情報処理学会論文誌 Vol.60 No.3 804–820 (Mar. 2019)

プログラムソースコードの分かりやすさの閾値に基づく評価基準とその導出手法群の評価

津田 直彦[1,a)] 鷲崎 弘宜[1,b)] 深澤 良彰[1,c)] 保田 裕一朗[2] 杉村 俊輔[2]

受付日 2018年6月6日, 採録日 2018年12月4日

概要：プログラムソースコード（以下コードと略す）における機能的不具合をともなわない構造上の問題は，コードスメル，発展性欠陥，不可視の技術的負債等と呼ばれ，長期的なソフトウェア開発において保守コストを増加させる原因の 1 つとして知られている．そのため，コードの分かりやすさをコードメトリクスと閾値で自動評価する様々な手法がこれまでに提案されてきた．しかし，必ずしもメトリクス間のトレードオフや冗長性が考慮されていないために，問題個所の正解（教師データ）を用いても自動評価の精度が改善されにくい場合があるという問題をかかえていた．一方で，分類木学習と呼ばれる機械学習手法では複数のメトリクスの影響関係を考慮できるため，過学習しやすくなる可能性があるものの，評価精度のより大きな改善を期待できる．従来，コードの分かりやすさの評価の分野では，一般的な閾値ベースの手法と，閾値に加えて解釈の形式表現も導出可能な分類木学習との性能比較事例は報告されていない．本論文では評価実験として，建機制御用の C++組込みソフトウェア群を題材としてソースコードファイル単位の評価基準の導出における手法の性能を比較した．そして，分類木学習アルゴリズム C5.0 では少量の教師データでも開発者の認識に反しない評価基準を導出でき，教師データの量を増加させた際には従来手法よりも自動評価の精度が改善され，実用的であることを確認した．自動評価の精度としてはエキスパートが実施したレビュー結果との一致度（F 値）を用い，比較対象の従来手法にはパーセンタイル関数，Alves 法，Bender 法，ROC 曲線法を用いた．

キーワード：自動評価，保守性，理解性，発展性欠陥，機械学習

Comparison of Methods to Derive Threshold-based Evaluation Criteria of The Understandability of Program Source Code

Naohiko Tsuda[1,a)] Hironori Washizaki[1,b)] Yoshiaki Fukazawa[1,c)] Yuichiro Yasuda[2] Shunsuke Sugimura[2]

Received: June 6, 2018, Accepted: December 4, 2018

Abstract: The low maintainability of program source code is an issue studied as code smell, evolvability defect, or technical debt. Evaluation of such issues is usually automated by measuring code metrics and interpreting the measured values by thresholds. An important action to make automatic evaluation precise is customization of thresholds by machine learning with training-data. However, gathering training-data of non-functional-defective structural issues needs manual inspection. Moreover, conventional methods to derive thresholds do not always interpret redundancy and non-orthogonal relations (e.g., trade-off) among metrics. Consequently, you might not obtain precise evaluation criteria even if you gather training-data in exchange for time of busy experts. In this paper, we defined a practical framework to customize evaluation criteria of the understandability of source code. In particular, we gathered a small amount of training-data by experts' manual inspection, and derive interpretation models and thresholds considering non-orthogonal relations among metrics by a classification-tree learning. In the experiments with source code files of embedded C++ systems developed by a company, we measured the F-measure, which indicating the accordance of manual inspection and automatic evaluation, and then compared classification-tree learning algorithm C5.0 with conventional methods (the percentile, Alves' method, Bender's method, and the ROC curve-based method).

Keywords: automatic evaluation, maintainability, understandability, evolvability defect, machine learning

1. はじめに

1.1 背景

長期にわたる継続的なソフトウェアの開発と保守において，プログラムソースコード（以下コードと略す）を分かりやすく書くことは重要である．複雑に書かれたコードは理解しにくく修正もしにくいので [1]，機能的な不具合が多発したり修正に時間がかかったりする等，保守のコストが増えやすいからである．そのため，コードの分かりにくさの評価とその自動化については種々の研究があり [2], [3], [4], [5], [6], [7], [8]，コードスメル [9]，発展性欠陥 [10]，不可視の技術的負債 [11] 等と呼ばれ，ソフトウェア開発における様々な分野で関心を向けられている．そして，これらはソフトウェアの機能上の直接の不具合ではなく，コードの分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のことである．

このようなコードの構造上の問題の自動評価においては，コードメトリクスが用いられる．コードメトリクスとは規模や複雑度といったコードの構造的特徴を定量的に定義したものである．メトリクスの測定値を解釈してコードを評価する方法としては，閾値の設定が一般的である．閾値とはメトリクスがその値以下（あるいは以上）ならば“問題なし”と見なすと定義するものである．高精度な自動評価の実現のためにはソフトウェアのコンテキスト（プログラミング言語，アプリケーションドメイン等）に応じた閾値の調整が必要となるが，それはコードメトリクスの分布（特に中央値）がソフトウェアのコンテキストによって異なりやすいからである [12], [13]．そして近年では，正解（教師データ）を用いずに導出した閾値では問題有無の評価精度が低いことも報告されており [14], [15], [16]，教師あり学習を用いた閾値導出手法に関する研究が増えつつある [17], [18], [19], [20]．

1.2 本研究で取り組む範囲

本研究では，コードの保守性のうち，機能的な不具合をともなわないコードの構造上の問題に焦点を当て，その高精度な自動評価を目的とし，教師あり学習を用いて評価基準をカスタマイズするための枠組みを提案する．

我々は，既存の体系的枠組みが提供するコードの評価基準を参考に，ある開発組織が評価観点とその測定用のメトリクスを選び，対象コンテキストを考慮して測定値の解釈方法を変更することを，評価基準のカスタマイズとした．そして，カスタマイズ方法の中でも特に，測定値や教師データを入力として統計的に解釈の形式表現（解釈モデル）と閾値を導出することを前提に，その実施手順を枠組みとして整理した．閾値に基づく評価基準に限定した理由は，評価過程や考慮されている範囲が人間にとって分かりやすく，開発者の認識と照らした妥当性の議論がしやすいと考えられるからである．

コードの分かりやすさの評価基準をカスタマイズする場合，開発組織で用意可能な少量の教師データの活用が重要な課題の 1 つである．その理由は，一般に発展性欠陥はバグ管理システムに登録されるものではないため，その教師データの収集には人手による評価が必要となりコストが大きいからである．

また，コード評価においては複数のメトリクスによる多面的な解釈が重要であることが知られているが，従来の一般的な閾値導出手法は複数のメトリクスを考慮した解釈モデルの最適化に対応していない．そのため，自組織向けに評価基準を最適化する際には解釈モデルも変更対象とすることで，過学習しやすくなる可能性があるものの，評価精度のより大きな改善を期待できると考えられる．

我々の枠組みでは，エキスパートがレビュー可能なコードが少量となることをふまえ，レビュー対象はコンテキストを代表するソフトウェア群（ベンチマーク）の一部に絞る．そして，複数のメトリクスの影響関係を分類木学習によって考慮することで，そのコンテキストに適した解釈モデルと閾値の組合せを導出する．

本研究では，少量の教師データにより，従来手法よりも高精度かつ，開発者の認識に反しない評価基準を得られることが実用的であると見なして，以下の研究課題（Research Questions: RQ）に取り組んだ．

- RQ1：我々の枠組みにより，実用的な評価基準を得られるか？
- RQ2：少量の教師データを入力とした場合でも，分類木学習により従来の閾値導出手法に比べ高精度な評価基準を導出できるか？
- RQ3：教師データの量は，自動評価の精度にどう影響するか？

最後に，本研究の貢献を以下にまとめる．

- コードの保守性のうち，機能的な不具合をともなわないコードの構造上の問題に焦点を当て，その高精度な自動評価を目的とし，コードメトリクスの最適な解釈モデルと閾値を導出するための実用的な枠組みを提案した．枠組みはプログラミング言語や評価対象の粒度に依存しない形に一般化しているため，汎用的に利用することが可能である．
- 評価実験 1：建機制御用の C++組込みソフトウェア群を題材にソースコードファイル単位で我々の枠組み

[1] 早稲田大学基幹理工学部情報理工学科
Department of Computer Science and Engineering, Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] 株式会社小松製作所開発本部 ICT 開発センタ
ICT Development Center, Komatsu Ltd., Hiratsuka, Kanagawa 254–8555, Japan
[a] 821821@toki.waseda.jp
[b] washizaki@waseda.jp
[c] fukazawa@waseda.jp

を適用し，適切に閾値をカスタマイズ可能か確認した．
- 評価実験 2：分類木学習アルゴリズム C5.0 と従来手法のそれぞれによる自動評価の精度を比較した．その際，学習時の教師データの数の違いが精度に与える影響も分析し，従来手法では教師データの増加に対して精度の改善が小さいことを確認した．

以降，本論文は次のように構成する．2 章では関連研究をあげて技術的背景と課題を説明する．3 章では閾値をカスタマイズするための枠組みを提案する．4 章では実験方法や利用したデータを説明する．5 章では評価実験の結果を示す．6 章と 7 章では RQ と妥当性への脅威について考察する．最後に，8 章では成果や展望をまとめる．

2. 関連研究

2.1 教師データを用いない閾値導出手法

教師データを用いない閾値導出手法の長所はソースコード以外の情報が不要であり，実施の手間が少ないことである．しかしながら，一般に，コードの規模や複雑度といった構造的メトリクスの分布（特に中央値）はソフトウェアのコンテキストによって異なる [12], [13]．そして，教師データなしでは必ずしも有用な閾値を導出できないという報告もあり [14], [15], [16]，これらの手法は自動評価の精度を重視する場合には実用性が低い．

以降，本節では代表的な手法について説明する．

ごく単純には，コードメトリック 1 種類ごとに測定値の平均値を計算して閾値とすることもできる [21]．ただし，規模や複雑度といったコードメトリクスは一般に正規分布とならないことが知られており [22], [23]，平均値よりもパーセンタイルの方が外れ値の影響を受けにくい（ロバストである）といわれている．p パーセンタイルとは，入力したメトリクス測定値を昇順に並べて小さい値から数え上げたときに全体の p%目となる値を意味する．たとえば，中央値は 50 パーセンタイルに，箱ひげ図の箱の下部と上部は 25 パーセンタイルと 75 パーセンタイルに相当する．

ソフトウェアメトリクスの分析においては，Alves らの方法（Alves 法）[2] が用いられることも多い [24], [25], [26]．Alves 法では，多くのソフトウェアにおいて頻出する値を重視し，一部のソフトウェアのみで測定される極端な値の影響の低減に取り組んでいる．内部的には通常のパーセンタイル関数とは異なり，各要素（クラスやファイル）の重み（行数等）の累積パーセンテージを，要素が属するソフトウェアを考慮しながら集計している．そして，70%，80%，90%の閾値を超えると，それぞれリスクの大きさが Moderate，High，VeryHigh と見なすと仮定している．

経験則としては 80 対 20 の法則は知名度が高く [27], [28]，80%とその近辺の 70%や 90%は利用されやすい傾向にある．あるいは，中央値（50 パーセンタイル）や，箱ひげ図における異常値境界[*1]が利用される場合もある [5], [8], [29]．

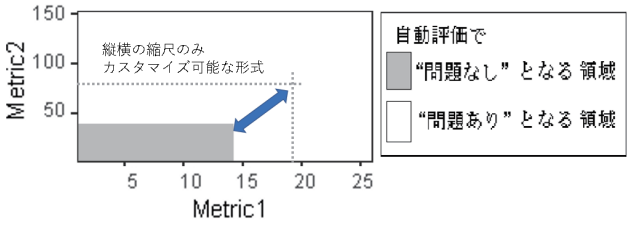

図 1 2 種類のメトリクスに 1 個ずつ閾値を設定した解釈モデル
Fig. 1 Example of a simple interpretation against two metrics.

