NAIST-IS-DD0961009

博士論文

判別モデルに目視評価を組合せた fault-prone モジュール判別手法

笠井 則充

2014 年 3 月 13 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

笠井 則充

審査委員：

松本 健一 教授	（主指導教員）
関　浩之 教授	（副指導教員）
門田 暁人 准教授	（副指導教員）
森崎 修司 准教授	（名古屋大学）

判別モデルに目視評価を組合せた fault-prone モジュール判別手法*

笠井 則充

内容梗概

ソフトウェアモジュールに対する fault-prone 判別を，従来の判別モデルに目視評価を組合せて行う方法を提案する．一般的な判別モデルではソースコードやその更新履歴の特性値群を入力データとし，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．ただし，コメントの説明と実際のソースコードの不一致やプラットフォームに依存した実装といった潜在的な不具合の存在や兆候をそれら特性値から捉えることは困難であり，モデルだけで判別精度を高めることには限界があるとの指摘がある．こういった不具合は目視評価によって検出できる可能性があるが，目視評価には多くの時間・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を評価することは現実的ではない．そこで，提案法では受入れ検査工程を対象に，従来の判別モデルによる評価結果から目視評価すべきモジュールを選択し，それにより目視評価のコストを抑えつつ，より高精度な fault-prone 判別を目指すこととした．

まず，従来の判別モデルの得点と目視による評価得点の和を fault-prone 判別得点とすることで，fault-prone 判別の精度が向上することを実験により確かめた．具体的には，全てのモジュールに判別モデルにより得点を与え，その得点を用いて，$\alpha\%$ のモジュールを目視評価の対象とする．目視評価の対象として選んだモジュールでは，判別モデルの得点に荷重 β を乗じた目視評価得点を加え fault-prone

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DD0961009, 2014 年 3 月 13 日.

判別得点とした．目視評価するモジュールの選び方は判別モデル得点の昇順，降順といった 4 種類の方法を比較した．いずれの方法においても，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で fault-prone モジュールを判別できることを確認した．また，判別モデルの判別得点をソースコード行数，ランダム値として，一般的な判別モデルであるサポートベクタマシンと比較したところ，サポートベクタマシンでの判別精度は，殆どの場合で精度が大きくなり，α が 40%のとき最も大きくなった．

次に，目視評価のコストを抑えることを目的として，規模の小さいモジュールにおいて目視評価の一部を省略するスコアリング方法を開発した．実験により，スコアリング方法が，目視評価の一部を省略しない場合と同程度の判別精度を保ちながら，43%の工数を削減できることがわかった．

キーワード

fault-prone モジュール，判別分析, サポートベクタマシン, 目視評価

An Approach for Fault-Prone Module Prediction Using a Discriminative Model and Manual Inspection*

Norimitsu Kasai

Abstract

This thesis proposes an approach for detecting fault-prone module by combining fault-prone detection model and manual inspection. Common fault-prone module detection models take as input source code metrics or source code change history and output fault-proneness of module. If fault-proneness exceeds threshold, the module is estimated as fault-prone. Source code metrics and source code change history might not capture some kind of potential fault such as platform dependent implementation and inconsistency between source code and its comment. Manual inspection is expected to detect such faults, however, manual inspection requires a lot of time and effort in actual scale of source code. Proposed approach aims to decrease manual inspection effort and increase detection accuracy by selecting source code modules to be inspected manually according to fault-proneness estimated by fault-prone detection model in acceptance testing.

First, an experiment was conducted to empirically investigate that detection accuracy was improved by combining fault-prone detection model and manual inspection. In the experiment, fault-proneness by detection model was used for selecting $\alpha\%$ of modules to be manually inspected. Fault-proneness of each selected module is sum of fault-proneness by detection model and manual inspection score with coefficient β. Fault-proneness of each module without manual

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0961009, March 13, 2014.

inspection is fault-proneness by detection model. Four ways of modules selection including ascending and descending order of fault-proneness are compared. The results of the experiment showed that detection accuracies of fault-proneness was increased with manual inspection with appropriate α and β. Also, accuracy of fault-proneness by combination of support vector machine and manual inspection was almost larger than lines of source code of module and random score. It was the maximum at $\alpha = 40\%$.

Second, manual inspection scoring procedure omitting part of manual inspection for smaller source code modules decreased manual inspection effort by 43 % without decreasing detection accuracy.

Keywords:

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

研究業績

学術論文誌

1. 笠井 則充, 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, September 2012.（第 3 章に関連する）

国際会議発表

1. Norimitsu Kasai, Shuji Morisaki, and Ken-ichi Matsumoto, Fault-Prone Module Prediction Using a Prediction Model and Manual Inspection, In Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC 2013), December 2013.（第 2 章に関連する）

査読付き国内研究集会発表

1. 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, September 2009.

目 次

図 目 次

表 目 次

第1章　序論

1．研究の背景と目的

企業におけるソフトウェア開発は，年々開発規模は大きく，開発期間は短くなる傾向にある [66]．文献 [31] によると，開発ライフサイクルモデルの 96.5% がウォータフォールモデルによる開発である．反復開発手法 [1] と比較して成果物の受託者との分担が契約形態になじむことや，業務分担によって生産性の向上が図れることが要因と考えられる [61]．文献 [30] 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受託者の工数であること，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 [29]，外部委託を行った上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，受託者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 (図 1.1)．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．

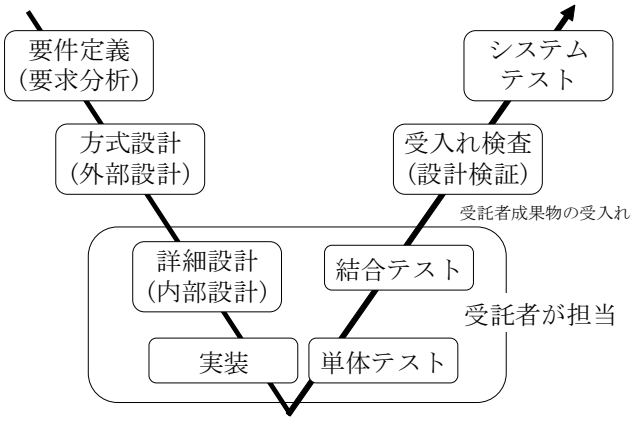

図 1.1 V 字モデル

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大きくなる．

一般に，手戻りが大きくなると修正にかかるコストは飛躍的に増大する [42]．受注から納入までの工期が短いプロジェクトでは，特に出荷前の最終プロセスであるシステムテストに割り当てる工数，時間が短くなる場合があり，この段階で発見された不具合は，出荷迄の短期間での修正が要求される場合がある．

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 [15] [16] [32] [40] [51] 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．これらの手法では，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

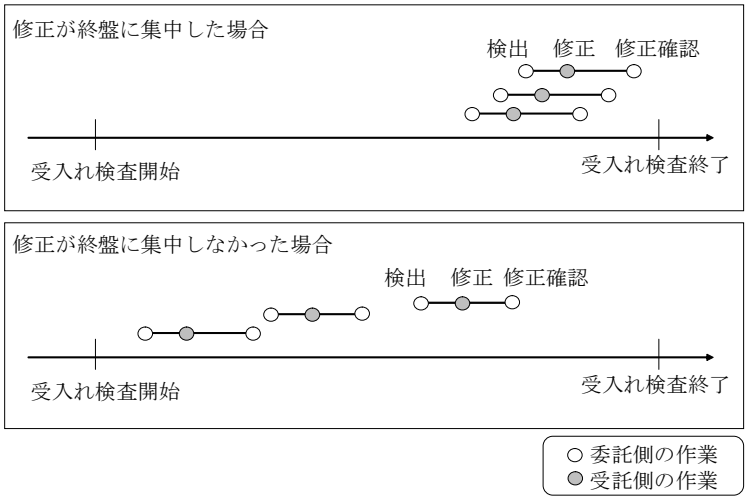

図 1.2 受入れ検査終盤での不具合検出の集中

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，全てのモジュールの目視が必要になる (図 1.2)．

本研究では，受入れ検査において不具合を含む可能性の大き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合せた fault-prone モジュールのランク付け手法を提案する．

2. 研究の方法

対象とするアプリケーションのソースコード全量を目視して詳細に評価することが可能であれば，大部分の欠陥を発見できる可能性は大きいと思われる．しかしこれには非常に大きな工数がかかることが容易に想像される．アプリケーションのソースコードを部分に分け，そのいくつかの部分のみを目視評価することで，含まれる欠陥の可能性を指摘することができれば，より早い段階でのソフトウェアの品質の推定，効率的なテスト工数の配分，手戻りの低減による納期遵守の可能性が大きくなる．

fault-prone 判別モデルに目視評価を組合せるとき，目視評価が判別分析結果を補強するならば，目視評価は判別分析結果の第一種及び第二種の過誤を修正する．判別分析の結果 fault-prone の度合いの大きいモジュールに対しては第一種の過誤を，fault-prone の度合いの小さいモジュールに対しては第二種の過誤を，それぞれ目視評価によって修正することができれば，目視評価を行うことで判別分析のみの場合よりも判別精度が大きくなる．

ソースコードの部分分けの方法について，アプリケーションの仕様書に記載されている機能単位，ソースコードのファイル単位，関数/メソッド単位の意味でのモジュール単位，クラス単位，ユースケース単位等が考えられる．ケーススタディの対象アプリケーションの初版が MS-DOS 上で動作する C 言語のものであったこと，本研究で評価に使用したときまでに 10 年以上に渡って機能追加や改版が重ねられ，関連する仕様書が膨大になっていたことを考慮して，モジュール単位 [40] とした．

