情報処理学会論文誌 Vol.53 No.9 2279–2290 (Sep. 2012)

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

笠井 則充[1,2,a)] 森崎 修司[3] 松本 健一[2]

受付日 2011年12月22日, 採録日 2012年6月1日

概要：不具合を含む可能性の高いモジュールから順に受入れ検査を実施することを目的とし，fault-prone 判別モデルの判別得点とソースコードの目視評価とを組み合わせたモジュールのランク付け手法を提案する．提案手法では，判別モデルから得た判別得点によりモジュールをランク付けし，上位 α%のモジュールを目視評価の結果により再度ランク付けする．判別モデルと目視評価の組み合わせによるランク付けの精度を評価することを目的として，商用ソフトウェアを対象として，判別モデルの判別得点順，モジュールの規模順，ランダム順と目視評価を組み合わせた場合のランク付けの精度を α の値を変化させて評価した．ランク付けの精度は AUC（Area Under the Curve: Alberg diagram の曲線下面積）により比較した．いずれの組み合わせにおいても α の値を大きくすることにより相対的に AUC が大きくなるという結果が得られ，判別モデルとの組み合わせにおいて最も大きな AUC となった．

キーワード：fault-prone モジュール，判別モデル，サポートベクタマシン，目視評価

An Approach for Prioritizing Fault-prone Modules by Combining Manual Inspection and Discriminant Model

Norimitsu Kasai[1,2,a)] Shuji Morisaki[3] Ken-ichi Matsumoto[2]

Received: December 22, 2011, Accepted: June 1, 2012

Abstract: This paper proposes an approach for prioritizing modules by their fault-proneness for acceptance test. The approach combines results of a fault-prone discriminant model and inspection by developers. First, the approach ranks modules by fault-proneness according to a discriminant model. Then, modules included in the top α% are inspected with questions. To evaluate the effectiveness of the approach, we conducted a case study with commercial software. In the case study, manual inspection is combined with fault-proneness by a discriminant model, size of each module and random. The accuracies are compared by the values of area under the curve of Alberg diagram with five α values. The result shows all accuracies are increased by combining inspection in three trials. In the case study, the accuracy is largest by the combination of fault-prone discriminant model and manual inspection.

Keywords: fault-prone, discriminant model, support vector machine, manual inspection

1. はじめに

近年ソフトウェアの複雑化，短納期化が進んでおり，生産性を維持しつつソフトウェア品質を確保する目的で外部の組織に開発業務委託（外部委託）を行う傾向がある [1]．また，文献 [2] によると，調査対象のプロジェクトの 7 割以上において，開発工数の半分以上が外部委託先の工数であること，特に規模が大きいプロジェクトで外部委託工数

1 三菱電機株式会社通信機製作所
Mitsubishi Electric Co. Communication Systems Center, Amagasaki, Hyogo 661–8661, Japan
2 奈良先端科学技術大学院大学
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
3 静岡大学
Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
a) Kasai.Norimitsu@bp.MitsubishiElectric.co.jp

の比率が大きいこと，が報告されている．

他方，ソフトウェアの外部委託によって品質が低下するという報告があり [3]，外部委託を行ったうえで納期と品質の両方を確保することが重要である．しかし現実には担当者が複数のプロジェクトの管理に並行して従事し，納期を確保しつつ成果物の品質を確保することが必要なため，実現は容易ではない．外部委託により開発されるソフトウェアの品質を低下させずに生産性を維持するには，委託先とリスクや課題を共有し，課題の早期発見と解決に努めることが必要である．

多くのウォータフォール型の委託開発では，委託者は要件定義，設計の一部を実施する．受託者は以降の設計，コーディング，単体テスト，結合テスト，システムテストの一部を実施する．委託者は受託者がテストを完了したソフトウェアを検収し，問題がなければ委託部分の作業は完了となる．検収の完了は受入れ検査と呼ぶ，委託者が実施するテストによって判断される．

受入れ検査時に不具合を多く含む機能やモジュールから順にテストを実施できれば品質向上，予定期間内での受入れ検査完了，不具合の修正工数削減につながる．受入れ検査はソフトウェア開発プロジェクトの終盤で実施される．委託者が受入れ検査で発見した不具合は受託者が修正し，不具合の修正と修正確認が完了するまで受入れ検査は完了しない．受入れ検査の終盤で不具合が発見されれば，修正と修正確認に時間を要し予定期間内で受入れ検査が完了しなかったり，期間内で完了するために場当たり的な修正方法しかとれなかったりする．また，不具合をまとめて発見できれば，回帰テストの実施工数が小さくなる場合があったり包括的な修正方法を採用したりすることによって，不具合の修正工数削減につながる可能性が高まる．

受入れ検査時にモジュールの検査実施順序を決定する方法の 1 つとして fault-prone モジュール判別手法の判別結果を用いることができる．これまで提案されている fault-prone モジュール判別手法 [4], [5], [6], [7], [8] は，判別コストを小さくするために主に計測を自動化できるソースコードメトリクスを入力とし判別結果を得ている．分岐が多数ネストしており複雑，規模が大きい，といった特徴が判別の根拠となる．一方，複数回のバージョンアップによりソースコード中のコメントの説明とステートメントが一致しておらず勘違いを起こしやすい，環境依存の実装方法となっている，といった構文解析だけでは得られないが有力な手がかりとなる兆候を判別結果に加味することが難しい．これらを加味するためには，すべてのモジュールの目視が必要になる．

本論文では，受入れ検査において不具合を含む可能性の高いモジュールからテストを実施することを目的として，fault-prone 判別モデルの判別得点とソースコードの目視の結果を組み合わせた fault-prone モジュールのランク付け手法を提案する．提案手法では，目視の対象をなるべく少なくすることにより目視コストを低減させることを目指す．具体的には，fault-prone 判別モデルの判別得点を用いてモジュールを順位付けし，上位のモジュールから順に目視評価を実施し，モジュールをランク付けする．

以降，2 章で提案手法について述べ，3 章で商用ソフトウェアを対象とした評価試行について述べる．4 章で考察を行い，5 章で関連研究に言及し，6 章でまとめる．

2. 提案手法

2.1 概要

提案手法は判別モデルにより得た判別得点と目視により得た評価点の 2 つを組み合わせて，モジュールを fault を含む可能性の大きい順にランク付けする．目視評価には観点（質問項目）を複数設定し観点ごとに評価する．判別モデルで fault-prone モジュールの可能性が大きい（判別得点が大きい）と判断された上位のモジュールから優先的に目視評価することにより，目視評価にかかるコストを抑えたうえでランク付けの精度を向上させる．

提案手法の概要を**図 1** に示す．図 1 中の左側が，対象のモジュール集合 M^t を表している（図 1 中 (1)）．判別モデルによる判別得点（fault-prone モジュールの可能性の大きさ）を $f_m(m_i)$ とし集合 M^t の要素を $(m_1^t, m_2^t, \cdots, m_{n_t}^t)$ の順序で並べる（図 1 中 (2)）．ここで，$f_m(m_i) \geq f_m(m_{i+1})$，$f_m(m_i)$ の値が大きいほど fault-prone モジュールの可能性が大きいと判別モデルにより判別されたものとする．

