2L-3

パターンに基づくフレームワーク API のリファクタリング方法の提案

高木 裕之† 河田 直人† 中道 上†‡ 青山 幹雄†‡

南山大学 数理情報学部 情報通信学科† 南山大学 情報理工学部 ソフトウェア工学科‡

1. 研究の背景と課題

フレームワーク API には，様々な理由で利用を制限する非推奨の変更が存在する．非推奨に変更された API(以下非推奨 API)にはその代替となる API(以下代替 API)が追加されるが，非推奨 API の検出と代替 API の置き換え方法が不明確であるため，リファクタリングが困難である．

2. 関連研究

2.1. API の呼び出し変更を用いた分析方法[1]

削除された API の代替 API を提示するレコメンデーションシステムを提案している．

2.2. バージョンアップに伴う変更のパターン化[2]

5 つのケーススタディを用いフレームワーク API のバージョンアップに伴う変更を分析し，既存のアプリケーションに影響を与える変更をパターン化している．

3. アプローチ

メタデータであるアノテーションの@Deprecated を用いて，非推奨 API を検出する．非推奨 API から代替 API への変更をパターン化し，パターン毎に置き換え方法を定義する．これによって非推奨 API のリファクタリングを支援する．

4. リファクタリング方法の提案

4.1. リファクタリング方法のプロセス

図 1 にフレームワークに存在する非推奨 API のリファクタリングプロセスを示す．本研究は Android V.3.2 を対象とした．

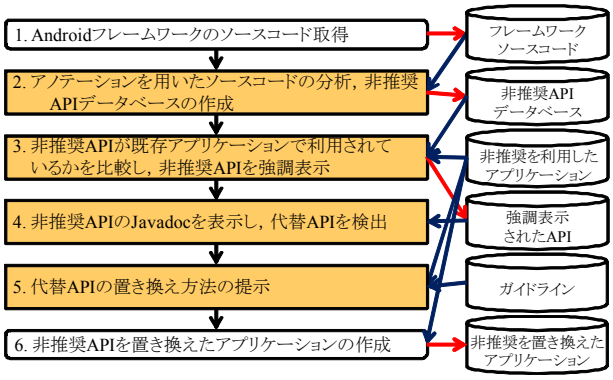


図 1 リファクタリング方法のプロセス

4.2. 非推奨 API の検出と代替 API の表示

Android アプリケーションフレームワークのソースコード内に存在する@Deprecated が付与された全ての API を検出し，API 名とソースコードのバージョン名をデータベースに格納する．非推奨 API データベースとアプリケーションを比較し，非推奨 API が使用されている場合は API を強調表示しマークすることで開発者に通知する．図 2 は比較から表示までの詳細なプロセスを示す．

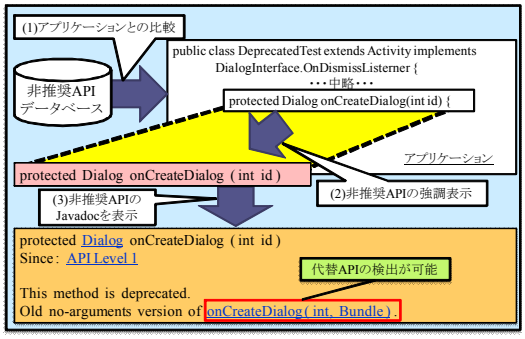


図 2 比較から表示までの詳細プロセス

4.3. 非推奨 API と代替 API の関係の分析

Android V.3.2 から検出した 208 個の非推奨 API の内，代替 API が提示されているものは 114 個であった(表 1)．代替 API が提示されている非推奨 API に対し，API の利用目的に変更が生じるかを分析したが，7 個の例外を除き利用目的の変更は無かった．そのため，パターンの前提条件として，非推奨 API に対する代替 API が存在し利用目的に変更がないものを対象とする．

表 1 代替 API が提示されている非推奨 API の数

代替APIが提示されている	代替APIが提示されていない	合計
114	94	208
54.8%	45.2%	100%

4.4. インタフェースの整合

非推奨 API を使用しているアプリケーションに対して代替 API を適用する際，インタフェースの変更を考慮する必要がある．インタフェースのシンタクスはシグネチャの変更に着目する．シグネチャは，メソッド名，パラメータの数と順序，パラメータの型及び戻り値の型を含む．インタフェースのセマンティクスは事前条件と事後条件の変更に着目する．

