

ソフトウェアリポジトリ解析に基づく リファクタリング推薦に向けて

市井誠^{1,a)} 川上 真澄¹

概要:

大規模なソフトウェアのリファクタリングを効果的に進めるためには、悪影響を生じさせるプログラム要素を的確に抽出し、その悪影響を取り除ける適切なリファクタリングを行う必要がある。本稿では、リポジトリ解析を用いたアンチパターン検出により、リファクタリングすべき箇所の悪影響の理解および改善方法の導出を支援するアプローチを提案する。

Towards Refactoring Recommendation based on Software Repository Analysis

1. はじめに

大規模な組込みソフトウェア開発では、同じコードベースを10年以上に渡り利用し続ける。リファクタリングは、繰り返される修正に伴う技術的負債の蓄積を避け、開発効率を維持するために必要であるが、従来、変更影響範囲を増やし、不具合リスクを有む余計な変更として避けられてきた。しかし、蓄積した技術的負債は工程を圧迫し、いずれ品質にも影響するため、リファクタリングの実施は避けて通れない道であると言える。

一方で、大規模化したソフトウェアに対し、無闇にリファクタリングを行っても、変更および検証のコストがかさむだけで、将来の開発効率や品質の向上に繋がらないおそれがある。そのため、悪影響を生む箇所を的確に抽出し、適切な方法でリファクタリングすることが必要である。しかし、そのためには製品開発状況の把握と、リファクタリングに関する習熟が必要となり、熟練者以外には難しい。

筆者らはこの課題に対しリファクタリングすべき箇所およびその方法を開発者に推薦することをめざした取り組みを進めている。

2. 関連研究

リファクタリングすべき箇所の特定には、不吉な香り、もしくはアンチパターンと呼ばれる考え方が用いられる [1]。これはリファクタリングすべきソースコードの性質に名前をつけたものであり、リファクタリング検討の指針となる。

アンチパターンの自動的な特定にはソースコード解析に基づく手法やリポジトリマイニングに基づく手法など様々な手法が提案されている。ソースコード解析に基づく手法としてはコードクローン解析 [2] やメトリクス値の組合せに基づく手法 [3] が存在する。改善すべき性質をそのまま指摘できる一方で、大規模な対象に対しては、検出数が膨大になり、優先度付けが困難になる。優先度付けの判断には、変更せず実害の無いところは経過観察など、変更のされ方を考慮する必要がある。

リポジトリ解析に基づく手法としては、ロジカルカップリング解析を用いた手法 [4] が存在する。リポジトリ解析に基づく方法は、変更履歴上で実害の痕跡を特定する方法とも言え、歴史の長い大規模システムに向いていると考えられる。ただし現象だけを見ているので、何が悪く、どうあるべきなのかの分析は人手に委ねられている。

リファクタリング方法の導出は、メソッド抽出範囲を Program Dependence Graph (PDG) を用いて特定する手法 [5] など、特定のリファクタリングパターンの具体的な適

¹ 株式会社日立製作所 研究開発グループ
システムイノベーションセンタ システム生産性研究部

^{a)} makoto.ichii.dn@hitachi.com

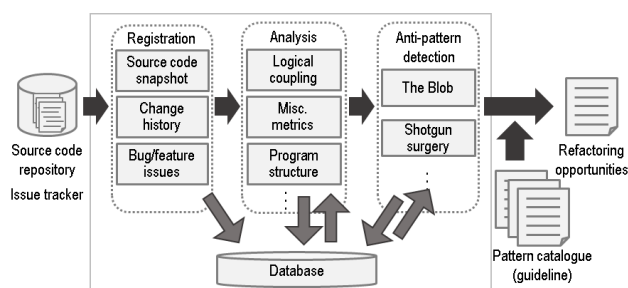


図 1 アンチパターン検出環境

用方法を導出するの試みが存在する。しかし、どのパターンをどう適用すべきかは設計意図に依存し、より大局的な指針決定の支援が必要である。

3. リファクタリング支援のアプローチ

大規模ソフトウェア開発において、開発者に対して適切なリファクタリングを促すための、アンチパターン検出に基づくリファクタリング支援のアプローチを提案する。

アンチパターン検出環境を図 1 に示す。基本的な考え方は、リポジトリ解析に基づく、悪影響を生む重要度の高いアンチパターンの検出、および、検出箇所の悪影響を理解して対策へと繋げるためのガイドラインとなる、アンチパターンの体系化である。

