チェックリストを用いたコードレビューと判別モデルの組合せによるモジュールの不具合リスクのランク付け



笠井　則充[†1]　　森崎　修司[†2]　　松本　健一[†3]

fault-prone モジュールのランク付け精度を向上させる手法を提案する．ランク付けには fault-prone 判別得点として，判別モデルにより得られる判別得点とコードレビューの評価得点を組合せた値を用いる．まず，判別性能が高いモデルを用いてコードレビューによる精度の向上を評価する．次に精度をできるだけ維持したまま工数を削減する 3 つの方法を提案する．削減方法は，1：構築コストの小さい判別モデルの利用，2：レビュー対象のモジュール数の削減，3：一部のレビュー観点の省略，である．これらの結果，従来の判別モデルにコードレビューによる評価得点を組合せることで fault-prone モジュールのランク付け精度が向上することが判った．また，レビュー工数の削減については，ランク付け精度を維持したまま 46% の工数を削減できることが判った．

An Approach for Prioritizing Fault-Prone Modules by Combining a Discriminative Model and Code Review with Checklist

Norimitsu Kasai[†1], Shuji Morisaki[†2], Kenichi Matsumoto[†3]

This paper proposes an approach for improving ranking accuracy of fault-prone module by combining discriminative model and code review. First, we evaluate ranking accuracy using support vector machine and random score. Next, we propose three ways to reduce the man-hours, I: using low cost discriminative model, II: reducing modules to review, III: using two aspects of the review to omit parts of review items. The result shows all accuracies are increased by combining code review in trials, and review effort is decreased by 46% without decreasing detection accuracy.

1. はじめに

顧客から受注したシステム開発のうち，ソフトウェア開発の一部を外部委託する業務形態において，委託先で開発された成果物に対して委託元がこれを検証するプロセスが重要視されている．委託元では V 字モデルに従い，受注したソフトウェアの要求仕様や外部装置とのインタフェース仕様を策定し，内部設計やコーディング，単体／組合せ試験を委託先に依頼することがある．委託先から成果物を受領後，システ

【脚注】
† 1　三菱電機株式会社 通信機製作所
† 2　名古屋大学
† 3　奈良先端科学技術大学院大学

ム統合試験として第三者検証を実施し出荷する．第三者検証中に不具合が多発した場合，限られた工期の中で可能な対策としては人員や時間を追加投入するクラッシング (crashing) しか無く，修正や再テストなど作業者の負担が増大する場合がある．これを避けるために委託元が成果物を受領した段階で設計検証を行い，不具合を可能な限り解消した上で第三者検証であるシステムテストを品質管理部門が実施するといったプロセス改善が進んでいる．

不具合を含んでいる可能性のあるソフトウェアモジュール（fault-prone モジュール）に対して，モジュールが不具合を含む度合いを得る方法として fault-prone 判別モデルが知られている．判別モデルでは行数やサイクロマチック数といったソースコードメトリクスや，内部仕様書のレビュー指摘件数のようなプロセスメトリクスといった特性値群を入力データとして用い，モジュールが不具合を含む度合いを出力する．これが基準値を超えるとそのモジュールが不具合を含むと推定する．

但し，「コメント行に記述された説明と実行ステートメントが一致していない」,「プラットフォーム依存の実装となっている」といった，潜在的な不具合の存在や兆候をそれら特性値群から捉えることは困難であり，ソースコードメトリクスやプロセスメトリクスから得た判別モデルだけで判別精度を高めることには限界がある．

このような，不具合の兆候となる情報はソースコードを直接確認するコードレビューによって検出することが可能である．しかし，コードレビューには多くの時間・工数が必要であり，実用規模のソフトウェアにおいて全モジュールに対してコードレビューを実施することは現実的ではない．著者らは文献 [kasai2012][kasai2013] において，コードレビューと判別モデルを組み合わせた場合の fault-prone モジュールのランク付け精度を学術面において議論してきた．本論文では，次の順序でコードレビューを組み合わせることの有効性を示し，開発現場での実践を前提としてこれにかかる工数の削減を試みる．

まず，従来の判別モデルにコードレビューによる評価を加えることで，判別モデル単独の場合よりも fault-prone モジュールのランク付け精度が向上するかをケーススタディによって示す．コードレビューの主目的は網羅的に欠陥を検出することではなく，ソースコードメトリクスやプロセスメトリクスからは得られにくい情報を補完的に得ることである．

レビューによりランク付け精度が向上すると判った場合は次の段階として，レビュー対象のモジュールを選択すること，レビュー手順の簡略化による評価工数の削減の効果，より低コストで導入が可能な判別モデルの導入を試行し，それぞれランク付け精度を評価する．

以降，2 章で判別モデルにコードレビューによる評価を加える方法とケーススタディにおけるランク付け精度について述べる． 3 章では工数の削減方法を述べ，ケーススタディによるランク付け精度と工数により評価する． 4 章で考察を行い，5 章でまとめる．

