4R−4

リファクタリングによるソフトウェア品質改善と品質劣化の予防 (1) リファクタリングプロセスの定義

片岡 欣夫,　本間 昭次,　深谷 哲司
株式会社東芝 研究開発センター システム技術ラボラトリー
{yoshio.kataoka|akitsugu.homma|tetsuji.fukaya}@toshiba.co.jp

1　はじめに

リファクタリングは「プログラムの意味・実現機能を変更すること無しに, プログラムの構造を変更する技術」と定義され, プログラムの保守性向上を目的とする技術である. リファクタリングを行うことにより,

- 常により良いシステム設計を保てる
- 可読性, 保守性の高さを保てる
- 潜在的バグを発見できる
- 拡張・改良の工数を短縮できる

など, 保守性や拡張性を向上させる改善効果や, 予め拡張性を確保する予防効果が期待される. また文献 [1], [2] 等に紹介されているように, リファクタリングは今やソフトウェア開発技術の基礎技術であり, ソフトウェア開発者に求められる必須技術となりつつある.

これらの有効性が一般的に認知されているにもかかわらず, 稼働中のプログラムに対して変更を加えることで新たな誤りを混入するかも知れないと言う危惧などから, リファクタリングが現場で積極的に行われることは少ない.

我々は, リファクタリング適用に関する問題点を解決し有効活用するために, 支援技術の研究を進めている. 具体的にはリファクタリングの適用過程をリファクタリングプロセスモデルとして定義し, それぞれの場面での支援技術について研究を進めている. 本稿ではこのリファクタリングプロセスモデルの構築について述べる.

2　リファクタリングプロセス

リファクタリングの適用過程は, (1) リファクタリング候補の同定, (2) リファクタリングの適用, 及び (3) リファクタリング結果の検証の三つのサブプロセスから構成される. これら三つのプロセスの相互関連と主な支援項目を図示したものが図 1 で, その形状から A(アルファ) プロセスモデルと名つけている.

A プロセスは, 大きく分けて二つの軸から構成される. 一つがプロセスの時系列を示す軸で, 図中では水平方向にあたる. もう一つはプロダクトの抽象度のレベル, あるいは担当者を示す軸で, 図中では垂直方向に当たる.

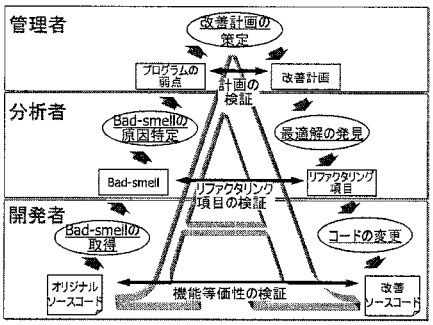

図 1: A プロセスモデル

まずベースとなるのがソースコードレベルであり, リファクタリングの妥当性の検証など最終的なチェックはこのレベルで行われる. このレベルの活動は主に開発者が担当することになるため, 開発者レベルとも言うことが出来る. 具体的には, プログラムの動作に変化が無いかということを, テストなどによって確認するレベルである.

その上に位置するのが分析者レベルであり, ソースコードの分析によって明らかにされた歪みと, 対応するリファクタリングとの検証を行うレベルである.

最上位に位置するのがプロジェクト管理者レベルである. このレベルでは, プロジェクト全体としてどのような改善を指向するのかと言った戦略的な判断が行われる. 具体的に問題になるのは, 改善にかかるコストとその効果とのトレードオフである. コストに関しては比較的定量化が行いやすいが, 効果に関しては定量化が困難であるため, 高度な判断が必要になる. 一つ重要なことは, このレベルでの方向つけが下位レベルでのリファクタリング戦術の選択に大きく影響を及ぼすという点である. 可能な改善項目を整理し, 達成すべきゴールの優先度つけを行った上でコストと相談をしながらリファクタリング計画を決定する必要がある.

次に三つのサブプロセスについて説明する.

