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

2008MI098 河田 直人 2008MI239 高木 裕之

指導教員 青山 幹雄

1. はじめに

フレームワークは，機能の向上，不具合の解消等を目的とするバージョンアップが行われる．しかし，バージョンアップに伴う API の変更により，既存のソフトウェアとの互換性が維持できなくなる可能性がある．本研究では Android アプリケーションフレームワーク[1]に着目する．

2. 研究課題

2.1. フレームワーク API の変更

Android のバージョン間に対するフレームワーク API を分析し，変更を以下の 4 つに分類した[6].

(1) 追加：新しい API が追加される
(2) 変更：API が変更される
(3) 削除：API が削除される
(4) 非推奨：既存の API の使用が制限される

2.2. 非推奨 API

非推奨 API は安全でない，バグが多い，処理が非効率的である，等の理由によって使用を制限された API である．非推奨 API は削除されることが前提であり，非推奨の代替となる API(以下代替 API)に置き換えられる．

2.3. API の変更を分析

(1) フレームワーク API の変更を分析

前述した変更を基に，V.2.1~V.2.2，V.2.2~V.2.3 のバージョン間で変更されたフレームワーク API を分析した結果を表 1 に示す．

表 1 フレームワーク API の分析結果

変更前→	変更後	追加	変更	削除	非推奨
2.1 →	2.2	254	7	1	18
2.2 →	2.3	505	7	1	6

分類した変更の内，追加は既存のアプリケーションに影響を与えない．また，変更，削除に関して，V.1.5~V.1.6，V.1.6~V.2.0，V.2.0~V.2.1 のバージョン間の変更も同様に分析をしたが，その数は非常に少なかった．また，非推奨の数に対して削除の数が少ないことが分かる．そのため，後続のバージョンにおいて非推奨 API が増加し続けていと考えられる．そのため，Android V.1.1~V2.1 での削除と非推奨の割合を更に分析した．

(2) 非推奨と削除の関係の分析

表 2 は Android V.1.1~V2.1 における削除と非推奨の数を示したものである．これにより，非推奨 API が削除されず残されていることが分かった．

表 2 バージョン間における削除と非推奨の数

変更前→	変更後	非推奨	非推奨クラス	削除	クラスの削除
1.1 →	1.5	44	3	4	0
1.5 →	1.6	12	4	2	2
1.6 →	2.0	27	37	0	0
2.0 →	2.1	2	0	0	0

上位の V.3.2 では非推奨 API を 208 個保持していることが確認できた．非推奨 API は今後のバージョンアップで削除される可能性がある．そのため，非推奨 API を代替 API に置き換える必要がある．

2.4. 解決すべき課題

(1) 非推奨 API の検出

Android のアプリケーション開発では Eclipse が推奨されている．しかし，アプリケーション内に非推奨 API が使用されていても検出することができない．そのため，非推奨 API を検出する必要がある．

(2) 代替 API の置き換え方法の明確化

代替 API 検出後，非推奨 API を代替 API にどのように置き換えるのか，API を比較するだけでは判断できない．そのため，非推奨 API を代替 API に置き換える方法を定義する必要がある．

3. 関連研究

3.1. API の呼び出しの変更を用いた分析方法[2]

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．変更セット内の呼び出しの違いから信頼度を計算し，代替 API を提供する．この分析は API が削除されることが前提であるため非推奨 API に対する代替 API の検出が困難である．

3.2. バージョンアップに伴う変更のパターン化[3]

5 つのケーススタディを用いフレームワーク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．しかし，パターン毎の置き換え方法が未定義である．

4. アプローチ

4.1. メタデータを用いた非推奨 API の検知

メタデータであるアノテーションを用いることで，非推奨 API を検出する．バージョンアップに伴い，非推奨に変更さ

れた API にはアノテーションの@Deprecated が付与される[4]. これは, Java のソース内で利用することが可能なアノテーションで, クラスや API 等に付与することで, それらが非推奨であることを示す. フレームワーク API のソースコードから@Deprecated が付与された API を判別することによって非推奨 API を検出する.