3.1 アンチパターンの検出

ソースコード解析および変更履歴・不具合(変更)管理情報の解析の組合せにより、プログラム要素もしくはプログラム要素間の関連を、改善すべきアンチパターンとして検出する。アンチパターン検出の例として“The Blob (肥満児)”の検出について述べる。The Blob はロジカルカップリング分析 [6] とメトリクス計測の組合せにより検出される。具体的には、他の多数の関数の変更影響を受け、かつ、複雑度メトリクスの値の高い関数として検出される。

3.2 悪影響および対策方針の導出

検出されたプログラム要素について、ソースコードや変更履歴などに観測される現象のパターンを用い、アンチパターンを放置した際の(または、既に起きた)悪影響を導出する。The Blob に関しては、関数の肥大化による理解性の悪化や、見落とし易い依存関係による変更漏れリスクが、悪影響として導出される。これは、過剰な複雑度、外部データを通じた間接的な依存関係など、アンチパターンごとにプログラム構造にあらわれる現象をパターン化しておく、それらが実際に存在するかどうか解析することで行う。

同様にして、観測された現象のパターンに基づき、カタログ化された対策方針を導出する。対策方針は、適用すべきリファクタリング、または、悪影響を顕在化させないための対処方法として示される。リファクタリングは、設計

思想(あるべき姿)を仮定し、それに近づくよう改善する一連の操作として定義される。例えばイベントの送受信そのものと、その際の処理が一体化していることで影響が生じている場合は、コマンドパターンのような形態で変更を局所化する。ソースコードの修正が困難な場合は、該当関数の変更時の注意書きをコメント欄に残すなど、悪影響の顕在化を防ぐ方法をガイドする。

3.3 適用と課題

本アプローチを、実際の組み込みソフトウェアの製品リポジトリに対して適用し、評価を進めている。対象製品は C 言語および C++ 言語で開発されている 100 万行規模の組み込みソフトウェア製品であり、Subversion および Git により管理されている。適用の結果、開発者の問題意識に一致した箇所を抽出できることを確認できた。一方、製品開発は 10 年以上に及ぶのに対してリポジトリ管理は最近数年であり、規模に対して履歴が不足し直近の変更にひきずられる点が、今後の課題として浮かび上がった。

4. おわりに

リファクタリング推薦に向けた取り組みを進めている。単に直すべき箇所を指摘するだけでなく、アンチパターンの悪影響や、それが生じた背景を示すことで、効果的なリファクタリングを促すことを目標としている。現在は検出されるパターンごとにカタログでリファクタリングをガイドするが、手作業コード理解に頼る部分も多い。履歴の解析方法の改善とともに、解析を自動化し、より高度な推薦を行うことを検討している。また、ケーススタディを重ね、より多くのパターンに対応していく。

参考文献

- [1] マーチン・ファウラー：リファクタリング－プログラミングの体質改善テクニック，ピアソン・エデュケーション (2000).
- [2] 神谷年洋，肥後芳樹，吉田則裕：コードクローン検出技術の展開，コンピュータソフトウェア，Vol. 28, No. 3, pp. 3.29-3.42 (2011).
- [3] Moha, N., Gueheneuc, Y. G., Duchien, L. and Meur, A. F. L.: DECOR: A Method for the Specification and Detection of Code and Design Smells, *IEEE Trans. Softw. Eng.*, Vol. 36, No. 1, pp. 20-36 (2010).
- [4] Palomba, F., Bavota, G., Penta, M. D., Oliveto, R., Poshyvanyk, D. and Lucia, A. D.: Mining Version Histories for Detecting Code Smells, *IEEE Trans. Softw. Eng.*, Vol. 41, No. 5, pp. 462-489 (2015).
- [5] Kanemitsu, T., Higo, Y. and Kusumoto, S.: A Visualization Method of Program Dependency Graph for Identifying Extract Method Opportunity, *Proc. the 4th Workshop on Refactoring Tools*, ACM, pp. 8-14 (2011).
- [6] Gall, H., Hajek, K. and Jazayeri, M.: Detection of Logical Coupling Based on Product Release History, *Proc. Int'l Conf. Softw. Maintenance*, pp. 190-198 (1998).