互換性が保持されてたか欠如されたかを検査する手順と，特定ブランチのコミット（または最新のコミット）について後方互換性が保持されてたか欠如されたかを検査する手順を提供している．

4.2　調査対象プロジェクト

APIDiff では類似度の閾値設定が精度に影響する．そこで，先行研究 [3] でこの APIDiff の動作の例として紹介されており，なおかつ APIDiff のダウンロードサイト [1] での説明で実行手順として紹介されていた mockito/mockito プロジェクト [3] を，先行研究の著者らが適切な閾値として設定できていると考えるプロジェクトとみなして対象にした．対象としたプロジェクトの概要を，表 1 に示す．

表 1　対象プロジェクトのサマリ

コミット数	コミッタ数	開発期間	実活動日数
5,576	263	2007/11/15 ～ 2021/7/20	1,430

Java ファイル		Java コード	
ファイル数	割合 (%)	行数	割合 (%)
949	84.8	90,430	85.4

4.3　APIDiff による後方互換性欠如の検出

APIDiff により，調査対象である mockito プロジェクトのリポジトリに対して，master ブランチの全コミットを対象に後方互換性の欠如を検出した．

4.4　検出結果

検出した後方互換性の欠如の個数を，表 2 に示す．表 2 において総数は，APIDiff で検出される後方互換性の種類ごとに検出された数の総和を，検出された箇所数は APIDiff で検出される後方互換性の種類ごとの検出されたコミットの数の総和を表す．また対象プロジェクトでは，Method Element に対する後方互換性の欠如のみが検出された．

表 2　検出された後方互換性欠如について

	総数	検出された箇所数
remove	385	123
change in return type	97	16
move	64	7
rename	26	19
lost visibility	23	13
change in exception list	7	4
remove static modifier	1	1

表 2 によると，remove による後方互換性の欠如が多く検出された．そこで後方互換性の欠如の予測にあたって，まず remove の予測を検討した．以降，remove の予測に利用可能なソフトウェアメトリクスを調査するために，remove が発生したコミットを定性的に観察する．

4.5　後方互換性削除の状況の観察

remove が発生したコミットを対象にその発生状況を観察した．APIDiff の出力結果から remove が発生したコミットを抽出し，GitHub からそのコミットの変更情報（差分，コミットメッセージ）を取得した．さらにそのメ

[1] https://github.com/aserg-ufmg/apidiff (2022/1/26 閲覧)

[2] 先行研究 [4] では類似度の閾値設定により精度に影響することを述べ，その改善のための手法が提案されているが，実装が公開されていないので，ここでは先行研究 [3] の実装を利用した．

[3] https://github.com/mockito/mockito (2022/1/26 閲覧)

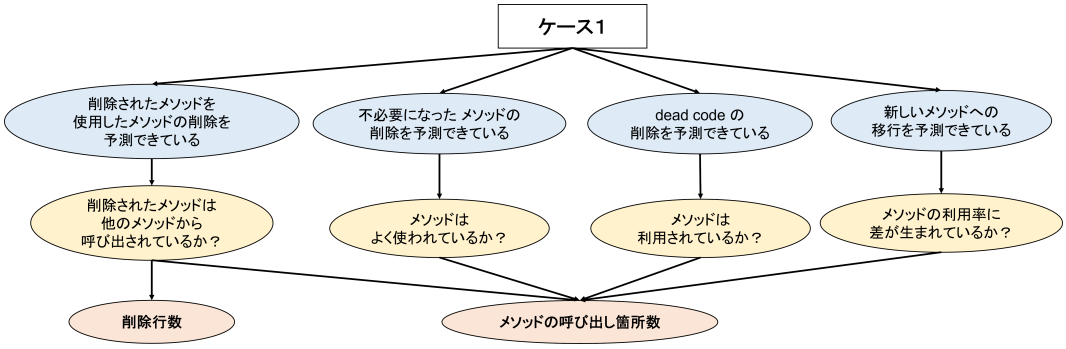

図 2　対象のメソッド自体の削除による remove の後方互換性欠如が発生する場合の分析例

ソッドに対する修正情報を取得し，remove が発生したメソッドが追加された時点からの様子を観察し，その特徴や remove が発生した理由を定性的に分析した．

観察の結果，対象のメソッド自体が削除される場合に加えて，クラスの移動や分割をメソッドの削除と新しいメソッドの追加として検出している場合，メソッドシグネチャを変更したことをメソッドの削除と新しいメソッドの追加として検出している場合の，大きく分けて 3 つのケースが確認できた．

4.6　ソフトウェアメトリクスの選定

観察されたケースごとに GQM パラダイムを用いて予測に利用できる可能性があるソフトウェアメトリクスを検討した．GQM パラダイムとは，[5] によって提案された測定のフレームワークである．