次に $(m_1^t, m_2^t, \cdots, m_{n_t}^t)$ の上位 α%のモジュール $(m_1^t, m_2^t, \cdots, m_k^t)$ を対象として目視評価を実施し，評価点 $f_v(m_i)$ を得て，モジュールを $(m_1'^t, m_2'^t, \cdots, m_k'^t, m_{k+1}^t, \cdots, m_{n_t}^t)$ に並べ替える（図 1 中 (3)）．ここで，$1 \leq k \leq \alpha/100 \times n_t$，提案手法において $0 < \alpha \leq 100$ である．k は式を満たす最大の整数とする．なお，$\alpha = 0$ の表記を目視評価を行わない意味に使用する．$f_v(m_i)$ の値が大きいほど目視において fault-prone モジュールの可能性が大きいと判断されたことを示し，$f_v(m_i') \geq f_v(m_{i+1}')$ である．

$f_v(m_i')$ が同じ得点のモジュールを $f_m(m_i')$ の値により再度並べ替え，$(m_1''^t, m_2''^t, \cdots, m_k''^t, m_{k+1}^t, \cdots, m_{n_t}^t)$ を得る（図 1 中 (4)）．目視評価点，判別得点ともに等しい場合にはモジュール名のアルファベット順とする．

得られた $(m_1''^t, m_2''^t, \cdots, m_k''^t, m_{k+1}^t, \cdots, m_{n_t}^t)$ の $m_1''^t$ から受入れ検査において優先的にテストを実施する．

なお，提案手法では，何らかの方法で受入れ検査のテスト項目とテスト対象のモジュールの対応づけができていることを前提とし，その対応からテスト項目を選ぶ．対応づけの方法に特に前提はおかないが，たとえば，要求，設計のフェーズでテスト計画の一環として作成される要求と設計，要求とテスト項目のトレーサビリティマトリクスを利用したり，詳細設計のフェーズで作成される機能名をモ

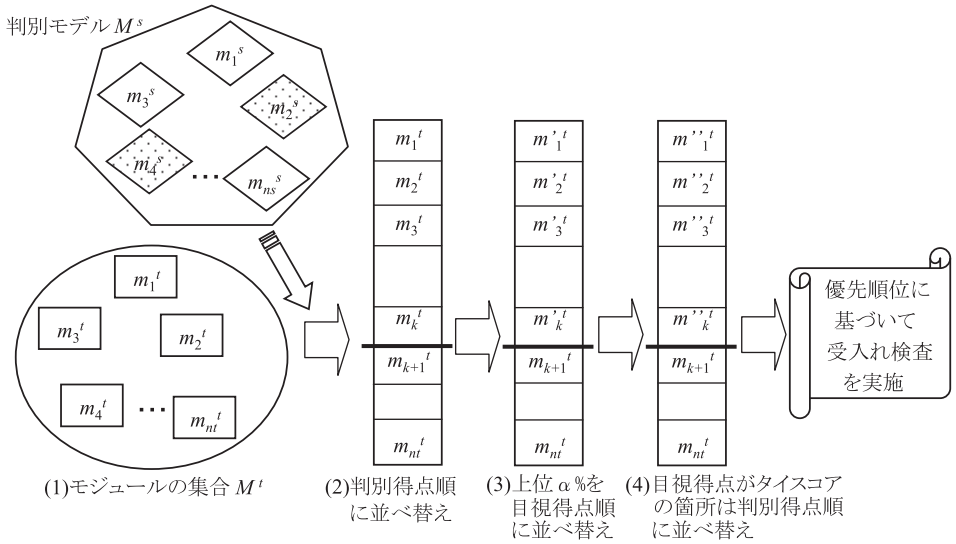

図 1 目視評価との組み合わせによる判別分析手順
Fig. 1 The procedure of discriminant analysis combining visual verification scoring.

ジュールの識別子に含めておき，その機能をテスト対象とするテスト項目を選択したりすることを想定している．

提案手法では特定の判別モデルを前提とはしないが，判別得点を用いて fault-prone モジュールの可能性が大きい順にモジュールを並べ替えることができるものを想定している．

2.2 判別モデルによるモジュールのランク付け

対象モジュールの過去のバージョンや類似のモジュールを用いて，判別モデルを構築する．fault-prone の可能性の大きい順にランク付けを行うモジュールの集合を M^t，判別モデル構築のためのモジュールの集合を M^s とし，それぞれ，$M^t = \{m_1^t, m_2^t, \cdots, m_{n_t}^t\}$，$M^s = \{m_1^s, m_2^s, \cdots, m_{n_s}^s\}$ とする．

M^s とその fault 有無の情報から判別モデル f_m を得る．対象とするモジュールの集合 M^t の特徴量を判別モデル f_m に与えることで m_i^t の判別得点 $f_m(m_i^t)$（$i = 1, 2, \cdots, n_t$）を得る．

なお，本章での説明は，モジュールの特徴量として代表的なソースコードメトリクスを用いて判別モデルを構築しているが，提案手法では，モジュールの特徴量として必ずしもこれに限定するものではなく，たとえば文献 [9], [10], [11], [12] のように派生開発に見られるソースコードの改変量や欠陥履歴，構成管理情報等を特徴量としてもよい．

2.3 目視によるモジュールのランク付け

目視評価は判別モデルと同様にモジュール単位で実施する．目視評価は事前に設定した質問項目により行う．質問項目は 2 種類の観点から設定し，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から不具合の有無を予測しようとしており，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)，(2) いずれの質問項目においても対象ソフトウェアや類似ソフトウェアにおいて起こりうる問題を予測し設定する．観点 (1) は過去の不具合情報やこれらの蓄積によって組織内で作成したチェックリストをもとに設定する．観点 (2) は ODC（Orthogonal Defect Classification）[13] を用いた欠陥分類，ISO9126 のような品質特性をもとに対象ソフトウェアやプロジェクトの特徴から起こりうる不具合を予想して設定することを想定している．観点 (1) の例を**表 1** に，観点 (2) の例を**表 2** に，それぞれ示す．

観点 (1)，(2) ともに複数の質問項目から構成される．観点 (1) の質問項目の集合を $Q^1 = \{q_1^1, q_2^1, \cdots, q_{n_1}^1\}$ とし，観点 (2) の質問項目の集合を $Q^2 = \{q_1^2, q_2^2, \cdots, q_{n_2}^2\}$ とする．Q^2 の各質問項目には，他の質問項目との相対的な重要度を示す重み w_l（$l = 1, 2, \cdots, n_2$）を設定する．Q^1，Q^2 ともどの質問項目から目視評価をはじめてもよい．観点 (1) は目視対象のモジュール $(m_1^t, m_2^t, \cdots, m_k^t)$ すべてに対するものである．Q^1 の質問項目は，いずれか 1 項目でも問題が発見されれば，そのモジュールには fault が含まれることを意味するので，他の質問項目による目視評価を止め，評価点を -1 とする．提案手法は受入れ検査で委託側が用いることを前提としているので，評価点が -1 のモジュールを受託側に伝えることにより，受託側は他の不具合の可能性の調査を含め再検討する．Q^1 の各質問項目に相対的な重みをつけない理由は，着目しているモジュールの不具

表 1 目視評価の観点（1）
Table 1 Perspective (1) of visual verification scoring.

Q^1	分類	質問項目
q_1^1	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）.
q_2^1	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.
q_3^1	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.
q_4^1	バッファオーバフロー対策漏れ	バッファオーバ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 2 目視評価の観点（2）
Table 2 Perspective (2) of visual verification scoring.

Q^2	品質特性	分類	重み	質問項目
q_1^2	保守性（変更性）	セルフチェック不足	1.0	コメント不備/更新忘れがないか.
q_2^2	保守性（変更性）	セルフチェック不足	1.0	コーディングルール非準拠箇所がないか.
q_3^2	使用性（運用性）	例外処理の検討不足	2.0	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.
q_4^2	機能性（セキュリティ）	考慮不足	3.0	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.