2. 判別モデルとレビューの組合せ

2.1. 概要

2.1.1. 判別モデル

fault-prone 判別モデルは，入力をモジュール m のメトリクス，出力を判別結果 $F(m)$ とする．判別結果 $F(m)$ は，fault-prone の度合いである．任意の二つのモジュール m_i と $m_j(i \neq j,i,j=1,2,\cdots)$ に対して $F(m_i)<F(m_j)$ が成り立つとき，m_i は m_j よりも fault-prone の度合いが大きいとする．

本論文では特定の fault-prone 判別モデルを前提としないが，モジュール $m_1,m_2,\cdots,m_n$ は判別結果によって並べられるものとする．

2.1.2. コードレビュー

コードレビューは，ソースコードの作成者やそれ以外の第三者が，作成したソースコードに目を通し確認する作業 [IPA2005] であるが，本論文でのコードレビューは，過去の類似システムで発生した不具合を元にしたチェックリストや評価者であるエンジニアが過去の類似案件で経験した情報等に基づいて評価を実施することを前提としている．

モジュール m の評価は，後述するチェックリストに含まれる質問項目の集合と各質問項目に対応した得点 $I(m)$ によって与えられる．質問項目や採点手順は，そのアプリケーションの背景などの豊富な経験と知識を有した開発者によってカスタマイズされ決められたものである．

チェックリストは l 個の質問項目 $q_i(i=1,2,\cdots,l)$ からなる．q_i により潜在的な欠陥の兆候となる可能性のある事項を検出する．各質問項目には採点手順が定められる．評価者は質問 q_i とその採点手順によりモジュール m に対して得点 $s_{im}(-1\leq s_{im}\leq 1)$ を与える．モジュール m に対するレビュー得点 $I(m)$ は，s_{im} の合計値を質問数 l で割った値とする．

質問項目 q_i は複数のモジュール間にまたがる評価は含まない．質問項目は，例外のハンドリング不足や，既存のソースコードのコメントの更新漏れといったコードスメル（不具合を含みそうなソースコードの兆候）[Fowler1999] によりモジュールを評価するものも含む．

2.1.3. 組合せ

本章による fault-prone 判別得点 $\hat{F}(m)$ は，判別モデルとレビュー評価による得点の合計に等しい．すなわち，$\hat{F}(m)=F(m)+\beta I(m)$.

ここで，βはレビュー評価点の係数である．βは，判別モデルの得点に対するコードレビューの評価点の重み付けであり，判別モデルによる得点とレビュー評価点のどちらに重みを置くか実績をみながら調整する．初めて提案手法を適用するときのように実績がない場合には，判別モデルの得点とレビュー評価点を同程度 (β=1) とする．

fault-prone 判別得点 $\hat{F}(m)$ によってモジュールを再度並び替える．

$$(m'_1,m'_2,\cdots,m'_n),\hat{F}(m'_1)\leq\hat{F}(m'_2)\leq\cdots\leq\hat{F}(m'_n).$$

並び替えた $m'_t(t=1,2,\cdots,n)$ の順序は不具合を含む可能性が大きいと判別したモジュールの順序である．

2.2. ケーススタディ

通信システムを監視，制御し，10 年以上に渡って保守，更新されてきたアプリケーションを用いてケーススタディを実施する．アプリケーションは C 言語で記述され，125 モジュールから成る．総行数は 15,100 行である．

ケーススタディ対象のプロジェクトは，ウォータフォールモデルによって実施された．コーディング，単体テスト，組合せテスト，及び，システムテストの一部のフェーズを外部委託している．ソースコードは受入れ検査前の最終版である．受入れ検査の間に 15 個のモジュールに欠陥が検出された．それぞれの欠陥は修正され，修正モジュールが保管されている．

このシステム開発に従事した熟練技術者がコードレビューのチェックリストを表 1 のように定義した．チェックリストは，委託先に提示した要求仕様書に記載している機能要求と非機能要求から抽出した．機能要求からはシーケンスタイミングやその条件等からクリティカルな部分を抽出してチェックリスト化した．非機能要求からは ISO9126 で定められた品質特性からこのシステムに必要とされるものを記載した項目に基づいてチェックリストに反映した．これに加えて，過去に実施したコードレビューで検出された不具合やレビュー時の指摘事項から得られた確認項目，過去に経験した不具合から類推される問題点等を抽出してリストへ反映した．

