修士学位論文

題目

Android アプリケーションを対象とした API 利用法の実態調査

指導教員
井上 克郎 教授

報告者
小笠原 康貴

令和 2 年 2 月 5 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和元年度 修士学位論文

Android アプリケーションを対象とした API 利用法の実態調査

小笠原 康貴

内容梗概

Application Programming Interface(API) とは，プログラムの機能を再利用できる形でまとめて提供したものである．API を利用することで開発者は実装の手間を削減できるが，API の利用法を学習することは必ずしも容易ではない．そこで，API の学習支援のためにソフトウェアリポジトリから API 利用法をマイニングする研究が多数行われている．

API 利用法をマイニングする際の課題の 1 つは，膨大な数の利用法から開発者にとって有用な利用法を評価して検出することが難しいことである．その要因は，開発者にとって有用な利用法を一概に利用数で評価することができない点である．例えば，開発者が知りたい利用法はリポジトリで頻出していない可能性があったり，リポジトリに選ぶプロジェクトによって含まれる API 利用法の数は変動する．そのため API 利用法のマイニングを行う研究者は，これらの懸念点の影響を理解して検出する利用法を評価する必要がある．

そこで本研究では，Andoroid アプリケーションで構成したデータセットを対象に API 利用法の実態調査を行い，API 利用法のマイニングの際に考慮すべき点について考察するとともに，API 利用法の評価においてプロジェクトのバージョンアップに伴う API 利用法の変遷の情報が有用であるかを検討した．実態調査では，データセット中で頻出する利用法は大半の利用法の部分列となっていること，API によってはプロジェクト毎に利用される利用法の傾向が大きく異なること，プロジェクトのバージョンアップに伴い，頻出する利用法は利用数が増加する一方で，利用数が少ない利用法は増減を繰り返す傾向にあることがわかった．また実態調査の結果を考察し，API 利用法のマイニングにおいて， API 利用法の利用数による評価，データセットの選び方に関して考慮すべき点を示すとともに，プロジェクトのバージョンアップに伴う API 利用法の変遷の情報を用いて API 利用法を評価できる可能性について言及した．

主な用語

API 利用法
リポジトリマイニング

目 次

1　はじめに

近年，ソフトウェアシステムの開発の際に Application Programming Interfaces (API) が広く使用されている．API とは，プログラムの機能を再利用できる形でまとめて提供したものである．API を利用することで，開発者は実装の手間を大幅に削減することができる．

開発者が API を利用するためには，その利用法を理解する必要がある．しかし，API の利用法がドキュメントに十分に記載されていなかったり，API に膨大な数のメソッドが定義されている場合，その利用法を学ぶことは困難である．

開発者の API 利用を助けるために，リポジトリマイニングを用いた API 利用法の研究が多数行われている．API を利用する際に，API メソッドから複数のメソッドを組み合わせて利用することが多い．そこで，リポジトリから頻繁に利用されているメソッドの組み合わせをパターンとしてマイニングすることで，その API の利用法を理解することができる．先行研究では，リポジトリから頻出パターンをマイニングして開発者に提示する [19]．また，別の先行研究では，頻出パターンの部分パターンとなっているソースコードを，API 利用法を違反している疑いのあるソースコードとして提示する [11][4]．API 利用法のマイニングによる API 利用法の学習支援の課題の 1 つは，膨大な数の利用法から開発者にとって有用な利用法を評価して検出することが難しいことである．API は典型的な 1 つの利用法だけが用いられるのではなく，オプション的な処理の有無，プロジェクトのコーディングスタイルの違いなどによって，複数のバリエーションが利用される．リポジトリマイニングにより開発者に必要な利用法の情報を得るためには，これらの利用法から有用なもの，正当なものを評価する必要がある．しかし，これらの複数の利用法の重要さを一概に利用数で評価することはできない．例えば，先行研究 では，頻出パターンの部分パターンであり利用数が低いソースコードは欠陥の疑いがあるソースコードとして検出されるが，実際は利用数が少なくても欠陥ではない利用法が多数存在する [4]．その結果，欠陥検出の精度は下がってしまう．また，実際のプロジェクトにおける API 利用法の利用数は，プロジェクトの規模や種類によって差が生じるため，リポジトリの選び方によって利用法毎の利用数の傾向が変動することで，検出される利用法が変動してしまう恐れがある．

これらの問題に対処するために，API 利用法のマイニングを行う研究者は，検出する利用法を評価するうえで考慮すべき点を理解する必要がある．そこで本研究では，次の項目に注目して API 利用法の実態調査を行い，その調査結果から API 利用法のマイニングの際に考慮すべき点について考察を行った．

- RQ1:利用数が多い利用法，少ない利用法にどのような違いがあるのか？
- RQ2:プロジェクト間で利用法に違いはあるのか？

- RQ3:バージョンアップに伴う利用法の変遷にどのような特徴があるのか？

以降，2 章では本研究の背景として，API 利用法に関連する研究とその課題の説明を行う．3 章では本研究で設定した RQ と調査手法について説明し，4 章では調査結果を述べる．5 章では調査結果に対する考察について述べる．6 章では本研究のまとめについて述べる．

2 背景

2.1 API の利用と障害

API(Application Programming Interface) は，プログラムの機能を再利用可能な形でまとめたものである．API はライブラリなどの形で開発者に提供され，開発者は API を利用することで機能を実装する負担を軽減することができる [10][6]．また，サードパーティなどによって十分にテストされた API を利用することはソフトウェア品質の向上につながる．さらに，API の開発者は新たな機能を追加したり，バグの修正を行うために API の更新を行っており，開発者は利用する API を最新版にすることでその恩恵を受けることができる [8]．

API にはその使い方を記述したドキュメントが存在することが一般的だが，ドキュメントは十分に記述されていないことが多く，開発者にとって API の使い方を学ぶことは容易ではない [13][14][5][18]．API の利用法を学ぶ際の障害として，ドキュメントなどのリソースに十分な情報が記述されていないことや，API の個々のメソッドの動作がわかっていても，開発者のタスクを達成するためにどのメソッドを組み合わせて利用すべきかがわからないことなどが挙げられる．

2.2 API 利用法の学習支援

ドキュメントなどのリソースから API の利用法を学ぶことは必ずしも容易ではない．そこで，API 利用法の学習を支援するための研究が多数行われている．

MAPO[19] は，ソースコードから API のメソッド呼び出しパターンを検出する手法である．クエリとして API のメソッドやクラス名を与えることで， クエリに関連するソースコードから頻出メソッド呼び出しパターンのリストを出力する．開発者はメソッド呼び出しパターンのリストを確認することで，どのように API のメソッドを組み合わせたらよいかを学習できる．しかし，MAPO の検出結果は冗長で重複した結果が多い傾向にあり，実用性に課題が残っている．

UP-Miner[17] は，MAPO と同様に API のメソッド呼び出しパターンを検出する手法だが，MAPO の課題を改善するために，2 段階のクラスタリングステップを導入することで，重複が少なく網羅性が高い結果を出力することができる．また，この研究では検出された API パターンを評価するために 2 種類のメトリクスを提案している．これらのメトリクスでは，検出された API パターンがどれだけ考えられうる API 利用法を網羅しているか，どれだけ重複していないかを評価することができる．

MLUP[16] は，MAPO，UP-Miner と同様に，API のメソッド呼び出しパターンを検出する手法である．この手法の特徴は，API メソッド同士の共起関係に注目し，共起関係の

強さに応じて API メソッドをクラスタリングすることである．これにより，常に一緒に利用されるメソッドのグループと，場合によって一緒に利用されるメソッドを区別して理解することができる．

MAPO，UP-Miner，MLUP のようなメソッド呼び出しのパターンを検出する手法に対して，具体的なコード例を提示することで API 利用法の学習を支援する手法が提案されている．竹之内ら [20] の手法では，変数のデータフローを考慮した検索クエリを用いることで，従来のコード検索エンジンに比べ，より限定的な文脈を持つコード例を検索することができる．また，検索対象となるソースコードをコード検索エンジンから取得することで，あらかじめコード集合を用意すること無く利用できる利点がある．

ここまで説明してきた手法と大きく異なるアプローチとして，ソフトウェアのバージョンアップに伴うソースコードの変化から API 利用法を予測する手法が用いられている [15]．この手法では，バージョンヒストリーから同時に編集される API のメソッド呼び出しをクラスタリングすることで，開発者が利用する API メソッド呼び出しを予測している．

