Code Smellの深刻度がリファクタリングに与える影響の調査

雜賀 翼[1,a)]　崔　恩瀞[1,b)]　吉田 則裕[2,c)]　春名 修介[1,d)]　井上 克郎[1,e)]

概要：リファクタリングとは, ソフトウェアの外部から見た振る舞いを変えずに, 内部構造を整理する作業である. リファクタリングすべき箇所を特定することの支援の 1 つとして, 開発者へ Code Smell を提示することでソースコードの問題を可視化することが行なわれている. Code Smell とは, ソースコードの設計上の問題を示す指標であり, リファクタリングの実施が推奨されている. どのような Code Smell に対して実際に開発者がリファクタリングを実施するかを知ることは支援方法の考察に有用である. そのため, 各種類の Code Smell がリファクタリングの実施に与える影響の調査が行なわれている. しかし, 一方で Code Smell の示す問題の大きさを考慮した研究は現在までに行なわれていない. リファクタリングの実施にはコストがかかるため, Code Smell を修正するコストと効果は, リファクタリングを実施するかの判断に影響すると推測できる. この推測が正しければ, 同種類の Code Smell でも軽度なものと重度なものでは, リファクタリングが実施されるかの傾向が異なると考えられる. 本研究では, Java で書かれた 3 つのオープンソースソフトウェアの開発履歴を調査対象として, Code Smell 検出ツールが提示する深刻度という重要さの指標と, リファクタリングの実施された頻度の関係について調査を行なった. 一部の Code Smell について，リファクタリングされるクラスの深刻度がされないものと比べて有意に高いことがわかった.

キーワード：リファクタリング Code Smell オブジェクト指向プログラミングソフトウェア保守

1. はじめに

リファクタリングとは, ソフトウェアの外部から見た振る舞いを変えずに, 内部の構造を整理する作業のことである [1]. リファクタリングの主な目的は, ソースコードの可読性や保守性の向上である. また, リファクタリングを実施することで, 機能の追加や, バグの発見が容易になり, ソフトウェア開発の効率を向上させることができる [2].
しかし, リファクタリングの実施するべき状況については明確な基準は無く, 開発者はソフトウェアのどの部分をどのようにリファクタリングを実施するかを判断する必要がある. また, リファクタリングの実施にかかるコストや, 保守性への効果を事前に予測することは難しく, 実施の判断は開発者の経験によるところが大きい.

[1] 大阪大学
Osaka University, Suita, Osaka, 565-0871, Japan
[2] 名古屋大学
Nagoya University, Nagoya, Aichi, 464-8601, Japan
[a)] t-saika@ist.osaka-u.ac.jp
[b)] ejchoi@osipp.osaka-u.ac.jp
[c)] yoshida@ertl.jp
[d)] haruna@ist.osaka-u.ac.jp
[e)] inoue@ist.osaka-u.ac.jp

この問題を解決するために, Folwer は, リファクタリングの実施を判断材料として Code Smell を提案した [1]. Code Smell とは, ソースコードの設計上の問題を示す指標であり, Code Smell が存在する箇所にはリファクタリングの実施が推奨されている.
現在, 開発者へ Code Smell を提示することでソースコードの問題を可視化し, リファクタリングすべき部分を特定することが支援の 1 つとして行なわれている [3]. その提示方法や優先順位の考察には, 開発者が実際にリファクタリングを実施するのはどのような Code Smell かを知ることは有用だと考えられる.
そのため, 各種類の Code Smell がリファクタリングの実施に与える影響の調査が行なわれている [4]. しかし, 一方で Code Smell の示す問題の大きさを考慮した研究は現在までに行なわれていない. リファクタリングの実施にはコストがかかるため, Code Smell を修正するコストと効果は, リファクタリングを実施するかの判断に影響すると推測できる. この推測が正しければ, 同種類の Code Smell でも軽度なものと重度なものでは, リファクタリングが実施されるかの傾向が異なると考えられる.
本研究では, Java で書かれた 3 つのオープンソースソフト

ウェアの開発履歴を調査対象として, Code Smell 検出ツールが提示する深刻度という指標と, リファクタリングの実施された頻度の関係について調査を行なった.
調査の結果, 一部の種類の Code Smell について, リファクタリングが実施された Code Smell の存在するクラスと, リファクタリングされなかった Code Smell の存在するクラスとの間で, Code Smell の深刻度に有意差が見られた. このことから, 深刻度の高い Code Smell を優先的に提示する支援が有用だと考えられる.