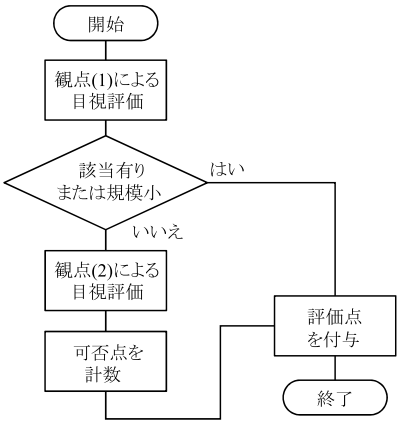

図 2 目視評価の手順
Fig. 2 The procedure of visual verification scoring.

合そのものを発見することになるため，質問項目に該当するモジュールを発見した場合に，それ以上の質問項目を評価しないからである．

観点 (2) は規模の大きなモジュールを対象としたものである．観点 (2) の評価は観点 (1) での評価において問題のなかったモジュールを対象とする．観点 (2) はすべての質問項目について，モジュール内での一貫性を評価し，“問題あり”，“問題なし”，“該当なし”，の 3 つのいずれかを評価結果とする．観点 (2) による評価を対象とするモジュールの規模の大きさは，評価対象のモジュール規模の分布や過去のバージョンにおける fault の分布にもよるが，一貫性を失うことにより問題が起こりやすくなる規模以上のものを対象とする．

目視評価は観点 (1) を先に実施し，次に観点 (2) を実施する．具体的な評価手順を**図 2** に示す．図 2 の手順は 1 つのモジュールに対するものであり，目視対象のモジュールの数だけ繰り返す．まず，観点 (1) による目視評価を実施する．問題があると判断されれば，評価点を -1 とし，そのモジュールの目視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再レビュー等の施策対象とする．観点 (1) による目視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，そのモジュールの目視評価を終了する．

観点 (2) はすべての質問項目 $\{q_1^2, q_2^2, \cdots, q_{n_2}^2\}$ について目視評価を実施し，モジュールごとに “問題なし”，“問題あり”，“該当なし” を判定し，それらの数および加重和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，数が多いほうを結果とする．具体的には，モジュール j（$1 \leq j \leq n_2$）の目視評価において問題なしと判定された質問項目の集合 C_j，問題ありと判定された質問項目の集合 U_j，該当なしの質問項目の集合を求め，次のとおりモジュール m_j の目視評価点 $f_v(m_j)$ を求める．

$$f_v(m_j) = \begin{cases} 1 & (|U_j| = 0 \text{ かつ } |C_j| \geq 1) \\ 0 & (|U_j| = |C_j| = 0) \\ 0 & (|U_j| \geq 1 \text{ かつ } \sum w_x > \sum w_y) \\ -1 & (|U_j| \geq 1 \text{ かつ } \sum w_x \leq \sum w_y) \end{cases}$$

ただし，$x : q_x^2 \in C_j$，$y : q_y^2 \in U_j$

3. ケーススタディ

提案手法による fault-prone モジュールのランク付けの精度を評価することを目的とし，ケーススタディを実施した．具体的には，判別性能が高いことが報告されている fault-prone モジュール判別モデル，判別性能がそれほど高くないと考えられるランダム，2 つのカテゴリの中間の判別性能となることが考えられるソースコードメトリクス，の 3 つのカテゴリにおいて，目視評価を組み合わせて評価する．それぞれのカテゴリにおいて，そもそもランク付け

表 3 対象ソフトウェア概要
Table 3 Outline of evaluation software.

プロセス	開発言語	LOC	GUI	主要機能	モジュール数
1	C++	6.7 k	無	監視結果から状態変更を検出	165
2	C++	8.4 k	無	対象装置とインタフェース	161
3	VB.NET	38.7 k	有	画面表示，操作受付	544

精度の向上があるか，向上する場合，どのカテゴリとの組み合わせが最も高いランク付け性能が得られるかを比較する．比較には AUC（Area Under the Curve）[14] の値を用いる．

3.1 準備

3.1.1 評価方法

対象プロジェクトで開発したソフトウェアを関数/メソッド単位で分割し，これを 1 モジュールとする．判別モデルの構築のため，全モジュールを規模順にソートし，偶数番目の群 A と奇数番目の群 B に分割し，それぞれを学習データ，予測データとする．目視評価前に実施する，モジュールの fault-proness のランク付け法として，以下に示す 3 通りの試行を行う．

- 試行 1．$f_m(m_i)$ として判別モデル（サポートベクタマシン）の予測結果を用いた結果
- 試行 2．$f_m(m_i)$ として対象モジュールの規模を用いた結果
- 試行 3．$f_m(m_i)$ をランダムとしたときの結果

試行 1 では，文献 [15], [16] で比較されている判別精度の高い fault-prone 判別モデルの中から，両文献において他の fault-prone モジュール判別モデルよりも再現率が高いとされ，提案手法で前提としている判別得点の得られるモデルとして，サポートベクタマシンを選んだ．受入れ検査では再現率の高さが優先されるからである．試行 2 では，ソースコードから直接測定できるメトリクスとして文献 [17], [18] で報告されているサイクロマチック数，ソースコード行数を候補として選び，より計測が容易なソースコード行数とした．試行 3 では，モジュールにランダムな判別得点を与えた．

予測データのうち目視評価の対象となるモジュールの割合 α を 0%（目視なし），25%，50%，75%，100%の 5 通りとした．ランク付けの評価には Alberg diagram [14]，および，AUC を用いる．Alberg diagram は，fault-prone モジュールの予測精度を可視化することを目的とした記法であり，予測手法による fault-prone の予測結果と実際に含まれていた不具合を比較する．具体的には，横軸を予測手法による fault-prone のランク，縦軸を実際に含まれていた不具合の累積件数としてプロットする．上に凸となっているほど，予測精度が高いことを示す．AUC は Alberg diagram における曲線下面積であり，$[0, 1]$ の値をとる．fault-prone モジュールと予測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 AUC は大きな値となる．

Alberg diagram，および，AUC を用いた fault-prone モジュール予測の評価は，テストのためのリソースが不足し，すべてのモジュールをテストできるとは限らない場合や，テスト工程の早い段階に不具合を見つけたい場合に適している [15], [19]．

3.1.2 対象ソフトウェア

対象ソフトウェアは，通信システムの設備機器を監視制御する商用アプリケーションソフトウェアである．その概要を**表 3** に示す．本アプリケーションは Windows XP 上で動作する 3 つのプロセスから構成されており，MS-DOS 版として作成されたソースコードを基に約 10 年前に Windows NT 4.0 用に移植されたものを機能拡張，変更する等して開発されている．納入先の要望に合わせて画面構成，表示色，表示内容や情報の集約方法を随時カスタマイズしており，以前のバージョンにない機能や画面を追加している．

評価対象ソースコードは Visual Studio 2005 で開発された 2 つのプロセス（表 3 中プロセス 1，および，2）である．規模はそれぞれ実行行数で 6,700 行，8,400 行である．これらの C++で実装した 2 つのプロセスのモジュール群のうち，ライブラリ関数として共通に使用している重複分，および，変数宣言のみのモジュールを除いた 125 モジュールを対象とした．対象ソフトウェアの開発では，要求定義，設計フェーズにおいて，受入れ検査をはじめとするテスト計画を立案する．その際に，要求と受入れ検査のテスト項目，要求と機能名，モジュール名の対応を明確にするためのトレーサビリティマトリクスを作成する．優先してテスト対象とすべきモジュールに対応する受入れ検査のテスト項目はこのトレーサビリティマトリクスを用いた．

試行 1 の判別モデルの構築は，測定したソースコードメトリクス（規模，McCabe のサイクロマチック数，ネスト数，関数呼び出し数）を説明変数，受入れ検査から出荷判定に合格した最終版（第 12 版）までに発見された不具合の有無（15 個，A 群には 8 個の不具合が，B 群には 7 個の不具合が含まれる）を目的変数とした．

文献 [20], [21] で報告されているように，判別精度の高い非線形サポートベクタマシンとし，カーネル関数はガウシアンカーネルを用いた．判別モデルは，判別得点が得られる判別モデルの中で最も性能の高いものの 1 つであ

表 4 目視評価の観点（1）
Table 4 Perspective (1) of visual verification scoring.

