4L-4

オブジェクト指向プログラムにおける リファクタリングの自動化支援について

佐野友昭† 古川善吾‡

(香川大学)

1. はじめに

近年のソフトウェア開発において、開発されるソフトウェアは大規模で複雑になり、度重なる仕様変更にも耐えうる必要がある。そのような中、ソフトウェアの外部的な振る舞いを保ちつつ、作業者から見た理解や修正が容易になるように内部構造を改善するリファクタリングという技術が注目されている[1]。

リファクタリングを行うプロセスを図 1 に示す。図に示すように、リファクタリングを行うにあたり、作業者には大きく分けて 2 段階の作業が必要とされる。

Step1. 作業者はソースコードの改善したほうがいいと思われる箇所（不吉な匂い）を検知する。

Step2. リファクタリング手法を施し、不吉な匂いを解消する。

いずれの段階も、作業者の知識や経験が必要とされる。そのため、リファクタリングの効果は、リファクタリンングを行う作業者の知識や経験の有無に左右される。

この問題点を解決するために、本研究では、Step1 に焦点をあて、ソースコード中の不吉な匂いを、人の知識や経験に頼らずに検出することを支援するシステムを試作した。なお、対象とするオブジェクト指向プログラミング言語としては、多くのプログラマに一般的であるという理由から、Java 言語を採用した。

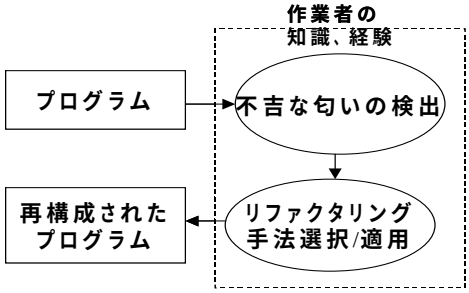

図 1 リファクタリングのプロセス

「A refactoring support method using some metrics values of an object-oriented program」
†Tomoaki Sano (Kagawa University)
‡Zengo Furukawa(Kagawa University)

2. 不吉な匂いの検出支援機構

ファウラーは、代表的な不吉な匂いを 22 種類、リファクタリング手法を 72 種類に分類し、それぞれの対応関係をカタログ化した[1]。

本研究では、ファウラーが分類した 22 の不吉な匂いのうち、4 つの不吉な匂いに対して、対応するメトリクスをそれぞれ定義することにより、自動的な検出を支援する。残る 18 の不吉な匂いに対しては、対応する有効なメトリクスを定義することができなかったため、今回は検出対象外とした。今回の研究において定義したメトリクスと、検出の支援の対象とした不吉な匂いを表 1 に示す。

表 1 不吉な匂いを検出するメトリクス

メトリクス名	説明	不吉な匂い
NoI	インスタンス変数の数	巨大なクラス
NoA	メソッドの引数の数	多すぎる引数
LoM	空白、改行を除いたメソッドの総文字数	長すぎるメソッド
NoG	他クラスへ get メソッドの数	属性、操作の横恋慕

表 1 のメトリクスの値と、対応する不吉な匂いの間には、メトリクスの値が大きくなれば、対応する不吉な匂いの兆候が強くなる正の相関関係があると考えられる。よって、これらのメトリクスを計測することにより、プログラムの不吉な匂いの兆候を判断する材料にすることができる。

しかし、メトリクスは、そのままの値では客観的な計測値の大きさの判断や、異なるメトリクス間での比較ができない。

そのため、本研究では、java で書かれたオープンソースからそれぞれのメトリクスの平均値、標準偏差を計測することにより、不吉な匂いの検出基準とした[2]。これにより、計測するプログラムにおいて、検出されたメトリクスの偏差値を計算することができ、計測値に客観性を持たせられるとともに、異なるメトリクス間での計測値の比較を行うことができる。以上を踏ま