以降, 2 節では調査対象のデータセットと調査手順を説明し, 3 節ではその調査結果を説明する. 4 節では本研究に関連する研究について述べる. 5 節では本研究のまとめと今後の課題を述べる.

2. 調査手法

2.1 調査対象のデータセット

本研究では Xerces-J と ArgoUML, Apache Ant という, Java で書かれた 3 つのオープンソースソフトウェアの, 合計 64 リリースバージョンを調査対象とした. データセットの概要を表 1 に示す.

これらは Bavota らの研究で調査対象となったデータセットであり, 本研究では彼らが検出したリファクタリングと, 検出元のリリースバージョンのソースコードを調査対象とする [4].

彼らは, Ref-Finder[5] というリファクタリング検出ツールを用いて, リリースバージョン間のソースコードの変化からリファクタリングを検出した. そして, Ref-Finder で検出されたリファクタリング 1 つ 1 つについて, それが本当にリファクタリングであるかの確認した結果を含めて検出結果を公開している. 本研究では彼らが正しく検出されたと判断したリファクタリングのみを調査対象とする.

検出されたリファクタリングの種類とそれぞれの検出数を表 2 に示す. 検出されたリファクタリング 1 つ 1 つについての情報は, リファクタリングの種類と, どのリリースバージョンのどのクラスから検出されたかの情報のみである.

2.2 Code Smell の検出方法

本研究では inFusion という商用の Code Smell の検出ツールを利用し, 調査対象の各リリースバージョンのソースコードから Code Smell を検出した [*1]. inFusion は Code Smell をソースコードのメトリクス値の特徴に基づいて抽出し, 24 種類の Code Smell を検出することが可能である. また, 検出された Code Smell には深刻度という, 1 から 10 までの整数値が付けられる.

inFusion を選択した理由は, 検出可能な Code Smell の種類の豊富なことと, 厳密な検出規則が定められていて検出の適合率の高いこと, そして深刻度によって Code Smell の重要さが表現されることである. また, inFusion は実際に企業等のソフトウェア開発現場で利用された実績がある.

本研究では inFusion の検出できる Code Smell のうち, 表 3 に示す 16 種類の Code Smell を調査対象とする. これらは Java のクラス単位またはメソッド単位で検出される Code Smell である. パッケージ単位で検出される Code Smell を除外する理由は, 調査対象のリファクタリングがクラス毎に検出されているのに対して, Code Smell からはパッケージ内のどのクラスに関係があるかが明確には表されないためである. その他には, Java を含むオブジェクト指向型言語からは検出されない Code Smell も除外している. 各種類の Code Smell の詳細な説明と検出規則は, inFusion のマニュアルに掲載されている.

2.3 調査手順

Code Smell の深刻度がリファクタリングに与える影響を明らかにするために, 以下の手順で調査を行なった.

(1) ソースコードから Code Smell を検出
調査対象の各リリースバージョンのソースコードから inFusion を利用して Code Smell の検出を行い, 各リリースバージョンについて Code Smell の一覧を得た. 検出した Code Smell のうちクラス単位またはメソッド単位の Code Smell を, 検出元のクラス毎に各種類の Code Smell の深刻度をまとめた.

(2) 相関分析
Code Smell の深刻度の大小がリファクタリングに影響するかを調べるために相関分析を行なった. Code Smell の深刻度の最大値と, リファクタリングの実施される割合との間でピアソンの積率相関係数を求めた.

(3) マン・ホイットニーの U 検定
リファクタリングが実施されたクラス郡と, リファクタリングされなかったクラス郡との間で, Code Smell の Code Smell の深刻度に有意差があるかどうかをマン・ホイットニーの U 検定で調べた. 検定のためのツールとしては, 統計分析ソフト R[*2] を用いた. Code Smell をもつクラスの深刻度の最大値と, Code Smell の種類毎の深刻度それぞれが, リファクタリングされるクラスの方が有意に高いかを知るために, マン・ホイットニーの U 検定は片側検定で行なった.