Q^1	分類	質問項目
q_1^1	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_2^1		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_3^1		変数を初期化せずに使用.
q_4^1		do〜while 文で while 条件はあるが do ループ内が空.
q_5^1		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_6^1	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_7^1		catch で適切な例外を個別に処理していない.
q_8^1		finally がなく，default の処理が規定されていない.
q_9^1	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}^1	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.

表 5 目視評価の観点（2）
Table 5 Perspective (2) of visual verification scoring.

Q^2	品質特性	分類	重み	質問項目
q_1^2	効率性	実装プロセス圧縮の悪影響	1.0	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_2^2	保守性	可読性の評価	1.0	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_3^2			1.0	switch 文で整理できる分岐を if 文で羅列していないか.
q_4^2			1.0	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_5^2			1.0	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_6^2		セルフチェック不足有無	1.0	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_7^2		実装作業時間不足の兆候	1.0	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_8^2		コーディングの適切さ	1.0	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_9^2			1.0	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_{10}^2		記述粒度の一貫性	1.0	プログラミングのルールを逸脱していないか.
q_{11}^2			1.0	記述粒度のばらつきがないか.
q_{12}^2	可搬性	環境依存の考慮不足	1.0	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_{13}^2			1.0	ログファイルの最大容量の考慮があるか.
q_{14}^2			1.0	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ないか.
q_{15}^2			1.0	必要な部分で volatile の付加漏れがないか.
q_{16}^2			1.0	必要でない割り込みの利用がないか.
q_{17}^2			1.0	演算結果がオーバフローする可能性がないか.
q_{18}^2			1.0	規定 API/インタフェースを使わず実装していないか.
q_{19}^2			1.0	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.

るサポートベクタマシンを用いた．サポートベクタマシン [22], [23], [24] 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おり [25]，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 [26].

試行 2 の規模順は，試行 1 の判別分析の説明変数に用いたソースコードの実行行数で計数した.

試行 3 のランダム順では，判別得点をランダムとし，個々のモジュールに与えた．具体的には，乱数の種を 100 個用意し，統計解析ソフトウェア R のメルセンヌツイスタ乱数を発生させ，判別得点とした.

3.1.3 目視評価

目視評価は，対象とするアプリケーションの委託組織に属する設計者が実施した．設計者はアプリケーションロジックを理解している．対象ソフトウェアやプロジェクトの特徴をふまえ，以下に示す方針で**表 4**（観点 (1)），**表 5**（観点 (2)）を目視評価の観点とした.

- 長期にわたって使用，保守されているシステムであり，今後も保守の計画があるため，環境非依存の実装，保守性，拡張性を十分に考慮する.
  - これまでにハードウェア，OS，API/ライブラリを更新した際の考慮点を委託側組織内でドキュメント化

したものを参考にし，今後起こりうる可能性のあるものを設定した．
  - 今後の拡張が容易になるよう，ソースコードの可読性，関数化の抽象度，適切な粒度での例外処理の定義等を設定した．
- 監視システムとして常時稼働しているため，長期間の稼働において問題となるリソース解放忘れがないか十分に考慮する．

また，観点 (2) の対象となるモジュールは 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，スクロールなしで見通すことができる行数であり，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．

3.2 適用結果

試行 1 の Alberg diagram を**図 3**（試行 1-1），**図 4**（試行 1-2）に示す．図 3 は学習データをモジュール群 A とし，予測データをモジュール群 B とした場合，図 4 は学習データをモジュール群 B とした場合である．すべての図において，次のように結果を示している．細い実線は目視評価を行わなかった結果（$\alpha = 0\%$）を表したもの，細い 1 点鎖線は最良値（すべての判別が正しい場合）を示している．他は提案手法の結果である．太い点線は α が 25%，細い 1 点鎖線は α が 50%，細い点線は α が 75%，太い鎖線は α が 100%の場合の累積不具合数を示す．

表 6 に後述する各試行を含めた AUC，および，AUC の最大値に対する割合（%）を示す．AUC の最大値は，すべての欠陥モジュール（n_f 件）を 1 位から順に n_f 位までランク付けし，$n_f + 1$ 位以降のモジュールには欠陥がないモジュールがランク付けされたときの AUC 値である．今回の試行では，予測モジュール群が B のときは 0.950，A のときは 0.942 となる．

試行 1 において，提案手法による AUC が判別モデル単体の AUC と比較して，大きくなるという結果が得られた．$\alpha = 25\%$の場合，すべての試行において，Alberg diagram の立ち上がりが目視評価を行わなかった場合と同等か，上回っており，このため表 6 において AUC が判別モデル単体より上回った．試行 1-1 の横軸 0.1 付近で判別モデル単体の不具合検出を下回っている箇所がある．$\alpha = 100\%$においては，試行 1-2 の一部で目視評価を行わなかった場合を下回っている箇所があるが，おおむねすべての試行で判別モデル単体を上回った結果となっている．

試行 2 における Alberg diagram を**図 5**，**図 6** に示す．表 6 に示すように，$\alpha = 25\%$の場合を除いて目視評価を行うことで AUC が大きくなった．試行 3 における Alberg diagram を**図 7**，**図 8** に示す．すべての α において目視評価を行うことで AUC が大きくなり，目視評価対象の割合 α の増加に従って AUC が大きくなるという結果が得られた．

観点に該当するモジュールの件数を**表 7** に示す．表にない観点は該当するモジュールがなかった．観点 (1) の 40%，観点 (2) の 26%の観点に該当するモジュールがあった．

表 8 に試行 1，2，3 の目視評価において必要となったコストを示す．すべての α において試行 1，2 の間に大きな目視コストの差はなかった．試行 3 は α が 50%以下で試行 1，2 と比較して最大で 1 人時程度小さい値となった．

4. 考察

いずれの試行においても目視評価を行うことにより，AUC が大きくなるという結果が得られた．特に，fault-prone モジュール判別モデルを用いた試行 1 では，$\alpha = 50\%$において，モジュール群 A，B ともに AUC が 0.8 に近い値とな

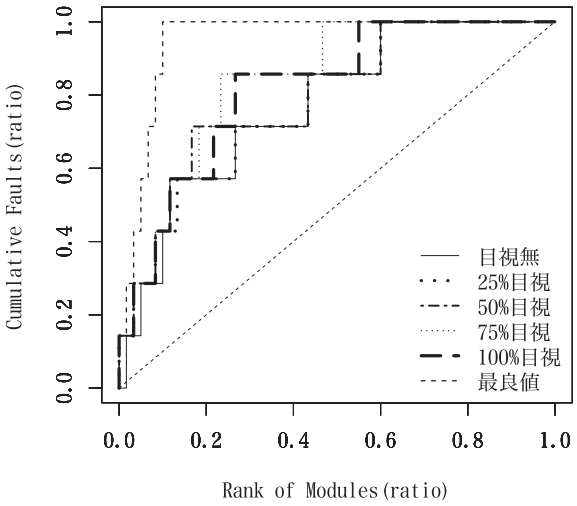

図 3 試行 1-1 サポートベクタマシン，予測データ：モジュール群 B
Fig. 3 Trial 1-1 support vector machine, test data: module group B.