判別分析に使用するソースコードメトリクスは，その利用と性質に関する研究が多数行われている．メトリクス間の相関の大きさについても言及されており [3][37][48]，判別モデルの構築時に問題となる多重共線性を回避する目的で，計測するメトリクスの種類は少数に絞った．判別分析に使用するモデルは，線形判別分析，ロジスティック回帰分析，主成分分析，ニューラルネットワーク，サポートベクタマシンを試行した．ケーススタディに使用しなかったものを含み比較的判別精度が大きいものを付録に記載した．

判別分析の結果として得られる判別得点が連続値であることを利用して，これを目視評価対象のモジュールの選択優先順位に用いることとした．目視評価対象のモジュールの選択を，判別得点の小 (=fault-prone の度合いの大きい）のモジュールから，判別得点の大 (=fault-prone の度合いの小さい）のモジュールから，を考える．これに判別分析の結果として特に注目されない部分である判別得点が中間値のモジュールから，判別得点が大及び小であるモジュールから，を加えて 4 通りの試行をする．

目視評価が fault-prone モジュールの判別精度の向上に寄与することが判れば，目視評価を行うモジュールの割合と判別精度の関係を評価する．判別モデルによっ

て得られる判別得点は標準化を行い，目視評価点がこれらに与える影響を等しくした．目視評価が判別分析の結果に与える影響の大小と，提案手法による判別精度の向上の関係を評価するために，目視評価得点に重み付けを行った．

目視評価得点の付与について，ソースコードを直接目視して評価を行うとき，評価者には開発言語に関する一般的な知識に加え，対象とするアプリケーションの動作環境などの特有の知識が有る方がより有効な評価が可能と考えられる．目視評価結果が属人性に依存することを避けるため，予め用意したチェックリストと採点基準を設けて評価を行う．目視評価作業を複数人で分担した場合や，単独の評価者で実施する場合も開始時と終了時で評価結果が揺らがないようにする．

目視評価工数を低減させるために，ソースコードをサイズによって大小に分け，異なる観点による二段階の目視評価を行う．直接的な欠陥の存在を確認する観点と全体の流れを追い潜在的な欠陥の有無を確認する観点である．統合開発環境では，クラスの初期化，setter や getter など小サイズのソースコードが自動生成されることが多い．これらは，前者の観点によって記述に問題が無いことが確認できれば以降の評価は必要ない．自動生成ではないが小サイズのモジュールについても直接的な欠陥が存在しないことが確認できれば以降の評価を省略できる．判別精度を維持したままで目視評価工数を削減できるかを試行する．

3. 論文構成

本論文の主要部分は大きく 3 つの章から構成される．

第 2 章では，ソースコードを目視評価することで fault-prone の度合いが改善するかを確認するために，高性能なモデル，シンプルなソースコードメトリクス，ランダムの 3 種の判別モデルと目視評価をそれぞれ組合わせて判別精度を評価する．目視評価を行うモジュールの選択基準として，判別分析の結果 fault-prone の度合いの大きさ順に並べたモジュールのどの部分を優先的に目視評価対象とするかを試行する．また，目視評価を行う対象のモジュール数（割合）を可変にして評価することと，判別分析モデルから得られた fault-prone の度合いと目視評価によって得られた評価点との荷重配分を変化させて判別精度を評価する．

第 3 章では，第 2 章の評価の結果に基づき，目視工数の削減と効率化を考慮し，2 種類に分けた目視観点について，対象とするモジュールの規模によって一方の観点による評価を省略することで，目視評価の工数の削減を図る．

第 4 章では，本研究の関連研究について，ソースコードメトリクスの先行研究，fault-prone モジュール判別の先行研究と本研究との関連について述べる．

第2章　判別モデルと目視評価の組合せによる判別精度の評価

1. はじめに

fault-prone モジュールの識別は，ソースコードの修正や再テストなどの手戻りの発生を考慮するとソフトウェアの品質保証において重要な課題であると言える [20][46][57]．fault-prone モジュール判別に適用するための様々な多変量モデル，例えば線形判別分析，ロジスティック回帰分析，分類木などが提案されている．二つの fault-prone 判別モデルを用いたハイブリッドな手法によるアプローチも提案されている [27][43].

従来手法である fault-prone モジュール判別モデル [15][16][20][32][51] は，入力情報のメトリクスとして機械的に収集できるものを用いている．Catal ら [12] によると，fault-prone モジュール判別モデルに関する文献のうち 84% において，メソッドの凝集度 [8][9]，コード行数，サイクロマチック数 [12] がソースコードメトリクスで構築したモデルの入力値として用いられていた．ソースコードの行数，分岐の数，ループの数を測定することにより得られるソースコードメトリクスは，潜在的な欠陥の兆候として用いることができる．コードスメル（code smell：欠陥を含みそうなソースコードの兆候）[17][38][53] を自動的に検出するアプローチや研究は，潜在的な欠陥の兆候を用いたアプローチに分類される．

しかし，潜在的な欠陥の兆候のすべてがソースコードメトリクスや静的な分析のみで見つかるわけではない．例えば，コピーされた一塊のソースコード中のコメントの更新漏れはソースコードメトリクスから検出することは困難である．ソースコードとコメントの間に不一致や矛盾が有ると，改版時の更新対象から漏れる可能性が発生する．これらの不一致はプログラマを混乱させ，欠陥の発生に

つながる．

目視評価は欠陥のきっかけとなる潜在的な兆候を検出することができる．しかしながら，目視評価には労力が必要である．モジュール数が増えると目視評価に必要な労力も増加する．多くの労力を必要とせずに判別精度を向上させるアプローチが必要である．

この章では fault-prone モジュール判別モデルの結果と目視評価を組合せることにより，判別精度を向上させ，目視評価にかかるコストを減少させることを実証的に評価する．第一に，判別モデルの結果によりモジュールを fault-prone の可能性の大きい順に並べる．次に，モジュールのうち目視評価により有効と思われる順から $\alpha\%$ を選択し，予め作成しておいたチェックリストに基づき目視評価を実施する．すべてのモジュールが判別モデルによって順序付けられ，目視評価の結果を持つ．予め用意したチェックリストは質問事項と対応する採点手順が組となって構成されている．質問事項と採点手順はモジュール毎に独立であり，特定の 1 モジュールに対する目視評価で回答が可能である．

この章では，判別モデルと目視評価の 4 つの組合せを示し，商用アプリケーションによる評価結果を示す．

仮説 1：fault-prone モジュールの判別精度は，判別モデルと目視評価の組合せで改善する．

もし，判別精度が改善された場合，次の仮説を調べるため，$\alpha\%$ のモジュールを調査する．

仮説 2：一定の割合以上のモジュールを目視評価すれば判別精度が向上する．

以降，第 2 節において fault-prone モジュールと目視評価の予備的定義を行う．第 3 節では設定とケーススタディの結果を示し，第 4 節で結果について議論を行う．第 5 節で結論と今後の方策について述べる．

2. 判別モデルと目視評価

2.1 判別モデル

目視評価と組合せる 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 $(i \neq j,\ \ i,j \in \mathcal{N})$ 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（$F(m)$ が小さい方が fault-prone の度合いが大きいと定義していることに注意）．以降，モジュール間の関係を明示するとき以外は簡単のために m の添字は省略する．

本研究では特定の fault-prone 判別モデルを仮定しないが，モジュール $m_1, m_2, \cdots, m_n$ $(n \in \mathcal{N})$ は判別結果によって並べられ得ると仮定する．対象とするアプリケーションに属する n 個のモジュール $(m_1, m_2, \cdots, m_n)$ に対して以下が成り立つとき，モジュール m_1 は最も fault-prone の度合いの大きいモジュールである．

$$F(m_1) \leq F(m_2) \leq \cdots \leq F(m_n).$$

2.2 目視評価

目視評価は，予め決められたチェックリストに従って評価者が実施する．モジュール m の評価は，後述するチェックリスト Q に含まれる質問の集合と各質問事項に対応した得点 $I(m)$ によって与えられる．質問事項や採点手順は，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．判別モデルと目視評価の組合せの効果を高めるため，判別モデルの入力であるソースコードメトリクスから現実的なコストで得られそうな質問事項は使用しないようにする．

チェックリスト Q は l 個の質問事項 $q_i (i = 1, 2, \cdots, l)$ からなるが，これにより潜在的な欠陥の兆候や雰囲気となる可能性のある事項を検出する．各質問事項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} を与える．モジュール m に対する評価得点 $I(m)$ は，

$$I(m) = \sum_{i=1}^{l} s_{im}/l \ \ , \ \ (-1 \leq s_{im} \leq 1)$$

によって得られる．

表 2.1 は，チェックリスト Q において質問事項と採点手順の例を示している．質問事項 q_i は複数のモジュール間の関連についての評価を要求しない．評価者は，特定の 1 モジュールの評価によって質問事項に答えることができる．チェックリストには，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（欠陥を含みそうなソースコードの兆候）により自明な欠陥を直接検出する質問事項を含む．

2.3 モデルと評価の組合せ

fault-prone の度合い $F(m)$ は，判別モデルによって得られ，目視評価すべきモジュールを選択するために用いられる．目視評価は個々のモジュール毎に実施するが，全てのモジュールに対して目視評価を実施しなければならないということは無い．目視評価を行うモジュールの割合を $\alpha\%$ と表記する．n 個のモジュールに対して $\alpha\%$ のモジュールに目視評価を行うとき，目視評価対象のモジュールの個数 k は $(\alpha n/100)$ の床関数とする．即ち，$k = [\alpha n/100]$.