対象のメソッド自体が削除される場合として，不要になったメソッドの削除，dead code メソッドの削除，非推奨のメソッドの削除，削除されたクラス内のメソッドを使用したメソッドの削除が観察された．そこで，それぞれを予測することを目標 (Goal) として設定し，メトリクスを整理した．対象のメソッド自体が削除されるケースについて，GQM パラダイムを用いた分析結果は図 2 の通りである．以降では，このケースについて分析を進める．

5　ソフトウェア進化履歴の分析

5.1　メトリクスの変化の観察

GQM パラダイムを用いた分析の結果得られたソフトウェアメトリクスと後方互換性の欠如との関係性について，本分析ではケース 1 について考える．ここでは，ケース 1 の remove を発生させた stubVoid メソッドと後方互換性の欠如を維持する変更（gain visibility）に分類された mock メソッドについてそれぞれのメトリクスの変化を観察した．ケース 1 では図 2 よりメソッド呼び出し箇所数と削除行数により予測を実施できる可能性があることを導出した．そこでここではそれらのメトリクスの変化に着目する．調査範囲としては，それぞれ後方互換性を欠如させる変更，維持する変更を行ったコミットから 100 コミット遡ってメトリクスを抽出した．各コミットごとの変化として，前後のソースファイルをもとに削除行数を取得し，さらに各メソッドに対するメソッドの呼び出し箇所数を取得した．なお削除行数は削除ファイルを除く削除行数を取得している．

5.2　観察結果

メソッドの呼び出し箇所数の推移を図 3 に示す．この図では後方互換性の欠如が発生した場合のメソッドについて赤色の線で，発生していない場合のメソッドについて青色の線で表している．横軸は後方互換性の欠如が発生したコミットを 0 番目として遡ったコミットの数を示しており，縦軸はメソッドの呼び出し箇所数を示している．後方互換性の欠如が発生している場合はメソッドの呼び出し箇所数に変化がないことが読み取れる．また誌面の都合上掲載しないが，後方互換性を欠如させるメソッドでは後方互換性の欠如の発生前に削除行数が少なくなっていた．

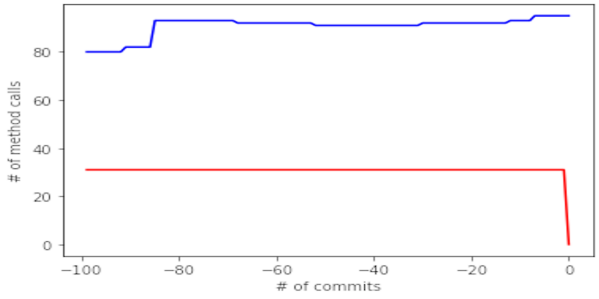

図 3　メソッドの呼び出し箇所数の推移

5.3　分析方法

5.3.1　進化履歴からのメトリクスの算出

5.2 節で観察したように，メソッドの呼び出し箇所数と削除行数の変化の推移は，後方互換性を欠如させる変更と維持する変更では異なる傾向がありそうなことがわかった．傾向が明確に異なるならば，後方互換性を欠如させる変更が発生するかどうかを，メトリクスの推移から予測できる可能性がある．そこで，実際にこれらのメトリクスの推移と後方互換性の欠如との関連を分析した．

ここでは，API 変更を後方互換性を欠如させる変更と維持する変更が行われたメソッドをそれぞれ 5 つずつ選択し，5.1 節と同様の操作で 2 つのメトリクスを抽出した．

5.3.2　算出したメトリクスに対する前処理

本研究では，API の変更コミットに対して直前の 100 コミットのメトリクスデータを利用して検定を実施した．

後方互換性の欠如が発生を予測すること目的としているので，API の変更が行われたコミットのデータは除外する．さらにメソッドの呼び出し箇所数，削除行数のそれぞれに前処理を行い，分析に使用した．メソッドの呼び出し箇所数については，各コミットに対して取得したメソッド呼び出し箇所数変化量と最小値との差分を利用し，また削除行数については，分析対象の各メソッド変更に対して，100 コミットでの中央値が等しくなるように正規化した削除行数を利用した．

5.3.3 後方互換性欠如の有無による有意差の検定

メソッドの呼び出し箇所数と削除行数について，後方互換性の欠如を発生させる変更と後方互換性を維持する変更の間に差があるかどうか t 検定による分析を行った．t 検定の結果，2 つの集合の間に有意差があれば予測に利用できる可能性のあるメトリクスと判断した．ここでは，5.3.2 節で前処理したメソッドの呼び出し箇所数と削除行数のデータの集合について，それぞれ後方互換性の欠如が発生した場合と後方互換性を維持する変更が発生した場合とで差があることを示すために，有意水準を 0.05(5%) として t 検定を実施した [4]．また 2 標本の母分散が等しいと限らないので非等分散を持つ可能性のある 2 つの標本に用いることを意図して定義された Welch の t 検定を使用した．