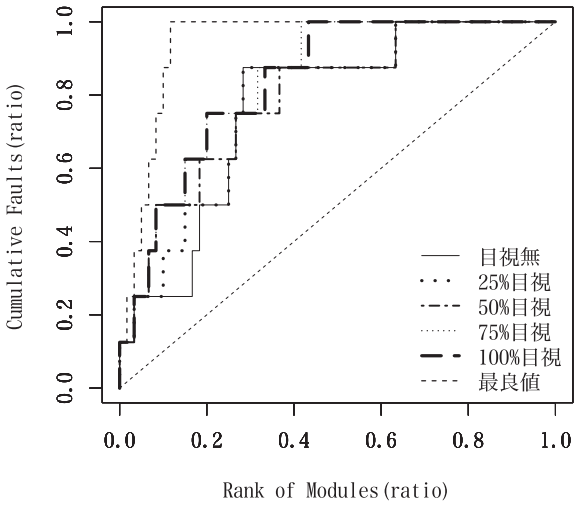

図 4 試行 1-2 サポートベクタマシン，予測データ：モジュール群 A
Fig. 4 Trial 1-2 support vector machine, test data: module group A.

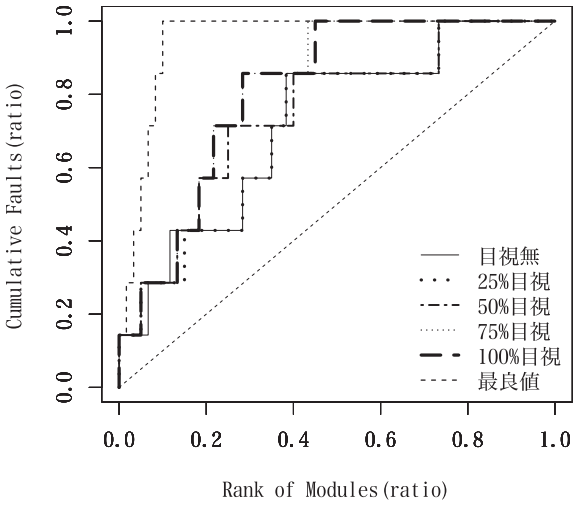

図 5 試行 2-1 規模順，予測データ：モジュール群 B
Fig. 5 Trial 2-1 module size order, test data: module group B.

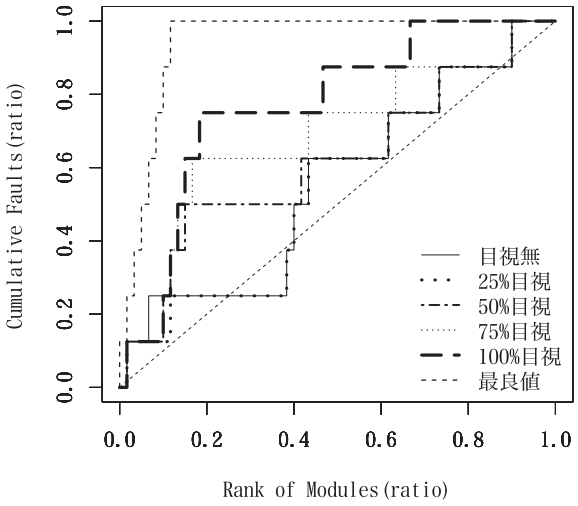

図 6 試行 2-2 規模順，予測データ：モジュール群 A
Fig. 6 Trial 2-2 module size order, test data: module group A.

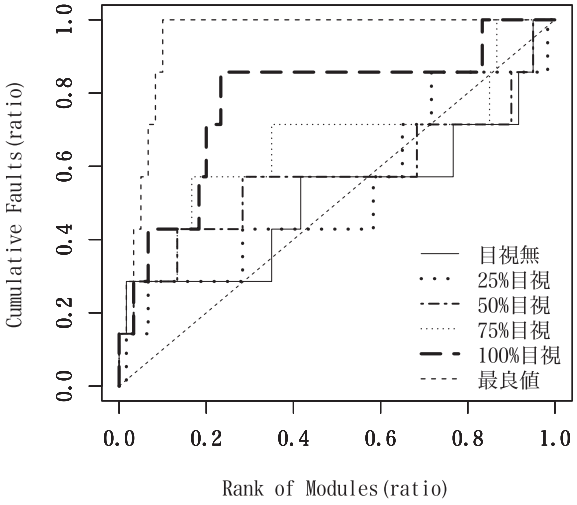

図 7 試行 3-1 ランダム順，予測データ：モジュール群 B
Fig. 7 Trial 3-1 random, test data: module group B.

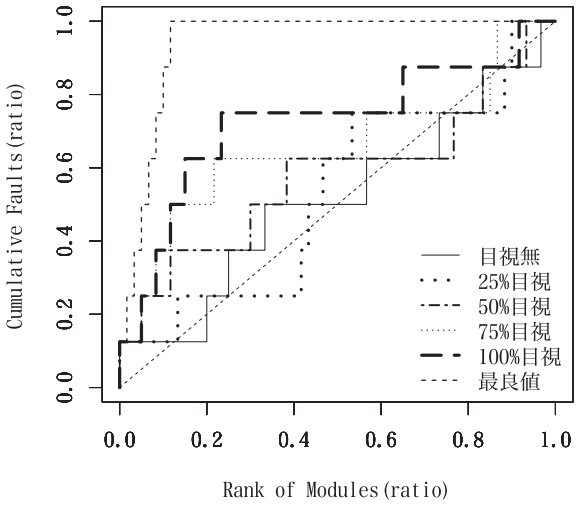

図 8 試行 3-2 ランダム順，予測データ：モジュール群 A
Fig. 8 Trial 3-2 random, test data: module group A.

表 6 各試行における AUC と最良値に対する割合 [%]
Table 6 AUC and ratio (%) for best value for each trial.

	予測モジュール群	試行 1（SVM）		試行 2（規模順）		試行 3（ランダム順）	
		AUC	ratio	AUC	ratio	AUC	ratio
目視なし	B	0.774	(81.5)	0.724	(76.2)	0.512	(53.9)
	A	0.773	(82.1)	0.556	(59.0)	0.515	(54.7)
$\alpha = 25\%$	B	0.779	(82.0)	0.721	(75.9)	0.529	(55.7)
	A	0.785	(83.4)	0.550	(58.4)	0.529	(56.2)
$\alpha = 50\%$	B	0.795	(83.7)	0.750	(78.9)	0.574	(60.4)
	A	0.796	(84.5)	0.619	(65.7)	0.577	(61.3)
$\alpha = 75\%$	B	0.840	(88.4)	0.814	(85.7)	0.657	(69.2)
	A	0.842	(89.4)	0.688	(73.1)	0.656	(69.7)
$\alpha = 100\%$	B	0.819	(86.2)	0.812	(85.5)	0.779	(82.0)
	A	0.838	(89.0)	0.771	(81.9)	0.760	(80.7)

り，ソースコードメトリクスのカテゴリを用いた試行 2，ランダムの試行 3 を上回る AUC の値が得られた．判別モデル単体でも判別性能が高いサポートベクタマシンとの組み合わせにおいても AUC の向上が見られ，すでに判別性能の高いモデルと目視評価を組み合わせた場合であっても提案手法が有効であることを示せた．AUC の最大値に対

表 7 観点（1），観点（2）とケーススタディの該当件数
Table 7 The number to correspond to perspective (1) and (2) in the case study.

