1B-06

コードの発展性欠陥の自動評価：コンテキストを考慮したメトリクス閾値の機械学習

津田 直彦† 鷲崎 弘宜† 深澤 良彰† 保田 裕一朗‡ 杉村 俊輔‡
早稲田大学† 株式会社小松製作所‡

1. はじめに

ソフトウェアの故障ではないものの，コードのわかりやすさや修正しやすさが通常よりも低くなってしまっている状態を発展性欠陥[1]と呼ぶ．これはコードのデザインレビューで発見できるが，レビューの実施コストが大きいため，メトリクスによる定量化と自動化が求められる．

本稿では開発のコンテキストを考慮したコードメトリクスの閾値を決定する手法を提案する．具体的には，コンテキストを代表するベンチマークの一部コードを人手で評価して発展性欠陥有無の正解を用意し，分類木学習で評価基準(メトリクス閾値とその解釈形式)を決定する．

2. 背景

故障を伴う機能性欠陥については発見済み欠陥の完全除去やテストケース実施数といった目標を掲げやすいが，一方で発展性欠陥はその抽象度の高さから様々な捉え方をできてしまい，目標の設定方法が必ずしも明らかでない．開発のコンテキスト（開発言語，アプリケーションドメイン，変更頻度等）が異なるプロジェクトでは多くのメトリクス(行数や複雑度等)で中央値の差があり[2]，どのシステムでも汎用的に異常値を検出できる閾値を決め難いことが示されている．そのため，効果的にコードを自動評価するためには自組織の開発コンテキストに合ったメトリクスと閾値の用意が欠かせない．

従来手法[3]を用いてコンテキストを考慮した閾値を構築するための手順を以下に示す．(1)主要なコンテキスト要素から重視するものを選ぶ．(2)そのコンテキストを代表するソフトウェアを集めベンチマークとする．(3)ベンチマークのメトリクス測定値を入力として閾値計算をする．．

3. 提案手法

2 章で述べた手順の適用事例の多くでは自動評価の精度検証において正解として発展性欠陥の有無を用いていない[3, 4]．我々は発展性欠陥の評価では以下の問題が存在すると考える．

*P1)*機能性欠陥有無の正解を用いない閾値決定の適用事例が多いが[3,4]，近年では評価精度が劣ると報告されている[3]．これは発展性欠陥でも同様となりえるため，正解を入力としたい．

*P2)*一般的な従来手法では個々のメトリクスで独立して閾値決定している．しかし，発展性欠陥はその抽象度の高さから様々な捉え方ができ，関連するメトリクスは必ずしも直交関係にない．そのため，メトリクス同士の関係を多面的に解釈[5]することが望ましい．

*P3)*高精度な閾値決定には欠陥有無の正解情報が必要だが，バグ管理システムには発展性欠陥の有無は登録されていないと考えられる．

P1～P3 を解決し，発展性欠陥の評価基準(メトリクス閾値とその解釈形式)を高精度に構築するための手順を以下に示す．

Step1) まずソフトウェア保守性の品質特性や発展性欠陥の細目[1]を参考に，Goal-Question-Metric(GQM)モデルを定義する．発展性欠陥が無いという目標(Goal)の達成可否の質問(Question)を多面的に用意し，対応するメトリクスを整理する．例えば，モジュールの(1)責務は小さいか，(2)複雑度は低いか，と質問する．

Step2) 自組織が重視するコンテキストを代表するソフトウェアを集めベンチマークとする．

Step3) P3 の解決のため，ベンチマーク内のモジュール(ファイルやクラス)を人手で評価する．評価時には Question 毎に欠陥のあり/なしを判断する．この Step3 ではコストを抑えるために一部モジュールのみを対象とすることも許容する．

Step4) メトリクスを静的解析により測定する．

Step5) Step3 で得た実際の欠陥有無と Step4 で測定したメトリクスを入力として機械学習する．その際，分類木学習を用いることで，Question に対応するメトリクスのうち必要なものが残り，また複数メトリクスの非直交関係を多面的に解釈した閾値が得られ，P1 と P2 の解決となる．

Step6) そのコンテキストに精通する開発者や品質管理者が評価基準の妥当性を検討する．不可の場合，最初の手順に戻る．

Evaluation of Evolvavility Defects: Determining Code Metrics Thresholds for a Given Context by Machine Learning
†Naohiko Tsuda, †Hironori Washizaki, †Yoshiaki Fukazawa
‡Shunsuke Sugimura, ‡Yuichiro Yasuda
†Waseda University, ‡Komatsu Ltd.