5.4 分析結果

t 検定の結果，算出された両側検定の場合の p 値は，メソッドの呼び出し箇所数が 9.01×10^{-23} ，削除行数が 9.55×10^{-4} であった．どちらも事前に設定していた有意水準 (0.05）を下回った．したがって，後方互換性の欠如が発生したメソッドの呼び出し箇所数と後方互換性を維持する変更が発生したメソッドの呼び出し箇所数，および後方互換性の欠如が発生した際の削除行数と後方互換性を維持する変更が発生した際の削除行数の間にはどちらも有意差があることが明らかになった．

6 考察

6.1 分析結果について

5.1 節の結果から，メソッドに対して後方互換性の欠如を発生させるコミットに着目した場合，その前の一定期間のコミットにおいて呼び出し箇所数の変化がなく，各コミットでの行の削除がほぼ発生しなかった．しかし，後方互換性の欠如を発生させる際の変更コミットではメソッドの呼び出し箇所数が 0 になり，削除行数が多くなっていることが確認できた．この結果から，メソッドが長期間にわたってメインテナンスされずにある時点で削除されるような状況が発生していることがわかる．また後方互換性を維持する変更では，呼び出し箇所数の変化するコミットが定期的に発生し，行の削除が多く発生するコミットも定期的に発生していることが確認できた．さらに削除行数とメソッドの呼び出し箇所数について，後方互換性の欠如を発生させる変更と後方互換性を維持する変更の間に差があるかどうか t 検定による分析を実施した．5.3 節の結果から，後方互換性の欠如を発生させる変更と後方互換性を維持する変更の間には有意差が確認できた．この結果からメソッドの呼び出し箇所数と削除行数が後方互換性の欠如の予測に利用できる可能性があることがわかった．

6.2 後方互換性の欠如を予測するために定義した分析プロセス

6.1 節で述べた状況から，後方互換性の欠如を予測する分析プロセスを次のように定義した．

Step 0) 期間基準値の設定 （前準備）
Step 1) ソフトウェアメトリクスの抽出
Step 2) 基準となる期間に対する検定
Step 3) 後方互換性の欠如が生じるかどうかを判定

このプロセスでは，対象のプロジェクトに対して予測に利用するべきコミット数およびその期間（期間基準値）を算出したのち，メソッド呼び出し箇所数や削除行数のようなメトリクス値を求め，後方互換性の欠如を予測する．

7 まとめ

本研究では，API の後方互換性の欠如の予測に利用できる可能性のあるメトリクスを，開発プロジェクトに対するソフトウェア進化履歴を用いて GQM パラダイムにより導出した．導出したメトリクスを t 検定を用いて分析した結果，後方互換性を欠如させる変更と維持する変更で値が有意に異なることが確認できた．そして API の後方互換性を欠如させる変更を，それらのメトリクスを利用することで予測する分析プロセスを定義した．

今後の課題としては，観察対象のより多くの箇所の後方互換性の欠如に対して調査すること，さらに調査結果が他のプロジェクトでも成り立つか明らかにすることが挙げられる．

参考文献

[1] Dig, D. and Johnson, R.: How do APIs evolve? A story of refactoring, Software Maintenance and Evolution: Research and Practice, Vol. 18, No. 2, pp. 83–107 (2006).

[2] Xavier, L., Brito, A., Hora, A. and Valente, M. T.: Historical and impact analysis of API breaking changes: A large-scale study, Proc. of 2017 IEEE 24th Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER), pp. 138-147 (2017).

[3] Brito, A., Xavier, L., Hora, A. and Valente, M. T.: APIDiff: Detecting API breaking changes, Proc. of 2018 IEEE 25th Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER), pp. 507-511 (2018).

[4] 入山 優，肥後 芳樹，楠本 真二：抽象構文木を利用した API の後方互換性が破壊される変更の検出，ソフトウェアエンジニアリングシンポジウム 2021 論文集，情報処理学会シンポジウムシリーズ，pp. 209–218 (2021).

[5] V.R. Basili, G. Caldiera, and H.D. Rombach,“The goal question metric paradigm,”in Encyclopedia of Software Engineering, ed. J.J. Marciniak, vol.1(1994), pp.528532.

[4] 後方互換性の欠如で対象にした変更には同一コミットのものが含まれる．それらについては削除行数の変化が変わらず，単純にデータに含めるとその影響を大きく受けることになるので，削除行数の検定では除外した．その結果，メソッドの変更 3 件を対象として，300 個のデータ数となった．