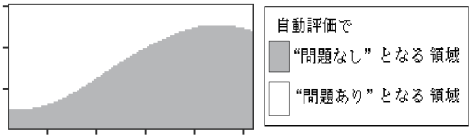

図 2 SVM による解釈モデルの例
Fig. 2 Example of an interpretation by SVM.

2.2 教師あり学習を用いた閾値導出手法

Bender の方法（Bender 法 [17]）と ROC 曲線による方法（ROC 曲線法 [18], [19]）では，任意のメトリック 1 種類の測定値と教師データを入力することで閾値を導出できる．しかし，コードの良し悪しは 1 種のメトリックの大小のみで判断することが難しく，複数のメトリクスによる多面的な解釈が重要となる [30], [31]．

Herbold らは複数メトリクスを同時に考慮する閾値導出に取り組んでいるが，1 つのメトリックにつき 1 つの閾値を設定する解釈モデルになっており，様々な解釈に対応した一般化には至っていない [20]．

このように，従来の閾値導出手法では必ずしもメトリクス間のトレードオフや冗長性が考慮されていない．結果として忙しいエキスパートの時間を割いて教師データを用意しても効果的な機械学習ができず，自動評価の精度が飛躍的には改善されない恐れがある．

これらの手法で 2 つのメトリクスの閾値を設定した場合の解釈モデルの例を**図 1** に示す．この図では 2 つのメトリクス Metric1 と Metric2 を軸にとり，それぞれ 1 個ずつ閾値を設定して点線を引いている．この解釈モデルにおいて，良い評価判定となるのは 2 つのメトリクスがともに閾値以下となる四角形の領域である．いい換えれば，この四角形のサイズ変更により表現できる範囲でしか，解釈モデルをカスタマイズできない．

2.3 教師あり学習を用いた解釈モデルの構築手法

SVM（Support Vector Machine）で 2 つのメトリクスの解釈モデルを構築した場合，**図 2** のように，問題の有無を

[*1] 箱ひげ図における異常値検出の境界はツールによって設定が異なりうるが，“75 パーセンタイル + 1.5*（75 パーセンタイル − 25 パーセンタイル）” とする設定がある．

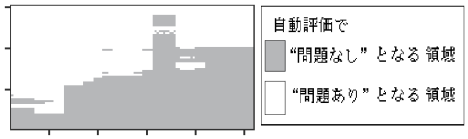

図 3 RandomForest による解釈モデルの例
Fig. 3 Example of an interpretation by RandomForest.

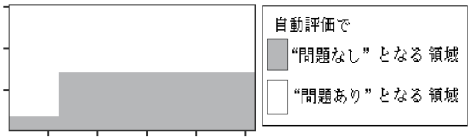

図 4 分類木による解釈モデルの例
Fig. 4 Example of an interpretation by Classification-Tree.

判別するための境界が曲線状になる．RandomForest で 2 つのメトリクスの解釈モデルを構築した場合，**図 3** のように，問題の有無を判別するための境界が複雑に入り組んだ形状になりやすい．これらの手法では高精度な解釈モデルを構築しやすいが，内容を閾値による条件の組合せで表現しにくく，評価過程や考慮されている範囲が人間にとって分かりにくいものになりやすい．モデルの分かりやすさは，人間がそのモデルを信用できるかどうかにも影響するため留意すべきである [32]．また，これらの手法では境界の形状をパラメータで制御できるが，境界の形状との関係が非線形的であり直感的な調整が難しい．

分類木学習アルゴリズム C4.5 [33] やその後継である C5.0 で 2 つのメトリクスの解釈モデルを構築した場合，**図 4** のように，閾値による条件式の組合せが得られる．分類木学習では特徴量空間を閾値で再帰的に分割して最適な解釈モデルを構築するため，図 1 のような四角形のサイズ変更よりも表現力が高い．

Fontana らは複数のコードスメルをメトリクスで予測することについて，SVM や RandomForest や C4.5 等の性能を比較した [34]．しかし，従来の閾値決定手法との性能の比較には踏み込んでおらず，C4.5 と従来手法との間に，どの程度の性能の差があるかは十分に明らかではない．パーセンタイル関数や Alves 法には教師データが不要という強みがあるため，C4.5 はそれより大きく優れていることを示せなければ，人手をかけてまで教師データを収集する動機付けをしにくい．一方，従来の閾値決定手法については，性能の上限において C4.5 に劣ることは推測できるものの，その単純さから過学習をしにくい可能性もあり，教師データが少ない状況下での性能比較には意義がある．

2.4 教師データの収集について

Mantyla らはコードの保守上の問題を，機能性欠陥と発展性欠陥という定義で区別した [10]．機能性欠陥とは，ソフトウェア実行時に観測されうる機能上の不具合である．発展性欠陥とは，ソフトウェア実行時に観測できるものではなく，コードの分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である．

しかし，一般に発展性欠陥はバグ管理システムに登録されるものではないため，その教師データの収集には人手による評価が必要となりコストが大きい．このようなコードの構造上の問題の人手による評価結果を公開しているデータセットの数には限りがあり [35]，自組織に類似した事例の収集も難しい．そのため，開発組織は独自に教師データを用意することになる．しかし，観点によってはドメイン知識を有するエキスパートでないと難しいうえに [10]，そのようなエキスパートには種々の業務も集中しやすい [36]．結果として，評価基準のカスタマイズにおいては，限られた少量の教師データの活用が重要な課題の 1 つとなる．

3. 枠組みの提案

本章ではコードの分かりやすさの評価基準（解釈モデルおよび閾値）をカスタマイズするための実用的な枠組みを提案する[*2]．枠組みの概要，枠組みで扱われる解釈モデルの一例，実施手順の順に説明する．

また，本枠組みで取り扱う「評価基準」と「解釈モデル」については以下のように定義した．

- 評価基準：**「ある観点でコードを評価するための定量的な基準」**．多数のメトリクスを説明変数とするものや，複数のメトリクス測定値を合成するものは，評価過程や考慮されている範囲が人間にとって分かりにくいものになりやすいと考えられるため，本研究で扱う評価基準は，説明変数を合成しておらず，その解釈を閾値を用いて形式表現可能なものに限定した．たとえば，「ファイル行数 <=100 AND 関数定義数 <=10 ならば問題なし」は 2 種類のメトリクスの条件式を組み合わせた評価基準である．逆に，「ファイル行数+10* 関数定義数 <=100 ならば問題なし」といった複数のメトリクス測定値を合成する条件式や，重回帰，SVM，RandomForest 等は対象外とした．
- 解釈モデル：**「閾値的な評価基準で用いる解釈の形式表現について，閾値が固定されずに抽象化されているもの」**．たとえば「ファイル行数 <=T1 AND 関数定義数 <=T2 ならば OK」という解釈モデルでは閾値 T1，T2 は固定されておらず，利用前にこれらを定めて具体化する必要がある．複数の既存のコード評価ツールにおいて，解釈モデルに相当する設定機能が用意されている [5], [8]．

3.1 概要

本研究ではコードの分かりやすさの高精度な自動評価を目的とし，その評価基準をカスタマイズするための実

[*2] この枠組みは我々が過去に定義した枠組みを拡張したものである [37]．

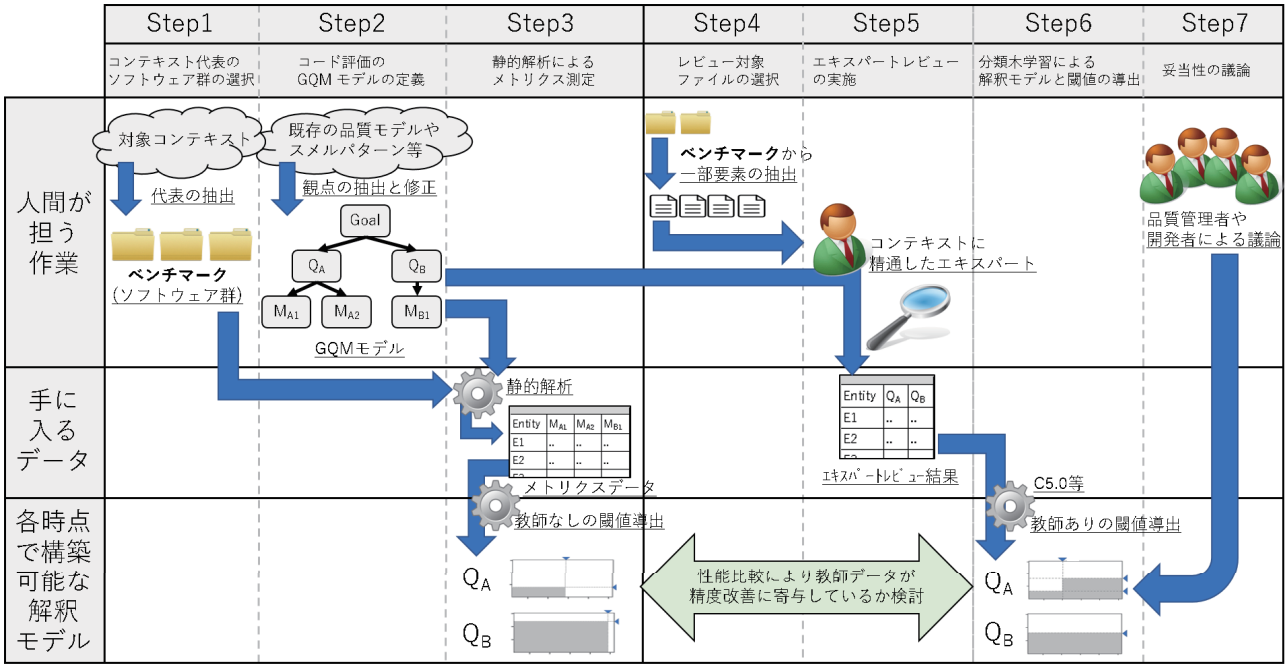

図 5 我々の枠組みの全体像
Fig. 5 Overview of our framework.