4. ケーススタディでの適用実験と考察

本稿では提案手法の有用性を検証するために，以下の二つの研究課題(RQ)について，小松製作所の建機用組込みシステムで適用実験をした．

RQ1:提案手法で高精度な評価基準が得られるか．
RQ2:提案手法により，人手によるコード評価の実施コストに見合う性能改善をできるか．

比較対象の従来手法には．機械学習をしないパーセンタイルと，教師あり学習をするROC曲線法とBender法(ロジスティック回帰ベース) [4]を用いた．その際，Bender法のp0パラメタは0.6とした．従来手法は後述のStep5で実施した．

提案手法は以下の手順で実施した．

Step1) 過去成果のGQMモデルを参照し，10個のQuestionと対応のメトリクスを設定した．尚，本稿では紙面の都合上，責務量(ファイル内の行数や関数定義数)や複雑度の二つのQuestionについて図表を掲載する．

Step2) 開発言語がC/C++であり，業務ドメインが小松製作所製の建機制御用システムとなるベンチマークデータ(C/C++)を用意した．

Step3) ベンチマークデータにおいて，特に保守性が重視されるファイル群を対象に，社内の熟練開発者一名を評価者とし．規模と複雑度の2つの質問では143個のファイルを，その他の8つの質問では55個のファイルを人手で評価し，合計で726件の発展性欠陥有無の正解を得た．

Step4) 社内で導入済みのツールで測定した．

Step5) 統計言語Rの環境にて，"C5.0"という分類木学習アルゴリズムで10個のQuestionそれぞれのための評価基準を構築した．

Step6) 企業の開発者3名と研究室で検討した．

RQ1:提案手法で高精度な評価基準が得られるか．

従来手法よりも高精度な評価基準を得られた．

構築した評価基準の抜粋(責務量 Question)を図 1に示す．プロットは人手による評価での発展性欠陥あり(×)となし(○)を表す．縦軸横軸には自動評価用の閾値が設定してあり，それを下回る範囲を灰色にした．灰色範囲に○が多く×が少ない程，人手による評価と自動評価の一致度が高い．一致度指標F1値の抜粋(責務量と複雑度のQuestion)を表 1に示した．

図 1の灰色範囲を見ると，従来手法では関数定義数(横軸)と実行コード行数(縦軸)それぞれで一個ずつ閾値が設定されているのに対し，提案手法では横軸の一点を境として縦軸に二個の閾値が設定されている．これにより，この事例では実行コード行数(空行や include 行を除く規模)の許容度合いを関数定義数に応じて多面的に解釈でき，評価精度が向上できたと考えられる．

RQ2:提案手法により，人手によるコード評価の実施コストに見合う性能改善をできるか．

発展性欠陥の種類によってF1値の性能改善量に違いがあり，明解な答えは得られなかった．

表 1を見ると，唯一機械学習をしていない80パーセンタイル(80%)では二つのQuestionにおいてF1値が最も小さく，提案手法との F1値の差は，責務量では0.33，複雑度では0.15となった．そのため，質問する発展性欠陥の種類によって期待できる性能改善量に差があると考えられる．

実用においては，試験的にStep1-6を実施し，大きな性能改善を期待できる質問でのみ本格的にStep3-6を再実施すれば，人手による評価の実施コストを効率的に削減できると考えられる．

表 1 質問別に構築した評価基準のF1値(抜粋)

GQM質問	提案	80%	Bender	ROC
責務量	0.88	0.50	0.63	0.64
複雑度	0.91	0.76	0.82	0.80

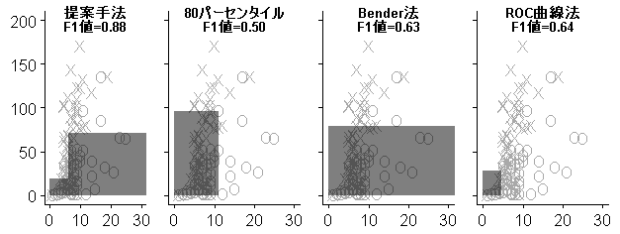

図 1 責務量のQuestionで構築した評価基準

5. おわりに

本稿では開発のコンテキストを考慮したコードメトリクスの閾値を決定する手法を提案した．今後の展望として，人手による評価で蓄積した正解データのうち入力として用いる量を増減させ，自動評価の精度に与える影響を調べたい．

参考文献

1) M.V. Mantyla, et.al: "What Types of Defects Are Really Discovered in Code Reviews?", IEEE Trans. on Soft. Engg., 35(3):pp.430-448, 2009

2)F. Zhang, et.al: "How does Context Affect the Distribution of Software Maintainability Metrics?", ICSM'13, 2013, pp.350-359

3) L. Lavazza, et.al: "An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measures", PROMISE'16, 2016, pp.6:1--6:10

4) L. Sánchez-González, et.al: "A study of the effectiveness of two threshold definition techniques", EASE'12, 2012, pp. 197-205

5) E. Bouwers, et.al: "Getting What You Measure", Communications of the ACM, 55(7):pp.54-59, 2012