2.3 API 利用法の制約違反の検出

API には，特定のメソッドは別の特定のメソッドと同時に利用しなければいけない，メソッドを呼び出す順番を入れ替えてはいけないなど，利用法に制約がある場合がある．そのような制約を満たさない ソースコードは，ソフトウェアにバグを起こしたり，脆弱性の要因になっている [9][3]．そこで，そのような制約を満たさないソースコードを自動で検出する研究が多数行われている [7][11][4]．これらの研究では，ソフトウェアリポジトリから頻出する API の利用パターンを検出し，そのパターンから要素が欠陥しているソースコードを違反として検出する．また，API 利用法の制約にはメソッド呼び出しだけでなくプログラムの制御構造やデータフローが関連することが多いことから，一部の研究では API パターンをメソッド呼び出しの系列ではなく，プログラムの構造を捉えたグラフで表現する手法を用いている．これらの研究の課題は欠陥検出の精度の低さであり，その主な要因として頻出はしないが利用法として誤りではない利用法が False-Positive として検出されることが挙げられる [1]．

2.4 API 利用法のマイニングにおける課題

2.2 節，2.3 節で述べたように，API 利用法の学習支援のために，リポジトリマイニングによる API 利用法の検出を行う手法が多数提案されている．これらの手法における課題の 1 つは，膨大な数の利用法から開発者にとって有用な利用法を評価する必要があるが，開発者にとって有用な利用法を一概に API 利用法の利用数で評価できない点である．既存研究

における API 利用法の検出結果は，リポジトリに含まれる API 利用法の利用数，API メソッドの呼び出し回数などによって変動し，多くの研究では頻出する利用法を典型的な利用法として検出する．しかし，開発者が学習したい利用法が頻出している典型的な利用法とは限らず，一概に利用数で評価することはできない．また，リポジトリに含まれる API 利用法はプロジェクトの選び方によって変動するが，1 つのプロジェクトで特定の利用法が大量に利用されている場合や，プロジェクト固有の利用法が複数存在する場合，そのプロジェクトをリポジトリに含んでいるかどうかは検出結果に変化を与える可能性が高い．しかし，実際にプロジェクトによってどれくらい利用される利用法の傾向に違いがあるのかは調査が行われていない．

3　研究概要

開発者が API を利用する際は，API メソッドから必要なメソッドを複数組み合わせて利用することが一般的である．このような API メソッドの組み合わせを本研究では API 利用法と呼ぶ．API 利用法の学習を支援するためにリポジトリマイニングによる API 利用法の検出が行われているが，2.4 節で述べたように API 利用法を一概に利用数で評価できないこと，リポジトリに含まれるプロジェクトの違いによる影響が把握されていないことが懸念点である．また，利用数以外の要素で API 利用法を評価できるかを検討する必要がある．

そこで本研究では，上記の懸念点をもとに研究者が API 利用法のマイニングの際に考慮すべき点，そしてソフトウェアのバージョンヒストリーを用いた API 利用法の変遷の情報が API 利用法の評価に有用であるかを検討するために，以下の 3 つの項目に注目して API 利用法の実態調査を行った．

- RQ1:利用数が多い利用法，少ない利用法にどのような違いがあるのか？
- RQ2:プロジェクト間で利用法に違いはあるのか？
- RQ3:バージョンアップに伴う利用法の変遷にどのような特徴があるのか？

この章では調査を行う手法，予備調査の結果について述べる．

3.1　調査方法

本研究では Java で記述されたプロジェクトを対象として API 利用法の調査を行うため，以降は Java の文法を基準とした説明を行う．

3.1.1　API 利用法の定義

本研究で調査を行う API 利用法の定義を説明する．API を用いた実装をする際は，使用したいクラスのインスタンスを生成し，そのインスタンスを用いてメソッドを呼び出す形が多い．そこで本研究では，クラスのインスタンスを生成する文 (メソッドあるいはコンストラクター) と，そのインスタンスを用いて呼び出されるメソッドの系列として API 利用法を定義する．次の 2 行のソースコードを具体例として考える．

1: Iterator it = new Iterator();

2: while(it.hasNext()) it.next();

この実装は，Iterator クラスを用いた繰り返し処理の典型的な実装である．1 行目は，Iterator クラスのインスタンスを生成する文であり，赤字の部分が生成されたインスタンス，下線部がコンストラクターを表している．2 行目では，生成されたインスタンスを用いて下線部の 2 つのメソッドが呼び出されている．このソースコードから インスタンスを生成する文と，そのインスタンスから呼び出されるメソッドの系列を抽出すると以下の様になる．

$$Iterator/hasNext/next$$

このような系列を本研究では API 利用法と定義する．系列の順序はソースコード中の出現順に依存する．また，メソッドは引数と共に使用されることがあるが，本研究では引数の情報は定義に含めない．また，同一名のメソッドが複数回使用されている場合，最初に出現したメソッドだけを利用法の要素として採用する．

3.1.2 API 利用法の検出

プロジェクト中で使用されている API 利用法を検出する手法を説明する．手法の全体像を図 1 に示す．まず構文解析を用いて対象のプロジェクトのソースファイルからメソッド宣言文を抽出する．そして各メソッド宣言文に対して API 利用法の検出を行う．ここでは，3.1.1 節で述べた定義にしたがい API のメソッド呼び出しの系列をメソッド宣言文から抽出する．この処理により，各メソッド宣言文で使用されている API 利用法を検出することができる．全てのメソッド宣言文に対してこの検出を行うことで，各 API 利用法が どれだけのメソッド宣言文で利用されたかを得ることができる．以降，各 API 利用法が使用されたメソッド宣言文の数を，その API 利用法の利用数と単純に呼ぶ．

3.2 調査対象

調査の対象とする API, データセットについて説明する．本研究では，Android SDK(Software Development Kit)[12] の 3 つのクラスについて調査を行う．Android SDK を調査の対象とした理由は，Android アプリケーション市場は近年急速に成長しており，android SDK がその開発のために広く使われているためである [2]．調査を行う 3 つのクラスについて表 1 に示す．

本研究では，Android アプリケーションの Java ソースコードを対象に調査を行う． RQ3 において，プロジェクトのバージョンアップに伴う API 利用法の変化について調査を行うため，対象のプロジェクトにはある程度の開発履歴が存在する必要がある．そこで，2016 年から 2018 年の間にリリース間隔が半年以上のバージョンが毎年 2 バージョン以上リリースされているプロジェクトを基準としてデータセットを選定した．データセットとして，表 2

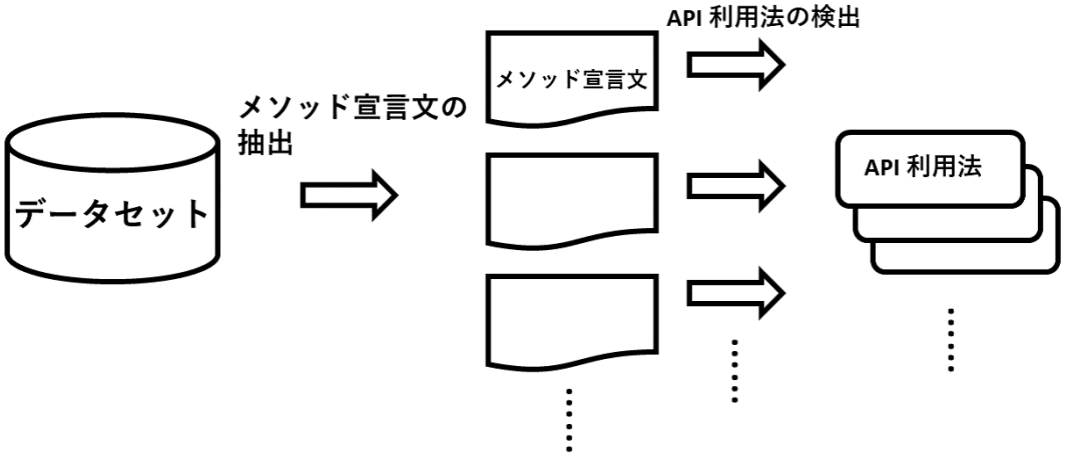

図 1: 調査手法の全体像

表 1: 調査対象の API

クラス名	説明
android.content.Context	アプリ環境の情報の管理
android.content.Intent	アプリ間の情報の受け渡し
android.database.Cursor	データベースの検索結果へのアクセス

に示すプロジェクトを使用する．各プロジェクトにつき 6 つのバージョンリリースをデータセットして使用する．バージョンの選び方は，リリース日が 2016 年 1 月から 2018 年 12 月にリリースされたバージョンから，バージョン間が約半年空くようなバージョンである．