用的な枠組みを提案する．具体的には，エキスパートがレビュー可能なコードが少量となることをふまえ，レビュー対象はコンテキストを代表するソフトウェア群（ベンチマーク）の一部に絞る．そして，複数のメトリクスの影響関係を分類木学習によって考慮することで，そのコンテキストに適した解釈モデルと閾値の組合せを導出する．

本研究では，C++のソフトウェア群を題材に評価実験をしているが，我々の枠組み自体はプログラミング言語に依存しない形に一般化しているため，汎用的に利用することが可能である．ただし，コードメトリクス閾値とその解釈モデルは対象とするコンテキスト向けにカスタマイズされるため，同一のコンテキスト（プログラミング言語とアプリケーションドメイン）でのみ共有することを制約とする．その際，アプリケーションドメインの同一性については開発組織が経験的に判断するものとする．コンテキストの同一性の判定方法の具体化と評価については今後の課題である．

我々の枠組みの全体像を**図 5** に示した．この図は実施手順を構成する以下の 7 つの Steps における前後関係と入出力関係，および閾値導出が可能となるタイミングを示している．その詳細は 3.3 節で個別に説明する．

- Step1) コンテキスト代表のソフトウェア群の選択．
- Step2) コード評価の GQM モデルの定義．
- Step3) 静的解析によるメトリクス測定．
- Step4) レビュー対象ファイルの選択．
- Step5) エキスパートによるレビューの実施．
- Step6) 分類木学習による解釈モデルと閾値の導出．
- Step7) 妥当性の議論．

Step1 から Step3 までを完了すると，パーセンタイル関数や Alves 法を利用できる状態になる．Step6 までを完了すると，教師あり学習を利用できる状態になる．最後に，Step7 では導出された解釈モデルおよび閾値の妥当性を議論する．必要に応じて，以前の Step からやり直す．

3.2 メトリクス測定値の様々な解釈モデルについて

規模や複雑度といったメトリクスの解釈には様々な場合がある．全体的にはメトリクス測定値の大きさがコードの構造上の問題を表しているものの，その関係が単調ではなく一定以上になるとむしろ問題は少なくなる場合がありうる．一部のクラスに処理を意図的に集中させる場合や，特定のコードが例外的に許容されている場合が知られている [38], [39]．また，他のメトリクスが別のメトリクスの判断に影響する非直交な関係性もありうる [30], [31]．たとえば，コードの規模の目標を「行数の削減」1 つだけ（One-track-metrics [30] の状態）にしてしまうと，開発者はその達成のために，1 行あたりのステートメントを複雑化させてしまう可能性があり（負のホーソン効果 [31]），結果としてエキスパートが行数以外の構造的特徴も考慮してレビューするようになることがありうる．

我々の枠組みでは，エキスパートによるレビュー結果を抽象化して模倣する形で解釈モデルを構築する．本節では構築されうる解釈モデルについて，4 つのパターンを例として説明する．実際には，これらの例の図形を左右上下反転や回転したパターンや，複数が同時に起こるパターンもありうる．

もし分類木学習によって得られた解釈モデルが開発者に

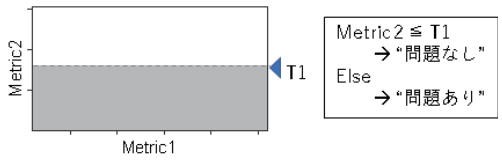

図 6 1 つのメトリックのみで閾値を設定しているパターン
Fig. 6 Interpretation pattern: Single threshold.

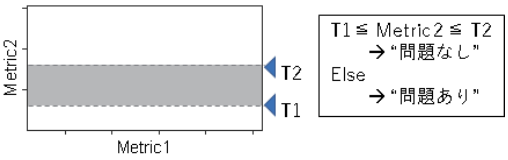

図 7 判別境界が中抜き状になるパターン
Fig. 7 Interpretation pattern: Between thresholds.

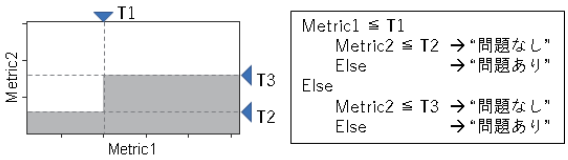

図 8 判別境界が階段状になるパターン
Fig. 8 Interpretation pattern: Stairs.

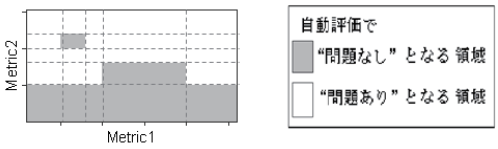

図 9 判別境界に突出した箇所があるパターン
Fig. 9 Interpretation pattern: Isolated area.

とって不自然であれば，教師データの偏りや過学習が原因の可能性もあるが，一方でエキスパートが独自の暗黙知を使っていることが原因の可能性もある．そのため，我々の枠組みを適用することは，エキスパートがどのようにレビューしているかを定量化し，暗黙知の発見を支援することにもつながる．

図 6 は 2 つのメトリクスを入力としたが，1 つのメトリックのみで閾値が設定されたパターンを示す．我々の枠組みでは，N 個のメトリクスを説明変数として学習していても，そのうちの一部のみで十分な精度の自動評価ができる場合には図 6 の解釈になりうる．説明変数どうしの相関が強い場合では，いずれかのメトリックのみで解釈をモデル化でき，冗長さを低減できる．一方，エキスパートの判断とほとんど関連していない説明変数がある場合では，そのメトリクスに閾値を設定しないことで，“問題あり” の誤検出数を抑制できる．

図 7 は判別境界が中抜き状になるパターンを示す．あるメトリックの値が一定範囲にあるかどうかが重要な場合にはこの解釈になりうる．行数や複雑度等は大きすぎることのほかに小さすぎることもまた実装の不自然さの兆候であり，ゴルディロックの原理と呼ばれて研究されてきた [40], [41]．ゴルディロックの原理は成り立つ場合とそうでない場合がそれぞれ報告されているため，我々の枠組みによるコンテキストを考慮したアプローチが有用であると考えられる．

図 8 は判別境界が階段状になるパターンを示す．あるメトリックの値の大きさ次第で他のメトリクスの判断基準が変わる場合にはこの解釈になりうる．たとえば規模については，担っている機能の違うコードどうしで行数や関数定義数の中心傾向が違う場合がある．また，自動生成コードやテストコード，MVC におけるコントローラクラス等では評価基準に違いが出やすいことも知られている [42]．枠組みを適用する際の方針として，これらのコードを例外として評価対象にしないようにすることも，枠組みの適用を経てどのように違っているかを確認することもできる．

図 9 は判別境界判別境界に突出した箇所があるパターンを示す．局所的に判断基準が変わることが特徴的である．突出した箇所に該当するコードの特徴が，そのコンテキストにおいて普遍的なものなのか，教師データの元であるソフトウェア特有のものであるかに注意すべきである．後者の場合，解釈モデルが同じコンテキストの他のソフトウェアにとって適していない恐れがある（過学習の恐れがある）．

3.3 実施手順の詳細

我々の枠組みは評価対象とするソフトウェア要素の粒度には依存せずに適用可能である．本節では記述を簡潔にするために，評価対象の要素がソースコードファイルの場合における実施手順を説明する．

3.3.1 Step1) コンテキスト代表のソフトウェア群の選択

- 開発組織においてコードの自動評価を導入したいコンテキストを特定し，そのコンテキストを代表するソフトウェア群（ベンチマーク）を用意する．

組込み製品ベンダ等，複数の製品機種を長期間にわたり開発・保守をしている組織では，近いコンテキストのソフトウェアが蓄積されやすく，ベンチマークを用意しやすいと考えられる．

この Step で構築したベンチマークは着目したコンテキストをよく代表する一方で，他のコンテキストまでに広げた一般性は欠きやすい．そのため，我々の枠組みで導出した評価基準は，同一のプログラミング言語とアプリケーションドメインのソフトウェアでのみ共有することを制約とする．その際，アプリケーションドメインの同一性については開発組織が経験的に判断するものとする．コンテキストの同一性の判定方法の具体化と評価については今後の課題である．

表 2 GQM モデルの例：Goal（目標），Question（質問），Metrics（測定法）
Table 2 GQM model: Goals, Questions, and Metrics.

Goal	Question	Metrics	ファイル単位メトリクスの説明
G：コードが分かりやすく書かれており，開発者にとって保守性が高いこと．	Q1：ファイルの責務の量は適切か?	FUN	ファイル内で定義されている関数の数．Number of Function definitions の略．
		ELOC	ファイル内のコード行数を，処理実行の主体以外の行を無視しながら数えたものであり，無視されるのは空行，コメント行，括弧のみの行，変数宣言のみの行等である．Executable LOC の略．
	Q2：関数の処理が複雑すぎないか?	AveDN	ファイル内の関数の DN の平均値．Average of DN の略．
		MaxDN	ファイル内の関数の DN の最大値．Maximum of DN の略．
		AveCC	ファイル内の関数の CC の平均値．Average of CC の略．
		MaxCC	ファイル内の関数の CC の最大値．Maximum of CC の略．

表 1 基礎的なソースコードメトリクス
Table 1 Basic metrics of the source code.