観点 (1)		観点 (2)	
観点	件数	観点	件数
q_3^1	10	q_8^2	6
q_2^1	3	q_9^2	6
q_4^1	3	q_3^2	3
q_6^1	1	q_2^2	2
		q_6^2	1

表 8 各試行における目視コスト [人時]
Table 8 Cost for each trial.

α	予測モジュール群	試行 1	試行 2	試行 3
25%	B	3.59	3.76	2.74
	A	3.59	3.59	2.57
50%	B	5.99	6.16	5.31
	A	5.99	5.99	5.48
75%	B	8.56	8.56	7.88
	A	8.39	8.39	8.05
100%	B	10.96	10.96	10.96
	A	10.96	10.96	10.96

する割合も AUC とほぼ同じ傾向の結果となり，判別モデルと目視評価を組み合わせた提案手法の優位性を示せた．試行 1 では，判別得点だけで A 群，B 群とも AUC の最大値に対して約 82%程度であったが，半数のモジュールの目視により，約 84%に，75%のモジュールの目視により，約 89%となった．すでにランク付け精度の高い判別モデルとの組み合わせにおいても精度の向上があった．

表 6 に示した 25%おきの区間で AUC の増加を比較すると，試行 1-1，1-2，2-1 では，50%から 75%の間で，増加が最も大きくなった．試行 2-2，3-1，3-2 では，75%から 100%において AUC の増加が他の区間よりも大きくなった．試行 3 では，判別得点がランダムであるため，目視得点の対象となるモジュールが増えるにつれ（α が増えるにつれ），AUC の増加が大きくなる．

提案手法では，α を設定することにより，目視評価対象モジュールを限定することができ，目視評価に必要なコストを低減させることができる．一般には，α を大きくするとより大きな AUC 値が得られることが期待されるが，その分，目視評価に必要なコストが大きくなる．ケーススタディでは，$\alpha = 75\%$と $\alpha = 100\%$の Alberg diagram 上での予測精度の差はほとんどなかった．α を 75%とすることにより，プロセス 1，2 をあわせると約 31 モジュールの目視評価を省くことができることを示している．ケーススタディでは，目視評価に要するコストは観点 (1) による目視評価で 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．受入れ検査における不具合の早期発見によって受入れ検査実施中と出荷後の両方においてメリットがある．受入れ検査序盤においては，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．受入れ検査終盤においては，スケジュールが圧迫された場合に修正中の部分を除外し，前倒しで部分的にテストを実施するコストを低減させることができる．また，受入れ検査終盤において，契約上の納期に間に合わせるために場当たり的な修正を施す可能性が低減できるため，出荷後，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．

表 8 に各試行に要した目視コスト（人時）を示す．ケーススタディでは，いずれの試行においても α を 75%とするのに必要な目視評価コストはほぼ 8 [人時] でよく，現場への適用には無理がないと考えられる．

試行で用いた観点 (1) には q_2^1，q_4^1，q_5^1 のように検出を自動化できるものや q_1^1，q_3^1，q_8^1，q_9^1，q_{10}^1 のように候補の列挙を自動化し，誤検出を目視で確認できるものがある．試行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，目視評価の項目としたが，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，今回の試行で準備コストが確保できれば，目視ではなく fault-prone モジュール判別モデルの説明変数に加えることができる観点 q_2^1，q_4^1，q_5^1 をサポートベクタマシンに与え，試行 1' を実施した．試行 1' では，3 つの観点を目視評価から削除し，サポートベクタマシンのモデル構築に用い，fault-prone モジュール判別時の説明変数として与えた．試行 1' の α の値を他の試行と同様に 5 段階に変化させて試行 1 の AUC 値と比較したところ同等か若干小さくなったが，試行 2 よりも大きな値となった．また，他の観点においても，候補の列挙を自動化し，候補から誤検出等を目視で選択する方法により，目視評価コストを小さくできる場合がある．提案手法の実施コスト低減のために，目視評価を自動化することは今後の重要な課題の 1 つである．

α は目視評価の実施時点において増減できる．α の設定は目視評価の時間とコスト以外の制約がないため，事前に決定しておく必要がなく，目視評価の途中であっても中断，追加が可能である．対象ソフトウェア，プロジェクトの状況を勘案することにより，ソフトウェアやプロジェクトごとに α を設定することができる．ケーススタディでは，α が 25%程度であっても AUC の向上がみられた．

提案手法で用いる判別モデルには，ケーススタディで用いたソースコード静的解析の結果を用いた判別モデルをはじめとして，プログラムが動作することを前提としないものを選択できる．ソースコードメトリクスはソースコードがコンパイル可能になった時点で収集でき，目視評価も実施可能である．ケーススタディでは，受入れ検査を想定したものとしたが，原理的には，委託先での単体テストや結合テストが進行している段階の中間成果物にも適用するこ

とができる [27]．これらの中間成果物の評価に提案手法を利用することで，受入れ検査の実施順序やその後のテストの実施順序やリソース割当てを再計画し，効率化できることが期待される．

提案手法では，観点 1 は既存のコードレビューの際に使うチェックリストや過去の不具合のうち，対象ソースコードにも起こりうるもの，かつ，局所的な目視評価で確認できるもの，としている．また，観点 2 は対象ソースコードに特に求められる品質について，品質特性と照らし合わせたものとしている．ケーススタディでは，対象プロジェクトやアプリケーションにおいて長年蓄積された過去の不具合情報から，両観点とも設定することができ，これらに関する問題を検出することができた．しかしながら，両観点のシステマチックな設定方法はまだ明らかではなく，今後の重要な課題の 1 つである．

5. 関連研究

ソースコードメトリクスを用いて fault-prone モジュールを予測する研究は多岐にわたり，ソースコードメトリクスの性質についての研究や考察 [14], [17], [18], [28], [29], [30], [31]，主としてソースコードメトリクスを用いた判別や予測を行った研究 [4], [5], [6], [7], [8] がある．文献 [4] では決定木を利用，文献 [5]，文献 [6] では文献 [14] で提示された新規/変更モジュールに注目，文献 [7] ではロジスティック回帰分析を用いた判別，文献 [8] では単純 Bayes 分類器を用いている．これらの研究は，ソースコードメトリクスを中心としたパラメータのみを単一の判別モデルに与えることを前提としている点で本研究とは異なる．

ソースコードメトリクスとその他のメトリクスを判別モデルに与え，fault-prone モジュールを予測する手法として，変更報告書を利用して改版モジュールと新規モジュールでサイクロマチック数と規模を分析 [32]，ソースコードの変更前後の不具合検出の有無の情報を併用 [11]，改版情報，欠陥履歴の併用 [10]，構成管理システムのログから得たデータの併用 [33]，3 段階の動的解析の結果の併用 [34]，構成管理システムから得たプロセスメトリクスの併用 [9]，設計メトリクスの併用 [12]，自動検査（Automated Software Inspection）ツールによる fault-prone モジュールの予測結果とソースコードの改変規模を併用 [35]，がある．これらはいずれもソースコードメトリクスとそれ以外の情報を併用している点で提案手法と共通しているが，同一の判別モデルにそれらをパラメータとして与える点で異なる．また，これらのいずれも提案手法の判別モデルとすることができる．