判別モデルとして，従来研究で判別精度が大きいと報告されているサポートベクタマシン (SVM) [Boser1992] を

表 1　チェックリスト

	分類	質問項目
q_1	実行環境依存の排除不足	カウンタを用いたループ処理による時間待ち.
q_2		OS が入れ替わったとき，使用できなくなる可能性のあるライブラリの使用.
q_3		変数を初期化せずに使用.
q_4		do ～ while 文で while 条件は有るが do ループ内が空.
q_5		for 文でループ内以外では使用していないローカル変数をカウンタに使用.
q_6	例外処理の対応不備	適切な箇所で構造化例外処理を記述していない.
q_7		catch で適切な例外を個別に処理していない.
q_8		finally がなく，default の処理が規定されていない.
q_9	デバッグ用メッセージ削除漏れ	デバッグ用 printf，外部出力命令の削除漏れがある.
q_{10}	バッファオーバフロー対策漏れ	get，gets，sprintf，strcat，strcpy，vsprintf を使用しているとき，バッファサイズの考慮が十分でない.
q_{11}	実装プロセス圧縮の悪影響	例外等による分岐によって，確保したメモリやリソースの解放処理が行われないことがないか.
q_{12}	可読性	分岐処理やループ処理，処理ブロックの先頭にその処理を説明するコメントがあるか.
q_{13}		switch 文で整理できる分岐を if 文で羅列していないか.
q_{14}		複雑な条件判断に下位関数の戻り値を使用しているか.
q_{15}		条件分岐後の複雑な処理を下位関数で定義しているか.
q_{16}	セルフチェック不足	コードを流用した場合，必要に応じてコメントも更新しているか.
q_{17}	実装作業時間不足の兆候	上位関数で実施すべき処理を下位関数で定義し，下位関数の引数を不必要に増やしていないか.
q_{18}	コーディングの適切さ	設計ロジックの複雑さに対してコードが必要以上に複雑になっていないか.
q_{19}		必要以上に if 文が連続し見通しが悪くなっていないか.
q_{20}	記述粒度の一貫性	プログラミングのルールを逸脱していないか.
q_{21}		記述粒度のばらつきが無いか.
q_{22}	環境依存の考慮不足	構造体をそのままファイル等に出力する場合，アライメントによる空のデータが混入しないか.
q_{23}		ログファイルの最大容量の考慮があるか.
q_{24}		ビッグエンディアン，スモールエンディアンの考慮漏れが無いか.
q_{25}		必要な部分で volatile の付加漏れが無いか.
q_{26}		必要でない割り込みの利用が無いか.
q_{27}		演算結果がオーバーフローする可能性が無いか.
q_{28}		規定 API/ インタフェースを使わず実装していないか.
q_{29}		ネットワーク通信やプロセス間通信用に規定の API を使っているか.

用いる．サポートベクタマシンは最も判別精度が大きくなる fault-prone 判別モデルの 1 つであると報告されている [Sebald2000]．他のモデルに比べて優れたパターン認識結果を得られることが知られており [Schölkopf1997]，局所解に陥ることがなく，特殊な場合を除いて解は一意に定まることが指摘されている [Burges1999]．また，ランダムな判別得点が得られるモデル (RND) を比較対象とする．評価指標は Alberg diagram [Ohlsson1996] の AUC(Area Under the Curve : 曲線下面積) を用いる．AUC は [0,1] の値をとり，fault-prone であると判別したモジュールに対して，実際に不具合が含まれているモジュールの割合が大きいほど大きな値となる．詳細は文献による．

モデル構築のためモジュール群を次のように 2 つのグループに分けた．モジュールを規模順に並べ，規模の小さい順に順位をつける．奇数の順位がついたモジュールをデータセット X とする．偶数の順位がついたモジュールをデータセット Y とする．

ケーススタディにおいて受入れ検査で検出された欠陥を含むモジュールの数は 15 モジュールであった．データセット X には 7 モジュールに，データセット Y には 8 モジュールにそれぞれ欠陥が有った．評価手順は以下の通り．

(1) チェックリストから質問項目と採点手順を決める．

(2) 以下のモジュールデータを用意する．

(a) 全てのモジュールのソースコードメトリクスを測定する．

(b) 以下のようにモジュールをデータセット X と Y に分割する．

$$X = \{M_1, M_3, \cdots, M_{2i+1}, \cdots, M_{125}\},$$

$$Y = \{M_2, M_4, \cdots, M_{2i}, \cdots, M_{124}\}.$$

データセット X は判別モデル (SVM) に与える学習データである．ここで，判別モデル (SVM) はデータセット Y のモジュールの fault-prone の度合いを判別する．逆もまた同様である．判別モデル (RND) は学習データを必要としない．

(3) 判別モデル (SVM)，(RND) の fault-prone の度合いに従ってモジュールを並べる．即ち，データセット X，Y について集合 X_{SVM}, X_{RND}, Y_{SVM}, Y_{RND} を得る．

$X_{SVM} = (m_{SVM1}, m_{SVM2}, \cdots, m_{SVM63})$ は判別モデル (SVM) によって並べられたデータセット X である．

(4) すべてのモジュールについてチェックリストによって採点を行い，本論文のアプローチによる fault-prone 判別得点 $\hat{F}(m)$ を得，この昇順にモジュールを並び替える．