3.3 予備調査

RQ1〜3 について調査する前に，予備調査として今回用いるデータセットにおいて，何種類の利用法が利用されているか，どれくらいの利用数があるかを調査した．さらに，3.1.1 節での利用法の定義から，インスタンスを取得するメソッドを除外した利用法についても同様の調査を行った．その理由は，インスタンスを取得するメソッドには，Java のコレクションからインスタンスを取得するメソッドなどが含まれており，API の利用法を知りたい開発者にとってあまり有益でない可能性があるためである．3 つの API についての調査結果を表 3 に示す．この調査結果から，`Intent` と `Cursor` については非常に多数の利用法が使用されていることがわかる．また，3 つの API ともに，インスタンスを取得するメソッドを除外した場合は利用法の種類が大幅に減少している．その要因は，どの API もインスタンスを取得するメソッドの種類が多いことである．`Context` では 14 種類，`Intent` では 66 種類，`Cursor` では 78 種類のインスタンスを取得するメソッドが使用されており，その多くが他のメソッドとの組み合わせではなく単独で使用されている．そのため，インスタンスを取得するメソッドを考慮しない場合の利用法の種類は大幅に減少する．

[1]リリースが最後のバージョンのもの
[2]インスタンスを取得するメソッドを除外したもの

表 2: データセット

プロジェクト	LOC[1]
AmazeFileManager	47,065
AndroidUtilCode	42,116
bitcoin-wallet	24,905
cgeo	109,288
codenameone	591,149
collect	75,790
ExoPlayer	195,539
fresco	109,099
k9	125,541
mapbox	40,437
news-android	20,828
owncloud	63,162
owntracks	14,812
photoview	2,305
realm-java	136,333
RedReader	45,837
robolectric	4,972
Signal-Android	111,234
skytube	20,083
wikipedia	76,402
WordPress	151,074
Total	2,052,505

表 3: API 利用法の種類と利用数

クラス名	利用法の種類	利用法の種類[2]	利用数
android.content.Context	44	18	218
android.content.Intent	213	144	1233
android.database.Cursor	206	136	380

表 4: API 利用法の利用数の分布

	Context	Intent	Cursor
[1∼5]	36(13)	186(121)	194(121)
[6∼10]	3(2)	16(13)	9(10)
[11∼100]	5(2)	9(8)	3(5)
[101∼]	0(1)	2(2)	0(0)

4 API 利用法の実態調査

4.1 RQ1

RQ1 では，利用数が多い利用法，少ない利用法にどのような違いがあるのか調査する．調査の手順として，まずデータセット中の API 利用法について利用数の分布を調査し，利用数が多い利用法，少ない利用法がどれぐらいの割合を占めているのかを確かめる．その結果から，利用数が多い利用法，少ない利用法の特徴の違いについて考察する．この違いを把握することで，API 利用法を検出する際に利用数が少ない利用法が与える影響について検討することが目的である．

利用数の分布の調査

まず，各利用法の利用数の分布について調査を行った．表 4 は，各利用法の利用数の分布を示したものである．3 つの API に共通する点は，利用数が 1 個から 5 個の利用法が大半を占める一方で，一部の利用法の利用数が突出している点である．例えば `Context` では，利用数の上位 5 件の利用法の利用数の合計は，全体の利用数の 65% ほどを占め，利用数が 5 個 以下である 36 種類の利用法の利用数の合計は全体の利用数の 25% ほどである．利用数が多い利用法の特徴は，表 5 に示すように，インスタンスを取得するメソッド単独で使用するもの，あるいはメソッドを 1 つだけ組み合わせたものが大半である．一方で，利用数が少ない利用法は比較的多数のメソッドを組み合わせた利用法であることが多い．この結果は，基本的な利用法にオプション的なメソッドを追加して利用する傾向を表している．

利用数が多い利用法，少ない利用法の特徴の調査

利用数の分布の調査では，利用数が少ない利用法は頻出する利用法に比べて，利用法に含まれるメソッド数が多いことがわかった．そこで，仮説として利用数が少ない利用法は頻出する利用法にメソッドを追加することで得られるではないかと考えた．そこで，次の調査と

表 5: 利用法の利用数 top5

	Context	Intent	Cursor
1	getContext/	Intent/putExtra/	query/
2	getApplicationContext/	Intent/	rawQuery/moveToFirst/
3	getContext/getString/	Intent/setAction/putExtra/	query/moveToNext/
4	getContext/getResources/	Intent/setData/	rawQuery/
5	getActivity/	Intent/setAction/	rawQuery/moveToFirst/moveToNext/

表 6: 利用法の利用数 top5(インスタンス取得文を除外した物)

	Context	Intent	Cursor
1	getString/	putExtra/	moveToFirst/
2	getResources/	setAction/putExtra	moveToFirst/close
3	startActivity/	setData	moveToNext
4	getPackageName	setAction/	moveToNext/getString
5	getSystemService/	addFlags	getCount/

して利用数が少ない利用法は利用数が多い利用法を拡張したものかどうかを検証する．それに加えて，利用数が少ない利用法が利用数が多い利用法の部分系列になっているかも検証した．これは，API の欠陥を検出する研究では，そのような部分系列となる利用法の存在が障害となっており，どの程度そのような利用法が存在するのかを確認するためである．各 API に対してこれらを検証するために，利用数が 10 回以上の利用法を頻出する利用法とみなして調査を行う．ただし，`Context` については全体の利用数，利用法の種類が多くないことから調査の対象外とする．また，RQ1 の結果から，多くのインスタンスを取得するメソッドが単独で利用されていることがわかっているので，ここでの調査ではインスタンスを取得するメソッドを除外した利用法について調査を行い，その結果を表 7 にまとめた．

表 7 の結果から，ほとんどの頻出利用法に対して，それを拡張した利用法が 10 個以上存在しており，頻出利用法の部分系列となる利用法はごく少数であった．またどちらにも該当しない利用法は `Cursor` で 15 個 (利用法の約 7 %)，`Intent` で 52 個 (利用法の約 24 %) であった．この結果から，ほとんどの利用法が頻出利用法を拡張したものであるといえる．ただし，`Intent` についてはどちらにも該当しない利用法が約 24 % 存在しており，基本的な利用法の拡張に限らない多様な利用法が乱立していることを示している．また，9 個の部分系列となる利用法は，そのうち 8 個が頻出している利用法であり，1 個だけが利用数の少ない利用法であり，関連研究〜で欠陥とみなされるような利用法は今回のケースではごく少数であった．該当なしに分類された利用法を確認すると，`Cursor`，`Intent` に共通する特徴は，頻出する利

表 7: 頻出利用法の拡張，部分系列数一覧

	頻出利用法	利用数	拡張	部分系列	該当なし
Intent	setAction/	36	30	-	52
	setAction/putExtra/	62	15	2	
	setData/	51	15	-	
	putExtra/	414	58	-	
	addFlags/	28	20	-	
	setType/putExtra/	24	14	-	
	setClass/setAction/	10	1	1	
	setType/	11	20	-	
	putExtra/addFlags/	13	10	2	
	putExtras/	13	1	-	
	setDataAndType/	10	14	-	
Cursor	moveToFirst/	16	67	-	15
	moveToNext/getString/	16	40	2	
	moveToFirst/close/	15	43	2	
	getCount/	12	28	-	
	moveToNext/	10	64	-	

用法と組み合わせて利用されることが多いメソッドが，単独で利用されているものが多い．例えば，getInt，getLong といったメソッドは，moveToFirst/getInt/,moveToFirst/getLong/ のように，別のメソッドと共に用いているケースが多い．しかし，該当なしの利用法では getInt/getLong,getLong のように単独で用いられていた．つまり，該当なしに分類される利用法は，追加処理的に用いられることの多いメソッドが単独で利用される場合もあることを示している．また，`Intent` においては該当なしに分類される利用法の割合が高い．この要因を確かめると，利用数の上位に入る利用法が全て set〜,put〜 といったデータを与えるメソッドで構成されており，`Intent` を利用する際はデータを与えるメソッドと取得するメソッドを同時に使うケースが非常に少ないことから，データを取得するメソッドを用いた利用法が該当なしに分類されていたためである．

RQ1:調査結果

RQ1 の調査結果を以下にまとめる．

RQ1:利用数が多い利用法，少ない利用法にどのような違いがあるのか?