4.5. 整合の対象範囲と解決方法

非推奨 API と代替 API のシグネチャの変更に応じて事前条件と事後条件を整合する必要がある．よって，シグネチャの変更をパターン化し，それに応じた事前条件と事後条件の変更を示したガイドラインを作成する．

A Reafactoring Method of Framework API Based on Pattern
†Hiroyuki Takagi, Naoto Kawada, Dep. of Information and Telecommunication Eng., Nanzan University.
‡Noboru Nakamichi, Mikio Aoyama, Dep. of Software Engineering, Nanzan University.

4.6. 代替 API の変更パターンモデルの定義

バージョンアップに伴う変更のパターン化[2]に基づき代替 API の変更をパターン化した. 定義されている Change Argument Type, Change Return Type, Rename Method, Extra Argument に加え, 本研究ではパッケージの変更, クラスの変更, 修飾子の変更を追加した(図 3).

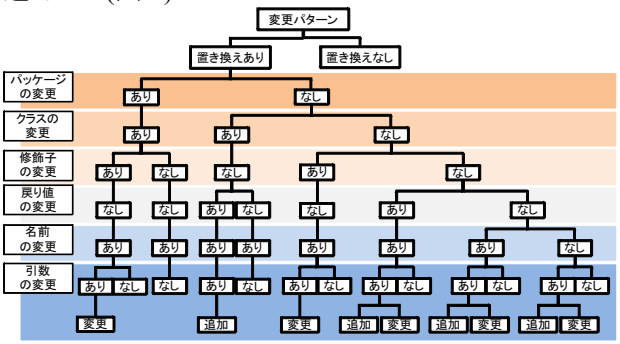


図 3 提案する変更パターンモデル

作成した変更パターンモデルに基づき非推奨 API のシグネチャを分析する. 分析によって確認できたシグネチャの変更の中で, 非推奨 API と代替 API の変更タイプが一致するものを詳細なパターンとして分類した(表 2).

表 2 シグネチャ変更の詳細パターン

変更パターン	変更パターンの詳細	非推奨APIとの比較
パッケージの変更		代替APIが存在するパッケージが異なる
クラスの変更		代替APIが存在するクラスが異なる
修飾子の変更	(a)abstruct > public	代替APIが抽象メソッドでなくなる
	(b)public static > public	代替APIでは動的な処理が必要になる
	(c)public > public static	代替APIでは静的な処理が必要になる
戻り値の変更	(a)void > int など	リソースの状態が増加
	(b)Object > void など	非推奨API実行後に, 開発者が記述する必要のあった処理が組み込まれる
	(c)リスト	戻り値がリスト構造に変更される
	(d)コンストラクタ	コンストラクタに処理が組み込まれる
	(e)String > Data	String以外の型が扱えるようになる
名前の変更		非推奨APIと代替APIの名前が異なる
引数の変更	(a)引数の追加	代替APIで引数が追加される
	(b)引数の変更	代替APIで引数が変更される
	(c)引数の削除	代替APIで引数が削除される

4.7. ガイドラインの作成

シグネチャ変更の詳細パターン毎に変更箇所, 非推奨 API へ移行した原因, 代替 API に移行する際の問題点とその対処法, 変更例を示した置き換え方法のガイドラインを作成した. 例として引数の追加に対するガイドラインを示す(表 3).