判別モデルによって各モジュールに fault-prone の度合い $F(m)$ が与えられたとする．モジュールが目視評価されたとき，評価得点 $I(m)$ が得られる．目視評価を実施しないモジュールに対しては評価得点 $I(m)$ は 0 である．本研究のアプローチによる fault-prone の度合い $\hat{F}(m)$ は，判別モデルと目視評価点の合計に等しい．すなわち，

$$\hat{F}(m) = F(m) + \beta I(m),$$
$$I(m) = \begin{cases} \sum_{i=1}^{l} s_{im}/l & (m \text{ は目視評価済}) \\ 0 & (m \text{ は目視評価されていない}) \end{cases},$$

表 2.1 目視評価のチェックリストの例

分類	ID	質問項目	採点手順
保守性，理解性	q_1	コードが他のモジュールからコピーまたは再利用されているとき、古いコメントが更新されているか？	-1：アップデートの必要なコメントが1つ以上残っている． 1：全てのコメントが最新版に従って更新されている． 0：更新が必要なコメントが無い．
	q_2	printf や beep を鳴動するようなデバッグコードが残っていないか？	-1：変数の値を参照したり条件分岐を確認する等の内部状態を出力するコードが残っている． 0：デバッグ用のコードが残っていない．
一貫性	q_3	例外のハンドリングがガイドラインや規約に従っているか？	-1：例外のハンドリングがガイドラインや規約に従っていないものが1つ以上ある．例えば catch 句が構造化されていなかったり finally 句が無かったりする． 1：ガイドラインや規約に従っていない例外のハンドリングが無い． 0：例外のハンドリングが存在しない．
	q_4	トラブルシューティングに対してログ記録の設定は適切か？	-1：必要なデバッグ情報や欠陥を検出するために必要なログ記録の粒度が十分で無い． 1：ログ記録がリソースに対して十分な粒度が有る． 0：ログ記録が必要で無い．
実行環境に対する考慮不足	q_5	コード実装はプラットフォームに依存していないか？	-1：メモリのアライメントやビッグ/スモールエンディアン、volatile の漏れ、CPU 固有の割り込み、数値演算のオーバーフローなどの考慮不足． 1：プラットフォーム独立性が十分考慮されている． 0：プラットフォーム独立性が必要な要素が無い．
	q_6	文書化されていなかったり、サポートされていない API やライブラリ呼び出しは無いか？	-1：サポートされていないか，文書化されていない API やライブラリー呼び出しが1つ以上存在する． 1：全ての API やライブラリ呼び出しがサポートされているか，または公式に文書化されている． 0：API やライブラリ呼び出しが存在しない．
...	...	...	...

ここで β は目視評価点の係数である．

判別モデルの選択，これにより得られる fault-prone の度合いの選択順，及び α の値によってどのモジュールを目視評価すべきかが決まる．図 2.1 に判別モデルと目視評価の組合せの概要を示す．評価者は $\alpha\%$ のモジュールを目視評価する．最初に，モジュールは fault-prone 判別モデルにより順序付けられる．次に，評価

者は判別モデルにより fault-prone の度合いの順序に並び替えられたモジュールを次に示す組合せによって目視評価対象として選択する．

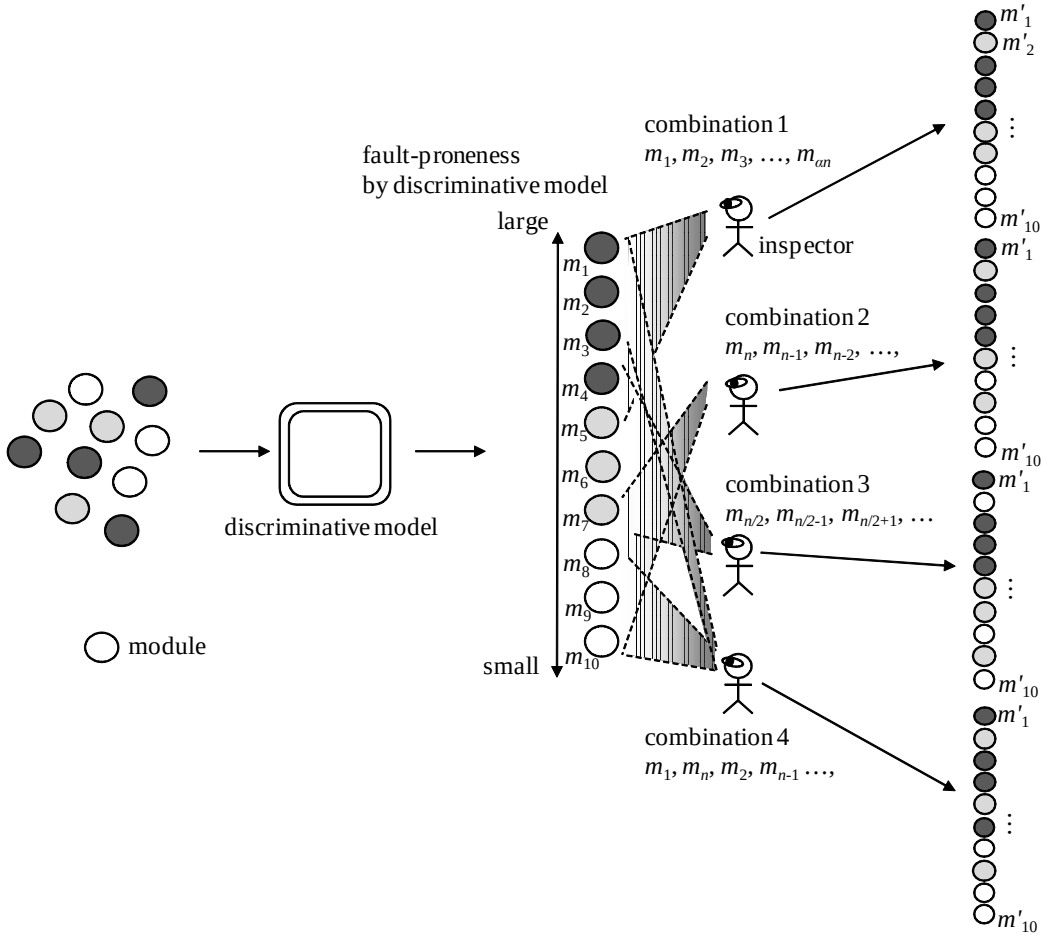

図 2.1 目視評価との組合せによる判別分析手順

組合せ 1 においては，評価者は fault-prone の度合いが大きいモジュールから順に目視評価対象のモジュールをその割合が $\alpha\%$ になるまで選択する．組合せ 2 においては，評価者は fault-prone の度合いが小さいモジュールから順に目視評価対象のモジュールをその割合が $\alpha\%$ になるまで選択する．組合せ 3 においては，評価者は fault-prone の度合いが中程度のモジュールから順に目視評価対象のモジュールをその割合が $\alpha\%$ になるまで選択する．組合せ 4 においては，評価者は

fault-prone の度合いが最大のモジュール，最小のモジュール，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 $\alpha\%$ になるまでのモジュールを目視評価対象として選択する．

目視評価の係数 β は，判別モデルでの検出を期待する欠陥の数と，過去の欠陥のデータや同様のプロジェクトで検出されたように，今回も目視評価によって検出を期待する欠陥の数とのバランスによって決定する．目視評価を行うモジュールの割合は目視評価にかけることが可能なコストによって決定する．

```
/* sort by fault-proneness by prediction model */
function m'[] sort_by_prediction_model(m[]){
   for(i = 1; i<=n; i++){ m[i].fault_proneness = F(m[i]); }
   m'[] = sort (m[],m[].fault_proneness);
   /* m'[1] stores the module with the largest fault-proneness */
      return m'[]
}
m'[] = sort_by_prediction_model(m[]);

/* conducting manual inspection from the largest fault-proneness */
/* by prediction model and adding inspection score */
for ( i = 1; i <=  floor( α * n / 100); i++){
   m'[i].fault_proneness +=  β*I(m'[i]);
}
```

図 2.2 組合せアルゴリズム 1

```
m'[] = sort_by_prediction_model(m[]);

/* conducting manual inspection from the smallest fault-proneness  */
/* by prediction model and adding inspection score */
for ( i = 1; i <=  floor( α * n / 100); i++){
   m'[n-i+1].fault_proneness +=  β*I(m'[n-i+1]);
}
```

図 2.3 組合せアルゴリズム 2

- 組合せ 1：fault-prone の度合いが大きいモジュールから順に目視評価対象のモジュールを選択する．つまり，$m_1, m_2, m_3, \cdots, m_k$，ここで $F(m_1) \leq F(m_2) \leq \cdots \leq F(m_k), k = [\alpha n/100]$. 図 2.2 に組合せ 1 のアルゴリズムを示す.