- 一部の利用法が頻出する一方で，利用数が少ない利用法の種類が膨大になる傾向がある．
- 頻出利用法は大半の利用法の部分列で，基本的な利用法を表現している．
- 利用数が少ない利用法には，頻出利用法とメソッドが重複していない利用法が含まれている．

4.2 RQ2

RQ2 ではプロジェクト間の利用法の違いについて調査する．調査の手順として，まず各利用法がどれくらいのプロジェクトに利用されているかを確認する．これにより，利用法の利用数とプロジェクト数の相関関係や，どの程度プロジェクト固有の利用法がデータセットに含まれているか把握することができる．その後，各プロジェクトで利用される頻出利用法とプロジェクト固有の利用法の利用率を比較する．その結果から，データセットのプロジェクトの選び方が API 利用法の検出に与える影響を検討する．

利用法の利用プロジェクト数の調査

まず各利用法がどれくらいのプロジェクトに利用されているかを調査した．図 2, 図 3, 図 4 は，各点が利用法を表しており，縦軸がその利用法の利用数，横軸がその利用法を利用しているプロジェクトの総数のグラフである．`Context` と `Intent` については，利用法を利用しているプロジェクトが多いほど利用数が多いという傾向が強く，その相関係数は 0.9 以上である．ただし，利用しているプロジェクトが 1 つだけでも，利用数が 10 回程度の利用法も存在している．`Cursor` については，利用しているプロジェクト数に対して，利用法の利用数が満遍なく分布しており，他の 2 つにみられた傾向が特に当てはまらず，相関係数は 0.6 程度である．特に，利用法全体で利用数が上位に入るような利用法が 1 つのプロジェクトでしか使われていないケースもあり，プロジェクト固有の利用法が大きな割合を占めている．

プロジェクト間における利用法の利用傾向の違いの調査

次に，頻繁に利用される利用法とプロジェクト固有の利用法について，それぞれのプロジェクトで利用傾向に違いがあるのか，それぞれがどの程度利用されているのかを調査する．表 8 は，表 5 で示した各 API の利用数 top5 の利用法と，プロジェクト固有の利用法が各プロジェクトでどれくらい利用されているかを示した表である．表の結果を確認すると，

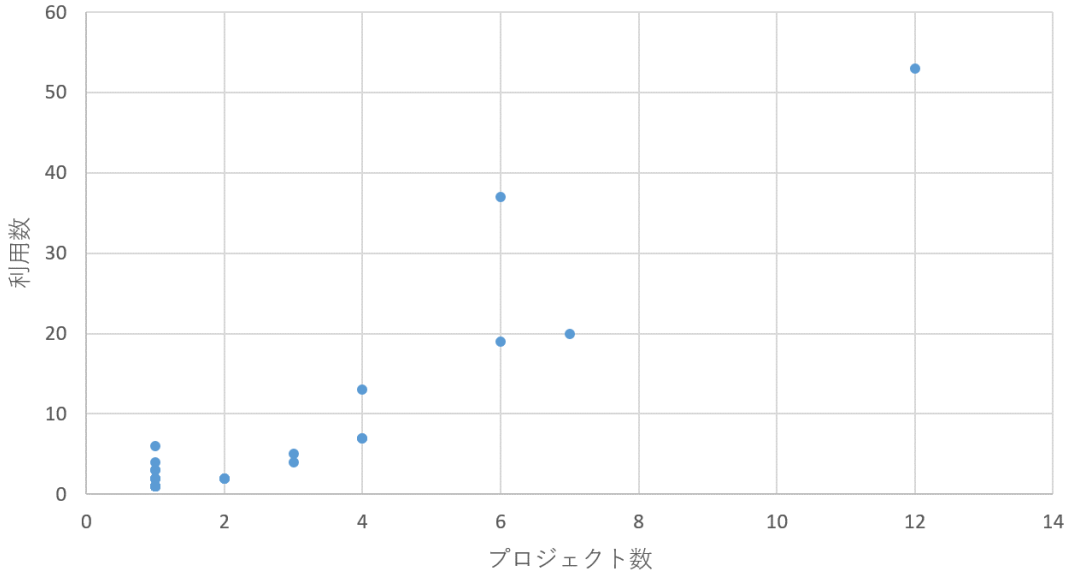

図 2: 利用法の利用数と利用しているプロジェクト数:Context

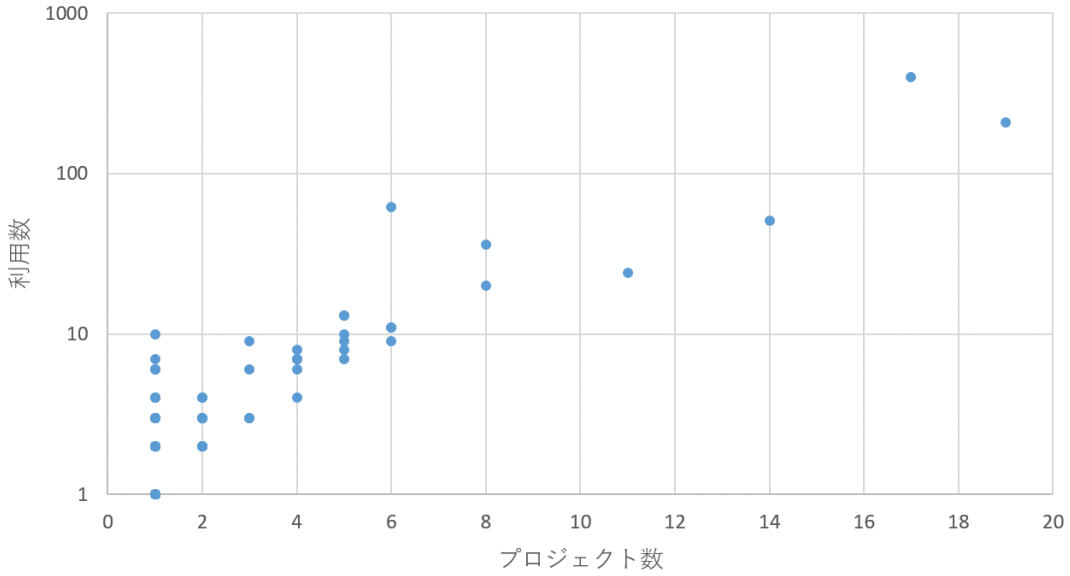

図 3: 利用法の利用数と利用しているプロジェクト数:Intent

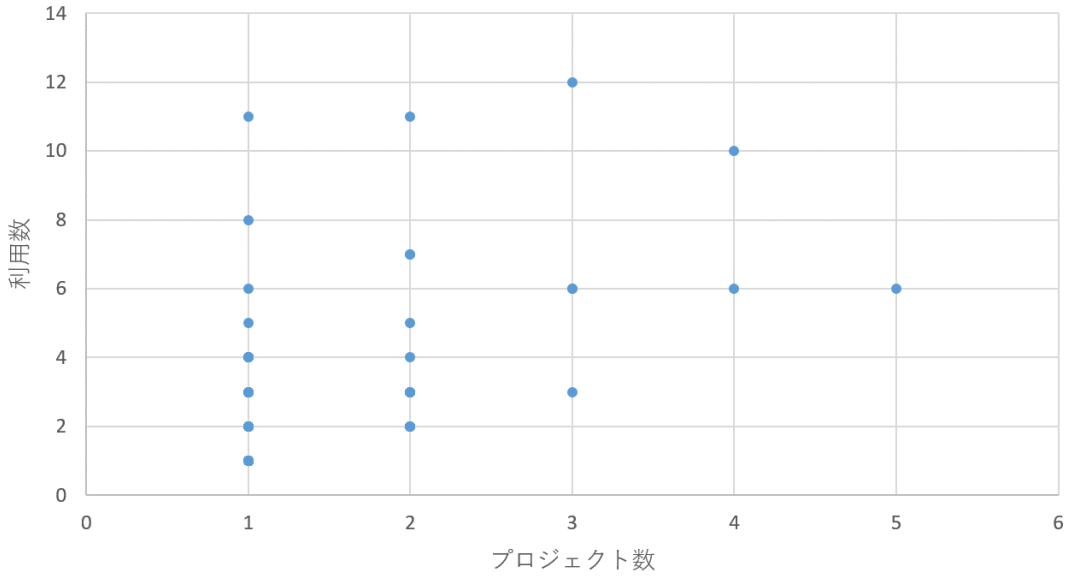

図 4: 利用法の利用数と利用しているプロジェクト数:Cursor

Context と Intent では，類似した傾向が見受けられる．利用数 top5 の利用法はプロジェクト全体で 60 % 以上の利用率で，プロジェクト個別でも，ある程度の利用数の母数があるプロジェクトでは高い割合を占めている．一方で，プロジェクト固有の利用法もプロジェクト全体で 20% 程度，プロジェクト個別で見ても大半のプロジェクトで 1 個以上利用されているなど，広い範囲である程度利用されていることを示している．これら 2 つの API に対して，Cursor については大きく異なる傾向を示している．プロジェクト全体でみると，利用数 top5 の利用法が 13%，プロジェクト固有の利用法が 71% と，他の API とは逆にプロジェクト固有の利用法が大半を占める結果になっている．プロジェクト個別でみたときもこの傾向は同様で，ほとんどのプロジェクトにおいてプロジェクト固有の利用法が大半を占めている．

