7A-1

効率的なレビューのための静的コード解析機能の拡張とその評価

野原賢二　坂田祐司　横山和俊

株式会社 NTT データ　技術開発本部

1　はじめに

コードを静的に解析し不適切な部分を指摘する多くのツールが登場し, 開発者に利用されている. しかし, これらのツールでは, 不適切な部分の修正を手動で行なうため, 初心者の開発者や, 大量の指摘がされる場合は, 修正が困難であるという問題があった. われわれは, 自動的に修正することにより, この問題を解決する方式を提案する.

2　静的コード解析ツール

静的コード解析とは, コードの構文や制御の流れ, データの流れを解析し, それに基づきプログラムの構造や意味の抽出, また不適切な部分の指摘を行う手法である. 本稿では, 開発者に広く利用されている, Java 言語を対象とし不適切な部分の指摘を行う静的コード解析ツールに注目する.

この種のツールには, 商用やオープンソースを含めて JTest, PMD, FindBugs, Checkstyle などがある. これらは, 不適切であると考えられるコードの特徴をチェックルールとして保持している. そして, 解析対象コードが, そのルールに適合する場合に不適切であるとし, 不適切な部分とその解決方針を開発者に提示する. これらのツールは, ① 初心者の開発者が自身のコードの問題点を発見する場合, ② プロジェクトとしてコードを一定の品質に保つことにより, 人間による単純なレビューの負担を軽減したい場合に, 有効である.

しかし, これらのツールは, 不適切な部分を指摘する機能しか持たない. すなわち, 不適切な部分の修正は手動で実施する必要がある. そのため, 以下の 2 つの課題がある. ① 初心者の開発者は, 問題を指摘されても, 修正の仕方が理解できず, 結局有識者の支援が必要となる場合がある ② 可読性の低さなど軽微と考えがちな問題が多く存在する場合, その修正のために必要なコストを考慮してツールの指摘事項を無視する傾向にある. これらの課題は, 既存のツールが不適切とする部分を指摘するのみであり, その修正に対する支援が不十分であることに起因する.

An Effective review by automatic debugging codes
Kenji Nohara, Yuji Sakata and Kazutoshi Yokoyama
NTT DATA CORPORATION Research and Development Headquarters

3　提案方式

以上の背景から, われわれは静的コード解析ツールによって指摘されたコードの問題点を半自動的に修正する機能を提案する. 本機能の概要を図 1に示す. 提案機能の実現では, 基盤ツールとして, PMD[1]を利用している. PMD は, 入力として与えられたソースコードに対する抽象構文木を構築し, その構文木と保持しているチェックルールを照合することにより, 不適切な部分のリストをチェック結果として出力する. 本研究で追加する修正機能は, PMD から出力されるチェック結果と抽象構文木を入力とし, 各チェックルールに対して実装される修正モジュールを用いて抽象構文木を修正, その抽象構文木からソースコードを生成し, 出力する.

なお, 類似研究・ツールとしては, Java のサブセット言語に対して自動的な修正機能を提供する Kenya[2]や商用ツールとしては JTest[3]の即時修正機能がある.

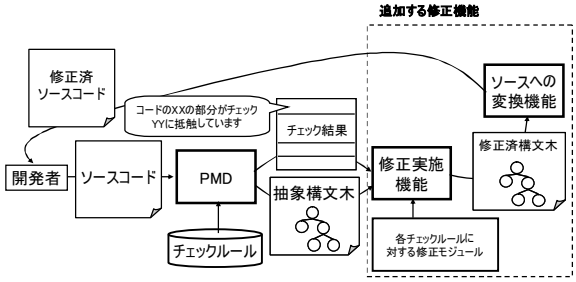

図 1: 自動修正機能の実現方式

4　評価

提案方式の有意性を判断するため, (1) チェックルールに対する修正の必要性, および, 自動的な修正の実現可能性, (2) 手動での修正と比較した場合の本提案手法による半自動的な修正の効果, について評価を行なった.

4.1　修正の必要性, および, 実現可能性の評価

提案方式の実現にあたり, PMD のチェックルールに対する修正の必要性, および, 自動的な修正の実現可能性について評価を行った. 具体的には,PMD のチェックルールの中に,修正の必要があるルール,および,修正の自動化が可能なルールがどの程度存在するかを評価した.

はじめに、どの程度修正が必要なルールが含まれているかを検討した. 具体的には、PMD ver. 3.2 が初期状態で保持している 145 個のルールを以下の 3 つに分類した. Ⅰ）修正必須ルール:修正が必須であるルール. バグ，性能，脆弱性に関する問題が相当する，Ⅱ）修正推奨ルール：修正が必須とはいえないルール．可読性，拡張性，無駄なコードが相当する，Ⅲ）それ以外のルール.