(5) $\hat{F}(m)$ の昇順に並び替えたモジュールに対してそれぞれ AUC を算出する．

モデル (SVM) の構築には，説明変数は McCabe のサイクロマチック数，ループや分岐の最大ネスト数，関数呼び出し数，コメントを含まない行数とし，目的変数は出荷判定中に検出された不具合の有無とした．モデル (SVM) の fault-prone の度合いは，統計解析ソフトウェア R を用いて構築したガウシアンカーネルによる非線形サポートベクタマシンを用いて算出した．モデル (RND) の fault-prone の度合いは，R によりメルセンヌツイスタ乱数を生成させて得た．

2.3. ケーススタディの結果

表 2 にそれぞれの判別モデルにおける最大の AUC とそのときの β の値 (複数存在する場合は最小値) を示す．表中，“SVM(レビュー無)”，“RND(レビュー無)” は判別モデルのみによって得られた AUC である．コードレビューを組み合わせることによって得られた $\hat{F}(m)$ による AUC が，レビューを行わない場合と比較して有意に大きいかを有意水準 5% の片側検定による 2 標本 t 検定によって実施した．いずれの場合においても判別モデルとレビューを組合せたときの AUC がレビューを行わないときの AUC よりも有意に大きい結果であった．

表 2　最大の AUC と β の値

判別モデル	データセット	AUC	β
SVM	X	0.831	1.6
	Y	0.854	1.4
RND	X	0.755	4.7
	Y	0.840	3.6
SVM(レビュー無)	X	0.774	—
	Y	0.773	—
RND(レビュー無)	X	0.512	—
	Y	0.515	—

適切なパラメータ β を与えることによって判別モデルのみを用いた場合と比較して，(SVM) のとき，データセット X では AUC が 0.057，データセット Y では 0.081 向上すること，(RND) のとき，データセット X では AUC が 0.243，データセット Y では 0.325 向上することが判った．ケーススタディでは最大の AUC が得られる β を求めた．β を 1 とした場合の AUC は (SVM) で 0.824 と 0.848, (RND) で 0.664 と 0.679 と小さくなるが，判別モデルのみを用いた場合の AUC と比較して大きな値であった．

3.　工数削減

3.1. 概要

本章では，予測精度をなるべく維持しつつ，前章で実施したランク付けに必要なコストを低減できるかを試みる．具体的には，コードレビューの対象となるモジュール数を減らし，モジュールの規模に応じてコードレビューのチェックリストの項目数を減らす．加えて，判別モデルの判別得点の代わりに単一のソースコードメトリクスを用いることで，モデル構築コストを低減する．

コードレビューの方法 I（対象モジュールの削減）：全モジュールをレビュー対象とするのではなく，レビューの効果がより大きいと思われる一部のモジュールを対象とする．2.1.3 節で述べた fault-prone の度合い $F(m)$ を，レビューすべきモジュールを選択する基準として用いる．レビューを行うモジュールの割合を α% と表記する．n 個のモジュールに対して α% のモジュールにレビューを行うとき，対象のモジュールの個数 k は $(\alpha n/100)$ の床関数とする．また，fault-prone 判別得点 $\hat{F}(m)$ を拡張し，レビューを行わないモジュールに対しては評価得点 $I(m)$ を 0 とする．α も β と同様にチェックリストと判別モデルの組合せの実績を見ながら決める．初めて提案手法を適用するときのように実績がない場合には，α の増加に従って提案手法の fault-prone モジュールのランク付け精度が増加するという前提を置き，コードレビューに割り当てることができる工数によって予測時に決めることを想定している．

コードレビューの方法 II（レビュー観点の省略）：レビュー観点の省略により個々のモジュールに割り当てるレビュー工数を削減する．前章で示したコードレビューでは，予め用意した質問項目全てを評価することを前提としているが，モジュールによっては質問項目の一部を評価するだけで十分な場合もある．そこで，モジュール規模に応じた観点を設定し質問項目を観点で分類する．モジュール規模によって省略する観点（質問項目）を決める．たとえば，コード全体を一目で見渡すことができるような小規模モジュールには該当しない観点を省略する．

判別モデル：既存の多くの fault-prone モジュール判別モデルでは，モデル構築のコストに加え，モデルを扱うための学習，教育，環境構築のコストが必要となる．特にモデル構築には過去の不具合がどのモジュールから検出されたかを収集しておく必要がある．そこで，多くの fault-prone 判別モデルの入力パラメータとなっているソースコードメトリクスを用いることによりモデル構築コストを低減する．

3.2. コードレビューの方法

3.2.1. 対象モジュールの削減 (方法 I)

図 1 に判別モデルの結果を用いたコードレビュー対象モジュールの選択方法を示す．モジュールの選択方法は明らかではないため，4 通りを考え，ランク付け精度と必要な工数を比較する．選択方法 1 においては，評価者は fault-prone の度合い $F(m)$ が大きいモジュールから順にレビュー対象のモジュールをその割合が α% になるまで選択する．選択方法 2 においては，評価者は $F(m)$ が小さいモジュールから順にレビュー対象のモジュールをその割合が α% になるまで選択する．選択方法 3 においては，評価者は $F(m)$ の中央値のモジュールから順にレビュー対象のモジュールをその割合が α% になるまで選択する．選択方法 4 においては，評価者は $F(m)$ が最大のモジュール，最小のモジュール，