メトリクス	説明
LOC	コードファイルの行数を，コメント行や空行も含めて数えたもの．Lines Of Code の略．
DN	ある関数について，その制御フローのネストの深さの最大値．ネストのある箇所とみなすのは *if, for, while, do, switch* を使用している箇所とする．Depth of Nest の略．
CC	ある関数について，そのサイクロマティック複雑度 [50]．計算方法は「制御フローの分岐の数+1」となる．Cyclomatic Complexity の略．

また，OSS（Open Source Software）と産業用ソフトウェアとでは多くの種類のメトリクスで分布が異なる傾向にあるとの報告があるため [43]，OSS をベンチマークに含める場合は，注意して対象を選ぶべきである．

3.3.2 Step2) コード評価の GQM モデルの定義

- 保守性の評価方法や，保守上の問題のパターン集・カタログ等を参考にして，対象コンテキストにおいてコードを評価する際の観点とメトリクスを定義する．

コードをどのように評価するかは，Goal-Question-Metric（GQM）モデルで定義すると分かりやすい [44]．具体的には，「コードが分かりやすく，開発者にとって保守性が高いこと」を目標（Goal）とし，達成可否について複数の質問（Question）を設け，定量的に回答するための測定方法（Metrics）を定義する．たとえば，クラス（ファイル）が担う責務量が大きすぎないか，という質問を設けたら，

– $Metric_1$：ファイル内の関数定義数．
– $Metric_2$：ファイルのコード行数．

といったメトリクスを関連付けていく．GQM モデルでは閾値までは定義していないので，大枠のみ他のコンテキストに流用することは許容できる．例として，我々が本研究で定義した基礎的なメトリクスを**表 1** に，GQM モデルを**表 2** に示した．

採用する Question や Metrics の定義をしていく際には，ソフトウェア品質評価の国際規格や [45]，発展性欠陥の分類 [10]，コードスメルのパターン [46], [47]，既存の GQM モデル [48], [49] 等が参考になる．これらの体系的枠組みには多量な観点とメトリクスを含まれているので，試験的導入の際には自組織が特に評価したい項目を抽出してカスタマイズすべきである．

3.3.3 Step3) 静的解析によるメトリクス測定

- GQM モデルで定義したメトリクスを測定する．測定対象はベンチマークに含まれる各ソースコードファイルとする．測定ツールの種類は問わない．

具体例として，代表的なソースコード静的解析ツールである Understand の説明をする[*3]．Understand は複数のプログラミング言語（C/C++，C#，Java 等）に対応しており，行数やサイクロマティック複雑度を測定可能である．メトリクス測定値は CSV ファイルとして出力できる．また，API を用いれば独自メトリクスの測定も可能である．

なお，静的解析ツールを利用する際には，結合度のメトリクスの測定仕様に注意すべきである．たとえば，Integer や String 等のプリミティブクラスも数えるか，外部ライブラリのクラスも数えるか，明示的でない依存関係も数えるかが異なる場合がある．また，行数やサイクロマティック複雑度といったファイル単体で完結して測定可能なメトリクスについても，複数の測定パターンが採用されている場合もあるため同様に注意すべきである．

3.3.4 Step4) レビュー対象ファイルの選択

- ベンチマークとしたソフトウェア群から一部のソースコードファイルを抽出し，次の Step5 でのエキスパートによるレビューの対象とする．

ベンチマーク全体のレビューは難しく，限られた範囲で教師データを集めることが現実的となる．その理由は，観点によってはドメイン知識を有するエキスパートでないと難しいうえに [10]，そのようなエキスパートには種々の業務も集中しやすい [36] からである．

抽出すべきファイルの数についてはいくつかの知見がある．Rahaman らは現実的かつ効果的な量として全体の 5%から 20%を提案している [51]．また，Fontana らは 100 件程度でもコードスメルの機械学習において有用であった

[*3] Understand: http://www.scitools.com/

と報告している [34]．100 という数値は一見大きいが，大規模なソフトウェアでは機能とファイルが細分化されている場合もあるため，非現実的な数とはいいきれない．また，本研究の実験結果を先にあげると，約 22 個の教師データで学習した場合でも，80 パーセンタイルより高精度な自動評価を実現できた．

抽出すべきファイルの内容はカスタマイズ方針によるため，どのような方法がより実用的な評価基準の導出に寄与するかは明らかではない．そのため，具体例を参考として開発組織が行動指針を決定するものとした．具体例は以下のとおりである．各方法の評価は今後の課題である．

- 仮定をおかずに，ファイルをランダムに選択する．
- Fontana らの研究では [34]，既成ツール（iPlasma 等）により問題個所の候補を絞り込んだ．さらに，人手でレビュー可能な数になるようにランダム選択した．
- 本研究では，協力企業が組込み製品（建機）の機種展開をしていくための母体となっている複数のソフトウェアシステムを代表とした．そして，人手でレビュー可能な数を選択する際には，まずレビュアが内容を熟知しているサブシステム群を特定した．最後に，教師データの正例と負例の偏りを低減させることを意図し，問題のある個所とない箇所を同程度含んでいると想定されるサブシステム群を数個選択した．

3.3.5 Step5) エキスパートによるレビューの実施

- GQM モデルの各 Question をチェックリスト項目として，各ファイルをエキスパートがレビューする．
- 各 Question について，各ファイルごとに “問題あり” か “問題なし” のいずれかを記録する．

エキスパートとしてはコンテキストに精通した熟練開発者や設計者等がふさわしい．M 個のファイルを N 個の Question でレビューする場合には，M 行 N 列のテーブルを結果として記録することになる．

問題の有無を二値に振り分けることが難しい場合には，何点未満なら “問題あり” なのかを明示したうえで，5 段階や 100 点満点で採点してもよい．レビュー後に自動的に二値に変換できる．

3.3.6 Step6) 分類木学習による解釈モデルと閾値の導出

- 分類木学習アルゴリズムにより，メトリクス測定値の解釈モデルを構築する．
- 各 Question について，それぞれ独立に実施する．

C4.5 系列の分類木学習アルゴリズムは複数の統計環境において実装および提供されている．本研究では統計言語 R で実装されている C5.0 アルゴリズムを用いた．

N 個の Question がある場合には，N 個の解釈モデルを構築することになる．この方式では，個々の Question ごとに関連するメトリクスの影響関係を考慮しつつ，簡潔なモデルを構築できる．現時点では，複数の Question をまたいだ影響関係は考慮していない．この制限の緩和や解決については今後の課題としたい．

3.3.7 Step7) 妥当性の議論

- コンテキストに詳しい開発者や品質管理者を交え，解釈モデルの妥当性を議論する．
- もし許容できない内容の場合，以前の Step からのやり直しを検討する．

コンテキストに詳しい開発者や品質管理者を交えることで，解析したデータに含まれない背景情報も含めた検討ができる [30], [31]．積極的で示唆のある議論をするためには，以下の 2 つの要素が重要であると我々は考えた．

– 性能指標：コードの自動評価がエキスパートのレビュー結果と一致している度合いが定量化されていないと比較や議論をしにくい．
– 可視化：解釈モデル（分類木）をテキストのまま見ても，特徴量空間上に分布する実データとの対応関係は直感的に理解しにくい．

性能指標については，Precision，Recall，F 値（F1 値とも呼ばれる）等を “問題あり” のファイルを陽性と見なして計算する．同じ F 値の手法どうしについては再現率と適合率を比較したトレードオフの議論が可能であるため，開発組織が目的に応じて判断する．たとえば，再現率が低くても適合率が高い場合は，問題個所の検出漏れがある代わりに検出した範囲内では正答率が高い．一方で，再現率が高くても適合率が低い場合は，問題個所の検出漏れが少ない代わりに，誤検出が多く含まれる．

可視化については，もし分類木が最終的に含むメトリクスが 2 個以下なら，図 6 から図 9 のような図を作成できる．この上にメトリクス測定値の散布図を重ねて，解釈モデルと閾値の内容が対象コンテキストの開発者の認識に反しないかを議論する際に利用する．確認手順の具体化と評価，説明変数が 3 つ以上の場合の効果的な可視化方法については今後の課題である．統計解析言語 R において contour 関数や ggplot2 パッケージの geom_tile 関数を用いれば，自動評価における “問題あり” と “問題なし” の境界を塗り分ける処理を実装できる．

また，教師なし手法と教師あり手法の性能比較により，教師データが評価基準の精度改善に寄与しているかを確認することも議論の際には有用である．もし，性能（F 値等）の差が小さい場合には，教師データ作成者が元々単純な解釈をしていることや，機械学習のチューニングパラメータが不適切であること等が理由として考えられる．前者の場合，教師データを増やし続けても導出される評価基準の精度は一定のままである可能性があり，教師データの作成において他の観点を優先する方が有用である可能性がある．

4. 実験方法

4.1 概要

我々の枠組みの実用性を評価するために，2 種類の実験

表 3 ベンチマーク規模の要約（C++部分のみ本実験で利用）
Table 3 Descriptive numbers of our initial benchmark. (C++ parts are mainly used in this study).

ソフトウェア (システム)	C 言語の部分 ファイル数	総 LOC	C++の部分 ファイル数	総 LOC
Sys1	79	24,956	502	154,004
Sys2	86	28,272	586	181,163
Sys3	81	42,237	319	47,938
合計	246	95,465	1,407	383,105

で RQ1 から RQ3 に取り組んだ．

- RQ1：我々の枠組みにより，実用的な評価基準を得られるか？
- RQ2：少量の教師データを入力とした場合でも，分類木学習により従来の閾値導出手法に比べ高精度な評価基準を導出できるか？
- RQ3：教師データの量は，自動評価の精度にどう影響するか？

コードメトリクス測定とエキスパートによるレビューの結果は，建機制御用の C++組込みソフトウェア群を題材として，3 章で説明した Step1 から Step5 までを実施することで得た．

実験 1 では，Step6 と Step7 を実施することで RQ1 に取り組んだ．

実験 2 では，閾値導出手法の性能を比較することで RQ2 から RQ3 に取り組んだ．比較した閾値導出手法は以下のとおりである．教師データを用いない手法については，パーセンタイル関数と Alves 法を用いた．教師あり学習を用いる手法については，Bender 法（ロジスティック回帰ベース），ROC 曲線法，C5.0 アルゴリズムを用いた．

4.2 データセット

コード自動評価の対象コンテキストとして，小松製作所が建機制御用の組込みソフトウェア群の共通プラットフォームとして継続的に開発・保守しているソフトウェアシリーズ（C++）を選んだ．そして，コンテキストの代表として一定の妥当性があると考えられる建機の機種展開における母体とされてる 3 個のソフトウェア（Sys1，Sys2，Sys3）をベンチマークとして選んだ．これら規模については**表 3** にまとめた．なお，3 つのソフトウェアは C/C++で多言語開発されていたが，ファイル数が占める割合は C++の方が 85%（1407/1653）と大きい．

4.3 GQM モデル

対象コンテキストで取り扱う GQM モデルを定義する際には，我々の過去の研究成果である GQM モデルを参考とした [48], [52]．メトリクスの粒度はファイル単位に揃え，保守性のうちの特にコードの分かりやすさや扱いやすさに着目して複数の Question とメトリクスを抽出し，一部変更を加えた．本研究では以下の 2 つの Question とそれに対応するメトリクスを題材として評価実験を実施した．

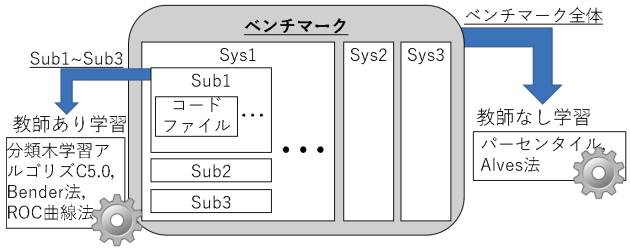

図 10 ベンチマーク構成と実験における利用方法
Fig. 10 Benchmark organization and usage in our experiments.