- 組合せ 2：fault-prone の度合いが小さいモジュールから順に目視評価対象のモジュールを選択する．つまり，$m_n, m_{n-1}, m_{n-2}, \cdots, m_{n-k+1}$，
ここで $F(m_{n-k+1}) \leq F(m_{n-k+2}) \leq \cdots \leq F(n), k = [\alpha n/100]$. 図 2.3 に組合せ 2 のアルゴリズムを示す.

- 組合せ 3：fault-prone の度合いが中程度のモジュールから順に目視評価対象のモジュールを選択する．つまり，$m_j, m_{j-1}, m_{j+1}, m_{j-2}, m_{j+2} \cdots$，ここで $F(m_1) \leq \cdots \leq F(m_{j-2}) \leq F(m_{j-1}) \leq F(m_j) \leq F(m_{j+1}) \leq F(m_{j+2}) \leq \cdots \leq F(m_k), j = [n/2 + 1]$. 図 2.4 に組合せ 3 のアルゴリズムを示す.

- 組合せ 4：fault-prone の度合いが大きいモジュールと小さいモジュールから交互に目視評価対象のモジュールを選択する．つまり，$m_n, m_1, m_{n-1}, m_2, m_{n-2}, \cdots$，
ここで $F(m_1) \leq F(m_2) \leq \cdots \leq F(m_{n-2}) \leq F(m_{n-1}) \leq F(m_n)$. 図 2.5 に組合せ 4 のアルゴリズムを示す.

```
m'[] = sort_by_prediction_model(m[]);

/* conducting manual inspection from the middle of the fault-proneness */
/* by prediction model and adding inspection score */
unsigned_index = 0;
sign = 1;
for ( i = 1; i <=  floor( α * n / 100); i++){
   index = sign * unsigned_index + int(n/2);
   m'[index].fault_proneness += β*I(m'[index]);
   sign = sign * -1;
   if(sign = = -1){ unsigned_index ++; }
}
```

図 2.4 組合せアルゴリズム 3

```
m'[] = sort_by_prediction_model(m[]);

/* conducting manual inspection from the smallest and largest fault-proneness */
/* in rotation and adding inspection score */
unsigned_index = 0;
sign = -1;
for ( i = 1; i <=  floor( α * n / 100); i++){
   if(sign = = -1){
      index = n + sign * unsigned_index;
   } else {
      index = sign * unsigned_index + 1;
   }
   m'[index].fault_proneness += β*I(m'[index]);
   sign = sign * -1;
   if(sign = = -1){ unsigned_index ++; }
}
```

図 2.5 組合せアルゴリズム 4

本研究のアプローチによる fault-prone の度合い $\hat{F}(m)$ 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 の順序はテストを行う順序である．

3. ケーススタディ

3.1 概要

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．表 2.2 に該当アプリケーションのソースコードの概要を示す．

表 2.2 ケーススタディにおけるソースコードの概要

目的	通信システムの監視と制御
保守期間	10 年以上
言語	C，C++
コード規模	15,100
モジュール数	125

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目視評価のチェックリストを決定した．その詳細は 3.3 節にて示す．

判別と評価用にモジュールとして関数，メソッドを選択した．以下の 3 つのモデル，即ち，fault-prone 判別モデル，単独のソースコードメトリクス，ランダムな順序，を判別モデルとして比較する：

(A) SVM ： サポートベクタマシン [6][10][62] 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 [26][54]．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 [52], 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 [11]．このケーススタディでは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を SVM に与えた．

(B) LOC ： コード行数はソースコードメトリクスの 1 つである．サイクロマチック数と LOC は fault-prone 判別のメトリクスとして最も正確である [2][55]．LOC を選択した理由は，本研究のアプローチを使用する場合，開発者への説明がサイクロマチック数と比較して容易であることによる．

(C) ランダム順 ： ランダムな値による順序数とする．

目視評価の係数 β の評価のため，3 つのモデルから得られた判別得点，行数，順序数を標準化しモジュール $F(m)$ の fault-prone の度合いとする．モジュール m のサポートベクタマシンによって得られた判別得点を $SVM(m)$，同様に行数を $LOC(m)$，ランダム順序数を $RND(m)$ とすると，3 つのモデルに対して標準化した fault-prone の度合いは以下の通りとなる．

$$
\begin{aligned}
F(m) &= (SVM(m) - E(SVM))/\sigma(SVM) \\
F(m) &= (LOC(m) - E(LOC))/\sigma(LOC) \\
F(m) &= (RND(m) - E(RND))/\sigma(RND)
\end{aligned}
$$

ここで，$E()$ はそれぞれ対象とするモジュールにおける平均，$\sigma()$ は同じく標準偏差とする．

3.2 目視評価のチェックリスト

表 2.3 は，質問項目と採点手順のチェックリストである．10 年間システムを保守してきた熟練のシステム開発者が既存のチェックリストをカスタマイズして質問項目を作成した．質問作成者は質問事項に関してこのケーススタディに合わせた採点手順を定義した．システムの背景として，このシステムはずっと保守され続けており，将来に渡っても保守が予定されている．質問事項や採点手順を決定するにあたり，システムの特徴や開発システムについての考慮の結果，以下の方針を決定した．

- 将来の発展：チェックリストの質問事項は既存のチェックリストの質問事項を基に決定しているが，保守性と将来への適応性について問う内容，例えばソースコードの理解性や条件分岐レベルの抽象度，構造化例外ハンドリングなどとしている．

- プラットフォーム独立の実装：チェックリストの質問事項は，既存のチェックリストを基に，プロセッサ依存の実装，OS 固有の実装，ログ記録の粒度，

表 2.3 ケーススタディにおける質問項目チェックリスト

分類	ID	質問項目	採点手順
保守性，理解性，可変性	q_1	分岐部，ループ部，コードブロック部に充分なコメントが記載されているか？	-1：分岐部やループ部，コードブロックに将来必要なコメントが無い． 1：理由や目的を説明するコメントが分岐部，ループ部，コードブロックに存在する． 0：説明を必要とする分岐やループやコードブロックが存在しない．
	q_2	switch 構文に集約可能な条件付き分岐が存在するか？	-1：将来の拡張や他のコードブロックとの一貫性を保つための強い要望により放置されている switch 構文に置き換え可能な条件付分岐のグループが 1 つ以上存在する． 0：置き換え可能な条件付分岐が無い．
	q_3	条件式が複雑過ぎるとき，その表記を関数として定義するか？	-1：1 つ以上の複雑な条件式が再利用可能な関数として抽象化されていない．4 つかそれ以上の結合した条件が存在する． 1：複雑な条件式が適度に抽象化され，再利用可能な関数として定義されている．4 つかそれ以上の結合した条件式が存在する． 0：複雑な条件式が存在しない．
	q_4	条件分岐の記載（then else 句が if の分岐中に有る）が長過ぎたり複雑すぎたりしているとき，これを適当な関数宣言に置き換えているか？	-1：条件分岐宣言中の 1 つ以上のコードブロックが再利用可能な関数としての抽象化がなされていない．10 行かそれ以上のコードブロックが存在し，3 つかそれ以上の分岐が存在する． 1：長かったり複雑な条件分岐行が適当に抽象化され，関数として定義されている．10 行より小さい行で 3 つより小さい分岐が存在する． 0：条件分岐が存在しない，または，長い複雑なコードが分岐部に存在しない．
	q_5	コードブロックが再利用されるとき，または他のモジュールからコピーされたとき，古いコメントが更新されるか？	-1： 1 つ以上の更新すべきコメントがそのままになっている． 1：全てのコメントが最新の更新に基づいて更新されている． 0：コメントを更新すべき変更が必要とされていない．
	q_6	ソースコードが設計時のロジックより複雑になっているか？	-1：関連する設計文書と比較して不必要な行、条件付分岐、またはループが存在する． 0：設計文書上の関連する記述において必要かつ充分な実装がなされている．
	q_7	不必要な条件付分岐が可読性を損ねていないか？	-1：冗長な条件付分岐が存在し，理解性を低下させている． 1：条件表現が適度に統合されシンプルさを保っている． 0：統合すべき条件分岐が存在しない．
実行環境における考慮	q_8	時間待ちのようなプラットフォーム依存の実装が存在するか？	-1：1 つ以上のプラットフォーム依存の実装が存在する． 0：実装に依存するプラットフォームや CPU が存在しない．
	q_9	文書化されていなかったりサポートされていない API やライブラリ呼び出しが使われているか？	-1：将来，または OS のバージョンアップによって妥当性を欠くような，サポートされていなかったり文書化されていない API やライブラリ呼び出しが 1 つ以上存在する． 1：すべての API やライブラリ呼び出しがサポートされているか公式に文書化されている． 0：API やライブラリ呼び出しが存在しない．
	q_{10}	プラットフォーム依存の方法で変数の初期化がなされているか？	-1：1 つ以上の変数がプラットフォーム依存の方法で初期化されている． 0：全ての変数が明確に初期化されている．
例外ハンドリングとデバッグ	q_{11}	例外のハンドリングが構造化されているか？	-1：構造化されていない例外ハンドリングが 1 つ以上存在する． 1：将来の拡張に備えて全ての例外が適切にハンドリングされている． 0：例外ハンドリングが存在しない．
	q_{12}	finally 句が必要な箇所で定義されているか？	-1：finally が必要な箇所で定義されていない箇所が 1 つ以上存在する． 1：全ての finally 句が適切に定義されている． 0：finally 句が必要とされていない．
	q_{13}	printf や beep 行のようなデバッグコードが残っているか？	-1：1 つ以上のデバッグ用コードが残っている． 0：デバッグ用コードが残っていない．

サポートされていない，または文書化されていない API の利用についての質問を設定している．

3.3 手順

ケーススタディにおいて，委託元はコーディング，単体テスト，組合せテスト，一部のシステムテストを外部委託する．判別結果は委託元が実施する受入れ検査にて優先順位付けされたモジュールに対して用いられる．評価指標は Alberg diagram[48] の AUC（Area Under the Curve : 曲線下面積）を用いる．ケーススタディにおいて，判別結果は受入れ検査における検査対象のモジュールの優先付けに使用されるが，これは初期の欠陥検出では毎回の不具合修正においてその場しのぎの修正や大規模な回帰テストを避けるためである．

交差検証法の実施に先立ち，モジュール群を 2 つのグループに分けるためにモジュールを規模順に並べ奇数番目と偶数番目のグループとする．奇数番号のグループはデータセット X とする．偶数番号のグループはデータセット 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 モジュールに欠陥が有った．データセット Y には 8 モジュールに欠陥が有った．

(1) 質問事項と採点手順を決める（表 2.3 を参照）．

(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．

$M_1, M_2, M_3, \cdots, M_{125}$．ここで，$size(M_k) \leq size(M_{k+1})$,

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

$Y = \{M_2, M_4, \cdots, M_{2l}, \cdots, M_{124}\}$, $\quad l = (1, 2, \cdots, 62)$.