複数の判別モデルを組み合わせて fault-prone モジュールを予測する手法として，ロジスティック回帰分析とラフ集合論から導出したルールを組み合わせた手法 [36]，ロジスティック回帰分析と相関ルール分析を組み合わせた手法 [37]，ロジスティック回帰分析と決定木を組み合わせた手法 [38]，ロジスティック回帰分析と分類木を組み合わせた手法 [39]，ロジスティック回帰分析と動的リスクモデルを組み合わせた手法 [40]，探索木とニューラルネットワークを組み合わせた手法 [41]，OSR（Optimized Set Reduction）を組み合わせた手法 [42] がある．これらの手法は判別方法（モデル）を組み合わせるという点で提案手法と共通しているが，提案手法のように人手による判別を組み合わせた手法ではない．また，提案手法では判別モデルを特定のものに限定しないため，原理的にはこれらのモデルを提案手法の判別モデルとして利用することができる．

一般に，多重共線性が避けられる場合において説明変数として得られる情報が多い方がより精度の高い判別や予測が可能となるが，説明変数として得られる情報を取得するのに必要となるコストとのトレードオフを考慮した文献として，文献 [43] では決定木を用いてより正確な情報を取得するコストとのトレードオフを最適化しており，文献 [44] では情報取得コストと誤判別コストの総和の最小化を行っている．誤判別コストとのトレードオフの研究では，文献 [9] や [45] があり，第 2 種の過誤により大きな損失コストを割り当ててこれを防ぐようにしている．テストの優先順位に関する研究では文献 [46], [47], [48] 等があり，文献 [47] では，対象ソフトウェアを機能単位に分割して評価点によるメトリクスを用いた荷重和によってテスト優先度を決定しており，文献 [46] では，ソフトウェアを構成するコンポーネント間の依存性に着目，文献 [49] では，fault 網羅率の増加値によってテスト優先順位を決定している．

6. まとめ

委託開発の受入れ検査において検査開始後の早い段階において多くの不具合を検出することを目的とし，判別モデルによる fault-prone モジュールの判別得点と目視評価を組み合わせ，検査を実施するモジュールの順番を決定する手法を提案した．具体的には，判別モデルから得た判別得点によって fault-prone モジュールの可能性が大きいと判別されたモジュールから順に上位 α%のモジュールを対象として目視評価を実施し，判別得点と目視評価の結果を用いて fault-prone モジュールの可能性の大きい順にモジュールをランク付けする．目視評価においてはプロジェクトや対象ソフトウェアに適合するよう，あらかじめ目視評価の観点を設定しておき，目視評価者が観点に沿って評価点を付与する．観点は，すべてのモジュールを対象とし目視評価で直接問題点を見つける観点 (1) と規模の大きなモジュールを対象とし，モジュール全体の整合性や一貫性を目視評価し，問題点を間接的に予測しようとする観点 (2) である．

提案手法の評価を目的として，長期にわたって稼働実績のある商用ソフトウェアの派生開発品を対象として，ケー

ススタディを実施した．ケーススタディでは，fault-prone モジュール判別モデル，ソースコードメトリクス，ランダムを判別モデルとして，目視評価を組み合わせた場合と比較した．交差検証法により判別モデル単体での判別精度と提案手法の α を，25%，50%，75%，100%と変化させ，Alberg diagram と AUC により比較した．いずれの判別モデルとの組み合わせにおいても，目視評価を組み合わせることにより，判別モデル単体のときよりも AUC が大きくなるという結果が得られた．また，判別モデル単体ですでに判別精度が高い fault-prone モジュール判別モデルであっても，目視評価との組み合わせにより判別精度が高まり，3 つのカテゴリの中で最も大きな AUC 値が得られた．

また，AUC が最も大きくなったのはサポートベクタマシンであり，提案手法の有効性を示せた．

ケーススタディでは，目視評価の対象となるモジュールの割合 α を増やすと AUC の値も大きくなっていた．提案手法では，α の値を事前に決めておく必要がないため，プロジェクトの状況に応じて目視評価の対象を増減させることにより，状況にあわせてランク付けの精度を高めることができる．

謝辞 本研究の一部は，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．本研究の一部は，文部科学省科学研究補助費（基盤 B：課題番号 23300009）および（若手 B：課題番号 20700033）による助成を受けた．

参考文献

[1] 角田雅照，門田暁人，宿久 洋，菊地奈穂美，松本健一：外部委託率に着目したソフトウェアプロジェクトの生産性分析，信学技報，Vol.106, No.16, pp.19–24 (2006).
[2] 経済産業省独立行政法人情報処理推進機構：ソフトウエア開発データ白書 2009，日経 BP (2009).
[3] 経済産業省独立行政法人情報処理推進機構：2005 年版組込みソフトウェア産業実態調査報告書，商務情報政策局情報政策ユニット情報処理振興課 (2005).
[4] Porter, A.A. and Selby, R.W.: Empirically Guided Software Development Using Metric-Based Classification Trees, *IEEE Softw.*, Vol.7, No.2, pp.46–54 (1990).
[5] Khoshgoftaar, T.M., Allen, E.B., Kalaichelvan, K.S. and Goel, N.: Early Quality Prediction: A Case Study In Telecommunications, *IEEE Softw.*, Vol.13, No.1, pp.65–71 (1996).
[6] Fenton, N.E. and Ohlsson, N.: Quantitative Analysis of Faults and Failures in A Complex Software System, *IEEE Trans. Softw. Eng.*, Vol.26, No.8, pp.797–814 (2000).
[7] Denaro, G., Morasca, S. and Pezzè, M.: Deriving Models of Software Fault-Proneness, *Proc. 14th Intl. Conf. Softw. Eng. and Knowl. Eng.*, pp.361–368 (2002).
[8] Menzies, T., Greenwald, J. and Frank, A.: Data Mining Static Code Attributes to Learn Defect Predictors, *IEEE Trans. Softw. Eng.*, Vol.33, pp.2–13 (2007).
[9] Moser, R., Pedrycz, W. and Succi, G.: A Comparative Analysis of The Efficiency of Change Metrics and Static Code Attributes for Defect Prediction, *Proc. 30th Intl. Conf. Softw. Eng.*, pp.181–190 (2008).
[10] Graves, T.L., Karr, A.F., Marron, J. and Siy, H.: Predicting Fault Incidence Using Software Change History, *IEEE Trans. Softw. Eng.*, Vol.26, pp.653–661 (2000).
[11] Yu, T., Shen, V.Y. and Dunsmore, H.E.: An Analysis of Several Software Defect Models, *IEEE Trans. Softw. Eng.*, Vol.14, No.9, pp.1261–1270 (1988).
[12] Jiang, Y., Cuki, B., Menzies, T. and Bartlow, N.: Comparing Design and Code Metrics for Software Quality Prediction, *Proc. 4th Intl. Workshop on Predictor Models in Soft. Eng.*, pp.11–18 (2008).
[13] Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B. and Wong, M.: Orthogonal Defect Classification — A Concept for In-Process Measurements, *IEEE Trans. Softw. Eng.*, Vol.18, pp.943–956 (1992).
[14] Ohlsson, N. and Alberg, H.: Predicting Fault-Prone Software Modules in Telephone Switches, *IEEE Trans. Softw. Eng.*, Vol.22, No.12, pp.886–894 (1996).
[15] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S.: Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, *IEEE Trans. Softw. Eng.*, Vol.34, No.4, pp.485–496 (2008).
[16] He, Z., Shu, F., Yang, Y., Li, M. and Wang, Q.: An Investigation on The Feasibility of Cross-project Defect Prediction, *Automated Softw. Eng.*, Vol.19, pp.167–199 (2012).
[17] Basili, V.R. and Hutchens, D.H.: An Empirical Study of A Syntactic Complexity Family, *IEEE Trans. Softw. Eng.*, Vol.9, No.6, pp.664–672 (1983).
[18] Selby, R.W. and Basili, V.R.: Analyzing Error-Prone System Structure, *IEEE Trans. Softw. Eng.*, Vol.17, No.2, pp.141–152 (1991).
[19] Jiang, Y., Cukic, B. and Ma, Y.: Techniques for Evaluating Fault Prediction Models, *Empirical Softw. Eng.*, Vol.13, pp.561–595 (2008).
[20] Sebald, D.J. and Bucklew, J.A.: Support Vector Machine Techniques for Nonlinear Equalization, *IEEE Trans. Signal Processing*, Vol.48, No.11, pp.3217–3226 (2000).
[21] Kamei, Y., Monden, A. and Matsumoto, K.: Empirical Evaluation of an SVM-based Software Reliability Model, *Proc. Intl. Symp. Empirical Softw. Eng.*, pp.39–41 (2006).
[22] Boser, B.E., Guyon, I.M. and Vapnik, V.N.: A Training Algorithm for Optimal Mergin Classifiers, *Proc. 5th ACM Workshop on Computational Learning Theory*, pp.144–152 (1992).
[23] Burges, C.J.C.: A Tutorial On Support Vector Machines For Pattern Recognition, *Data Min. Knowl. Discov.*, Vol.2, No.2, pp.121–167 (1998).
[24] Vapnik, V.: An Overview Of Statistical Learning Theory, *IEEE Trans. Neural Networks*, Vol.10, No.5, pp.988–999 (1999).
[25] Scholkopf, B., Sung, K.-K., Burges, C., Girosi, F., Niyogi, P., Poggio, T. and Vapnik, V.: Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers, *IEEE Trans. Signal Processing*, Vol.45, No.11, pp.2758–2765 (1997).
[26] Burges, C.J.C. and Crisp, D.J.: Uniqueness Of The SVM Solution, *NIPS*, pp.223–229 (1999).
[27] 笠井則充，森崎修司，松本健一：中間成果物のサンプリングによる全体品質の推測に向けた分析，ソフトウェア品質シンポジウム 2009 発表報文集，pp.185–190 (2009).
[28] McCabe, T.J.: A Complexity Measure, *Proc. 2nd Intl.*