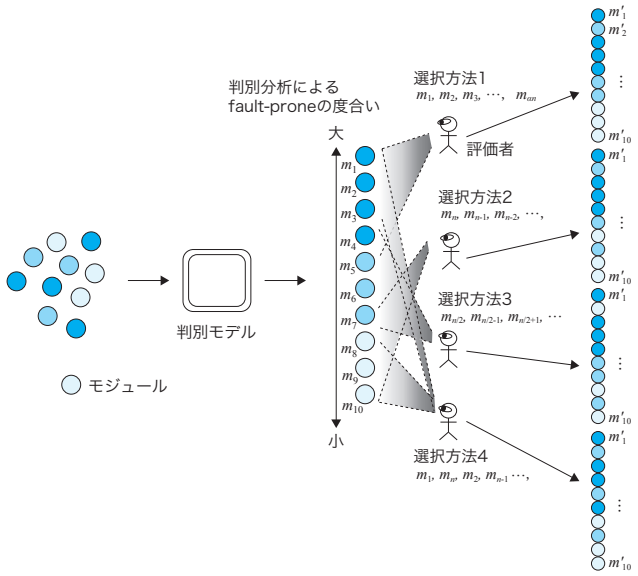

図 1　方法 I によるレビュー対象の選択方法

最大から 2 番目のモジュール，最小から 2 番目のモジュール，とモジュールの割合が α% になるまでのモジュールをレビュー対象として選択する．

選択方法 1 と 2 は，それぞれコードレビューが $F(m)$ の偽陽性，偽陰性を補完することを期待している．選択方法 3 と 4 は $F(m)$ がとりたてて注目されていないモジュール，及びその逆のものに着目してコードレビューの効果を期待している．

3.2.2. レビュー観点の省略 (方法 II)

コードレビューに用いる質問項目は 2 種類の観点から設定する．すなわち，観点 (1) 局所的な目視で不備や不具合の存在自体を即座に発見できる質問項目，観点 (2) モジュール全体にわたって整合性等を確認し，潜在的不具合を予測しようとする質問項目，である．観点 (1) で直接的な不具合を予測すること，観点 (2) で問題の兆候を予測することを想定している．観点 (2) による評価は規模の大きなモジュールでのみ実施する．観点 (2) の質問項目はモジュール内の一貫性の不備に起因する不具合の兆候を検出しようとしており，規模の小さいモジュールではそのような問題が起きにくいからである．観点 (1) の例を表 3 に，観点 (2) の例を表 4 に，それぞれ示す．

レビューの具体的な手順を図 2 に示す．この手順は 1 つのモジュールに対するものであり，レビュー対象のモジュールの数だけ繰り返す．

まず，観点 (1) によるレビューを実施する．問題があると判断されれば，そのモジュールには不具合や問題が含まれることを意味するので，他の質問項目による評価を止め，評価点を -1 とし，そのモジュールの評価を終了してモジュール内の他の不具合の可能性を考慮した施策対象とする．観点 (1) による評価において問題がないと判断されたモジュールのうち，規模の小さいモジュールは評価点 1 を付与し，

表 3　方法 II によるレビュー観点 (1) の例

分類	質問項目
環境依存の考慮不足	プロセッサ処理速度に依存した時間待ち処理 (ハードウェアのリプレース時に本来の時間よりも短い時間で処理が終わる).
例外処理対応不備	例外の発生する可能性のある箇所で例外処理が未定義.
デバッグ用メッセージ削除漏れ	printf, beep 等，実行通知のための外部出力命令の削除漏れ.
バッファオーバーフロー対策漏れ	バッファオーバーフローが発生する可能性の大きい関数を使用しているとき，バッファのサイズをチェックする等の対策をしていない.

表 4　方法 II によるレビュー観点 (2) の例

分類	質問項目
セルフチェック不足	コメント不備 / 更新忘れがないか.
	コーディングルール非準拠箇所がないか.
例外処理の検討不足	例外処理の一貫性（同一の例外に対して同一の例外処理）を維持しているか.
考慮不足	異常終了時等のログ出力情報は一貫しているか. 外部からの入力に一貫したサニタイズを実施しているか.

そのモジュールの評価を終了する.

観点 (2) は含まれる全ての質問項目について評価を実施し，モジュール毎に “ 問題なし ”，“ 問題あり ”，“ 該当なし ” を判定し，それらの数によって評価点を決定する．問題ありと問題なしに該当する質問項目数を比較し，数が多い方を結果とする．表 5 に評価点とその付与条件を示す．なお，方法 II ではモジュールによってレビューを途中で終了する場合があり判別得点とレビュー評価点を合算しないため，β は使用しない.