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

(3) 判別モデル (A)，(B)，(C) 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 $X_A, X_B, X_C, Y_A, Y_B, Y_C$ を得る．$X_A = (m_{A1}, m_{A2}, \cdots, m_{A63})$ は判別モデル (A) 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，fault-prone の度合い $\hat{F}(m)$ を得る．

(5) すべての組合せについて，AUC を算出する．

目視評価はそのシステムやソースコードのエキスパートである者が実施する．モデル (A) の fault-prone の度合いは，統計解析ソフト R によりガウシアンカーネルを用いたサポートベクタマシンにより算出した．モデル (C) の fault-prone の度合いは，統計解析ソフト R によりメルセンヌツイスタ乱数を生成させて得た．

3.4 結果

判別の正確性

表 2.5 に最大の AUC とパラメータを示す．表において，第 1 列は 2.3 節の目視評価対象の選択組合せと，3.1 節の判別モデルを示す．AUC が最大となるときの α と β の値はモデル間の組合せによって異なる．目視評価を行うモジュールの割合 ($\alpha\%$) は 1-A のデータセット X を除いていずれも 70% 以上である．

表 2.5 は目視評価を含まない判別モデルによる AUC 値を含む．表中，“A(without inspection)”，“B(without inspection)”，“C(without inspection)”の各行である．結果において，目視評価と判別モデルの全ての組合せにおいて AUC が増加している．AUC が大きな値をとる組合せと α の値は，4-B-X のとき 0.871 で α は 70%，2-A-Y のとき 0.863 で α は 95%，1-A-X のとき 0.860 で α は 40% であった．α の値の小ささは 1-A-X が突出している．最も AUC が増加した組合せは判別モデルによって異なる．組合せ 1 では，目視評価無と比較して 1-C において AUC が 0.340 と 0.325 増加した．A による組合せでは最大で 0.09(2-A-Y の場合) 増加している．組合せ 4 では AUC は最大で 0.243 と 0.325 増加した (4-C)．B の LOC による組合

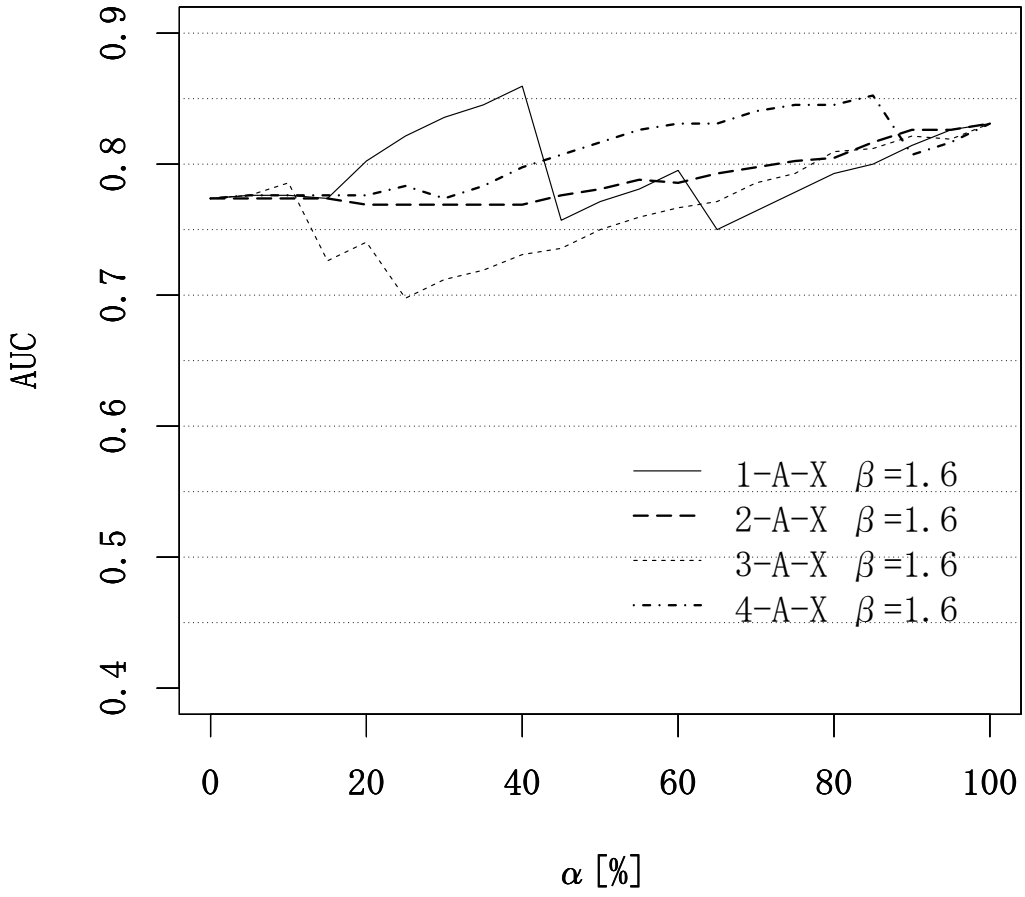

図 2.6 サポートベクタマシンと目視評価の組合せ データセット X

せでは 0.102 かそれ以上の増加が有った．C のランダムモデルにおいても AUC は 0.243 以上増加した．

パラメータ β も判別モデルによって異なる．1-A と 4-A ではデータセット X と Y の両方で β が 1.6 のときなどで AUC が増加した．2-A と 4-B においてデータセット X で β が 2.0 以下のときで AUC は増加した．一方で，1-B，2-B，2-C，3-B において β が 5.9 のときなど大きな値で AUC が増加した．

提案手法によって得られた AUC が目視評価を行わない場合と比較して有意に大きいかを確認した．目視評価を行わない，判別モデルのみの場合と提案手法のそれぞれの組合せとで，得られたモジュールの優先順位に対して，欠陥を含むモ

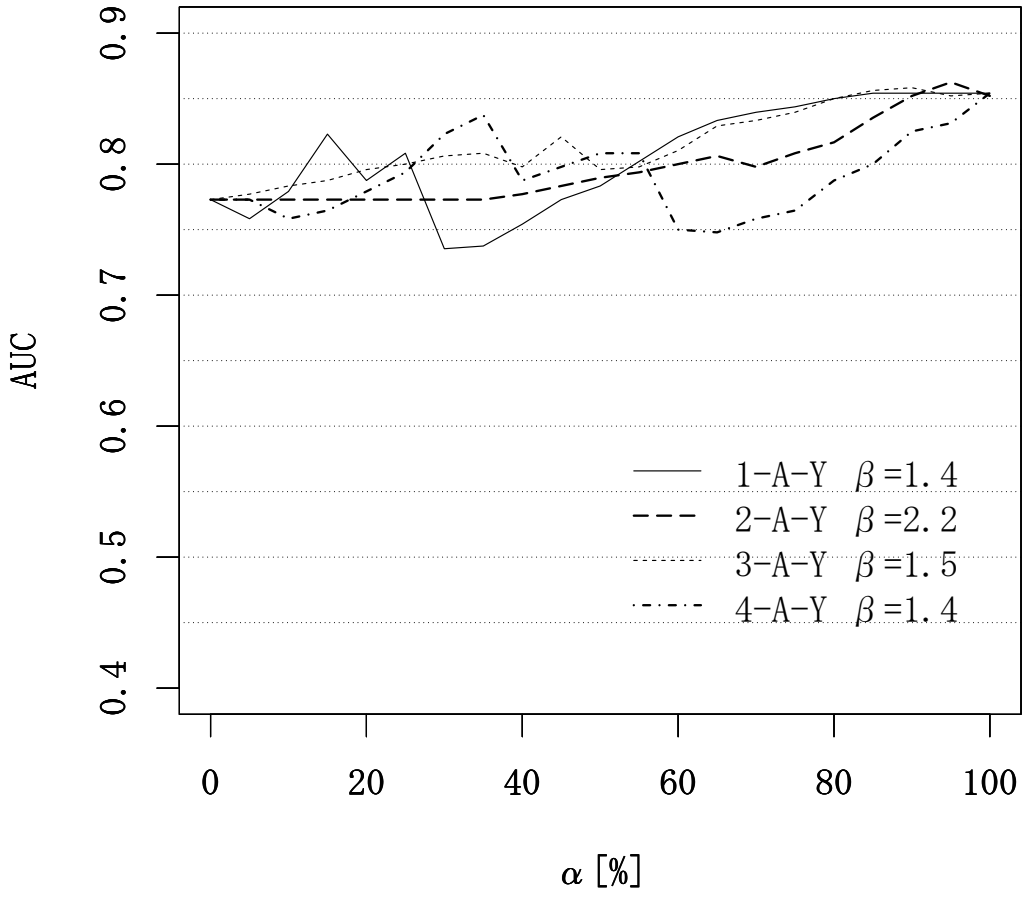

図 2.7 サポートベクタマシンと目視評価の組合せ データセット Y

ジュールの序数を抽出した数列を考える．判別モデルのみの場合の数列とそれぞれの組合せのそれとで有意水準 5% 及び 1% の片側検定による 2 標本 t 検定を実施した．表 2.5 の最終列に結果の有意水準を示す．いずれの組合せにおいても判別モデルと目視評価を組合せたときの AUC が目視評価を行わないときの AUC よりも有意に大きい．

