B-06

ソースコード履歴情報に基づくリファクタリングと欠陥の関係分析

An Analysis of Relationship between Software Refactorings and Defects in Development History

藤原　賢二 †　伏田　享平 †　吉田　則裕 †　飯田　元 †
Kenji Fujiwara　Kyohei Fushida　Norihiro Yoshida　Hajimu Iida

１．　はじめに

ソフトウェアの設計品質を向上させる技術として，リファクタリングがある．リファクタリングとは，ソフトウェアの外部的な振る舞いを変更することなく，内部の構造を改善することを言い，Fowler によって典型的なリファクタリングパターンがまとめられている [4]．Fowler は，リファクタリングの効果としてソフトウェアの欠陥が少なくなると述べている．リファクタリングを行うことが欠陥の発生を抑えるなら，リファクタリングを定常的に行うことでソースコードの品質を向上させることができ，逆にリファクタリングを行っていない場合は，品質の悪い部分が残存すると考えられる．そこで，本研究では，リファクタリングを行った場合の影響に加えて，行わなかった場合の影響を通して，リファクタリングを定常的に行うことの必要性を評価する．

リファクタリングが欠陥に与える影響を評価するに当たって，リファクタリングを行った場合の影響と，リファクタリングを行わなかった場合の影響を計測する．初めに，リファクタリングを行った場合の影響を考える．Fowler は，リファクタリングを行うことでソースコードを，開発者が容易に理解可能なものに改善でき，変更に対して柔軟に対応できるように設計を改善することができると述べている [4]．一方，欠陥の発生要因として，保守作業における人為的なミスが挙げられる．これは，開発者のソフトウェアに対する理解不足や，要求される変更作業が煩雑であることに起因する．これらを踏まえると，ソフトウェア開発においてリファクタリングを定常的に行い，ソースコードを常に理解し易く，変更の容易な状態に保つことで，欠陥の発生を抑えることができると考えられる．そこで，我々は次に示す仮説を立てた．

仮説１　リファクタリングが定常的に行われている開発プロジェクトは欠陥の発生率が低い

この仮説を検証することで，欠陥の発生率という観点からリファクタリングがソフトウェアの品質を向上させるかどうかを確認することができる．

† 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

次に，リファクタリングを行わなかった場合の影響について考える．リファクタリングを行わずにいると，先ほど述べた改善が行われないため，ソースコードが次第に可読性が低く，変更が容易でない状態になると考えられる．このようなリファクタリングが要求される状態であるかを判断する基準として，“重複したコード” や，“長すぎるメソッド” などの，“コードの不吉な匂い [4]” が Fowler によって提案されている．そして，リファクタリングはこの不吉な匂いを除去するために行われるため，リファクタリングが必要であるにも行われていないソースコードには，不吉な匂いが長い期間存在すると考えられる．そこで，我々は次に示す仮説を立てた．

仮説２　不吉な匂いの滞留期間が長い開発プロジェクトは欠陥の発生率が高い

この仮説が正しい場合，リファクタリングが必要であるにも関わらず，リファクタリングを行わずに開発を続けると欠陥の発生率が高くなることが分かる．このことから，この仮説を検証することで，リファクタリングの必要性を確認することができると考える．

本研究では，**仮説１**及び**仮説２**を検証するために，版管理システムに記録されたソースコード編集履歴から，リファクタリング履歴とリファクタリングの契機となるコードの不吉な匂いの検出箇所の変遷を抽出する．そして，両仮説を検証することで，ソフトウェア開発におけるリファクタリングと欠陥の関係を分析する．版管理システムとは CVS[2] や Subversion[8] のようにソフトウェアの構成管理に用いられるシステムである．また，バグ管理システムは開発プロジェクトにおいて，欠陥を集中管理し，各欠陥の修正状況を追跡するためのシステムであり，代表的なものに Bugzilla[1] や Trac[9] がある．

本稿では，リファクタリング履歴およびコードの不吉な匂いと欠陥の関係を分析する手法を提案する．以降，２節では本研究において重要な用語であるコードの不吉な匂いについて説明し，実験で用いるリファクタリング検出手法について説明する．また，本研究と同様に，リファクタリングと欠陥の関係について分析している先行研究を紹介し，本研究との違いを述べる．そして，３節で先に述べた仮説を検証するための実験方針について説明した後，実験において測定する各値の定義と測定方法

について述べる．最後に，４節でまとめと今後の展開を述べる．

２． リファクタリング

2.1 コードの不吉な匂い

リファクタリングがどのようなときに要求されるかを判断する厳密な基準は定義されていないが，先に述べた通り，Fowler はリファクタリングが要求される可能性のあるいくつかの兆候としてコードの不吉な匂いを定義している．通常，開発者は不吉な匂いを見つけた場合，自身の経験に従ってリファクタリングを行うかどうかを決定する．この過程の属人性を軽減するために，“重複したコード” や “長すぎるメソッド” など一部の不吉な匂いに対して，リファクタリングの必要性を評価するためのメトリクスを定義し，それらを用いてリファクタリングを支援する手法が提案されている [13, 14].