3.3. ケーススタディ

前節のコードレビューの方法 I (対象モジュールの削減)，方法 II (レビュー観点の省略) によるレビュー工数の削減，判別モデルの判別得点に代わり単一のソースコードメトリクスを用いた方法の有効性を確認する．対象とするアプリケーションは 2 章と同じものを用い，使用した判別モデル (SVM と RND) やチェックリストは 2.2 節及び 3.1 節と同じとする.

方法 I (対象モジュールの削減) において，レビュー対象とするモジュールの選択方法は 3.2 節の 4 種類の方法とする．方法 II (レビュー観点の省略) において，観点 (2) の対象は 70 行以上のモジュールとした．70 行は，対象ソフトウェアの開発においてディスプレイにソースコードを表示したとき，スクロール無しで見通すことができる行数を想定しており，これを超えるものに一貫性の不備が混入しやすいと考えた.

表 5　方法 II による観点 (2) の評価点と付与条件

評価点	付与条件
1	“ 問題あり ” が無く，“ 問題なし ” がある.
0	(i)“ 問題あり ” は有るが，“ 問題なし ” の方が多い. (ii)“ 問題あり ” も “ 問題なし ” も無い.
-1	“ 問題あり ” の方が “ 問題なし ” よりも多い.

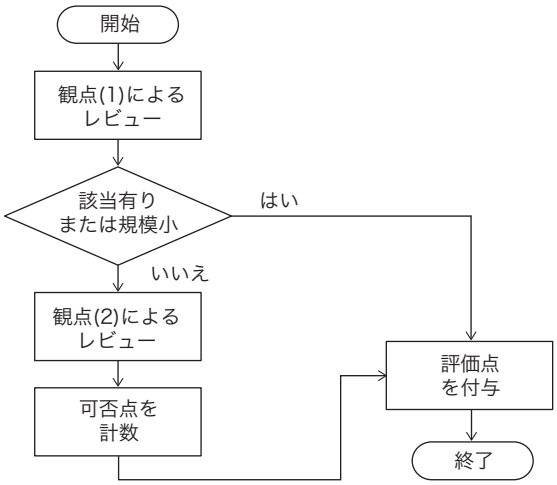

図 2　方法 II によるレビューの手順

判別モデルの判別得点の代わりにソースコード行数 (LOC) を選んだ．モジュール $m_i(i=1,2,\cdots)$ に対する $F(m_i)$ を LOC に (-1) を乗じた値とした．LOC を選んだ理由は，多くの fault-prone 判別モデルの入力パラメータであること，先行研究により LOC と複雑度メトリクスとの相関が大きいこと [Basili1984]，複雑度メトリクスが大きいと不具合が含まれやすいこと [Ohlsson1996] ，LOC と不具合に相関があること [Selby1991] が報告されているからである.

3.4. ケーススタディの結果

表 6 に最大の AUC 及び α と β の値 (複数存在する場合は最小の β の値とそのときの α の値を示す) を示す．表において，方法 I の列はレビュー対象とするモジュールの選択条件を 4 通り試行したときの AUC が最大となるときの値とパラメータを示し，方法 II の列は，レビューを観点 (1) と観点 (2) に基づいて実施した結果である．すべての選択方法とモデルにおいて，方法 I の AUC の値のほうが方法 II の AUC の値よりも大きくなった．RND を除くと，差の最大は 0.059 であった.

方法 I (対象モジュールの削減) において，レビューを行うモジュールの割合 (α%) は 1-SVM のデータセット X を除いていずれも 70% 以上である．3 つの判別モデルとレビュー対象モジュールの選択方法 1 ～ 4 において AUC が最大となるのは，判別モデルが LOC で選択方法 4 のとき 0.871，このとき α は 70% である．次に AUC が大きい組合せは，判別モデルが SVM で選択方法 1 のとき 0.860，このとき α は 40% である.

表 6　最大の AUC とパラメータ

選択方法とモデル	データセット	方法 I			方法 II	
		AUC	α	β	AUC	α
1-SVM	X	0.860	40	1.6	0.840	65
	Y	0.854	85	1.4	0.850	65
1-LOC	X	0.826	100	5.9	0.814	75
	Y	0.788	100	5.2	0.771	95
1-RND	X	0.852	95	4.7	0.740	95
	Y	0.840	100	3.6	0.817	100
2-SVM	X	0.831	100	1.6	0.819	100
	Y	0.863	95	2.2	0.854	95
2-LOC	X	0.833	95	4.2	0.812	95
	Y	0.788	100	5.2	0.771	100
2-RND	X	0.755	100	4.7	0.748	95
	Y	0.846	95	5.7	0.819	95
3-SVM	X	0.831	100	1.6	0.831	75
	Y	0.858	90	1.5	0.854	90
3-LOC	X	0.826	100	5.9	0.812	90
	Y	0.788	100	5.2	0.771	95
3-RND	X	0.845	90	3.0	0.760	95
	Y	0.840	100	3.6	0.817	100
4-SVM	X	0.852	85	1.6	0.819	100
	Y	0.854	100	1.4	0.838	100
4-LOC	X	0.871	70	1.8	0.812	100
	Y	0.800	75	3.8	0.771	100
4-RND	X	0.755	100	4.7	0.738	100
	Y	0.840	100	3.6	0.817	100