RQ2:調査結果

RQ2 の調査結果を以下にまとめる．

表 8: top5 利用法の利用数とプロジェクト固有の利用法の利用数

API 名	Context		Intent		Cursor	
プロジェクト名	top5(%)	固有の利用法	top5(%)	固有の利用法	top5(%)	固有の利用法
全プロジェクト	142(65)	47(21)	754(61)	241(19)	52(13)	271(71)
AmazeFileManager	2(40)	1(20)	31(65)	6(12)	0(0)	15(65)
AndroidUtilCode	0(0)	10(100)	47(58)	15(18)	0(0)	0(0)
bitcoin-wallet	3(75)	1(25)	15(51)	10(34)	0(0)	1(100)
cgeo	2(50)	1(25)	68(72)	14(14)	3(20)	6(40)
codenameone	20(90)	1(4)	26(50)	15(29)	0(0)	6(54)
collect	0(0)	1(25)	45(54)	25(30)	0(0)	55(96)
ExoPlayer	2(66)	0(0)	4(36)	5(45)	0(0)	0(0)
fresco	7(100)	0(0)	4(57)	1(14)	0(0)	2(66)
k9	37(80)	5(10)	77(52)	45(30)	6(8)	49(68)
mapbox	5(62)	2(25)	1(100)	0(0)	0(0)	0(0)
news-android	0(0)	0(0)	32(80)	3(7)	2(20)	8(80)
owncloud	1(33)	2(66)	67(68)	8(8)	0(0)	12(80)
owntracks	1(100)	0(0)	5(29)	2(11)	0(0)	2(100)
photoview	0(0)	0(0)	1(100)	0(0)	0(0)	0(0)
realm-java	3(60)	1(20)	8(88)	0(0)	0(0)	0(0)
RedReader	6(60)	4(40)	34(52)	11(16)	0(0)	2(40)
robolectric	0(0)	0(0)	0(0)	1(100)	0(0)	0(0)
Signal-Android	20(71)	4(14)	102(61)	31(18)	9(11)	51(64)
skytube	0(0)	1(100)	23(76)	2(6)	0(0)	17(80)
wikipedia	5(45)	4(36)	24(51)	16(34)	10(55)	5(27)
WordPress	28(65)	9(20)	140(66)	31(14)	22(45)	40(83)

RQ2:プロジェクト間で利用法に違いはあるのか?

- Intent,Context では利用数が多い利用法ほど，多数のプロジェクトで利用される傾向にある．一方で，Cursor では 少数のプロジェクトでしか利用されない利用法が多くの割合を占めている．
- 頻出利用法はどのプロジェクトでも多数利用されていて，プロジェクト間で利用傾向に大きな違いは存在しない．

4.3 RQ3

RQ3 では，バージョンアップに伴う利用法の変遷の特徴について調査する．調査する内容は，プロジェクトのバージョンアップに伴う，利用法の利用数の変遷の傾向と，利用法の種類の変遷の傾向である．これらの変遷の情報を用いて，開発者にとって有益な利用法を評価することができるか検討する．

利用法の利用数の変遷の調査

まず，プロジェクトのバージョンアップに伴う利用法の利用数の変遷の傾向を調査する．図 5 は，Context について，2016 年 1 月から 2018 年 12 月までに約半年間隔でリリースされた 6 個のバージョンに対して利用法の利用数を古いバージョンから順に集計したものである．色分けされた部分は各利用法を表している．横軸はプロジェクトのリリース時期であり，左から順に古いバージョンのリリース時期を表している．縦軸は利用法の利用数を表している．主な特徴として，旧バージョンの時点で利用数が多い利用法は増加傾向であり，新しいバージョンにおいても利用数が多い．一方で，利用数が少ない利用法については，細かい増減を繰り返す傾向にある．この点についてさらに詳しく調べるために，利用数の少ない利用法の増減を表すグラフを図 6 に示す．このグラフは，Context の利用法のなかから利用数が 10 回未満の利用法を厳選して，バージョンアップに伴う利用数の増減を折れ線で表したものである．グラフを確認すると，多くの利用法が増減を起こしており，特に利用数が 0 回から数回，数回から 0 回に変化するケースが多い．この傾向は，Intent,Cursor でも同様である．

利用法の種類の変遷の調査

次に，プロジェクトのバージョンアップに伴う利用法の種類の変遷についてプロジェクトごとに調査を行った．表 9 は，各プロジェクトについて，データセットのうち最も古いバージョンで利用されている利用法の種類と，最も新しいバージョンで利用されている利用法の

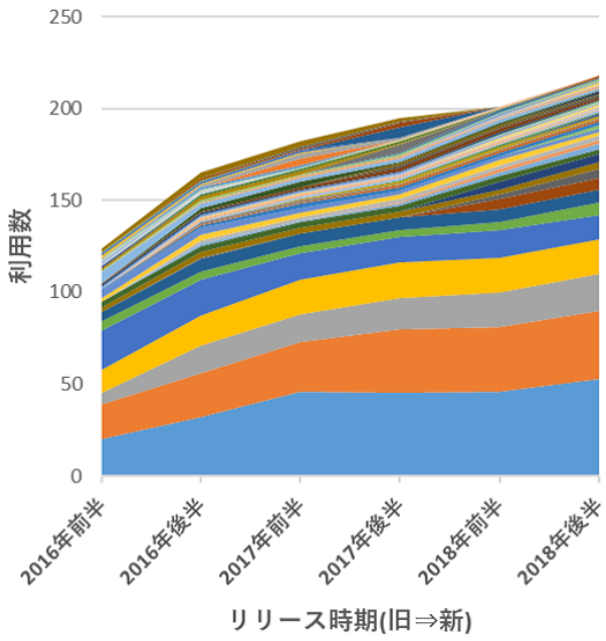

図 5: プロジェクトのバージョンアップに伴う各利用法の利用数の推移：Context

種類を比較したものである．基本的な傾向として，ほとんどのプロジェクトにおいて 旧バージョンに対して新バージョンは利用法の種類は増加している．全プロジェクトの合計で利用法の種類の変化をみると，どの API についても約 1.5 倍かそれ以上の増加をしており，プロジェクトのバージョンアップによるソースコード行数の増加は，利用法の種類の増加につながっていることが確認できる．一方で，一部のプロジェクト，API では利用法の種類の減少が起こっている．特に，WordPress における `Cursor` の利用法では，ソースコード行数が増加しているにも関わらず，利用法の種類が大幅に減少している．

ここで，インスタンスを取得するメソッドを除外した利用法の変遷についても同様に調査を行った．表 9 における括弧内の数値がその値になっている．表の結果を確認すると，いずれの API についてもインスタンスを取得するメソッドを含めた場合と比較すると利用法の種類があまり増加していない．つまり，インスタンスを取得するメソッドを含めた場合では種類の増加が多かったことと合わせると，利用法の種類の増加は，インスタンスを取得する新しいメソッドが追加されることで起きることが多いと推測できる．ただし，`Cursor` についてはインスタンスを取得するメソッドを除外した場合でもある程度の利用法の種類の増加が起こっている．

利用法の種類の増加はインスタンスを取得する新しいメソッドが追加されることで起きる

[3]括弧内はインスタンスを取得するメソッドを除外した利用法についての数値

表 9: プロジェクトのバージョンアップに伴う利用法の種類の変遷[3]