2.2 リファクタリング検出手法

リファクタリングの有効性や実施状況について分析する際は，“いつ”，“どのような” リファクタリングが行われたかを知る必要がある．この要求に対するアプローチとして，

(a) 版管理システムに記録されたコミットログを用いる
(b) ソースコードの編集履歴を解析する
(c) 開発者の行動を監視する
(d) リファクタリングツールの使用を記録する

の４つのアプローチがそれぞれ研究されている [5]．(a) は，開発者が版管理システムに変更をコミットする際に “refactor” や “extract”，“rename” など，リファクタリングに関する単語をログメッセージとして残している場合，その変更でリファクタリングが行われたと判断する手法である．(b) は，異なるバージョンにおけるソースコード間の差違を解析することで，“変数名の変更” や “メソッドの抽出” などのリファクタリングを検出する．(c) は，開発者が “どのように” リファクタリングを行うかを，直接またはツールを使って監視する．このアプローチは適用範囲が限られるが，必要な情報を詳細に集めることが可能である．(d) は，統合開発環境が提供するリファクタリング支援機能をユーザが “いつ “，“どのような場所に” 使用したかを記録することで，リファクタリングに関する情報を集める．

(a)，(b) は既存の版管理されているソフトウェア開発プロジェクトに適用できるが，過去の履歴を用いた推定であるため，検出精度が低い．(c)，(d) は (a)，(b) と比べて高精度にリファクタリングを検出可能だが，実施に準備が必要なため，実施中のプロジェクトには適用できない．本研究では，(a)，(b) のアプローチによってリファクタリングを検出する．

2.3 欠陥との関係を分析している研究

リファクタリングと欠陥の関係を分析している研究として，Ratzinger らはコミットログを用いてリファクタリングを検出し，欠陥との関係を分析している [6]．彼らは分析の結果，リファクタリングは欠陥の発生を抑える効果があると述べている．しかし，リファクタリングを行わず品質の低いソースコードが残存した状態で開発を続けた場合の影響については評価していない．本研究の手法では，不吉な匂いを判断基準としてリファクタリングの必要性があるにも関わらず，リファクタリングを行っていない場合の欠陥への影響を評価する．また，本手法ではリファクタリングの検出手法として，コミットログを用いた手法以外に，ソースコードの変更履歴を解析する手法も利用する．

３． 実験計画

実験では，**仮説１**を検証するためにリファクタリングが定常的に行われているかどうかを調べる．また，**仮説２**を検証するために不吉な匂いの滞留期間を調べる．そして，それらと欠陥の発生頻度の間で相関を取ることで両仮説を検証する．実験の概観を図１に示す．

3.1 リファクタリング頻度の測定

仮説１を検証するには開発プロジェクトにおいて “リファクタリングが定常的に行われているかどうか” を定量的に評価する必要がある．その指標として，実験では一定期間にどれだけリファクタリングが行われたのかを示す，リファクタリング頻度を測定する．版管理システムに記録されたソフトウェアの全リビジョンを V，各リビジョンを v_i とすると，$V = [v_1, v_2, ..., v_n]$ と表せる．次に，リビジョン v_i から v_{i+1} の間にソースコードに対して行われた変更を op_i とし，op_i においてリファクタリングが行われたかを返す $r(op_i)$ を次のように定義する．

$$r(op_i) = \begin{cases} 1 & (\text{if } op_i \text{ is refactoring}) \\ 0 & (\text{otherwise}) \end{cases}$$

$r(op_i)$ は，op_i においてリファクタリングが行われた場合は 1，行われていない場合は 0 を返す．$r(op_i)$ を用いて，リビジョン v_j から v_k におけるリファクタリング頻度 $f_r(j, k)$ を次のように定義する．