3. 調査結果

3.1 Code Smell の検出結果

データセットから検出された Code Smell を表 4 に示す. システム毎に各種類の Code Smell を持つクラスの数を表

[*1] http://www.intooitus.com/products/infusion

[*2] https://cran.r-project.org

表 1　調査対象のデータセットの概要

プロジェクト	対象期間	対象リリース	リリース数	クラス数	リファクタリング数	リファクタリングの種類数
Xerces-J	2003 年 10 月-2006 年 11 月	1.0.0-2.9.0	34	19,567	7,502	24
ArgoUML	2002 年 10 月-2011 年 4 月	0.10.1-0.32.2	12	43,686	3,255	21
Apache Ant	2003 年 8 月-2010 年 12 月	1.1-1.8.2	18	22,768	1,289	16
全体	-	-	64	86,021	12,043	28

表 2　調査対象のリファクタリングの概要

Type of Refactoirngs	Xerces-J	ArgoUML	Apache Ant	全体
add parameter	929	491	128	1,548
consolidate cond expression	196	45	32	273
consolidate duplicate cond fragments	474	95	72	641
extract method	184	135	73	392
extract superclass	0	13	0	13
form template method	0	10	0	10
inline method	83	22	24	129
inline temp	103	97	51	251
introduce assertion	0	14	0	14
introduce explaining variable	184	102	114	400
introduce null object	0	10	0	10
introduce parameter object	16	0	0	16
move field	1,183	389	64	1,636
move method	1,053	326	27	1,406
pull up field	11	0	0	11
pull up method	13	0	0	13
push down field	60	0	0	60
push down method	45	0	0	45
remove assignment to parameters	83	40	48	171
remove control flag	222	224	25	471
remove parameter	584	427	111	1,122
rename method	1,061	261	157	1,479
replace data with object	45	10	0	55
replace exception with test	18	0	0	18
replace magic number with constant	597	145	314	1,056
replace method with method object	201	367	36	604
replace nested cond guard clauses	137	32	13	182
separate query from modifier	20	0	0	20

している. 1 つのクラスからは複数の Code Smell が検出される場合があるため, Code Smell の数の合計の代わりとして 1 つ以上の Code Smell を持つクラスの数を表している.

また, 表 5 は, 1 つ以上の Code Smell を持つクラスについて, Code Smell の深刻度の最大値を求め, 深刻度の最大値毎に該当するクラスの数を表したものである.

3.2　深刻度の最大値とリファクタリング割合との相関分析

Code Smell を持つクラスの深刻度の最大値と, そのクラスがリファクタリングが実施された割合の関係を表 6 に表した. 各システムの Code Smell の深刻度の最大値が 1 から 10 までのクラスがリファクタリングされる割合を表している. 例えば, Xerces-J で深刻度の最大値が 10 のクラスがリファクタリングされる割合は 31.8%であった.

また, 1 から 10 までの深刻度の最大値とリファクタリングが実施された割合との相関を求めた. 相関係数は-1 から 1 までの値をとり, 1 に近いほど正の相関が強くなる. この場合の正の相関は深刻度の最大値が高いほどリファクタリングされる割合が高いことを示す. Xerces-J とデータセット全体では強い正の相関が見られたが, ArgoUML では弱い正の相関, Apache Ant では 0 に近く無相関という結果になった.

3.3　マン・ホイットニーの U 検定の結果

マン・ホイットニーの U 検定を行なった結果の p 値を表 7 に示す. p 値が 0.05 以下で有意差があるとされるものは黄色でハイライトしている. また, n/a と表記している部分は, その種類の Code Smell が検出されなかった場合も含めて, その種類の Code Smell のあるクラスに対してリファクタリングが実行されなかったものである.