えた、本研究における不吉な匂いの検出支援機構を図2に示す。

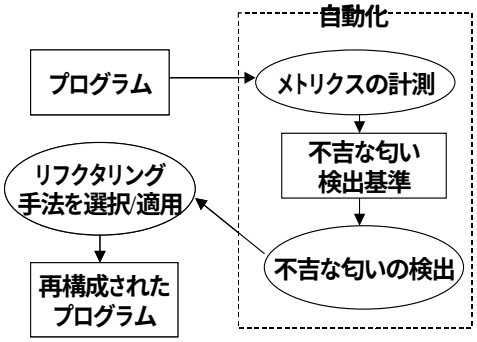

図2　不吉な匂いの検出支援機構

3. 検出ツールの実装

これらの機構を実現するために、本研究では、対象とするメトリクスの値を読み込んだ java のプログラムから検出するツールを実装した。ツールの開発言語としては C 言語を使用し、コマンドラインで実行する。

実装の仕組みとしては、読み込んだファイルを単語ごとに分割し、解析することにより、各種メトリクスの値を計測し、不吉な匂いの検出基準から、メトリクスの偏差値を計算し、検出されたメトリクスの値を、偏差値が大きい順に出力する。

4. 不吉な匂い検出基準

不吉な匂い検出基準としての、各種メトリクスの平均、標準偏差を求めるにあたり、オープンソースソフトウェアとして公開、運用されているソフトウェア”JUnit”[3]を計測対象とした。

JUnit のファイルを対象としてそれぞれのメトリクスを計測した結果を表2に示す。

表2　JUnit から計測した不吉な匂い検出基準

	NoI	NoA	LoM	NoG
平均値	0.333	0.234	67.44	0.059
標準偏差	1.414	0.695	97.61	0.316

5. 計測実験と考察

実装したツールと、不吉な匂い検出基準を用いて、実際のプログラムから計測実験を行った。計測例として、JUnit に含まれるファイルの１つである SimpleTest.java の計測結果を表 3 に示す。

表3の計測結果より、SimpleTest.java においては、まずソースコードの 11 行目のクラスにおいて、「巨大なクラス」の兆候が高く、次には「多すぎる引数」の兆候が 63 行目のメソッドにおいて高いことが分かる。作業者は、この結果に基づき、測定したプログラムにおいて、不吉な匂いの兆候が高い箇所を順に確認することができる。

問題点として、NoA、NoG、NoI は、表2に示した不吉な匂い検出基準において、平均値が極めて小さな値になっており、そのために、小さな計測データも大きな偏差値として出力されてしまう。このため、作業者はこれらのメトリクスの偏差値を、本当に問題点であるかどうかの判断材料とすることが難しい。

表3 SimpleTest.java の計測結果

メトリクス	値	偏差値	行番号	不吉な匂い
NoI	2	61.78	11 行目	巨大なクラス
NoA	1	61.02	63 行目	多すぎる引数
LoM	147	58.15	55 行目	長すぎるメソッド
LoM	52	48.41	45 行目	長すぎるメソッド
NoG	0	48.12	16 行目	属性、操作の横恋慕

6. おわりに

オブジェクト指向プログラムにおけるリファクタリングの自動化支援について検討し、メトリクスを自動的に検出するツールを実装し、実際のプログラムに対して計測を行った。本ツールを用いることにより、作業者は、リファクタリングを行う際に、不吉な匂いを検出する指針を持つことができる。

今後の課題としては、まず、不吉な匂いの検出基準についての妥当性の検証がある。今回は Junit を不吉な匂いの検出基準の測定対象としたが、 Junit 以外の環境で検出基準を計測することで、検出基準の設け方や妥当性について議論する必要がある。また、今回定義したメトリクスの値と対応する不吉な匂いの相関性の妥当性に関しても、検証する必要がある。

さらに、今回対象にしたメトリクス以外に、新たにメトリクスを定義することにより、より多くの不吉な匂いを検出できるようにすることや、検出した不吉な匂いに対して、どのリファクタリング手法を適用するかの判断を支援するアプローチの考案、ツールへの組み込みも今後の課題として挙げられる。

【参考文献】

[1]マーチン・ファウラー,”リファクタリング”, ピアソン・エデュケーション (2000)
[2]秦野克彦,乃村能成,谷口秀夫,牛島和夫,”リファクタリングの自動化を支援する機構”, オブジェクト指向最前線 2002, pp.75-82(2002)
[3]http://www.junit.org/index.htm,”JUnit, Testing Resources for Extreme Programming”