図 2.6，2.7，2.8，2.9，2.10，2.11 に，AUC と目視評価のモジュールの割合の関係を示す．グラフの横軸は目視評価の割合 α% である．縦軸は AUC を示す．各グラフは同じモデルで異なった組合せを示している．パラメータ β の値は表 2.5 で説明したもので，各組合せにおいて AUC が最大になるときの値である．例え

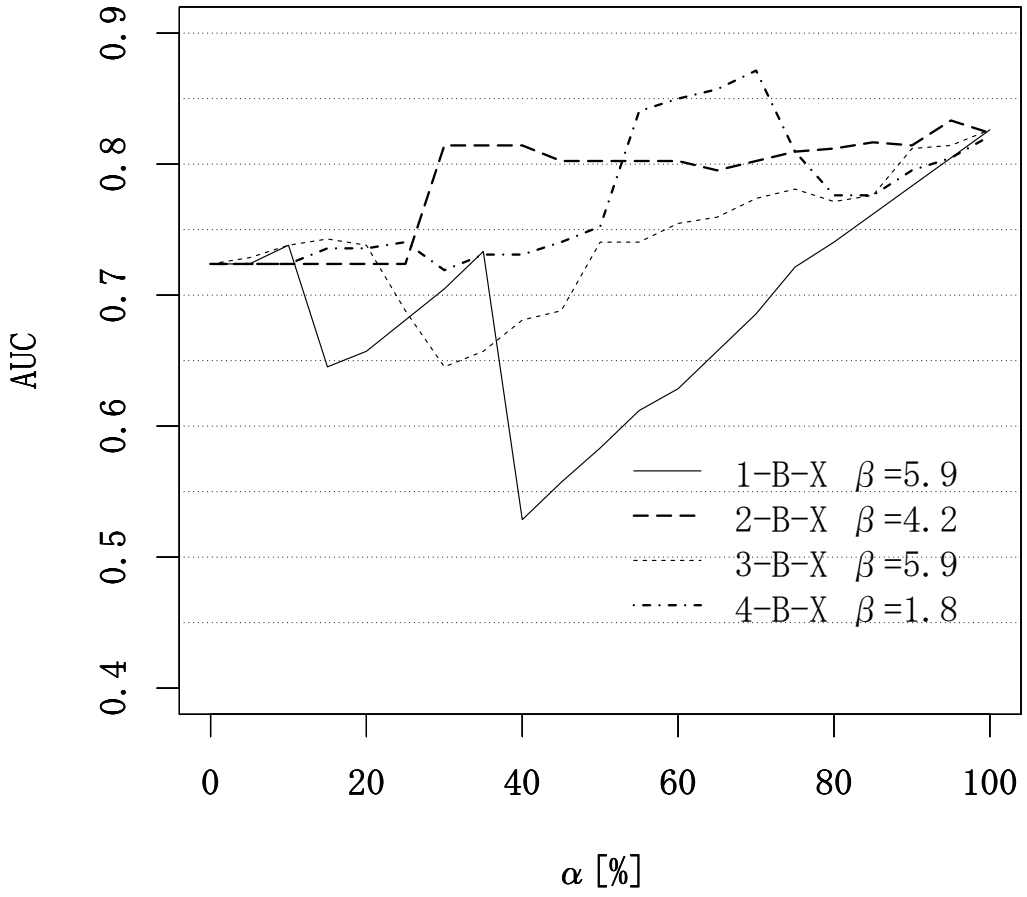

図 2.8 LOC と目視評価の組合せ データセット X

ば，図 2.6，2.7 はサポートベクタマシンと目視評価の 4 つの組合せの結果を示す．図 2.6 において，1-A-X の AUC が最大となるのは 0.860 であり，1-A における β の値は 1.6 である．ここで，表 2.5 において $\beta = 1.6$，$\alpha = 40\%$ である．

殆どの組合せにおいて，目視評価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 の増加に対して単調増加するとは必ずしも言えない．図 2.6 と図 2.7 において，2-A の AUC はデータセット X では α が 15% と 20% の間を除き、データセット Y では α が 65% と 70% の間を除き単調増加している．図 2.8 と図 2.9 において，2-B の AUC はデータセット X では 40% と 45%，60% と 65%，95% と 100% の間を除き AUC が単調増加し，デー

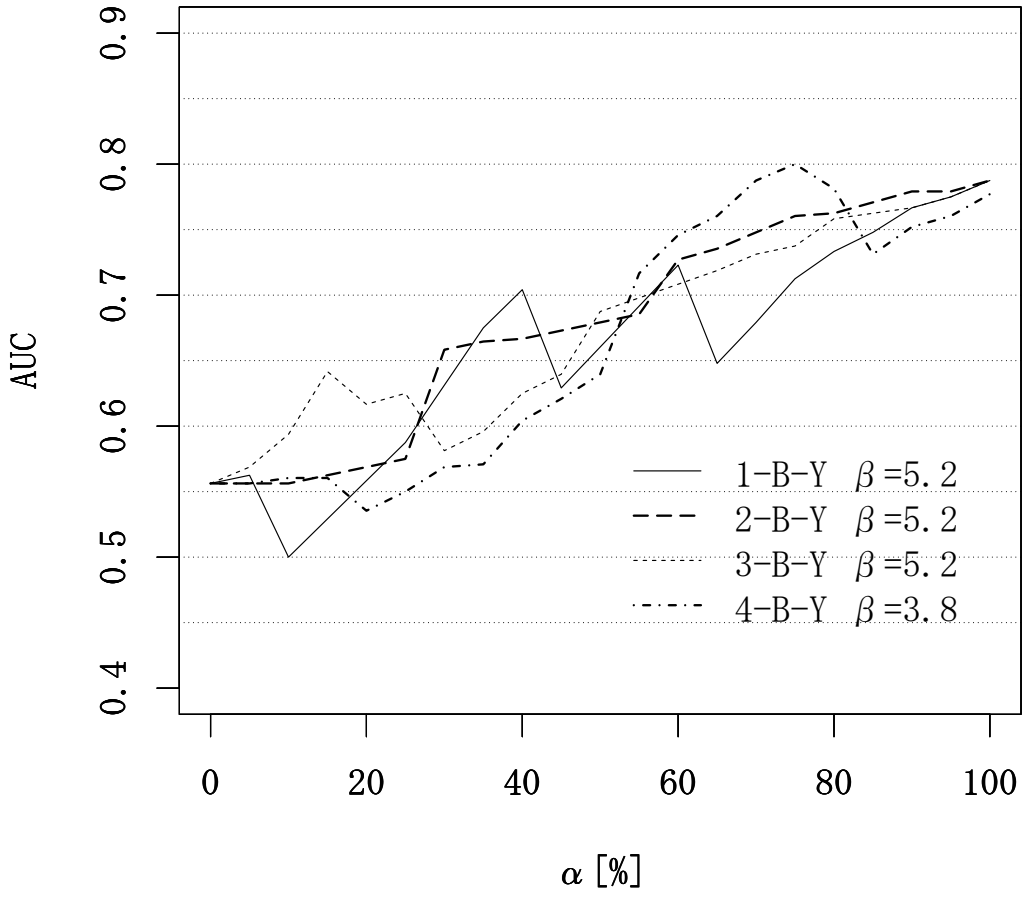

図 2.9 LOC と目視評価の組合せ データセット Y

タセット Y では α の全域において AUC が単調増加している．図 2.10 において，3-C-X では α が 90% 以下の範囲では AUC が単調増加している．図 2.11 において，2-C-Y では α が 50% と 55%，70% と 75%，95% と 100% の間を除き，AUC が単調増加している．

判別モデル単体で判別精度が大きい SVM の場合は，2-A で示すように AUC が α に対して他の判別モデルと比較して単調増加に近い傾向を示す．一方で，判別モデルとしての判別精度が小さい場合は，LOC の 1-B，ランダムの 1-C の AUC は α の値に対する増減傾向は単調増加からはかけ離れた状況を示している．