表 3　調査対象の Code Smell の種類の説明

Code Smell の種類	検出単位	説明
Blob Class	クラス	コードの量が多く複雑なクラス
Data Class	クラス	フィールドと getter, setter 以外の機能を持たないクラス
Distorted Hierarchy	クラス	狭く深い継承階層にあるクラス
God Class	クラス	他のクラスと比べて責務の多過ぎるクラス
Refused Parent Bequest	クラス	親クラスから継承したものの一部しか利用しないサブクラス
Schizophrenic Class	クラス	鍵となる概念を複数持つクラス
Tradition Breaker	クラス	親クラスの提供する機能を打ち消してしまうサブクラス
Blob Operation	メソッド	コードの量が多く複雑なメソッド
Data Clumps	メソッド	複数個所で一緒に現れるデータ郡をパラメータに持つメソッド
External Duplication	メソッド	他のクラスの複数のメソッドと著しく重複するメソッド
Feature Envy	メソッド	所属するクラスとは違うクラスのデータを多く用いるメソッド
Intensive Coupling	メソッド	複数の他のクラスのメソッドの呼び出しが多過ぎるメソッド
Internal Duplication	メソッド	同じクラスの複数の他のメソッドと著しく重複するメソッド
Message Chains	メソッド	他のクラスから取得するオブジェクトが多過ぎるメソッド
Shotgun Surgery	メソッド	変更を行なうと複数のクラスに変更が必要になるメソッド
Sibling Duplication	メソッド	継承階層で兄弟関係にあるクラスに重複するコードがあるメソッド

表 4　プロジェクト毎の検出された Code Smell の数

Code Smell の種類	Xerces-J	ArgoUML	Apache Ant	全体
Blob Class	665	83	87	835
Data Class	71	3	28	102
Distorted Hierarchy	9	0	0	9
God Class	952	160	143	1,255
Refused Parent Bequest	114	14	81	209
Schizophrenic Class	39	48	4	91
Tradition Breaker	99	3	0	102
Blob Operation	2,428	545	413	3,386
Data Clumps	834	434	54	1,322
External Duplication	713	269	46	1,028
Feature Envy	723	110	220	1,053
Intensive Coupling	476	463	132	1,071
Internal Duplication	701	160	85	946
Message Chains	19	9	1	29
Shotgun Surgery	39	10	34	83
Sibling Duplication	927	545	131	1,603
Code Smell があるクラス数	3,907	1,992	897	6,796
Code Smell がないクラス数	15,660	41,694	21871	79,225
総クラス数	19,567	43,686	22,768	86,021

結果から, Schizophrenic Class と Blob Operation については有意差があると考えられる. しかし, その他の種類の Code Smell については, 各データセットで一貫した結果が出なかった. この原因の 1 つとしてデータの量が少なかったことが考えられるため, 今後更に調査対象を増やして調べる必要がある.

4. 関連研究

Code Smell がリファクタリングに与える影響についての既存研究の 1 つとして, Bavota らはリファクタリングが Code Smell のあるクラスに実施される割合と, リファクタリングによって Code Smell が完全に取り除かれる割合を調査した [4]. 彼らは Code Smell の種類毎にリファクタリングに与える影響を分析し, リファクタリングと Code Smell の間に明確な関係は見られなかったとしているが, Code Smell の深刻度については考慮されていなかった. Code Smell の修正にもコストがかかるので, 深刻でない Code Smell は修正されずに放置されるやすいと推測できる. また, 深刻度の低い Code Smell が問題視されにくいとすると, 開発者は Code Smell を完全に取り除くのではなく, 深刻度を和らげることを目標にして修正を行なうと考えられる. そのため, 本研究では Code Smell の深刻度について考慮して調査を行なった.

リファクタリングと Code Smell の関係についての研究

表 5　プロジェクト毎の検出された Code Smell の深刻度の最大値

深刻度の最大値	Xerces-J	ArgoUML	Apache Ant	全体
1	268	115	8	391
2	390	450	118	958
3	277	504	154	935
4	301	241	102	644
5	452	226	155	833
6	553	145	108	806
7	752	154	99	1,005
8	398	83	34	515
9	157	48	94	299
10	359	26	25	410
Code Smell があるクラス数	3,907	1,992	897	6,796

表 6　Code Smell の深刻度の最大値とリファクタリングが実施された割合

深刻度の最大値	Xerces-J	ArgoUML	Apache Ant	全体
1	0.4%	4.3%	0.0%	1.5%
2	6.4%	5.3%	4.2%	5.6%
3	12.6%	5.0%	4.5%	7.2%
4	8.0%	7.5%	5.9%	7.5%
5	6.4%	6.6%	0.0%	5.3%
6	12.1%	14.5%	11.1%	12.4%
7	17.2%	9.7%	5.1%	14.8%
8	27.1%	6.0%	8.8%	22.5%
9	20.4%	0.0%	1.1%	11.0%
10	31.8%	11.5%	0.0%	28.5%
相関係数	0.902	0.214	0.037	0.848