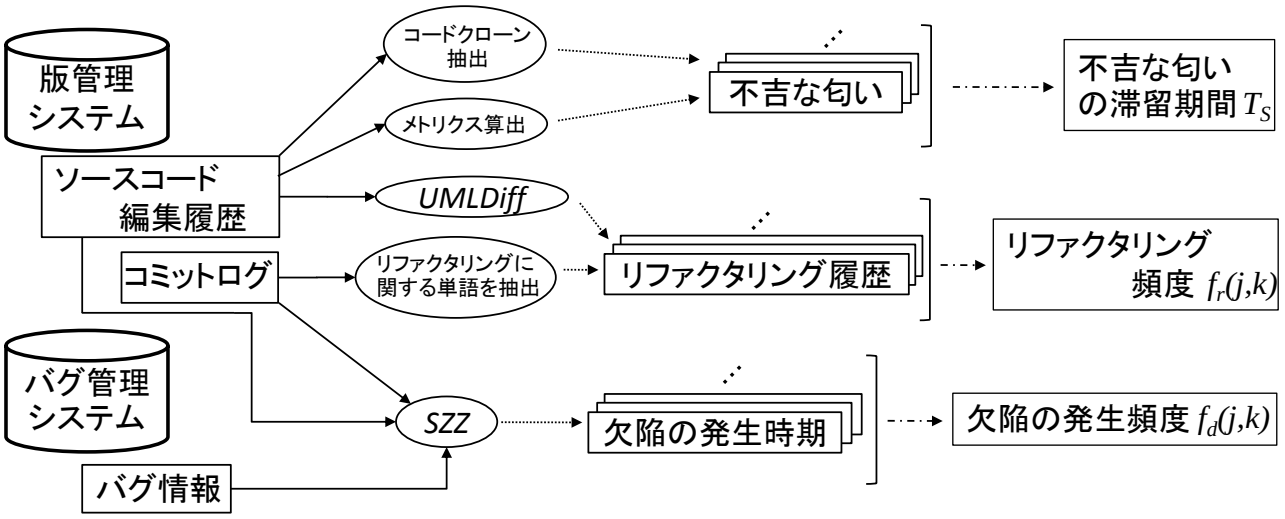

図 1: 実験の概観

$$f_r(j,k) = \frac{\sum_{i=j}^{k} r(op_i)}{v_k - v_j} \qquad (j < k, \quad v_j, v_k \in V)$$

$r(op_i)$ の測定には，バージョン間の設計情報の差分を抽出する *UMLDiff* アルゴリズム [10] を用いてリファクタリングを検出する手法 [11] と，コミットログを利用した手法 [6] を利用し，それぞれの手法から求めたリファクタリング頻度と後述する欠陥の発生頻度について相関を求める．

3.2 不吉な匂いの滞留期間の測定

まず，本実験で扱う不吉な匂いを定義する．吉田らは不吉な匂いの一つである “重複したコード” の検出をコードクローン検出技術により行う手法を提案している [14]．また，三宅らはリファクタリングの一つであるメソッド抽出の必要性を評価するためのメトリクスを算出する手法を提案している [13]．本実験ではこれら二つの手法を用いて得られるコード断片を不吉な匂いとして，その滞留期間を測定する．

リビジョン v_i と v_i のソースコード中に存在する不吉な匂いの集合 S_i との関係を

$$\begin{aligned} s(v_i) &= S_i, \\ S_i &= \{smell | smell = (classID, \\ &\qquad beginLine, endLine)\} \end{aligned}$$

と定義する．なお，ここでは不吉な匂い（$smell$）を不吉な匂いであるコード断片が記述されているクラスを一意に特定できる値（$classID$），不吉な匂いの開始行（$beginLine$），不吉な匂いの終了行（$endLine$）の組で表すものと定義している．次に，リビジョン v_i における，ある不吉な匂い $smell_a$ に対して，同一のコード断片を示している $smell_b$ がリビジョン v_{i+1} に存在するとき，$smell_a$ と $smell_b$ には履歴関係があると定義する．そして，履歴関係にある $smell$ の集合 S に対応する滞留期間 T_S を次のように定義する．

$$\begin{aligned} T_s &= v_{sd} - v_{so}, \\ v_{so} &= \min(\{v \in V | s(v) \cap S \neq \phi\}), \\ v_{sd} &= \max(\{v \in V | s(v) \cap S \neq \phi\}) \end{aligned}$$

v_{so} は，不吉な匂いの発生時期，v_{sd} は消滅時期を表している．v_{so}，v_{sd} を求めるには，あるリビジョンに存在する不吉な匂いが，別のリビジョンに存在するものと履歴関係にあるかを調べる必要がある．“重複したコード” については，川口らがコードクローンの履歴関係を抽出する手法 [12] を提示しており，この手法を応用して “重複したコード” とメソッド抽出における不吉な匂いの履歴関係を抽出する．