- **Q1：ファイルの責務の大きさが適切であるか？** 2 つのメトリクスを関連付けた．ファイル内の関数定義数（FUN）とファイル内の実行部分のコード行数（ELOC）である．コメント行や空行を含む LOC ではなく，これらを除外する ELOC をより責務量を表すメトリクスとして採用した．また，FUN を Q1 の説明変数として加えたのは，ファイルの責務の適切さの評価に関するエキスパートの暗黙的な観点を評価基準に反映させようとしたためである．たとえば，行数が多いファイルでも機能が適切に分割されていれば許容する，あるいは関数定義数が多くても行数が少なければ許容するといった観点がありうる．
- **Q2：（ファイル内の）関数の処理が複雑すぎないか？** 4 つのメトリクスを関連付けた．この 4 つは関数の制御フローに関する 2 つのメトリクスを集計したものであり，ネスト深さ（DN）とサイクロマティック複雑度（CC）のファイル単位の平均値（AveDN，AveCC）・最大値（MaxDN，MaxCC）である．DN はネストの深さであり，CC はサイクロマティック複雑度 [50] である．

上記の Question とメトリクスの詳細については表 1 と表 2 に示した．

4.4 エキスパートによるレビューの結果

対象コンテキストのソフトウェア群のアークテクチャに精通し，10 年以上の開発経験を持つエキスパート 1 名によりコードレビューを実施した．その際，ベンチマークのすべてのファイルをエキスパートがレビューすることは難しいため，3 個のサブシステム（Sub1，Sub2，Sub3）をレビュー対象として選定した．まずレビュアが内容を熟知しているサブシステム群を特定した．次に，教師データの正例と負例の偏りを低減させることを意図し，問題のある箇所とない箇所を同程度含んでいると想定されるサブシステム群を 3 個選定した．結果的に，**図 10** に示すように，サブシステム群は Sys1 のみからの選定となった．

表 4 サブシステムの規模とエキスパートによるレビュー結果の要約
Table 4 Descriptive numbers of subsystems and results of the manual inspection.

対象のサブシステム	ファイル数	総 LOC	Q1 問題あり	Q2 問題あり
Sub1	57	13952	17	14
Sub2	31	6926	11	11
Sub3	55	8875	32	20
合計	143	29753	60	45

レビューの結果については**表 4** にまとめた．各 Question について問題ありと判定されるファイルは半数以下であり，Q1 では 42%（60/143），Q2 では 31%（45/143）であった．

なお，約 40%のファイルが問題ありとなっているが，今回のレビューではコードの静的構造上の問題が指摘されており，この数値が小松製作所のソフトウェアの機能的な不具合の多さを直接には意味していない点に留意してほしい．

また，エキスパートは 1 つのファイルにつき数分をかけてレビューしており，ベンチマーク全体（1,407 個）のうち 143 個のファイルのレビューには約 8 時間かかった．

4.5 評価基準

本研究の実験では，複数の閾値導出手法を比較するための性能指標として F 値を用いた．また，実験 2 の教師あり学習の比較では，より詳細な分析のために Precision と Recall も用いた．

F 値の測定の際には，ファイルが “問題あり” であることを陽性と見なして計算した．

F 値の値域は [0.0, 1.0] であり，1 に近づくほど性能が高いことを表す．F 値は Precision と Recall の調和平均であるが，Precision は偽陽性の少なさを表し，Recall は偽陰性の少なさを表す．偽陽性は，エキスパートが “問題なし”（陰性）と判定しているにもかかわらず，自動評価では陽性と誤判定してしまったものを意味する．偽陰性は，エキスパートが陽性と判定しているにもかかわらず，自動評価では陰性と誤判定してしまったものを意味する．偽陽性と偽陰性はそれぞれ自動評価の誤りを表すが，一般に互いにトレードオフの関係にあることが知られており，調和平均によって総合的に評価するのが望ましいとされている．

4.6 実験 1 の手順

3 章で説明した Step6 と Step7 を実施した．Step6 では，統計解析言語 R の C50 パッケージにより，Q1 と Q2 のそれぞれの評価用の分類木を構築した．Step7 では，構築した分類木の内容について開発担当者 3 名を含む 5 名で議論した．また，構築した分類木で Sys2 と Sys3 を評価し，その結果について協力企業の開発者からフィードバックを得た．

4.7 実験 2 のデザインと手順

実験 2 では主に，教師あり学習を用いる閾値導出手法を比較をした．加えて，教師データを用いるからには越えるべきと期待されるターゲットとして教師データを用いない手法とも比較をした．取り扱う閾値導出手法は以下の 5 つである．C5.0 以外の方法では 2.2 節の図 1 で説明した形式でメトリクスの解釈モデルを構築した．

- 教師データを用いない閾値導出手法：パーセンタイル関数，Alves 法．
- 教師あり学習を用いる閾値導出手法：Bender 法，ROC 曲線法，分類木学習アルゴリズム C5.0．

また，図 10 に示すように，閾値導出手法に応じて以下のデータを入力とした．

- 教師データを用いない手法では，ベンチマーク全体に基づいて閾値を決めることが一般的であるため，1407 個の C++ファイルのメトリクスデータを入力とした．
- 教師あり学習を用いる手法では，エキスパートによるレビューの対象とした 143 個の C++ファイルのメトリクスデータと教師データを入力とした．

そして，教師あり学習を用いる手法については，それぞれ，以下の 3 つの規則での実験を 100 回繰り返し，平均値を測定結果とした．

- 規則 1) 143 個の教師データから 72 個（50%）をランダム選択して学習用候補とし，残りをテストデータとする．
- 規則 2) 学習用候補からさらに n%をランダム選択して学習に用いる．その際，n の値は 10 から始め，5 刻みで 100 までを対象とする．
- 規則 3) 規則 3 のランダム選定では正例と負例が必ず 1 件以上含まれるように調整する．学習データに正例か負例の片方しか存在しない場合には，正例と負例を隔てる閾値の導出ができないため，ランダム選定をし直す．

また，パーセンタイル関数，Alves 法，Bender 法では，リスクの大きさを仮定して設定する引数の値次第で導出される閾値が変わる．そのため，本論文では以下のように表記してそれぞれを区別する．

- パーセンタイル関数の引数に 10 パーセントを指定したものは P10 と表記する．
- Alves 法の引数に 10 パーセントを指定したものは A10 と表記する．
- Bender 法の引数に 0.10 を指定したものは B10 と表記する．

5. 実験結果

5.1 実験 1 の結果

5.1.1 Step6 の結果

構築された分類木を二次元で可視化した結果を**図 12** と

図 11 図 12 と図 13 の凡例
Fig. 11 Legend of Figs. 12 and 13.

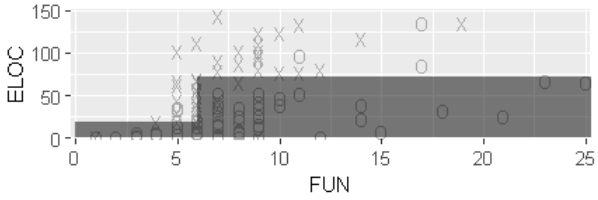

図 12 Q1 について構築した分類木を可視化した散布図
Fig. 12 Visualization of the Classification-tree for Q1.

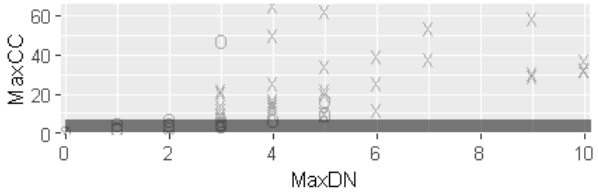

図 13 Q2 について構築した分類木を可視化した散布図
Fig. 13 Visualization of the Classification-tree for Q2.

図 13 に示した．また，これらの図におけるプロット形状と背景色の意味を**図 11** に示した．プロットの形状は各ファイルのエキスパートによるレビューの結果を表しており，“X” と “O” が “問題あり” と “問題なし” を意味する．また，背景色は自動評価の解釈モデルを表しており，淡い灰色と濃い灰色が “問題あり” と “問題なし” を意味する．構築した分類木のテキストベースの表現は以下の内容となった．

- Q1) ファイルの担う責務が大きすぎないか？
  If(FUN $\leq$ 6)
  IF(ELOC $\leq$ 19): 問題なし
  ELSE : 問題あり
  Else
  IF(ELOC $\leq$ 71): 問題なし
  ELSE : 問題あり
- Q2) ファイル内の関数が複雑すぎないか？
  IF(MaxCC $\leq$ 7): 問題なし
  ELSE : 問題あり

Q1 の分類木は FUN と ELOC の非直交な関係性に解釈を与えるものとなった．FUN の値に応じて，ELOC では 2 つの閾値が使い分けられている．Q2 の分類木では MaxCC のみ用いられ，他の 3 つのメトリクスは除外された．

5.1.2 Step7 の結果

Q1 と Q2 の分類木の内容について開発担当者 3 名（教師データ作成者ではない）と研究室メンバ 2 名で議論し，その内容に一定の妥当性があることを確認した．Q1 と Q2 の分類木の性能指標である F 値はそれぞれ 0.88 と 0.91 であり，エキスパートによるレビュー結果との一致度が高かった．また，解釈モデルと閾値の内容が対象コンテキストの開発者の認識に著しく反するものではないことを可視化を通じて確認した．以下に議論の結果を示す．

Q1 で得られた解釈モデルは対象コンテキストにおいて妥当と判断した．このコンテキストではソフトウェアの機能は細分化されており，個々の機能を担うファイルには以下の傾向があった．

- ある程度の数の関数を持つファイルでは，それらの関数を組み合わせて使うため，実行コード行数がやや多い．
- コアでない周辺ファイルでは，少数の関数のみが含まれ規模も小さい．

そのため，関数の定義数に応じて，許容される実行コード行数の大きさに数通りのパターンがあるのは自然と考えた．F 値も 0.88 と高いため，現状ではこの分類木を採用した．ただし，今後教師データを追加して再学習することで分類木が同様の評価基準を維持するかどうかを確認することも視野に入れることとした．