次に，修正の自動化の実現性について検討を行った. 具体的には，PMD のルールを以下の 5 つに分類した．ⅰ）一意に修正：不適切な部分を修正する変換が一意に定まるもの, すなわち修正が完全に自動化できるもの, ⅱ）一部のみ修正：一部を修正し開発者が残りの修正を手動で行うもの, ⅲ）選択すべき候補を提示：コードに対する開発者の意図を選択してもらい, その選択に応じて一意に修正が可能であるもの, ⅳ）不可能：自動的な修正が不可能であるもの, ⅴ）要調査：今回の検討では判断ができなかったもの.

これらの結果を図 2 に示す. 有効性に関して言えば, 修正が必須なルールが 44%, 修正が推奨されるルール 49%存在し, 両方のルールを総計すると 93%のルールの修正が有効であるという検討結果になった. また, 実現可能性に関して言えば, 53%のルールは一意に修正が可能であり（ⅰ）, ⅱ）, ⅲ）の合計, すなわち 84%のルールは何らかの形で修正に対する支援が可能であることがわかった.

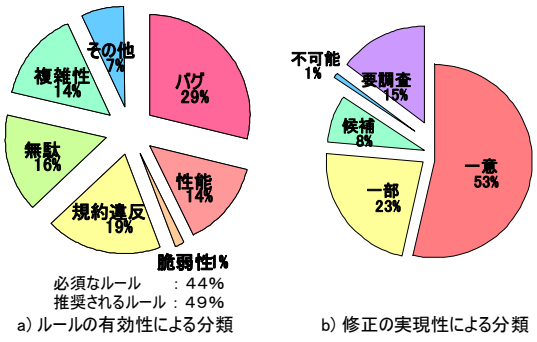

図 2: PMD のチェックルールの分類結果

4.2 修正時間の改善度の評価

実機による自動的な修正の効果を評価するために, 図 1に示す修正機能を PMD の拡張としてプロトタイプを実装した. 今回, 修正機能は 60 個のルールに対して実装した.次に, 実装したプロトタイプを用いて, 修正時間の改善度の評価実験を行なった. 具体的には, PMD を実行して得られる問題に対して開発者が手動で修正する通常の修正とプロトタイプを用いた自動的な修正の修正時間の比較を行った. 評価対象プログラムは, 無作為に選んだ 10 種類のオープンソースのプログラムを使用した. また, 手動修正は修正方法を理解している開発者で行なった.

実験結果を図 3 に示す. 図 3 は問題検出数に対する修正時間の関係を表している.■は, 通常の手動修正, ▲は, 本手法による修正である.

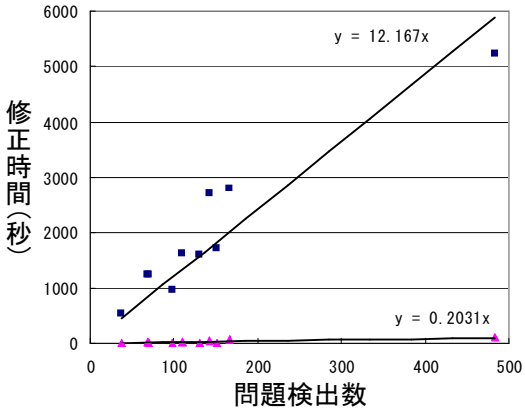

図 3: 問題検出数と修正時間の関係

4.3 考察

4.1 より，93%のルールは修正の必要性があり，かつ，84%のルールは修正の自動化が可能であることがわかった. よって, チェックルールに関して, 提案方式の有意性は高いといえる.

4.2 の結果より,問題検出数と修正時間が比例の関係であると仮定して線形近似を行った結果, 通常の手動修正は問題 1 個につき平均 12.2 秒, 本手法は, 平均 0.2 秒, 修正に時間がかかっていることがわかった. つまり, 本手法は, 通常の手法と比較して修正時間を約 98%短縮できることがわかった. また, 今回は, 修正方法を理解している開発者で実験を行なったが, 修正方法がわからない, もしくは, 問題の意味を理解できない開発者であれば, 修正方法を理解する時間や問題の意味を理解する時間が今回の結果に加わるため, より有意な効果が得られると考えられる.

5 おわりに

今後は, 実際の開発現場での実検証として, 修正機能の評価だけでなく, 通常のレビューから修正までの工程全体の工数と比較した有意性の評価を行なう予定である.

参考文献

[1] PMD: http://pmd.sourceforge.net/.
[2] Robert Chatley and Thomas Timbul, “KenyaEclipse: learning to Program in Eclipse”, ESEC-FSE ’05, Lisbon, Sep. 2005.
[3] JTest. http://www.parasoft.com/jsp/home.jsp