3.3 欠陥の発生頻度の測定

一定期間内における欠陥の発生数を欠陥の発生頻度とする．欠陥の発生時期の抽出方法として，SZZ アルゴリズム [7] を用いる．SZZ アルゴリズムは，バグ管理システムに記録されたバグ情報とソースコードの編集履歴を対応付けることで，欠陥の発生要因となったコードの修正を特定する．この修正が行われた時期を欠陥の発生時期とし，変更 op_i において，欠陥が発生したかどうかを

返す $d(op_i)$ を次のように定義する．

$$d(op_i) = \begin{cases} 1 & (\text{if defects are induced at } v_{i+1}) \\ 0 & (\text{otherwise}) \end{cases}$$

$d(op_i)$ は，op_i に欠陥の発生要因となったコード修正が行われた場合は 1，そうでない場合は 0 を返す．リビジョン v_j から v_k における欠陥の発生頻度 $f_d(j,k)$ を次のように定義する．

$$f_d(j,k) = \frac{\sum_{i=j}^{k} d(op_i)}{v_k - v_j} \qquad (j < k, \quad v_j, v_k \in V)$$

実験では，$f_d(j,k)$ と先に述べたリファクタリング頻度と不吉なコードの滞留期間についてそれぞれの相関を求める．

3.4 実験対象に求められる要件

実験の対象となる開発プロジェクトは，版管理システムとバグ管理システムが利用されており，分析可能であることが前提となる．また，コミットログを利用するリファクタリング検出手法を用いる場合には，コミットログが詳細に記述されているプロジェクトであることが求められる．また，SZZ アルゴリズムを適用するため，バグ管理システムの欠陥情報をコミットログに記載するなど，バグ管理システムと版管理システムを連携して扱っているプロジェクトが望ましい．

4． まとめと今後の展開

本稿ではソフトウェア開発過程における，リファクタリング履歴及びコードの不吉な匂いと，欠陥の関係を分析する手法を提案した．本稿で示した実験を行い，仮説を検証することで，リファクタリングを定常的に行った場合の欠陥への影響に加え，リファクタリングを行っていない場合の影響も定量的に評価することができる．また，コードの不吉な匂いを利用し，リファクタリングが必要であるにも関わらず，リファクタリングが行われていない開発プロジェクトを評価する手法を提案した．今後は，実験計画で示した手順に従って実験を行っていく．なお，実験対象として，Eclipse[3]，Bugzilla[1] プロジェクトを予定している．

謝辞

本研究は一部，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．

参考文献

[1] Bugzilla, http://www.bugzilla.org/.

[2] CVS, http://www.cvshome.org/.

[3] eclipse, http://www.eclipse.org/.

[4] Fowler M.:*Refactoring: improving the design of exsiting code.*, Addison Wesley, 1999.

[5] Murphy-Hill E., Black A.P., Dig D. and Parnin C.:Gathering refactoring data: a comparison of four methods. In *Proc. of WRT 2008*, pp. 1–5, 2008.

[6] Ratzinger J., Sigmund T. and Gall H.C.:On the relation of refactorings and software defect prediction. In *Proc. of MSR 2008*, pp. 35–38, 2008.

[7] Śliwerski J., Zimmermann T. and Zeller A.:When do changes induce fixes? In *Proc. of MSR 2005*, pp. 1–5, 2005.

[8] subversion, http://subversion.tigris.org/.

[9] trac, http://trac.edgewall.org/.

[10] Xing Z. and Stroulia E.:UMLDiff: an algorithm for object-oriented design differencing. In *Proc. of ASE 2005*, pp. 54–65, 2005.

[11] Xing Z. and Stroulia E.:Refactoring Detection based on UMLDiff Change-Facts Queries. In *Proc. of WCRE 2006*, pp. 263–274, 2006.

[12] 川口真司, 松下誠, 井上克郎:版管理システムを用いたクローン履歴分析手法の提案. 電子情報通信学会論文誌, Vol.J89-D, No.10, pp.2279–2287, 2006.

[13] 三宅達也, 肥後芳樹, 井上克郎:メソッド抽出の必要性を評価するソフトウェアメトリックスの提案. 電子情報通信学会論文誌, Vol.J92-D, No.7, pp.1071–1073, 2009.

[14] 吉田則裕, 肥後芳樹, 神谷年洋, 楠本真二, 井上克郎:コードクローン間の依存関係に基づくリファクタリング支援. 情報処理学会論文誌, Vol.48, No.3, pp.1431–1442, 2007.