Q2 で得られた解釈モデルは対象コンテキストにおいて妥当と判断した．ネストの深さに関するメトリクスが評価基準に含まれなかったが，このコンテキストでは多分岐の switch-case 文を使う傾向があり，ネストが浅くても実行経路が複雑になっている関数が多々あった．また，AveCC も評価基準に含まれなかったが，MaxCC と相関しており片方は冗長となる関係にあった．

特徴的であったのは解釈モデルの形状であった．Q1 の解釈モデルは開発者が明示的には意識していなかった階段状パターンになった．一方で，Q2 の解釈モデルは 1 種のメトリックと 1 個の閾値からなる単純な形式になった．この結果ついては，Q1 と Q2 の性質の違いが影響していたと考えられる．1 つ 1 つのファイルの規模は必要な機能（責務）の内容や量によって必ずしも小さくしきれない一方で，関数の構造的複雑度は関数分割を徹底すれば一律に抑制できると考えられる．そのため，少なくともこのコンテキストにおいては，人間によるコードファイルの規模の解釈は多面的なものに，構造的複雑度の解釈は単純なものになりやすい可能性がある．

5.1.3 評価基準の妥当性に関するフィードバック

本実験では，結果的に，教師データを作成するためのレビュー対象ファイルを Sys1 のみから選定していた．そのため，機械学習時に過学習をし，Sys2 と Sys3 に適さない評価基準が導出される恐れがあった．しかしながら，Sys1，Sys2，Sys3 は同一の開発組織によって開発されていたため，一定の共通性が認められると我々は考えた．また，導出した分類木による Sys2 と Sys3 の評価結果について，同組織の開発者から一定の妥当性があったというフィード

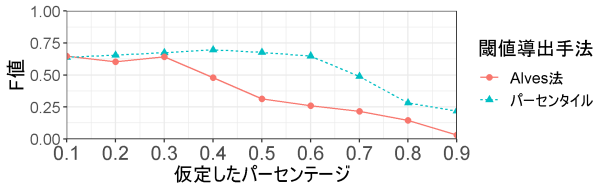

図 14 パーセンタイル関数と Alves 法の F 値（Q1）
Fig. 14 F-measure of PERCENTILE and ALVES for Q1.

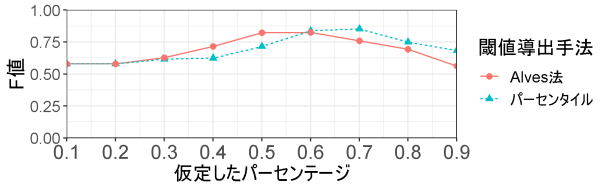

図 15 パーセンタイル関数と Alves 法の F 値（Q2）
Fig. 15 F-measure of PERCENTILE and ALVES for Q2.

バックを得た．そのため，導出した評価基準には一定の外的妥当性があったと考えられる．また，同一のコンテキストで評価基準を共有するという我々の想定にも一定の妥当性があったと考えられる．

5.2 実験 2 の結果

本節では実験結果を要約する折れ線図を示し，その読み方を説明する．図から読み取れる事象と，それに対する考察は 6 章で説明する．

5.2.1 教師データを用いない閾値導出手法ついて

パーセンタイル関数と Alves 法のそれぞれの方法において自動評価を実施することで F 値を測定した．Q1 と Q2 の測定結果はそれぞれ**図 14** と**図 15** に示したこの図の横軸と縦軸はそれぞれ，パーセンテージと F 値の大きさを意味する．パーセンタイル関数と Alves 法では，リスク境界として仮定するパーセンテージ次第で導出される閾値が変わる（自動評価の結果も変わる）．本実験では，10%ごとのパーセンテージそれぞれを仮定した場合の F 値を測定し，折れ線を引いて示した．

5.2.2 教師あり学習を用いる閾値導出手法ついて

Bender 法，ROC 曲線法，分類木学習アルゴリズム C5.0，それぞれの方法において自動評価を実施することで F 値，Precision，Recall を測定した．

Q1 と Q2 の F 値の測定結果はそれぞれ**図 16** と**図 17** に示した．この図の横軸と縦軸はそれぞれ，手持ちの教師データのうち学習に用いた割合と F 値の大きさを意味する．図 16 と図 17 の横軸の意味が，図 14 と図 15 とは異なる点に注意してほしい．

本実験では，5%ごとにそれぞれの割合で学習した場合の F 値を測定し，折れ線を引いて示した．加えて，比較対象として教師データを用いない閾値導出手法で測定した F 値について特徴的な位置に直線を引いて示した．

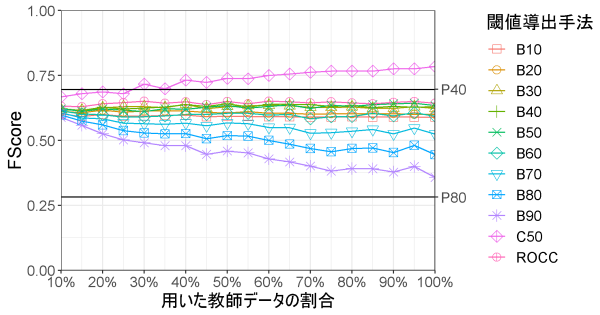

図 16 教師データの割合と F 値の関係（Q1）
Fig. 16 Sampling percentages and the F-measure (Q1).

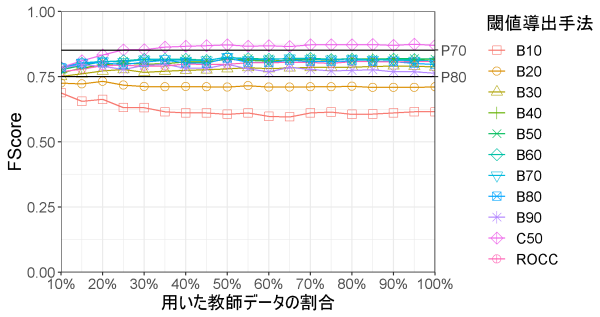

図 17 教師データの割合と F 値の関係（Q2）
Fig. 17 Sampling percentages and the F-measure (Q2).

図 16 と図 17 で直線を引いたのは，P80 と一番 F 値が大きかったパーセンタイルの位置である．P80 については，80 対 20 の法則の知名度が高く，教師データなしの場合に利用される可能性が高いと考えられる．一番 F 値が大きかったパーセンタイルについては，Q1 では P40，Q2 では P70 であった．あらかじめ “問題あり” のファイルの割合のノウハウがあるならば，教師データなしでも高性能なパーセンタイルを導ける可能性があり，比較対象としての意義がある．教師あり学習を用いる閾値導出手法には，これらの 2 つのターゲットを越えることが期待されている．

そして，図 16 と図 17 は C5.0 が比較対象の中では最も高性能であることを示している．また，従来手法では教師あり学習を用いているものの，パーセンタイル関数の性能を下回る場合があることを示している．

ここまでの F 値の結果に加えて，教師あり学習の比較では，より詳細な分析のために Precision と Recall も同様に測定した．Q1 と Q2 の Precision は**図 18** と**図 19** に示した．Q1 と Q2 の Recall は**図 20** と**図 21** に示した．それぞれ，F 値の図と同様の形式である．

Q1 と Q2 の双方において，C5.0 では教師データを増やすにつれ，再現率と適合率が双方とも上昇する傾向であった．ROC 曲線法や Bender 法の B40，B50 等は再現率と適合率が双方ともほぼ一定という傾向であった．また，B10，B20 等は再現率が高い一方で適合率が低いという傾向であった．逆に，B90，B80 等は再現率が低い一方で適合率が高いという傾向であった．

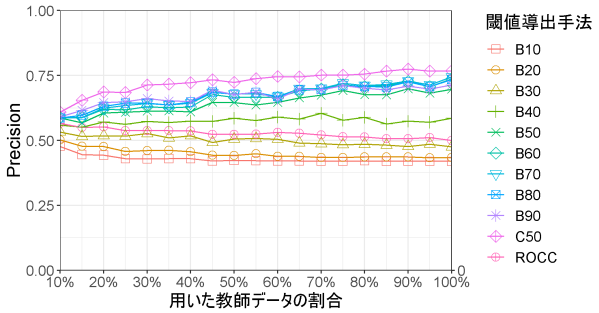

図 18 教師データの割合と Precision の関係（Q1）
Fig. 18 Sampling percentages and the Precision (Q1).

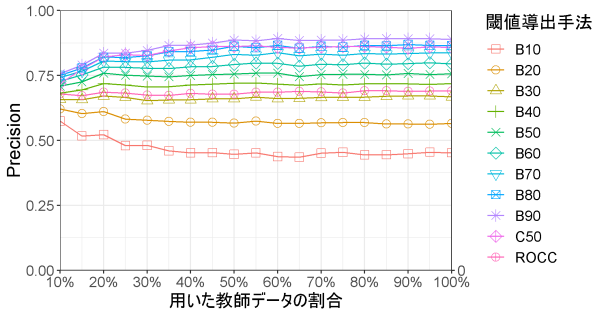

図 19 教師データの割合と Precision の関係（Q2）
Fig. 19 Sampling percentages and the Precision (Q2).

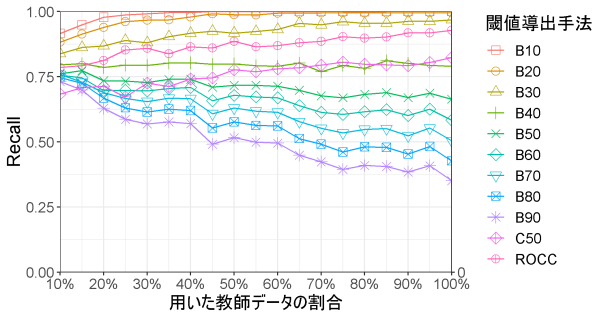

図 20 教師データの割合と Recall の関係（Q1）
Fig. 20 Sampling percentages and the Recall (Q1).

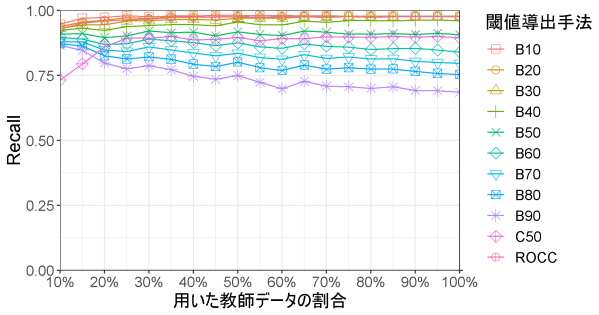

図 21 教師データの割合と Recall の関係（Q2）
Fig. 21 Sampling percentages and the Recall (Q2).