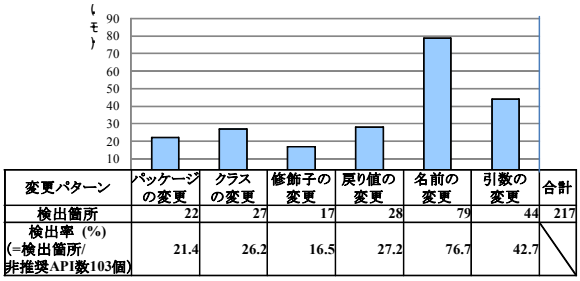
表 3 引数の追加に対するガイドライン

引数の変更	引数の追加
[変更箇所] 非推奨APIの引数の宣言に対して, 代替APIでは新たに引数が追加される. (代替APIは非推奨APIで宣言される引数を全て保持している)	
[原因] 非推奨APIで外部機能やリソースと連携する際に, それらの状態(連携できるか, データを取得できるか)を考慮していない. そのため非推奨APIでは実行時に予期せぬエラーが発生することがある. 代替APIでは, 連携先やデータの取得先の状態を追加することにより, エラーの回避が可能になる.	
[例] 非推奨API : public void setButton(CharSequence text, Message msg) 代替API : public void setButton(int whichButton, CharSequence text, Message msg) 非推奨APIではメソッドの処理の際, ボタンの種類を設定することができなかった. 代替APIではint whichButtonが追加され, Button_Positive, Button_Nutral, Button_Negativeの状態を設定することが可能になった.	
[移行への問題点] 代替APIで必要なパラメータが不足している.	
[対処方法] 追加される引数のパラメータが, 連携先やデータの取得先などの状態を表すものである. そのため代替APIを呼び出す場合は事前に連携先やデータの取得先などの状態を対応する引数の型で取得する必要がある.	

5. 提案するガイドラインの評価

5.1. ガイドラインの有用性の評価方法

有用性の評価として, Android V.3.2 に含まれる非推奨 API に適用可能であるガイドラインの割合を明らかにする. 図 4 で Android V3.2 に存在する変更パターンを示す. ガイドラインの適用は, 103 個の非推奨 API に含まれるシグネチャの変更箇所に対して行う. 変更箇所は非推奨 API に複数存在するため合計 217 箇所となる.



変更パターン	パッケージの変更	クラスの変更	修飾子の変更	戻り値の変更	名前の変更	引数の変更	合計
検出箇所	22	27	17	28	79	44	217
検出率 (%) (=検出箇所/非推奨API数103個)	21.4	26.2	16.5	27.2	76.7	42.7	

図 4 Android V.3.2 に存在する変更パターン

5.2. Android V.3.2 への適用結果

図 4 で示す 6 変更パターンの内, パッケージの変更, クラスの変更, 名前の変更はシグネチャの詳細パターンが存在しないため評価対象外とする. 表 4~表 6 は, 修飾子の変更, 戻り値の変更, 引数の変更パターンに含まれるすべての変更箇所に対し, 適用可能であったガイドラインの割合を示している.

表 4 修飾子の変更パターン(API 数 17)

「修飾子の変更」パターン						
	合計	シグネチャ変更のパターン			適用結果	
		abstruct > public	public static > public	public static > public Static	可能	不可能
検出箇所	17	7	9	1	17	0
内訳(%)	100	41.2	52.9	5.9	100	0

表 5 戻り値の変更パターン(API 数 28)

「戻り値の変更」パターン								
	合計	シグネチャ変更パターン					適用結果	
		void > int boolean > int void > boolean	view > void Object > void	リスト	コンストラクタ	String > Data	可能	不可能
検出箇所	28	13	3	2	2	3	23	5
内訳(%)	100	46.4	10.7	7.1	7.1	10.7	82.0	18.0

表 6 引数の変更パターン(API 数 44)

「引数の変更」パターン						
	合計	シグネチャ変更のパターン			適用結果	
		引数の追加	引数の変化	引数の削除	可能	不可能
検出箇所	44	19	13	4	36	8
内訳(%)	100	43.1	29.5	9.0	81.6	18.4

以上の結果より, 89 箇所の内 76 箇所(85.3%)の変更に適用することができた.

6. まとめ

Android のアプリケーションフレームワークを対象とした. 非推奨 API の検出とガイドラインを作成し, ガイドラインの有用性を評価した.

参考文献

(1) B. Dagenas, et al., Recommending Adaptive Changes for Framework Evolution, Proc. ICSE 2008, ACM, pp. 481-490.

(2) D. Dig, et al., How Do APIs Evolve? A Story of Refactoring, J. of Software Maintenance and Evolution, Vol. 17, 2006, pp. 1-26.