API 名	Context		Intent		Cursor	
プロジェクト名	旧	新	旧	新	旧	新
全プロジェクト	27(15)	44(18)	158(123)	213(144)	141(96)	206(136)
AmazeFileManager	0	4(3)	15(14)	17(15)	7(7)	9(9)
AndroidUtilCode	0	2(2)	6(6)	20(16)	1(1)	0
bitcoin-wallet	4(4)	4(4)	11(8)	12(8)	6(5)	1(1)
cgeo	1(1)	3(2)	26(20)	28(21)	10(10)	12(11)
codenameone	3(3)	6(4)	21(18)	24(19)	10(9)	10(9)
collect	1(1)	2(2)	14(14)	23(20)	9(7)	44(25)
ExoPlayer	3(3)	3(3)	4(4)	8(5)	0	0
fresco	0	3(2)	1(1)	5(4)	2(2)	3(3)
k9	8(5)	11(7)	36(33)	49(33)	27(24)	44(39)
mapbox	3(3)	4(2)	2(2)	1(1)	0	0
news-android	1(1)	1(1)	9(8)	10(9)	6(6)	6(6)
owncloud	0	2(2)	26(24)	25(24)	8(6)	9(6)
owntracks	0	1(1)	15(14)	10(10)	3(3)	2(2)
photoview	1(1)	1(1)	1(1)	1(1)	0	0
realm-java	2(2)	3(2)	3(2)	3(2)	0	0
RedReader	5(3)	5(3)	15(14)	21(19)	5(5)	4(4)
robolectric	1(1)	0	2(2)	1(1)	0	0
Signal-Android	8(5)	10(5)	31(23)	44(34)	16(8)	56(40)
skytube	0	1(1)	5(4)	9(8)	0	11(9)
wikipedia	1(1)	8(5)	14(9)	19(12)	3(3)	7(7)
WordPress	9(4)	13(7)	26(25)	41(33)	54(34)	22(21)

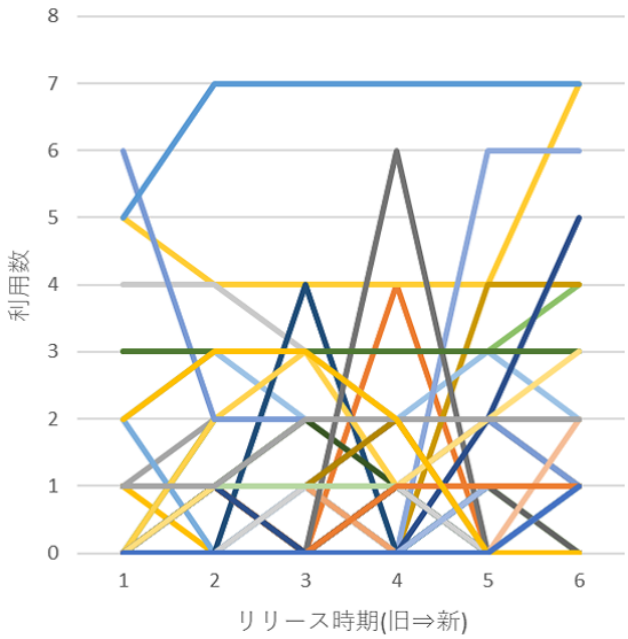

図 6: 少数利用法の利用数の推移：Context

ことが多いという推測を検証するために，新しく増えた利用法のうち，インスタンスを取得する新しいメソッドを用いた利用法がどれくらい存在するかを調査した．その結果，Intent においては 55 個のうち 34 個，Context においては 17 個のうち 5 個,Cursor については 65 個のうち 32 個の利用法がインスタンスを取得する新しいメソッドを用いていた．この結果から，多くのプロジェクトで，バージョンアップに伴って API のインスタンスを取得するメソッドを追加しながら API を利用するケースが多いことがわかる．さらなる調査として，インスタンスを取得するプロジェクト固有のメソッドはどの程度存在するのかを調べたところ，インスタンスを取得するプロジェクト固有のメソッドは Context で 4 個,Intent で 53 個，Cursor で 47 個あることがわかった．この結果は多くのプロジェクトがインスタンスを取得するプロジェクト固有のメソッドを定義して利用していることを示している．

最後に，各プロジェクトにおいて利用法の増減がどの程度起こっているかについて調査を行う．ただし，先ほど述べたように多くのインスタンスを取得するメソッドがプロジェクト固有のものであるため，ここではプロジェクト間の共通点を調査するためにインスタンスを取得するメソッドを除外した利用法を対象とする．図 7，8，9 は，点が各プロジェクトを表していて，横軸を利用法の減少数，縦軸を利用法の増加数としてプロットしたものである．グラフからわかることは，大半のプロジェクトにおいて，増加だけでなく減少も同時に起こっていること，そして合算すると増加傾向にあることがわかる．増加と減少が同時に起

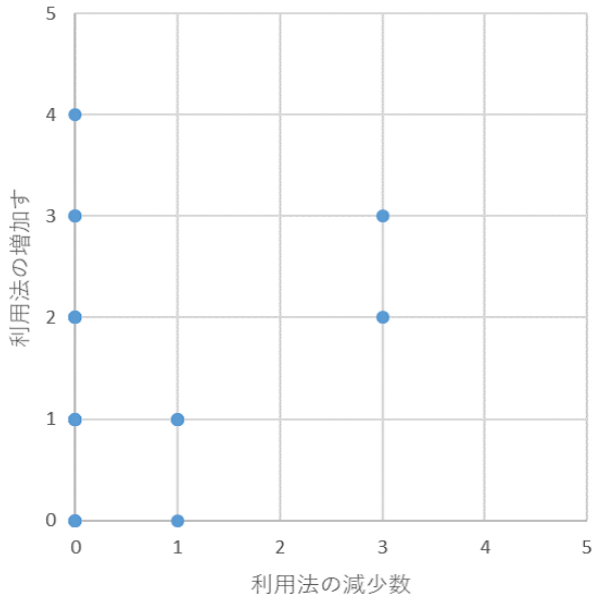

図 7: 各プロジェクトの利用法の種類の増減:Context

こっていることは，図 6 で示したような，利用法が消滅したり新しくできたりしている結果と合致する．

RQ3 の結果を以下にまとめる．

RQ3:バージョンアップに伴う利用法の変遷にどのような特徴があるのか?

- 旧バージョンにおいて利用数が多い利用法は，バージョンアップに伴い利用数が増加し続ける傾向にある．一方で，利用数が少ない利用法は増減を繰り返すものが多い．
- プロジェクトのバージョンアップに伴い，利用法の種類は増加する傾向にある．ただし，ほとんどのプロジェクトで増加だけでなく減少も同時に起こっている．
- 利用法の種類の増加の内訳は，インスタンスを取得するメソッドの増加によるものが多い．ただし，Cursor においてはそのような傾向は見られない．

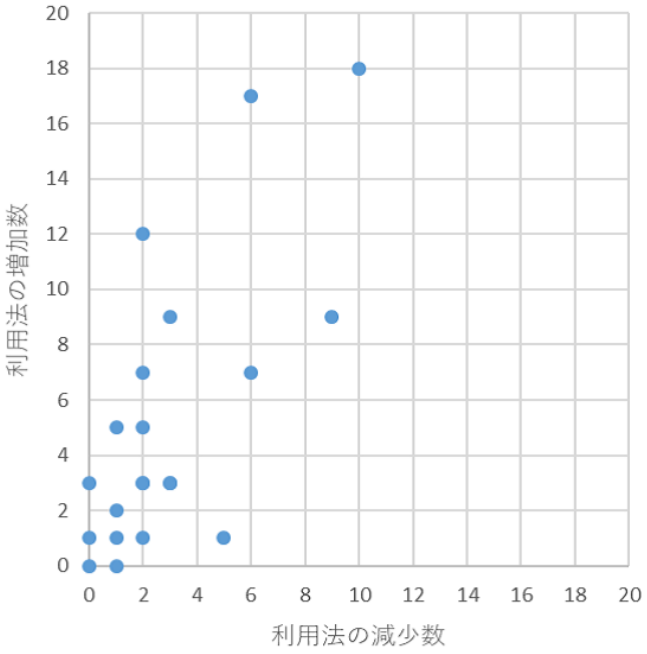

図 8: 各プロジェクトの利用法の種類の増減:`Intent`

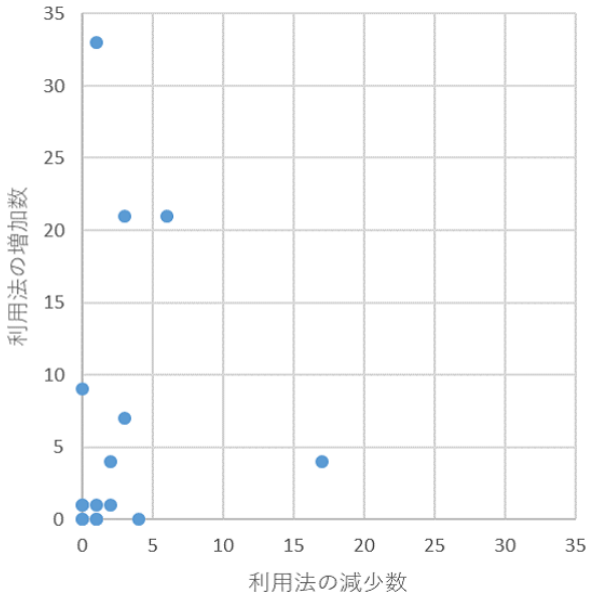

図 9: 各プロジェクトの利用法の種類の増減:`Cursor`

5　考察

リポジトリマイニングによる API 利用法の学習における懸念点として，利用法を一概に利用数で評価できないこと，リポジトリに含まれるプロジェクトの違いによる影響が把握できていないことを述べた．ここでは，4 章の実態調査の結果をもとに，上記の懸念点が与える影響について考察するとともに，プロジェクトのバージョンアップに伴う変遷の情報が API 利用法の評価に有用であるか検討する．