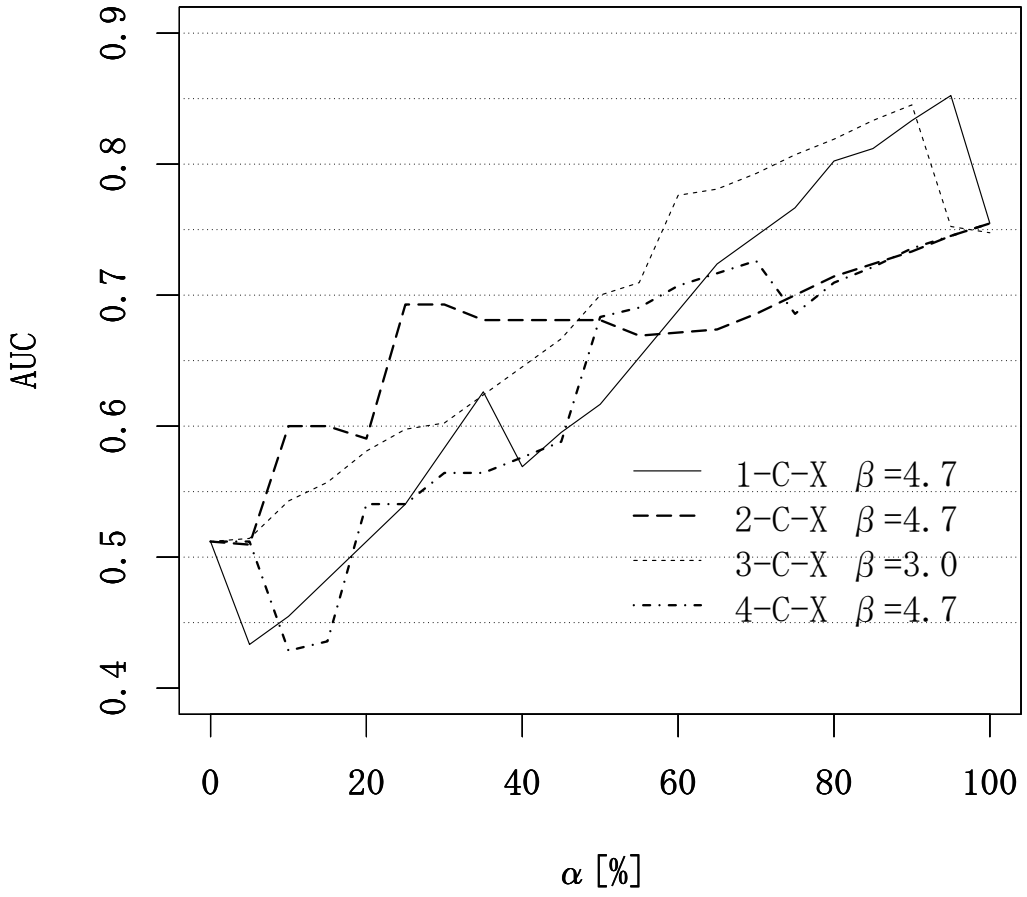

図 2.10 ランダムと目視評価の組合せ データセット X

目視評価

表 2.4 は，目視評価の結果を示しているが，得点が 0 未満のもののみを含む．欠陥を含むモジュールの数を真陽性の数，欠陥を含まないモジュールの数を偽陽性の数として示す．1 つ以上の質問が 1 つのモジュールに適用される．質問 q_5 と q_{11} は 3 つの真陽性を含み，偽陽性は含まないこと，質問 q_2 は真陽性を含まず 9 つの偽陽性を含むこと，が結果から示された．

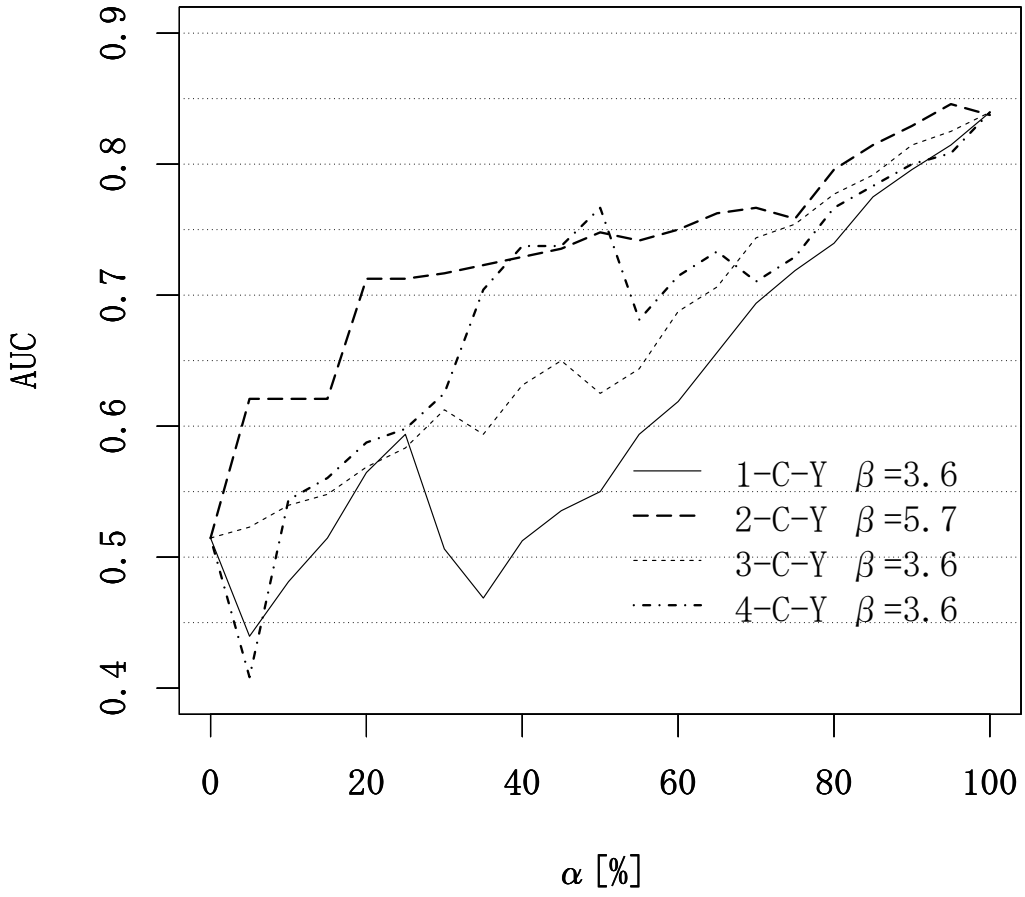

図 2.11 ランダムと目視評価の組合せ データセット Y

表 2.4 目視評価の結果

質問事項	真陽性	偽陽性
q_1	3	3
q_2	0	9
q_5	3	0
q_6	9	9
q_9	6	3
q_{10}	8	22
q_{11}	3	0

表 2.5 最大の AUC とパラメータ

combination and model	dataset	β	$\alpha(\%)$	AUC	significance level
1-A	X	1.6,1.9,2.0	40	0.860	< 0.05
	Y	1.4,1.5,1.6	85,90,95,100	0.854	< 0.01
1-B	X	5.9,6.0	100	0.826	< 0.05
	Y	5.2–6.0	100	0.788	< 0.01
1-C	X	4.7–4.9,5.9,6.0	95	0.852	< 0.05
	Y	3.6–4.0,4.2–4.6	100	0.840	< 0.01
2-A	X	1.6,1.7	100	0.831	< 0.05
	Y	2.2–3.1	95	0.863	< 0.01
2-B	X	4.2–6.0	95	0.833	< 0.01
	Y	5.2–6.0	100	0.788	< 0.01
2-C	X	4.7–4.9,5.9,6.0	100	0.755	< 0.05
	Y	5.7–6.0	95	0.846	< 0.01
3-A	X	1.6,1.7	100	0.831	< 0.05
	Y	1.5–1.8	90	0.858	< 0.01
3-B	X	5.9,6.0	100	0.826	< 0.05
	Y	5.2–6.0	100	0.788	< 0.01
3-C	X	3.0-3.2	90	0.845	< 0.05
	Y	3.6–4.0,4.2,4.4,4.6	100	0.840	< 0.01
4-A	X	1.6-2.0	85	0.852	< 0.05
	Y	1.4–1.6,2.6–2.9	100	0.854	< 0.01
4-B	X	1.8	70	0.871	< 0.05
	Y	3.8-4.0	75	0.800	< 0.01
4-C	X	4.7–4.9,5.9,6.0	100	0.755	< 0.05
	Y	3.6–4.0,4.2,4.4,4.6	100	0.840	< 0.01
A(without insection)	X	—	—	0.774	—
	Y	—	—	0.773	—
B(without insection)	X	—	—	0.724	—
	Y	—	—	0.556	—
C(without insection)	X	—	—	0.512	—
	Y	—	—	0.515	—

4. 考察

4.1 判別モデルと目視評価の組合せ

仮説 1 「fault-prone モジュールの判別精度は，判別モデルと目視評価の組合せで改善する.」 の結果は，適切なパラメータ α と 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 AUC が 0.057 かそれ以上向上することが判った．3 つの判別モデルと組合せ 1〜4 において AUC が最大となるのは，判別モデルが LOC で組合せ 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 で組合せ 1 のとき 0.860，このとき α は 40% である．

結果より，判別モデル単体の判別精度の大きさと目視評価点を組合せて得られた fault-prone の度合いは相補的であること，AUC は増加傾向があることが解る．判別モデルが SVM のとき，目視評価を行っていない状態で AUC は既に 0.77 以上であるが，目視評価を行うことによって AUC は増加している．