4.2. パターンを用いた代替 API 置き換え方法の提示

代替 API をアプリケーションに適用させるために, 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.

5. リファクタリング方法の提案

5.1. リファクタリング方法のプロセス

バージョンアップに伴う非推奨への変更に対し, 代替 API をアプリケーションに適用するための支援としてリファクタリング方法を提案する . 図 1 で提案するリファクタリング方法のプロセスを示す[7].

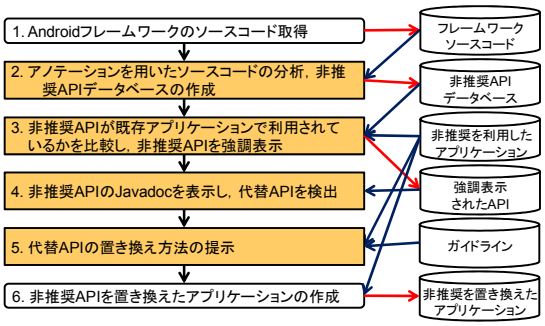

図 1 リファクタリング方法のプロセス

5.2. ソースコードの取得

Android のソースコードの取得は, Android Open Source Project で公式サポートが行われている OS の Ubuntu Linux を用いる. 本研究では, 日本語版が容易に入手できる Ubuntu 8.04 を VMware Player を用いて Windows Vista 上で動作させる.

5.3. ソースコードの分析と非推奨 API データベースの作成

Android ソースコードに存在する非推奨 API を検出し, 非推奨 API データベースの作成を行う. アノテーションを用いて非推奨 API をソースコードから検出し, 非推奨 API をデータベースに格納する.

5.3.1. フレームワーク API のデータモデル

ソースコードの分析を行うために図 2 フレームワーク API のデータモデルを示す. Android の各バージョン内には, 複数個のフレームワーク API が存在している. フレームワーク API は 0 個以上のアノテーションと Javadoc を保持しており, 非推奨 API, 代替 API, その他の API の 3 種類に分類できる[5].

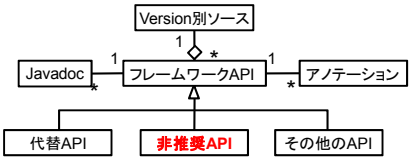

図 2 フレームワーク API のデータモデル

5.3.2. ソースコードの分析

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非推奨 API には, アノテーションである @Deprecated が付与される. この@Deprecated が記述された API をデータベースに格納する.

5.3.3. 非推奨 API データベースの作成

非推奨 API を格納するデータベースの構成を図 3 に示す. Android アプリケーションフレームワークを用いてアプリケーションを開発する際は, そのアプリケーションを動作させるターゲット(フレームワークのバージョン)を選択しなければならない[5]. ターゲット毎に非推奨 API は異なるため, データベースに格納する要素は非推奨 API 名と API が非推奨に変更した際のバージョン名とする.

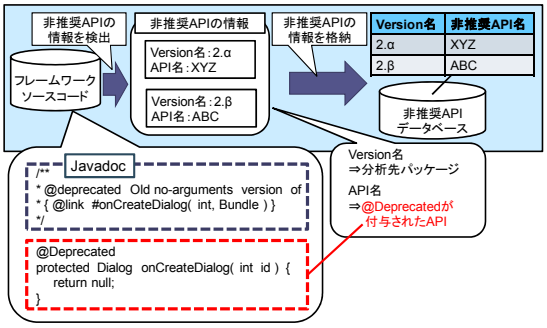

図 3 データベースの構成

5.4. 非推奨 API の比較と代替 API 表示

作成した非推奨 API データベースとアプリケーションを比較し, 一致する非推奨 API が存在すれば開発者に通知する機能を提供する. また, アプリケーションは Eclipse を用いて開発されることを前提する. そのため, 実現する機能は, Eclipse のプラグインを想定している. 図 4 に比較から表示までの詳細プロセスを示す.