方法 II (レビュー観点の省略) において，規模の小さなモジュールに対する評価工数を削減させることにより評価に必要なコストを低減させた．一般には，α を大きくすると，より大きな AUC が得られることが期待されるが，その分，評価に必要なコストが大きくなる．ケーススタディでは，α=75% のとき，データセット X では AUC が 0.840，データセット Y では 0.841，α=100% のときそれぞれ 0.819 と 0.838 であり，fault-prone モジュールのランク付け精度の差はほとんどなかった．α を 75% とすることにより，約 31 モジュールのレビューを省くことができる．方法 II (レビュー観点の省略) では，評価に要するコストは観点 (1) による評価で 1 モジュールあたり 0.16[人時] 程度，観点 (2) で 0.33[人時] 程度であった．いずれの判別モデルにおいても α を 75% とするのに必要な評価コストはほぼ 8[人時] でよく，現場への適用には無理が無いと考えられる．また，方法 I (対象モジュールの削減) のうち，α が 40% と小さな値となった選択方法 1 によるレビューコストと，方法 II (レビュー観点の省略) の各判別モデルの場合を比較して α が 25%,50%,75%,100% のそれぞれの値 (人時) を表 7 に示す．いずれにおいても評価コストは方法 II のほうが方法 I よりも小さくなり，α が 100% の場合で評価コストは約 46% 削減した．

表 7　各判別モデルにおける目視コスト［人時］

α	データセット	方法 I	方法 II		
			SVM	LOC	RND
25	X	5.12	3.59	3.76	2.74
	Y	5.12	3.59	3.59	2.57
50	X	10.23	5.99	6.16	5.31
	Y	10.23	5.99	5.99	5.48
75	X	15.35	8.56	8.56	7.88
	Y	15.35	8.39	8.39	8.05
100	X	20.46	10.96	10.96	10.96
	Y	20.46	10.96	10.96	10.96

表 8　判別モデルの比較

項目	SVM	LOC	RND
予測精度	大	中	小
モデル構築作業と必要な知識	要	不要	不要
学習データ（過去の蓄積情報）	要	不要	不要
導入容易性	やや難	易	易

4.　考察

判別モデルとレビューの組合せに関して考察する．今回のケーススタディでは表 1 のチェックリストの質問項目には，偽陽性がなかったものが含まれていた．質問項目 q_6 と q_{16} は真陽性が 3 件，偽陽性は 0 件であったことから，コードレビュー結果は判別モデルを補完し得ることが解る．

ケーススタディでは，SVM と LOC をランダムな順序と比較して評価した．比較を表 8 に示す．SVM は他のモデルに比べて優れたパターン認識結果を得られることが知られている．モデルに与えるメトリクスなど，過去のデータを持っている場合や，判別分析が実施可能な環境にあれば，SVM によるモデルを構築することで高い精度での fault-prone モジュールのランク付けが可能となる．一方で類似ソフトウェアによる蓄積情報などが必要だったり判別モデルの構築作業にある程度の知識が必要であったりするため，現場への導入が容易とは言えない．

工数削減に関して考察する．コードレビューの方法 I (対象モジュールの削減) において，コードレビューを行うモジュールの割合が大きくなると概ね AUC が増大する結果となった．しかし，AUC は α の増加に対しすべての場合において単調増加するとは言えず，方法 I によってレビュー工数が削減できるとは必ずしも言えない．原因はレビューの質問項目に含まれる偽陽性であり，こうした質問項目を減らすことが今後の課題である．α の増加に対して AUC が単調増加すれば，コードレビューに投入できるコストに応じて α を決めることができる．また，コードレビュー対象モジュールを選択方法 1 によって選んだときに α の値が最小となった．レビューの質問項目は，判別モデルに与える

ソースコードメトリクスでは捉えることが難しいものを前提とし，ケーススタディで用いたソフトウェアに求められる非機能要件に合わせてテーラリングしたものである．普遍的なものではないので対象とするソフトウェアにあわせてテーラリングする必要がある．