C5.0 は他の手法に比べて Precision が高く，Recall も減少傾向ではなかったため，高い F 値が得られたと考えられる．他の手法は，Precision と Recall のどちらか一方が高くもう片方が低い，あるいはどちらも中程度という傾向であった．

6. 考察

本章では RQ1 から RQ3 について，本研究の評価実験の結果に基づいて考察を深める．

6.1 RQ1：我々の枠組みにより，実用的な評価基準を得られるか？

YES．Q1 と Q2 の双方において妥当な解釈モデルを構築できたことを，開発担当者 3 名を含む 5 名による議論を通じて確認した．議論の経緯については，実験結果として 5.1.2 項で説明した．レビューを実施したエキスパートや議論に参加した開発者は対象コンテキストに精通していたため，対象コンテキストにおける成果は妥当と考えた．

今後は，教師データを追加して再学習することで，分類木が同様の評価基準を維持するかどうか確認することを検討している．たとえば Q1 については，濃い灰色の矩形の理想的な形について議論の余地がある．対象コンテキスト内で教師データを増やしたり，対象コンテキストの範囲をやや広げたりという変化に対して，矩形がどのように変化するかである．フラットな形状と階段形状のどちらがより良いのか，階段形状ならば 2 段か，あるいはそれ以上の段数が良いのか，この疑問については判別領域の境界に教師データを加えていく能動学習のアプローチで取り組むことを検討している．

6.2 RQ2：少量の教師データを入力とした場合でも，分類木学習により従来の閾値導出手法に比べ高精度な評価基準を導出できるか？

YES．図 16 と図 17 は，C5.0 が従来の閾値導出手法よりも高性能であることを示している．Q1 と Q2 のいずれにおいても，教師データを用いる割合が 35%以上の場合，C5.0 が最も高性能であった．また，72 個の 35%は 26 個（小数繰り上げ）であるが，これは十分に少ない量と考えられ，我々の枠組みの実用性を示している．

以下，比較対象である C5.0 以外の手法について考察する．

Question ごとに P80 の F 値を見ると，Q1 では 0.28 であり低性能であるが，Q2 では 0.75 であり上位の性能であった．Q1 の P80 の自動評価結果を分析したところ，偽陰性（誤って “問題なし”）の判定結果が 34%もあり，P80 では閾値が大きすぎることが分かった．

Question ごとに一番 F 値が大きかったパーセンタイルの F 値を見ると，Q1 の P40 では 0.70 であり，Q2 の P70 では 0.85 であった．双方ともに ROC 曲線法と Bender 法と同等あるいは上回る性能であった．

ROC 曲線法と Bender 法において問題となるのは，パーセンタイルを大きく差をつけて上回ることがなかった点である．Q1 のように，エキスパートがコードの複数の特徴を複合的に解釈している場合には，メトリック 1 種ごとに

閾値を最適化することが適さない可能性がある．また，Q2のように，エキスパートがパーセンタイルに類する判断をしている場合には，C5.0も含めて教師あり学習とパーセンタイル関数とで性能に差が出にくいと考えられる．

6.3 RQ3：教師データの量は，自動評価の精度にどう影響するか？

C5.0では教師データを増やすことによる精度の改善が見られた．一方で，ROC曲線法とBender法では，教師データの割合を増やしてもF値があまり向上しない，あるいは逆に下がるという結果になった．1種のメトリックに1個の閾値を設定する形式が要求されていないならば，C5.0を用いる方がよいと考えられる．

まず，C5.0の結果について考察する．図16と図17を見ると，Q1とQ2では教師データの割合に対する性能の変化量が異なる．学習用候補から使用する教師データが10%の場合と100%の場合とで，F値の変化量は，Q1では0.11（=0.78−0.67），Q2では0.08（=0.87−0.79）であった．

6.2節でも考察したコンテキストの背景をふまえると，以下のことが推測できる．

- Q1のように複合的な解釈の学習の場合では，より多くの教師データが役立つ．
- 一方で，Q2のように元々が単純な解釈の場合では，非常に少ない教師データでも十分となりうる．

今回の実験結果から，評価対象の観点（GQM Question）によって十分となる教師データの数が異なると推測できる．たとえば，処理の複雑度の低減は関数分割によって比較的一律にできるが，ファイルが担う規模（必要な機能）の削減には突き詰めるとアルゴリズムや設計の見直しが必要となる．そのため，コードの規模の評価は比較的に難しく，エキスパートは複合的な観点になりやすい可能性がある．

以下，比較対象であるC5.0以外の手法について考察する．図16と図17を見ると，ROC曲線法とBender法では，教師データの割合を増やしてもF値の向上が小さい，あるいは逆に下がるという結果になった．

ROC曲線法やB40，B50等は再現率と適合率が双方ともほぼ一定という傾向であったが，これらの従来手法ではメトリクスの複合的な解釈の最適化ができずに性能が伸び悩んでいたと考えられる．

B90，B80等は再現率が低い一方で適合率が高いという傾向であったが，Q1では教師データを増やすにつれてF値が下がっていた，Bender法において引数p0を0.90にすると（B90），楽観的に問題個所を少なく見積もることになる．そのため，Q1では極端に大きな閾値が導出され，特に明白な問題箇所以外が検出漏れしてしまっていたと考えられる．

B10，B20等は再現率が高い一方で適合率が低いという傾向であったが，Q2では教師データを増やすにつれてF値が下がっていた．Bender法において引数p0を0.10にすると（B10），悲観的に問題個所を多く見積もることになる．そのため，Q2では極端に小さな閾値が導出され，多くのファイルが問題個所として誤検出されてしまっていたと考えられる．

6.4 評価基準の精度に関する関連研究

2.3節で述べたFontanaらの研究では[34]，4種類のコードスメルの自動評価について我々の実験よりも高いF値が出ていた．本研究との差の原因としては，機械学習時に説明変数としたメトリクスの数の影響が考えられる．Fontanaらは60個以上のメトリクスを使用しており，一方で我々はQ1とQ2でそれぞれ2個と4個のメトリクスを使用していた．本研究では，評価過程や考慮されている範囲が人間にとって分かりやすい評価基準の導出を意図し，コード評価基準は閾値に基づくものに限定し，GQM（Goal-Question-Metrics）法により説明変数となるメトリクスも限定していた．メトリクスを限定する方法の評価については今後の課題である．

7. 妥当性への脅威

内的妥当性：教師データを作成したエキスパートが1名だけであり，レビュー実施時の集中力や先入観といった個人的要因の影響を排せていない点は内的妥当性への脅威である．しかし，このエキスパートは対象企業において10年以上の開発経験を持つ設計者であったため対象コンテキストには十分に精通していた．また，枠組みのStep7では，学習結果である評価基準を可視化して判別境界付近のデータに着目し，評価基準が対象コンテキストの開発者の認識に著しく反するものではないことを，対象企業の開発者3名と研究室メンバ2名で確認した．そのため，教師データには一定の内的妥当性があったと考えられる．

外的妥当性：本研究では，結果的に，教師データを作成するためのレビュー対象ファイルをSys1のみから選定していた．そのため，機械学習時に過学習をし，Sys2とSys3に適さない評価基準が導出される恐れがあった．しかしながら，Sys1，Sys2，Sys3は同一の開発組織によって開発されていたため，一定の共通性が認められると我々は考えた．そして，導出した分類木によるSys2とSys3の評価結果について，同組織の開発者から一定の妥当性があったというフィードバックを得た．そのため，導出した評価基準には一定の外的妥当性があったと考えられる．また，同一のコンテキストで評価基準を共有するという我々の想定にも一定の妥当性があったと考えられる．より具体的な実証実験については今後の課題である．

本研究では，元々の教師データが143個と少量であったため，サンプルの割合を変動させるのに十分な学習データを確保するために，2-fold交差検証に類する形で実験をし

た．しかし，2 回しか機械学習を実施しない 2-fold の形式では，10-fold の形式に比べてデータの偏りの影響を低減させにくいという欠点がある．本研究では実験を 100 回繰り返して，その平均値を測定結果とすることで外的妥当性への脅威の低減に取り組んだ．

8. おわりに

本研究ではコードの保守性のうち，機能的な不具合をともなわないコードの構造上の問題に焦点を当て，閾値に基づく評価基準を統計的に導出するための手順を枠組みとして整理した．さらに，従来の一般的な閾値導出手法と分類木学習の実用性を比較するために，少量の教師データにより高精度かつ開発者の認識に反しない評価基準を導出できるかという観点で評価実験をした．

- 分類木学習アルゴリズム C5.0 により，少量の教師データでも高精度かつ開発者の認識に反しない評価基準を導出できることを確認した．少量の教師データでも，80 パーセンタイルや ROC 曲線法等の従来手法を上回る性能を期待できる．
- 分類木学習アルゴリズム C5.0 では教師データを増やすことで自動評価の精度（F 値）が改善された．一方で，1 種のメトリックに 1 個の閾値を設定する形式である Bender 法と ROC 曲線法では，教師データを増やしても顕著な改善は見られなかった．
- 本研究ではコードの規模と複雑度の 2 つの観点の評価基準をカスタマイズしたが，規模の観点の方が多数の教師データを高精度化のために必要とした．各観点において，エキスパートのコードの解釈が複雑であるかどうかが影響したと考えられる．

ただし，今回の評価実験の結果は 1 つの企業における 1 つのコンテキストにおいて限定的に得られたものである．そのため今後は，題材としたコンテキストにおける教師データを追加して分析を深めるとともに，異なるコンテキストでの調査を開始することを検討している．

参考文献

[1] Buse, R.P. and Weimer, W.R.: A Metric for Software Readability, *Proc. 2008 International Symposium on Software Testing and Analysis, ISSTA '08*, pp.121–130, ACM (2008).

[2] Alves, T.L., Ypma, C. and Visser, J.: Deriving Metric Thresholds from Benchmark Data, *Proc. 2010 IEEE International Conference on Software Maintenance, ICSM '10*, pp.1–10, IEEE Computer Society (2010).

[3] Foucault, M., Palyart, M., Falleri, J.-R. and Blanc, X.: Computing Contextual Metric Thresholds, *Proc. 29th Annual ACM Symposium on Applied Computing, SAC '14*, pp.1120–1125, ACM (2014).

[4] Oliveira, P., Valente, M.T. and Lima, F.P.: Extracting relative thresholds for source code metrics, *2014 Software Evolution Week – IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering* (*CSMR-WCRE*), pp.254–263 (2014).

[5] 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).

[6] Tsantalis, N. and Chatzigeorgiou, A.: Identification of Move Method Refactoring Opportunities, *IEEE Trans. Softw. Eng.*, Vol.35, No.3, pp.347–367 (2009).

[7] Marinescu, R.: Assessing technical debt by identifying design flaws in software systems, *IBM Journal of Research and Development*, Vol.56, No.5, pp.9:1–9:13 (2012).