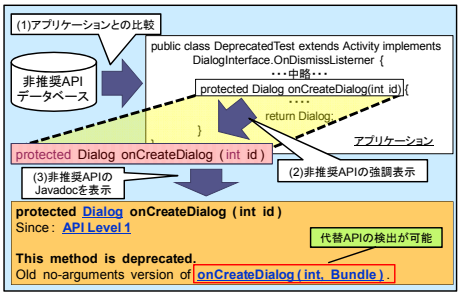

図 4 比較から表示までの詳細プロセス

5.4.1. アプリケーションとの比較

非推奨 API データベースに格納された非推奨 API 名と

バージョン名の情報を用いて，Eclipse で開発されたアプリケーションが非推奨 API を使用しているかを確認する．

5.4.2. 非推奨 API の強調表示

アプリケーションに存在する非推奨 API の検出後，ユーザにその使用を通知する．通知の方法には Eclipse の強調表示機能を用い，自動的に API の呼び出し部分をマークする．これにより，非推奨 API の強調表示が可能になり，アプリケーション開発者は非推奨 API の存在を視覚的に理解可能になる．

5.4.3. 非推奨 API の Javadoc 表示

強調表示によって非推奨 API の呼び出し部分がマークされる．開発者はアプリケーション内でマークされた非推奨 API にマウスカーソルを合わせることで，Javadoc を表示可能になる．

5.5. 代替 API の置き換え方法

非推奨 API と代替 API の関係を分析することによって，変更をパターン化する．この変更パターンを用いて置き換え方法を定義する．

本研究では，Android V.3.2 において確認した 208 個の非推奨 API を分析し，パターン化を行い，置き換え方法を定義する．

5.5.1. 非推奨 API と代替 API の関係の分析

V.3.2 に存在する 208 個の非推奨 API の内，Javadoc によって代替 API が提示されているものは 114 個であった(表 3)．さらに，代替 API が提示されている非推奨 API に対し，追加された代替 API の利用目的に変更がされているかを分析した．7 個の例外を除き利用目的の変更は無かった．そのため，パターン化の前提条件として，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 3 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%

5.5.2. インタフェースの整合

非推奨 API を使用している既存の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や事後条件の変更に着目する．

5.5.3. 整合の対象範囲と解決方法

本研究では非推奨 API と代替 API の利用目的に変化がないものを対象としている．そのため，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．これにより，代替 API の置き換えが可能になる．

5.5.4. 代替 API の変更のパターン

バージョンアップに伴う変更のパターン化[3]に基づき代替 API の変更をパターン化する．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．さらに，自動化を目的とするためメソッドの宣言に対応した順序で作成する必要がある．そのため，パッケージ，クラス，アクセス修飾子，修飾子，戻り値，API 名，引数の順序でパターンモデルを作成した(図 5)．追加以外のパターン(Change Argument Type 等)は変更として分類した．

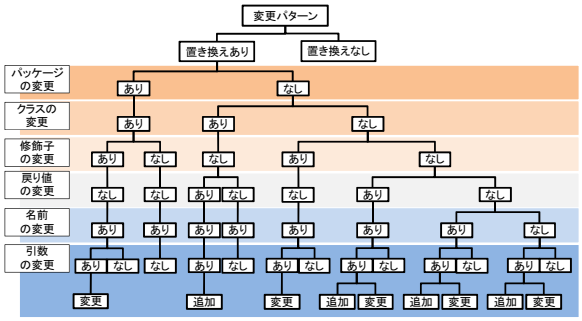

図 5 提案する変更パターンモデル

作成した変更パターンモデルのパッケージの変更，クラスの変更，修飾子の変更，戻り値の変更，名前の変更，引数の変更に基づき非推奨 API のシグネチャを分析する．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 4)．