利用法の利用数による評価の影響

RQ1 で述べたように，頻出する利用法は大半の利用法の部分列であり，その API の基本的な利用法であるといえる．一方で，利用数が少ない利用法のには頻出する利用法に別のメソッドを追加したもの，あるいは頻出する利用法と共通のメソッドを利用していない利用法が存在する．この結果から，利用法をマイニングする際に利用法の利用数を用いて評価する場合，頻出する利用法に含まれていないメソッドを用いた利用法が十分に学習できない可能性がある．API 利用法のマイニングにおける考え方として，基本的な利用法だけでなく，その API を用いて実現できる処理を網羅的にカバーするように利用法を検出すること目指す場合がある．そのような場合，利用数が少ない利用法にも注目する必要があるといえる．

API 利用法をマイニングする際に考慮すべき点をまとめると，

知見

頻出利用法に注目することで基本的な利用法を提示できるが，その API で実現できる処理を幅広く提示したい場合，利用数が少ない利用法にも注目する必要がある

プロジェクトの違いによる影響

RQ2 では，頻出利用法は大半のプロジェクトで多数利用されていて，プロジェクト間で利用傾向に大きな違いが存在しない一方で，利用数が少ない利用法にはプロジェクト固有のものが多く存在することを述べた．特に `Cursor` においては，データセット全体でみるとプロジェクト固有の利用法が利用数の 70% ほどを占めていた．この結果は，データセットに用いるプロジェクトを変更することで，データセット中の利用法の種類，呼び出されるメソッドの種類や呼び出し数などの要素が大きく変更することを示している．特にデータセットに用いるプロジェクト数が少ない場合，どのようなプロジェクトを選ぶかによって含まれる利用法は大きく異なる．API 利用法をリポジトリマイニングにより検出する既存研究では利用法の利用数やメソッド同士の共起関係をもとに提示する利用法の決定を行うため，そのような要素の変更は検出結果を大きく左右する．

データセットに含まれるプロジェクトの違いの影響と開発者が考慮すべき点についてまとめると，

知見

API によってはプロジェクト固有の利用法の割合が高い傾向にあるため，プロジェクトの選び方によってデータセットに含まれる利用法の種類や割合は大きく異なる．そのため，利用法の利用数を用いたマイニングによる検出結果はデータセットに含まれるプロジェクトの種類や量によって大きく異なることを考慮しなければならない．

利用法の変遷の情報を用いた有用な利用法の評価

ここでは，利用法の変遷の情報が開発者に利用法の評価に有用であるか検討する．RQ3では，利用数が少ない利用法の利用数は増減を繰り返すものが多いこと，多くのプロジェクトでは利用法の種類が増加するだけでなく減少も同時に起こっていることを述べた．この結果から，一部の利用法は複数のプロジェクトで同時に増加したり減少したりしていると推測を行った．仮にそのような共通の変遷があった場合，利用法の評価に有用である可能性がある．例えば，複数のプロジェクトで同時に利用数が減少している場合，その利用法は何らかの理由で実装に適さなくなったり，開発者に好まれなくなっている可能性がある．このように，利用数の変遷の共通点から開発者により適した利用法を推薦できる可能性がある．この点について検証するために，各利用法についてプロジェクトの利用数の増減について調査した．図 11,10,12 は，点が各利用法を表しており，横軸をバージョンアップに伴いその利用法の利用数が減少したプロジェクト数，縦軸をその利用法の利用数が増加したプロジェクト数として点をプロットしたグラフである．各利用法の利用数に応じて点は色分けされている．各 API に共通する点は，利用数の多い利用法ほど多くのプロジェクトで増加傾向を示している．逆に利用数が少ない利用法は，増加傾向，減少傾向の双方にまばらに点在する傾向にある．しかし，一部の利用法では，利用している全てのプロジェクトで増加，または減少しているものが存在する．そこで，複数のプロジェクトで減少傾向を示している幾つかの利用法について，そのソースコードを確認して利用法が減少した要因を調査した．もしそれらの利用法の変更の要因が一致していれば，先ほど推測したように開発者に利用されなくなる利用法を発見するケースを示すことができる．調査として，`Intent` から 3 つの利用法，`Cursor` から 1 つの利用法について実装を確認した．調査した利用法と，その変更の要因を以下に示す．

- setNotificationUri/
  - メソッドを呼び出していた行が削除されたため
  - ファイルごと削除されたため
  - インスタンスが外部から与えられ，調査の対象外に変更されたため
- putStringArrayListExtra/
  - 利用法を利用していたソースコードがプロジェクトの外部ファイルへ移動したため
- setPackage/putExtra/
  - メソッドを呼び出していた行が削除されたため
  - ファイルごと削除されたため
- setType/setAction/
  - メソッドを呼び出していた行が削除されたため
  - メソッドが追加されて別の利用法に変更されたため

調査した結果，利用法が減少した要因はファイルごと削除されるなど，そもそも利用しているソースコードが消えてしまうものや，メソッドの追加，削除によって別の利用法に変更されたものなどが存在した．しかし，プロジェクト間で比較するとその要因はプロジェクト毎で別々であり，プロジェクト間で一貫した実装の変更が加えられたケースを確認することはできなかった．しかし，この調査では複数のプロジェクトで利用数が増加，または減少するケースを確認することができ，利用法をプロジェクトのバージョンアップに伴う変遷の情報から評価できる可能性を確認できた．

この調査の結果をまとめると，

知見

一部の利用数の少ない利用法では，利用するプロジェクトで共通して利用数が増加，または減少している．このようなプロジェクト間の共通した変遷の情報から，開発者から利用されやすい，または利用されにくい利用法を評価できる可能性がある．