と近い内容で, リファクタリングとソフトウェア品質メトリクスの関係について研究が行なわれている [6], [7], [8], [9]. 本研究で調査した Code Smell の検出規則にはメトリクス値の条件に含まれているため, Code Smell の深刻度がリファクタリングに影響するように, メトリクス値の大小がリファクタリングに影響することが予想できる.

Szoke らの研究によると, 単一のリファクタリングは Code Smell などの保守性の問題をあまり改善しないが, 複数のリファクタリングを連続して実施することで保守性が大きく向上することができる [8]. また, 雑賀らの研究によると一度に複数種類のリファクタリングが組み合わせて実施される [10]. そのため, Code Smell の存在するクラスに対してどのようなリファクタリングが実施されるかを調査する場合には, 単一のリファクタリングではなくリファクタリングの組み合わせで調査するべきだと考えられる.

Yamashita らの研究によると, 複数種類の Code Smell が集中することは, 保守性を低下させたりバグの要因となるとされている [11]. 本研究では各種類の Code Smell の深刻度とその中の最大値について個別に調査したが, 複数種類の Code Smell の集中がリファクタリングにどのように影響するかは, 今後の研究課題である.

Tufano らの研究によると, あるクラスに Code Smell の発生するタイミングは, そのクラスが作成されたときの場合がほとんどである [12]. また, Kim らの研究によると, リファクタリングはリリースの直前に集中して実施される [13]. したがって, リリースバージョンのソースコードから Code Smell を検出した場合には, リリース間で発生した Code Smell の多くを見落としている可能性がある. 本研究ではリファクタリング検出の制限からリリースバージョン間に限定した調査を行なったが, 任意のリビジョンで調査可能なデータセットやリファクタリング検出ツールを探して調査する必要がある.

5. まとめと今後の課題

Code Smell の深刻度がリファクタリングに与える影響を明らかにするために, 3 つの Java のオープンソースソフトウェアの合計 64 リリースバージョンを対象に調査を行なった. クラス毎の各種類の Code Smell の深刻度と実施されたリファクタリングを元に, 相関分析とマン・ホイットニーの U 検定の 2 つの分析手法を用いて影響の分析を行った.

相関分析の結果, 一部のシステムにおいては強い正の相関が見られ, Code Smell のあるクラスの深刻度の最大値が高いほど, そのクラスがリファクタリングの実施される割合が高いことがわかった. 一方, マン・ホイットニーの U 検定の結果では, 一部の種類の Code Smell については

表 7　マン・ホイットニーの U 検定の p 値

	Xerces-J	ArgoUML	Apache Ant	全体
深刻度の最大値	$2.200*10^{-16}$	$3.967*10^{-3}$	$5.197*10^{-1}$	$2.200*10^{-16}$
Blob Class	$5.237*10^{-6}$	$8.820*10^{-1}$	$2.189*10^{-2}$	$1.251*10^{-9}$
Data Class	n/a	n/a	n/a	n/a
Distorted Hierarchy	$1.000*10^{0}$	n/a	n/a	$1.000*10^{0}$
God Class	$2.221*10^{-6}$	$9.129*10^{-1}$	n/a	$2.200*10^{-16}$
Refused Parent Bequest	$6.779*10^{-1}$	n/a	n/a	$1.740*10^{-4}$
Schizophrenic Class	$2.832*10^{-4}$	$1.644*10^{-4}$	n/a	$1.037*10^{-4}$
Tradition Breaker	$1.762*10^{-4}$	$1.000*10^{0}$	n/a	$7.306*10^{-10}$
Blob Operation	$2.200*10^{-16}$	$4.530*10^{-2}$	$3.403*10^{-3}$	$2.200*10^{-16}$
Data Clumps	$2.550*10^{-5}$	$9.940*10^{-1}$	$2.029*10^{-1}$	$2.200*10^{-16}$
External Duplication	$1.000*10^{0}$	$3.272*10^{-1}$	$9.445*10^{-1}$	$2.200*10^{-16}$
Feature Envy	$6.640*10^{-2}$	$3.640*10^{-1}$	$9.359*10^{-1}$	$2.921*10^{-14}$
Intensive Coupling	$1.179*10^{-1}$	$2.874*10^{-1}$	$2.617*10^{-1}$	$2.200*10^{-16}$
Internal Duplication	$6.917*10^{-1}$	$7.703*10^{-1}$	$3.306*10^{-1}$	$2.200*10^{-16}$
Message Chains	$6.668*10^{-1}$	n/a	n/a	$6.718*10^{-2}$
Shotgun Surgery	$6.576*10^{-1}$	$1.749*10^{-1}$	n/a	$2.046*10^{-2}$
Sibling Duplication	$7.338*10^{-1}$	$9.790*10^{-1}$	n/a	$3.970*10^{-13}$