表 4 シグネチャ変更の詳細パターン一覧

変更パターン	シグネチャ変更パターン	非推奨APIと代替APIの変更タイプ
パッケージの変更		存在するパッケージが異なる
クラスの変更		存在するクラスが異なる
修飾子の変更	abstruct > public	抽象メソッドでなくなる
	public static > public	動的な処理が必要になる
	public > public static	静的な処理が必要になる
戻り値の変更	void > int	リソースの状態が増加する
	boolean > int	
	void > boolean	
	view > void	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
	Object > void	
	リスト	戻り値がリスト構造に変更される
	コンストラクタ	コンストラクタに処理が組み込まれる
	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	引数の追加	引数が追加される
	引数の変更	引数が変更される
	引数の削除	引数が削除される

5.5.5. ガイドラインの作成

分類した変更パターンに含まれる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 5)．

表 5 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．	

6. 提案するガイドラインの評価

6.1. 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 6 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．

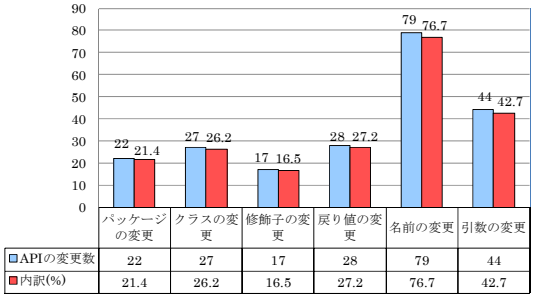

図 6 Android V.3.2 に存在する変更パターン

6.2. Android V.3.2 への適用結果

図 6 で示される 6 つの変更の内，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 6~表 8 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 6 修飾子の変更パターン(API 数:17)

	abstruct > public	public static > public	public > public static	合計
API数	7	9	1	17
内訳	41.2%	52.9%	5.9%	100%

表 7 戻り値の変更パターン(API 数:28)

	Void > int boolean > int void > boolean	View > void Object>void	リスト	コンストラクタ	String > Data	合計
API数	13	3	2	2	3	23
内訳	46.4%	10.7%	7.1%	7.1%	10.7%	82.0%

表 8 引数の変更パターン(API 数:44)

	引数の追加	引数の変更	削除	合計
API数	19	13	4	36
内訳	43.1%	29.5%	9.0%	81.6%

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．

7. 考察

既存の技術では，非推奨 API の検出が困難であること．また，変更パターンの置き換え方法が未定義であったため，非推奨 API のリファクタリングが困難であった[2][3]．本研究ではアノテーションを用いた非推奨 API の検出と非推奨 API を代替 API に置き換えるガイドラインを作成し，その手順を示した．これにより非推奨 API のリファクタリングが期待できる．

8. 今後の課題

非推奨 API はフレームワークに依存しているため，他のフレームワーク API を分析する必要がある．また，非推奨 API の検出機能とガイドラインを提供する機能を Eclipse のプラグインとして実装する必要がある．

9. まとめ

本研究では，Android のアプリケーションフレームワークを対象とし，非推奨 API のリファクタリング方法を提案した．非推奨 API の検出にはアノテーションを用い，代替 API の検出には Javadoc を用いた．代替 API の置き換え方法を明確にするために，変更パターンモデルを定義した．また，変更パターンに対する置き換え方法を示すガイドラインを作成し有用性を評価した．

参考文献

[1] Android Developers, http://developer.android.com.

[2] B. Dagenas, et al., Recommending Adaptive Changes for Framework Evolution, Proc. ICSE 2008, ACM, pp. 481-490.

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

[4] G.James, et al., JAVA Language Specification, Addison Wesley, 2005 [村上 雅章(監訳), JAVA 言語仕様 第 3 版, ピアソン・エデュケーション, 2006].

[5] 木南 英夫, Google Android アプリケーション開発入門, 日経 BP 社, 2009.

[6] R. E. Johnson, et al., パターンとフレームワーク, 共立出版, 1999.

[7] 結城 浩, Java 言語で学ぶリファクタリング入門, ソフトバンククリエイティブ, 2007.