更なる工数削減に向けてレビューの質問項目の自動化の可能性を考察する．ケーススタディで用いた観点 (1) に含まれる質問項目には，OS 依存のライブラリの存在，do ～ while 文や for 文で想定される問題のように検出を自動化できるものや，変数の初期化漏れ，例外処理中の finally や default 処理の不足，デバッグ用の printf 文の存在，get や strcpy などのようなバッファサイズの考慮が必要な関数の使用といった，問題点の候補を自動的に列挙し，目視で確認できるものがある．これらの観点について，ケーススタディでは，自動化のための準備のコストのほうが自動化によって省略できるコストよりも大きかったため，コードレビューの質問項目としたが，将来のバージョンでの再利用等を加味すれば，自動化によるメリットが得られるものもある．そこで，レビューではなく判別モデルの説明変数に加えることができる 3 つの質問項目をレビューにおける質問項目から外し，3 つの質問項目を SVM のモデル構築に用いて試行した．αの値を 25%,50%,75%,100% と変化させて 3 つの質問項目をレビューにおいて実施した場合と比較したところ，AUC が同等か若干小さくなった．判別モデルを LOC とした場合においても同様に試行したところ，LOC を判別モデルとした場合よりも AUC が大きくなった．

本論文による提案手法が有効に使用できた場合，受入れ検査における不具合の早期発見によって以下のメリットが生じる．検出された不具合が複数ある場合にその回帰テストを 1 回にまとめることができる．あるモジュールで検出した不具合箇所の修正作業と，残りの部分のテストを並行して実施できるため，手戻りを回避しながらテスト期間を短縮することができる．特に，受入れ検査終盤において，納期に間に合わせることを優先するための場当たり的な修正を防げるため，出荷後の保守や次バージョン以降の拡張において，改変コストを小さくできる点においてもメリットがある．

5. まとめ

本論文では，従来の判別モデルによる fault-prone モジュール判別にコードレビューを加えることで fault-prone モジュールのランク付け精度が向上するかを，ケーススタディによって確認した．また，ランク付け精度を維持したままレビュー工数を削減する方法についてケーススタディによって確認した．

まず，従来の判別モデルの得点とレビューによる評価得点の和を fault-prone 判別得点とし，その判別得点によりモジュールをランク付けした．レビューによる評価得点に適切な荷重βを乗じて加えることで判別モデルの得点のみによるランク付けよりも精度の高いランク付けができることをケーススタディを通じて確かめた．

判別モデル構築の必要がないモデル（モジュールのコード行数に (-1) を乗じた値を与えるモデル）を評価対象に加えて，次の方法でランク付け精度を維持しつつレビュー工数を削減できるか確かめた．方法 I（対象モジュールの削減）としてレビュー対象を全モジュールのα% として評価した．レビューするモジュールの選び方は判別得点の昇順，降順といった 4 種類の方法を比較した．適切なα，βを与えることにより，判別モデル単体よりも高精度な fault-prone モジュールのランク付けができることを確認した．サポートベクタマシンでの精度は，殆どの場合で大きくなり，αが 40% のとき最も大きくなった．

方法 II（レビュー観点の省略）として，規模の小さいモジュールにおいてレビューによる評価の一部を省略する方法をケーススタディにより試行した．ケーススタディの結果，方法 I と同程度のランク付け精度を保ちながら，方法 I よりも小さいレビュー工数で方法 II を実施できることが判った．αが 100% の場合，方法 II が方法 I よりも 46% の工数を削減できた．

【参考文献】

[Basili1984] V. R. Basili and B. T. Perricone, Software errors and complexity: An empirical investigation. Communications of the ACM, Vol.27, No.1, pp.42–52 (1984)

[Boser1992] B. E. Boser, I. M. Guyon, and V. N. Vapnik, A training algorithm for optimal margin classifiers, In Proceedings of the 5th annual ACM workshop on computational learning theory, pp.144–152 (1992)

[Burges1999] C. J. C. Burges and D. J. Crisp. Uniqueness of the svm solution. In NIPS, pp.223-229 (1999)

[Fowler1999] M. Fowler. Refactoring: Improving the design of existing code. Addison-Wesley (1999)

[IPA2005] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター編．組込みソフトウェア開発における品質向上の勧め (コーディング編)，翔泳社 (2005)

[Kasai2012] 笠井則亮，森崎修司，松本健一．目視評価と判別モデルを組み合わせた fault-prone モジュールのランク付け手法．情報処理学会論文誌 , Vol.53, No.9, pp.2279-2290 (2012)

[Kasai2013] N. Kasai, S. Morisaki, K. Matsumoto, Fault-prone module prediction using a prediction model and manual inspection, In Proceedings of the 20th Asia-pacific software engineering conference, pp.106-115 (2013)

[Ohlsson1996] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. IEEE Transactions on Software Engineering, Vol.22, No.12, pp.886-894 (1996)

[Schölkopf1997] B. Schölkopf, K. Sung, C.J.C. Burges, F. Girosi,P. Niyogi, T. Poggio, V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. IEEE Transactions on Signal Processing, Vol.45, No.11, pp.2758-2765, nov (1997)

[Sebald2000] D. J. Sebald and J. A. Bucklew, Support vector machine techniques for nonlinear equalization. IEEE Transactions on Signal Processing, Vol.48, No.11, pp.3217-3226 (2000)

[Selby1991] R. W. Selby and V. R. Basili. Analyzing error-prone system structure. IEEE Transactions on Software Engineering, Vol.17, No.2, pp.141-152 (1991)