リファクタリングされる Code Smell のあるクラスと，リファクタリングされない Code Smell のあるクラスとの間に，有意差が見られた．これらより，Code Smell の深刻度がリファクタリングに影響する可能性が示された．

調査結果には 3 つのシステムについて一貫なかった部分があるため，今後更にデータセットを増やして調査を行なうことが必要である．また，Code Smell の種類による重要さの違いや，複数の Code Smell が集中することによる影響を今後調査する予定である．

謝辞　本研究は JSPS 科研費 25220003，26730036 の助成を受けたものです．

参考文献

[1] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).
[2] Bavota, G., Carluccio, B. D., Lucia, A. D., Penta, M. D., Oliveto, R. and Strollo, O.: When does a Refactoring Induce Bugs? An Empirical Study, *Proc. of SCAM*, pp. 104–113 (2012).
[3] Emden, E. V. and Moonen, L.: Java Quality Assurance by Detecting Code Smell, *Proc. of WCRE*, pp. 97–106 (2002).
[4] Bavota, G., Lucia, A. D., Penta, M. D. and Oliveto, R.: An Experimental investigation on the Innate Relationship between Quality and Refactoring, *Journal of Systems and Software (in press)*, Vol. 107, pp. 1–14 (2015).
[5] Kim, M., Gee, M., Loh, A. and Rachatasumrit, N.: RefFinder: a refactoring reconstruction tool based on logic query templates, *Proc. of FSE*, pp. 371–372 (2010).
[6] Stroggylos, K. and Spinellis, D.: Refactoring–Does It Improve Software Quality?, *Proc. of WoSQ*, No. 10 (2007).
[7] Szoke, G., Antal, G., Nagy, C., Ferenc, R. and Gyimothy, T.: An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model, *International Journal of Software Engineering and Its Applications*, Vol. 5, No. 4, pp. 127–150 (2011).
[8] Szoke, G., Antal, G., Nagy, C., Ferenc, R. and Gyimothy, T.: Bulk Fixing Coding Issues and Its Effects on Software Quality: Is It Worth Refactoring?, *Proc. of SCAM*, pp. 95–104 (2014).
[9] Kannangara, S. H. and Wijayanake, W. M. J. I.: An Empirical Evaluation of Impact of Refactoring on Internal and External Measures of Code Quality, *International Journal of Software Engineering and Its Applications*, Vol. 6, No. 1, pp. 51–67 (2015).
[10] Saika, T., Choi, E., Yoshida, N., Goto, A., Haruna, S. and Inoue, K.: What Kinds of Refactorings are Co-occurred? An Analysis of Eclipse Usage Datasets, *Proc. of IWESEP*, pp. 31–36 (2014).
[11] Yamashita, A. and Moonenl., L.: Exploring the Impact of Inter-smell Relations on Software Maintainability: An Empirical Study, *Proc. of ICSE*, pp. 682–691 (2013).
[12] Tufano, M., Palomba, F., Bavota, G. and Oliveto, R.: When and Why Your Code Smell Starts to Smell Bad, *Proc. of ICSE*, pp. 403–414 (2015).
[13] Kim, M., Cai, D. and Kim, S.: An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution, *Proc. of ICSE*, pp. 151–160 (2011).