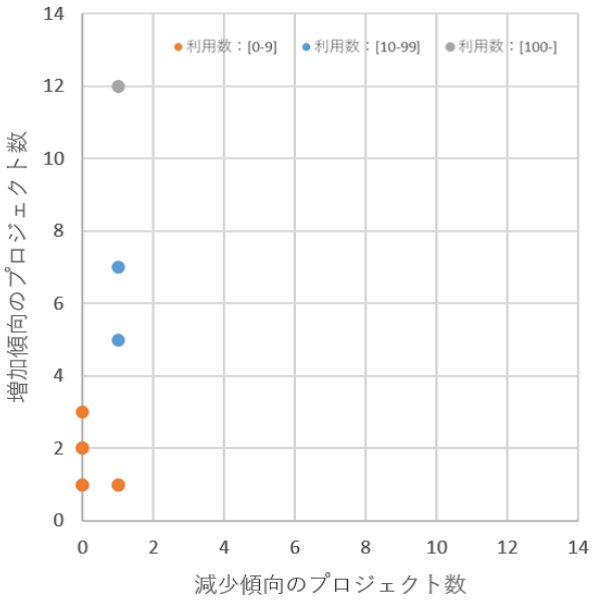

図 10: 利用法の利用数の変化の傾向:Context

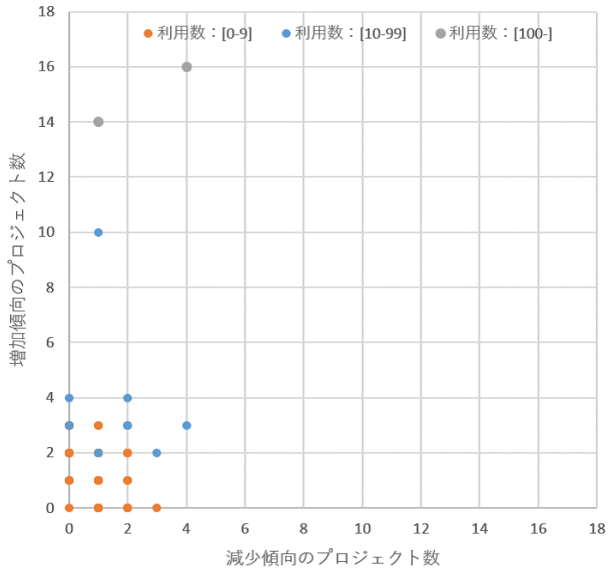

図 11: 利用法の利用数の変化の傾向:Intent

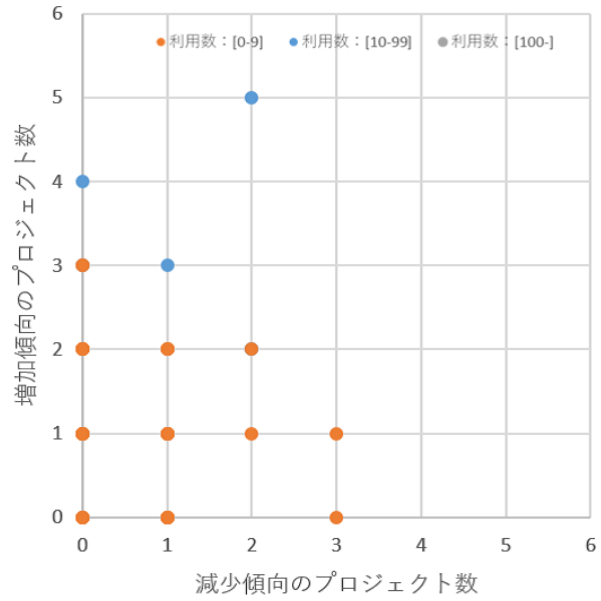

図 12: 利用法の利用数の変化の傾向:Cursor

6 まとめ

本研究では，Andoroid アプリケーションで構成したデータセットを対象に API 利用法の実態調査を行い，API 利用法のマイニングの際に考慮すべき点について考察した．

実態調査では 3 つの RQ について調査を行った．RQ1 では，一部の利用法の利用数が突出すること，またそれらの利用法にメソッドを追加することで大半の利用法が得られることを示した．RQ2 では，API によって調査結果が異なり，利用数が多い利用法ほど多数のプロジェクトで利用される傾向がある API，プロジェクト固有の利用法が多くの割合を占める API があることを示した．RQ3 では，利用数が多い利用法はバージョンアップに伴い利用数が増加し続ける傾向にあること，一方で利用数が少ない利用法は増減を繰り返す傾向にあることを示した．

実態調査により得られた知見をもとに，API 利用法のマイニングの際に考慮すべき点を以下にまとめる．

- API の利用法を網羅的に検出したい場合，利用数が少ない利用法にも注目する必要がある．
- API によってはプロジェクト毎に利用される利用法が大きく異なるため，データセットに選ぶプロジェクトによって API 利用法の検出結果は大きく変動する．
- 一部の利用法では，利用するプロジェクトで利用数の変遷に共通点がみられる．そのような共通点から，開発者にとって有益な利用法を推測できる可能性がある．

謝辞

大阪大学情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授には，研究に関して多くの助言を頂くだけでなく，井上　教授の監督のおかげで学生生活において多くの見識を深めることができました．井上　教授の御指導を受けることができ光栄に思っております．井上　克郎　教授に深く感謝いたします．

大阪大学情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には，その論理的な視点から，研究に関して的確な助言をいただきました．松下　准教授のおかげで本論文を完成させることができました．松下 誠 准教授に深く感謝いたします．

大阪大学情報科学研究科コンピュータサイエンス専攻 春名修介 特任教授には，研究に関して貴重な助言をいただきました．春名　特任教授のおかげで，自身の研究をよりよいものにすることができました．春名　修介特任教授 に深く感謝いたします．

大阪大学情報科学研究科コンピュータサイエンス専攻 神田 哲也 助教には，研究について何度も相談させていただきました．神田　助教の親身な担当指導のおかげでこの論文を完成させることができました．神田 助教の御指導が無ければ私の学生生活は成り立ちませんでした．神田　哲也助教に深く感謝いたします．

大阪大学情報科学研究科コンピュータサイエンス専攻事務職員である 軽部 瑞穂 氏には，いつも気さくに話しかけていただき，研究室生活をより楽しいものしていただきました．いつも研究室を支えてくださることに，心より感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上研究室の先輩方におきましては，たくさんの研究に関する御助言をいただきました．特に，大阪大学大学院情報科学研究科コンピュータサイエンス専攻　伊藤　薫氏，嶋利 一真 氏には，研究に関する助言，本論文の添削など，様々な場面でご協力していただきました．先輩方の御指導のおかげで本論文を完成させることができました，深く感謝いたします．

研究生活において大いにお世話になりました，大阪大学情報科学研究科コンピュータサイエン ス専攻 井上研究室の皆様に感謝いたします．

参考文献

[1] S. Amann, H. A. Nguyen, S. Nadi, T. N. Nguyen, and M. Mezini. A systematic evaluation of static api-misuse detectors. *IEEE Transactions on Software Engineering*, 2018.

[2] J. Businge, M. Openja, D. Kavaler, E. Bainomugisha, F. Khomh, and V. Filkov. Studying android app popularity by cross-linking github and google play store. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering*, 2019.

[3] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, Vol. 22, No. 1, pp. 73–84, 2013.

[4] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. Investigating next steps in static api-misuse detection. *Proceedings of the 16th International Conference on Mining Software Repositories*, p. 265–275, 2019.

[5] D. Hou and L. Li. Obstacles in using frameworks and apis: an exploratory study of programmers ' newsgroup discussions. *2011 IEEE 19th International Conference on Program Comprehension*, pp. 91–100, 2011.

[6] D. Konstantopoulos, J. Marien, M. Pinkerton, and E. Braude. Best principles in the design of shared software. *2009 33rd Annual IEEE International Computer Software and Applications Conference*, Vol. 2, pp. 287–292, 2009.

[7] Z. Li and Y. Zhou. Pr-miner: Automatically extracting implicit programming rules and detecting violations in large software code. *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 3016–315, 2005.

[8] T. McDonnell, B. Ray, and M. Kim. An empirical study of api stability and adoption in the android ecosystem. *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, 2013.

[9] M. Monperrus and M. Mezini. Detecting missing method calls as violations of the majority rule. *ACM Transactions on Software Engineering and Methodology*, Vol. 22,

No. 1, pp. 1–25, 2013.

[10] S. Moser and O. Nierstrasz. The effect of object-oriented frameworks on developer productivity. *Computer*, Vol. 29, No. 9, pp. 45–51, 1996.

[11] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen. Graph-based mining of multiple object usage patterns. *Proceedings of the 7th ACM Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, pp. 383–392, 2009.

[12] Android API Reference. https://developer.android.com/reference.

[13] M. P. Robillard. What makes apis hard to learn? answers from developers. *IEEE Software*, Vol. 26, No. 6, pp. 27–34, 2009.

[14] M. P. Robillard and R. DeLine. A field study of api learning obstacles. *Empirical Software Engineering*, Vol. 16, No. 6, pp. 703–732, 2011.

[15] P. C. Rigby S. Azad and L. Guerrouj. Generating api call rules from version history and stack overflow posts. *ACM Transactions on Software Engineering and Methodology*, Vol. 25, No. 4, 2017.

[16] M. A. Saied, O. Benomar, H. Abdeen, and H. Sahraoui. Mining multi-level api usage patterns. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering*, pp. 23–32, 2015.

[17] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. Mining succinct and high-coverage api usage patterns from source code. *2013 10th Working Conference on Mining Software Repositories*, pp. 319–328, 2013.

[18] W. Wang and M. W. Godfrey. Detecting api usage obstacles:a study of ios and android developer questions. *2013 10th Working Conference on Mining Software Repositories*, pp. 61–64, 2013.

[19] T. Xie and J. Pei. Mapo: Mining api usages from open source repositories. *Proceedings of the 2006 International Workshop on Mining Software Repositories*, No. 4, p. 54–57, 2006.

[20] 竹之内啓太, 石尾隆, 井上克郎. プログラミング言語の構造を考慮した api 利用例検索ツール. ソフトウェア工学の基礎 23, pp. 23–32, 2016.