1. リファクタリング候補同定
   図中で左下からプロジェクト管理者レベルに向か

"Software Quality Improvement and Deterioration Prevention using Refactoring – Definition of Refactoring Process –" by Josh KATAOKA, Akitsugu HOMMA, and Tetsuji FUKAYA, System Engineering Laboratory, Corporate Research and Development Center, TOSHIBA Corporation

う流れがこれに当たる．ミクロには，ソースコード中のリファクタリングすべきプログラムポイントを同定する作業，マクロにはソフトウェア全体の改善方針を探る作業である．

2. リファクタリング実施
図中でプロジェクト管理者レベルから右下へと向かう流れがこれに当たる．ミクロには各リファクタリング項目の実施，マクロにはソフトウェア全体の品質向上を行う作業である．

3. リファクタリング結果検証
図中で左右のプロダクト間を繋ぐ部分がこれに当たる．具体的には上記同定作業で列挙された項目が，正しく実施されていることを確認する作業が各レベルで行われる．

3 主な作業項目

上述したような A プロセスモデルに沿って，どのような作業項目が必要かを列挙してゆく．以下改善方針の策定と改善の実施との二段階に分けて，作業順に説明する．更に各レベルで実施されるべき検証作業項目についても述べる．

3.1 改善方針の策定 (リファクタリング候補同定)

1. Bad-smell の取得 (開発者レベル)
A プロセスは，個々の開発者が自分が保守すべきソースコードを分析し，可読性や保守性を低下させている要因を見つけ出すことから始める．“Bad-smell” とは，[3] の中で述べられている，プログラム中の問題を示唆するプログラム特性のことである．単純な例では，巨大なメソッド，多すぎる引数などであり，少し複雑なものでは，継承関係の冗長性や無駄なクラスの存在などがこれに当たる．

2. Bad-smell の原因特定 (分析者レベル)
次になすべきことは，Bad-smell の原因を特定することである．Bad-smell の原因とは，即ちソースコード上の弱点と言い替えることが出来る．これらの弱点は，Bad-smell から自明の場合もあるが，一般には対象となるソースコード部分の分析作業が必要とされる．

3. 改善計画の策定 (管理者レベル)
前段で示されている通り，特定のコードに現れる Bad-smell を潰すためには，コード内で閉じた解決策もあれば，他のコードの方で問題を修正すると言うアプローチもある．これらのうちどのアプローチを取るかと言うことは，マネジメントも含めた高度な判断が必要である．そのための材料となる効果とコストの評価を行う．

3.2 改善の実施 (リファクタリング実施)

1. 改善計画に対応する最適解の発見 (分析者レベル)
策定の段階で検討されたように，特定のゴールに対応する解決策は複数考えられるが，ここでは分析時に得られた情報を元に，誰の担当部分がどのゴールに責任を負うかと言うことを確認し，作業の割り振りが行われる．

2. コードの変更 (開発者レベル)
各担当者は割り振られたリファクタリングを実施する．ここでは全体として整合が取れた状態でリファクタリングが進んでいるかと言うことを常に確認しておく必要がある．

3.3 検証作業項目 (リファクタリング結果検証)

1. 改善計画の検証 (管理者レベル)
策定した改善計画が，目的としたソースコードの弱点を確実に押えているかを確認する．

2. リファクタリング項目の検証 (分析者レベル)
導かれたリファクタリング項目が，検出された Bad-smell を潰すことが出来ることを確認する．

3. 機能等価性の検証 (開発者レベル)
リファクタリング前後のプログラムが，比較可能な単位で機能的に等価であることを検証する．

4 まとめ

本稿では我々が提案する A プロセスモデルについて説明した．我々の提案するモデルは，候補同定，適用，及び検証という三つのサブプロセスから構成され，作業担当者の観点から，開発者，分析者，及び管理者の三つの階層が存在し，それぞれに対応するサブプロセスのアクティビティが定義される．このようなプロセスモデルを準備することで，それぞれに適した支援項目を設定することができ，効率の良いリファクタリング支援環境開発が可能となる．

今後はこのモデルに従って，順に支援環境の構築を進めて行く予定である．

参考文献

[1] K. Beck. *eXtreme Programming eXplained: Embrace Change*. Addison–Wesley, 2000.

[2] W. Brown, R. Malveau, H. McCormick III, and T. Mowbray. *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley Computer Publishing, 1998.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison–Wesley, 1999.