[8] Marinescu, R.: Detection strategies: Metrics-based rules for detecting design flaws, *Proc. 20th IEEE International Conference on Software Maintenance*, pp.350–359 (2004).

[9] Lanza, M. and Marinescu, R.: *Object-oriented Metrics in Practice*, Springer-Verlag, Berlin (2006).

[10] Mäntylä, M.V. and Lassenius, C.: What Types of Defects Are Really Discovered in Code Reviews?, *IEEE Trans. Softw. Eng.*, Vol.35, No.3, pp.430–448 (2009).

[11] Kruchten, P., Nord, R.L. and Ozkaya, I.: Technical Debt: From Metaphor to Theory and Practice, *IEEE Software*, Vol.29, No.6, pp.18–21 (2012).

[12] Zhang, F., Mockus, A., Zou, Y., Khomh, F. and Hassan, A.E.: How Does Context Affect the Distribution of Software Maintainability Metrics?, *2013 IEEE International Conference on Software Maintenance*, pp.350–359 (2013).

[13] Gil, J.Y. and Lalouche, G.: When do Software Complexity Metrics Mean Nothing? – When Examined out of Context, *Journal of Object Technology*, Vol.15, No.1, pp.2:1–25 (2016).

[14] Lavazza, L. and Morasca, S.: An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measures, *Proc. 12th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2016*, pp.6:1–6:10, ACM (2016).

[15] Shatnawi, R.: A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems, *IEEE Trans. Softw. Eng.*, Vol.36, No.2, pp.216–225 (2010).

[16] Mäntylä, M.V. and Lassenius, C.: Subjective Evaluation of Software Evolvability Using Code Smells: An Empirical Study, *Empirical Softw. Eng.*, Vol.11, No.3, pp.395–431 (2006).

[17] Bender, R.: Quantitative risk assessment in epidemiological studies investigating threshold effects, *Biometrical Journal*, Vol.41, No.3, pp.305–319 (1999).

[18] Sánchez-González, L., García, F., Ruiz, F. and Mendling, J.: A study of the effectiveness of two threshold definition techniques, *16th International Conference on Evaluation Assessment in Software Engineering* (*EASE 2012*), pp.197–205 (2012).

[19] Shatnawi, R., Li, W., Swain, J. and Newman, T.: Finding Software Metrics Threshold Values Using ROC Curves, *J. Softw. Maint. Evol.*, Vol.22, No.1, pp.1–16 (2010).

[20] Herbold, S., Grabowski, J. and Waack, S.: Calculation and Optimization of Thresholds for Sets of Software Metrics, *Empirical Softw. Eng.*, Vol.16, No.6, pp.812–841 (2011).

[21] Erni, K. and Lewerentz, C.: Applying design-metrics to object-oriented frameworks, *Proc. 3rd International Software Metrics Symposium*, pp.64–74 (1996).

[22] Concas, G., Marchesi, M., Pinna, S. and Serra, N.: Power-Laws in a Large Object-Oriented Software System, *IEEE Trans. Softw. Eng.*, Vol.33, No.10, pp.687–708 (2007).
[23] Wheeldon, R. and Counsell, S.: Power law distributions in class relationships, *Proc. 3rd IEEE International Workshop on Source Code Analysis and Manipulation*, pp.45–54 (2003).
[24] Fontana, F.A., Ferme, V., Zanoni, M. and Yamashita, A.: Automatic Metric Thresholds Derivation for Code Smell Detection, *Proc. 6th International Workshop on Emerging Trends in Software Metrics, WETSoM '15*, pp.44–53, IEEE Press (2015).
[25] Vale, G., Albuquerque, D., Figueiredo, E. and Garcia, A.: Defining Metric Thresholds for Software Product Lines: A Comparative Study, *Proc. 19th International Conference on Software Product Line, SPLC '15*, pp.176–185, ACM (2015).
[26] Veado, L., Vale, G., Fernandes, E. and Figueiredo, E.: TDTool: Threshold Derivation Tool, *Proc. 20th International Conference on Evaluation and Assessment in Software Engineering, EASE '16*, pp.24:1–24:5, ACM (2016).
[27] Ostrand, T.J., Weyuker, E.J. and Bell, R.M.: Where the Bugs Are, *Proc. 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA '04*, pp.86–96, ACM (2004).
[28] Chidamber, S.R., Darcy, D.P. and Kemerer, C.F.: Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis, *IEEE Trans. Softw. Eng.*, Vol.24, No.8, pp.629–639 (1998).
[29] Lochmann, K.: A benchmarking-inspired approach to determine threshold values for metrics, *SIGSOFT Softw. Eng. Notes*, Vol.37, No.6, pp.1–8 (2012).
[30] Bouwers, E., Visser, J. and van Deursen, A.: Getting What You Measure, *Comm. ACM*, Vol.55, No.7, pp.54–59 (2012).
[31] Washizaki, H.: Chapter One - Pitfalls and Countermeasures in Software Quality Measurements and Evaluations, *Advances in Computers*, Vol.106, pp.1–22 (2017).
[32] Ribeiro, M.T., Singh, S. and Guestrin, C.: "Why Should I Trust You?": Explaining the Predictions of Any Classifier, *Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pp.1135–1144, ACM (2016).
[33] Quinlan, J.R.: *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993).
[34] Fontana, F.A., Mäntylä, M.V., Zanoni, M. and Marino, A.: Comparing and Experimenting Machine Learning Techniques for Code Smell Detection, *Empirical Softw. Eng.*, Vol.21, No.3, pp.1143–1191 (2016).
[35] Palomba, F., Nucci, D.D., Tufano, M., Bavota, G., Oliveto, R., Poshyvanyk, D. and De Lucia, A.: Landfill: An Open Dataset of Code Smells with Public Evaluation, *Proc. 12th Working Conference on Mining Software Repositories, MSR '15*, pp.482–485, IEEE Press (2015).
[36] Bosu, A. and Carver, J.C.: Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation, *Proc. 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14*, pp.33:1–33:10, ACM (2014).
[37] Tsuda, N., Washizaki, H., Fukazawa, Y., Yasuda, Y. and Sugimura, S.: Machine Learning to Evaluate Evolvability Defects: Code Metrics Thresholds for a Given Context, *2018 IEEE International Conference on Software Quality, Reliability and Security* (*QRS*), *QRS '18*, pp.83–94 (2018).
[38] Yamashita, K., Huang, C., Nagappan, M., Kamei, Y., Mockus, A., Hassan, A.E. and Ubayashi, N.: Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density, *2016 IEEE International Conference on Software Quality, Reliability and Security* (*QRS*), pp.191–201 (2016).
[39] Taube-Schock, C., Walker, R.J. and Witten, I.H.: Can We Avoid High Coupling?, *Proc. 25th European Conference on Object-oriented Programming, ECOOP'11*, pp.204–228, Springer-Verlag (2011).
[40] El Emam, K., Benlarbi, S., Goel, N., Melo, W., Lounis, H. and Rai, S.N.: The Optimal Class Size for Object-Oriented Software, *IEEE Trans. Softw. Eng.*, Vol.28, No.5, pp.494–509 (2002).
[41] Fenton, N.E. and Neil, M.: A Critique of Software Defect Prediction Models, *IEEE Trans. Softw. Eng.*, Vol.25, No.5, pp.675–689 (1999).
[42] Aniche, M., Treude, C., Zaidman, A., v. Deursen, A. and Gerosa, M.A.: SATT: Tailoring Code Metric Thresholds for Different Software Architectures, *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation* (*SCAM*), pp.41–50 (2016).
[43] Robinson, B. and Francis, P.: Improving Industrial Adoption of Software Engineering Research: A Comparison of Open and Closed Source Software, *Proc. 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '10*, pp.21:1–21:10, ACM (2010).
[44] Basili, V.R. and Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data, *IEEE Trans. Softw. Eng.*, Vol.10, No.6, pp.728–738 (1984).
[45] ISO/IEC: ISO/IEC 25010 – Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, Technical Report (2010).
[46] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley (1999).
[47] Mäntylä, M.V.: Empirical Software Evolvability – Code Smells and Human Evaluations, *Proc. 2010 IEEE International Conference on Software Maintenance, ICSM '10*, pp.1–6, IEEE Computer Society (2010).
[48] Washizaki, H., Namiki, R., Fukuoka, T., Harada, Y. and Watanabe, H.: A Framework for Measuring and Evaluating Program Source Code Quality, *Proc. 8th Int. Conf. PROFES, PROFES'07*, pp.284–299, Springer-Verlag (2007).
[49] Monden, A., Matsumura, T., Barker, M., Torii, K. and R., B.V.: Customizing GQM Models for Software Project Monitoring, *IEICE Trans. Information and Systems*, Vol.95, No.9, pp.2169–2182 (2012).
[50] McCabe, T.J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol.2, No.4, pp.308–320 (1976).
[51] Rahman, F., Posnett, D. and Devanbu, P.: Recalling the "Imprecision" of Cross-project Defect Prediction, *Proc. ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12*, pp.61:1–61:11, ACM (2012).
[52] 鷲崎弘宜, 波木理恵子, 福岡呂之, 原田陽子, 渡辺博之：プログラムソースコードのための実用的な品質評価枠組み, 情報処理学会論文誌, Vol.48, No.8, pp.2637–2650 (2007).

津田 直彦 （正会員）

2013 年早稲田大学基幹理工学部情報理工学科卒業．2014 年同大学大学院基幹理工学研究科修士課程修了．2017 年同大学院基幹理工学研究科博士課程単位取得後退学．2016 年より同大学助手．現在に至る．日本ソフトウェア科学会会員．

鷲崎 弘宜 （正会員）

早稲田大学グローバルソフトウェアエンジニアリング研究所所長，早稲田大学理工学術院基幹理工学部情報理工学科教授，国立情報学研究所客員教授，株式会社システム情報取締役（監査等委員），株式会社エクスモーション社外取締役，ガイオ・テクノロジー株式会社技術アドバイザ．文部科学省社会人教育事業 enPiT-Pro スマートエスイー事業責任者．

深澤 良彰 （正会員）

1976 年早稲田大学理工学部電気工学科卒業．1983 年同大学大学院博士課程修了．同年相模工業大学工学部情報工学科専任講師．1987 年早稲田大学理工学部助教授．1992 年同教授．工学博士．主として，ソフトウェア再利用技術を中心としたソフトウェア工学の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

保田 裕一朗

2010 年株式会社小松製作所入社．現在，開発本部 ICT 開発センタ在籍．

杉村 俊輔

1992 年株式会社小松製作所入社．現在，開発本部 ICT 開発センタ在籍．