Conf. Softw. Eng., p.407 (1976).
[29] Li, H.F. and Cheung, W.K.: An Empirical Study of Software Metrics, *IEEE Trans. Softw. Eng.*, Vol.13, No.6, pp.697–708 (1987).
[30] Lehman, M.M., Perry, D.E. and Ramil, J.F.: Implications of Evolution Metrics on Software Maintenance, *Proc. Intl. Conf. Softw. Maintenance*, p.208 (1998).
[31] Menzies, T., Stefano, J.S.D., Chapman, M. and McGill, K.: Metrics That Matter, *Proc. IEEE/NASA Goddard Softw. Eng. Workshop* (2002).
[32] Basili, V.R. and Perricone, B.T.: Software Errors And Complexity: An Empirical Investigation, *Comm. ACM*, Vol.27, No.1, pp.42–52 (1984).
[33] Phadke, A.A. and Allen, E.B.: Predicting Risky Modules In Open-Source Software For High-Performance Computing, *Proc. 2nd Intl. Workshop on Software Engineering for High Performance Computing System Applications*, pp.60–64 (2005).
[34] Csallner, C., Smaragdakis, Y. and Xie, T.: DSD-Crasher: A Hybrid Analysis Tool For Bug Finding, *ACM Trans. Softw. Eng. Methodol.*, Vol.17, No.2, pp.1–37 (2008).
[35] Nagappan, N., Williams, L., Hudepohl, J., Snipes, W. and Vouk, M.: Preliminary Results on Using Static Analysis Tools for Software Inspection, *Intl. Symp. Softw. Reliability Eng.*, pp.429–439 (2004).
[36] Morasca, S. and Ruhe, G.: A Hybrid Approach to Analyze Empirical Software Engineering Data and Its Application to Predict Module Fault-Proneness in Maintenance, *J. of Systems and Software*, Vol.53, No.3, pp.225–237 (2000).
[37] 亀井靖高，森崎修司，門田暁人，松本健一：相関ルール分析とロジスティック回帰分析を組み合わせた Fault- Prone モジュール判別方法，情報処理学会論文誌，Vol.49, No.12, pp.3954–3966 (2008).
[38] Selby, R.W. and Porter, A.A.: Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis, *IEEE Trans. Softw. Eng.*, Vol.14, No.12, pp.1743–1757 (1988).
[39] Morasca, S.: A Proposal for Using Continuous Attributes in Classification Trees, *Proc. 14th Intl. Conf. on Softw. Eng. and Knowl. Eng.*, pp.417–424 (2002).
[40] Wong, W.E., Qi, Y. and Cooper, K.: Source Code-Based Software Risk Assessing, *Proc. 2005 ACM Symp. Applied Computing*, pp.1485–1490 (2005).
[41] Pendharkar, P.C.: Exhaustive and Heuristic Search Approaches for Learning A Software Defect Prediction Model, *Eng. Appl. Artif. Intell.*, Vol.23, No.1, pp.34–40 (2010).
[42] Briand, L.C., Basili, V.R. and Thomas, W.M.: A Pattern Recognition Approach for Software Engineering Data Analysis, *IEEE Trans. Softw. Eng.*, Vol.18, No.11, pp.931–942 (1992).
[43] Tan, M.: Cost-Sensitive Learning of Classification Knowledge and Its Applications in Robotics, *Mach. Learn.*, Vol.13, pp.7–33 (1993).
[44] Bilgic, M. and Getoor, L.: VOILA: Efficient Feature-Value Acquisition For Classification, *22nd Conf. Artificial Intelligence* (2007).
[45] Khoshgoftaar, T.M., Yuan, X. and Allen, E.B.: Balancing Misclassification Rates in Classification-TreeModels of Software Quality, *Empirical Softw. Eng.*, Vol.5, pp.313–330 (2000).
[46] Borner, L. and Paech, B.: Integration Test Order Strategies to Consider Test Focus and Simulation Effort, *Intl. Conf. Advances in System Testing and Validation Lifecycle*, pp.80–85 (2009).
[47] 平山雅之，山本徹也，岡安二郎，水野 修，菊野 亨：機能モジュールに対する優先度に基づいた選択的ソフトウェアテスト手法の提案，信学技報 SS，ソフトウェアサイエンス，Vol.101, No.98, pp.1–8 (2001-05-22).
[48] Wang, Q., Wang, S. and Ji, Y.: A test sequences optimization method for improving fault coverage, *Proc. 2nd IEEE Intl. Conf. on Information Management and Engineering*, pp.80–84 (2010).
[49] Gangaram, V., Bhan, D. and Caldwell, J.K.: Functional Test Selection for High Volume Manufacturing, *Proc. 7th Intl. Workshop on Microprocessor Test and Verification*, pp.15–19 (2006).

笠井 則充

1991 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．同年三菱電機（株）入社．衛星通信システムのソフトウエア設計に従事．2009 年奈良先端科学技術大学院大学情報科学研究科博士後期課程．

森崎 修司 （正会員）

2001 年奈良先端科学技術大学院大学情報科学研究科博士課程修了．同年（株）インターネットイニシアティブ入社．2007 年奈良先端科学技術大学院大学助教．2012 年静岡大学講師．ソフトウェアレビュー，エンピリカルソフトウェア工学の研究に従事．博士（工学）．電子情報通信学会，プロジェクトマネジメント学会，IEEE 各会員．

松本 健一 （正会員）

1985 年大阪大学基礎工学部情報工学科卒業，1989 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．1993 年奈良先端科学技術大学院大学助教授．2001 年同大学教授．工学博士．エンピリカルソフトウエ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，ACM 各会員，IEEE Senior Member．