本研究のアプローチにおいて，判別モデルに対する要求については更なるケーススタディと考察が必要である．ケーススタディにおいて，判別モデルがソースコードメトリクス (LOC) のとき AUC が増加した．もし提案手法において SVM と LOC で AUC が同等であれば，LOC はモデル構築が不要であるため実施コストが小さくて済む．

目視評価において偽陽性となる質問事項を減らすように選択することで，判別の正確性を増加させることが期待できる．ケーススタディでは，質問事項は 10 年間に渡る保守において検出された欠陥やその因果分析に基づいて維持されているチェックリストから選択されている．しかしながら，質問 q_{10} は目視評価において 8 個の真陽性と 22 個の偽陽性が有り，q_{11} は真陽性は 3 つで偽陽性は無い．提案手法においては，一般的なチェックリストからのテーラリングについては更なる議論が必要である．

4.2 目視評価を行うモジュールの数

仮説 2 「一定の割合以上のモジュールを目視評価すれば判別精度が向上する.」の結果は，モデルと目視評価を行うモジュールの選択順位との組合せで共通となる割合は見つからなかった．目視評価を行うモジュール数の決定のための手順については，更なる議論とケーススタディが必要である．しかしながら，ケーススタディにおいて，2-B-Y は目視評価するモジュール数の割合に合わせて AUC が単調に増加した．2-B-Y のように AUC が単調増加する場合には，利用できるコストに応じて目標とする α を決定できるので，開発者は予め何個のモジュールを目視評価しなければならないかを決定しなくて済む.

全てのモジュールについて目視評価を実施するのに必要な時間は約 41 時間であった．ケーススタディに関する開発者との議論で，以下の意見を得た.

- fault-prone の度合いによってモジュールを優先順位付けすることで，その場しのぎの改修や将来の改善への労力を軽減することができる.
- 早い段階での欠陥の検出は，2 つまたはそれ以上の改修が同じ回帰テストのテストケース群を共有する可能性があるため回帰テスト群の集約につながる可能性がある.

目視評価における偽陽性をより少なくするような質問項目の選択によって，AUC は α により単調増加するかもしれない．ケーススタディにおいて，質問事項 q_5 と q_{11} は目視評価では偽陽性は発生しなかった.

目視評価得点は，提案手法による falut-prone の度合いの評価の他に，判別モデルの構築にも用いることが可能である．しかし，目視評価得点を判別モデルの構築のためのパラメータとして与えるためには，判別モデルの構築の前にすべてのモジュールに目視評価得点を付与する必要がある.

4.3 妥当性

ケーススタディに適用したソフトウェアは注意深く選択したものだが，ソフトウェアを代表する共通的なものではない．本研究のアプローチを一般化するには

更なるケーススタディと議論が必要である．ケーススタディにて行った開発者との議論では，ソースコードメトリクスでは捕らえることが困難な潜在的な欠陥の兆候を目視評価によって捕らえることを期待している．適当なチェックリストによる質問項目，判別モデルと目視評価の組合せ，判別モデルはソフトウェアやテストフェーズに合わせて検討する必要がある．

目視評価による得点は，個人の能力に依存する可能性があるが，ケーススタディの採点手順はソフトウェアの 10 年以上に渡る保守によるチェックリストの質問項目に基づいている．偽陽性を減らしたり，保守や改良を通して検出した欠陥に基づいた兆候を捕らえるためのより確実な手順を確立するには更なる議論が必要である．

ケーススタディにおける目視評価の質問事項 q_9 は，サポートされていない API の全リストを定義したり静的解析ツールに定義を加えることで自動化できる可能性がある．ケーススタディでは，目視評価の方がリストの定義よりも低コストで実施できると考え，目視評価を実施した．評価対象に応じて目視評価を行うか自動化するかの選択は，コストや対象とするアプリケーションによって決定することができる．

5. まとめ

本章では，既存の fault-prone 判別モデルに目視評価を組合せることによって，fault-prone モジュールの判別精度が向上するか実際のソースコードを用いて評価した．目視評価には質問と採点手順で構成されるチェックリストを用いた．目視評価は実施コストを要するため，目視評価対象のモジュールが一部であっても判別精度が大きくなるかを調べた．

評価では，対象となる全てのモジュールは fault-prone 判別モデルによって fault-prone の度合いを点数付けした．次に，4 種類の方法で目視対象とするモジュールを選んだ．選ぶモジュールの割合は $\alpha\%$であり，目視評価の結果得られた得点を β 倍した上で fault-prone 判別モデルによる fault-prone の度合いと合算する．モジュールの選択方法は，(1) 最も fault-prone の度合いが大きいモジュールから

$\alpha\%$，(2) 最も fault-prone の度合いが小さいモジュールから $\alpha\%$，(3)fault-prone の度合いが中間のモジュールから $\alpha\%$，(4)fault-prone の度合いが最大，最小となるモジュールから交互に $\alpha\%$，である．

10 年以上に渡って保守されてきた通信システムを監視制御する商用アプリケーションを用いて，ソースコードと受入れ検査において検出された欠陥データを用いてケーススタディを実施した．(A) サポートベクタマシン，(B) コード行数，(C) ランダム順，の判別モデルとこの 4 つの組合せによって Alberg diagram の AUC を比較した．その結果，全てのモデルと適切なパラメータ α と β によって AUC が増加した．全ての組合せにおいて AUC が最大となったのは判別モデルがコード行数の場合であり，α が 70% のときであった．次点であるサポートベクタマシンでは α は 40% で最大の AUC が得られた．判別モデルの判別精度が大きい場合，目視対象となるモジュールの割合をより小さくできる可能性がある．

判別モデルをコード行数，サポートベクタマシンとし，目視対象のモジュールの選択方法を (2)fault-prone の度合いが小さいモジュールから $\alpha\%$とした場合に，α の増加に対して，AUC が単調増加すると見なせる結果が得られた．AUC が単調増加すれば，目視評価を行うモジュールの数を事前に決定する必要が無い為，目視評価に利用できるリソースに応じて可能な限り多くのモジュールを評価することができる．

ケーススタディにより判別モデルと目視評価を組合わせることによって，判別精度が大きくなることがわかった．今後の課題は目視コストの低減である．ケーススタディにおいて 40%の目視評価で 2 番目に大きな判別精度が得られた．この組合せを第 3 章で検討する．また，チェックリストの質問の体系的な定義も今後の課題である．偽陽性が少ない質問項目を提供することにより，AUC はより大きくなり，AUC を単調増加させることが可能となる．

第3章　判別モデルと目視評価を組合せたスコアリング手法

1. はじめに

第2章のケーススタディにおいて，判別分析で得たfault-proneの度合いの大きいモジュールから順に目視評価を行うことで目視評価対象のモジュールの割合αを40%と抑えながらfault-proneモジュールの判別精度が向上した．本章では，目視評価対象のモジュールの選択方法はこの結果を反映し，目視評価コストをより低減する方法を試行する．

ソースコードの実装プロセスにおいて，ソースコードの単位として例えばモジュールを実装し単体試験に先立ってソースコードレビューを実施する場合がある．特に，複雑だったり諸条件を加味する必要が有る処理を含むような場合においてソースコードレビューは重要なプロセスである．レビューは，出力したソースコードの他に関連する詳細設計書，過去の類似システムの開発時に蓄積したレビュー観点を記載したチェックリスト，ユースケースやテストケースを記載した資料等を用いる [58].

第2章で実施したソースコードの目視評価は，予め用意したチェックリストに記載された質問事項に沿ってソースコードを確認するが，目視評価を行う観点は静的解析ツールや検索ツールのような機械的な処理で実現できない事項である．例えば，ソースコードのコメント部に記載された改定内容の日付がそのモジュールの生成日より明らかに古い場合や，コメントの署名者が既に設計チームに居ない場合，ソースコードの整形規則が新規に作成したものと異なる場合なども，そのコードが十分な注意無しにコピーされてきたことの痕跡であったり，検討や確認が不十分なまま流用されたものである可能性がある．

前章のケーススタディで実施したソースコードの目視評価作業は，チェックリストの全ての質問項目に対する回答を行わないと評価が完了しない．一方，コード全体を一目で見渡すことができるようなサイズの小さなモジュールは非該当となる質問項目は相対的に多いと考えられる．また，このような小サイズのモジュールは処理内容の把握が比較的容易であり，潜在的な不具合を見つけるよりも直接的な欠陥を見つけるような評価方法がより適していると考えられる．

本章では，目視評価を行うモジュールに適用する観点を 2 種類に分け，目視評価にかかるコストの低減を図る．

以降，2 節で目視評価手順について述べ，3 節で商用ソフトウェアを対象としたケーススタディについて述べる．4 節で考察を行い，5 節で本章をまとめる．

2. スコアリング手法

2.1 判別モデルと目視評価の組合せ

組合せ方法について，目視評価対象の選択方法は，第 2 章のケーススタディで小さい α で大きな AUC が得られた組合せ 1(fault-prone の度合いが大きいモジュールから順に $\alpha\%$ を目視評価の対象とする) を用いる．目視評価得点の影響をより大きくする目的で，目視評価対象のモジュールは目視評価得点順に並び替えることとした．非対象のモジュール，及び目視評価得点がタイスコアのモジュールについては判別得点順（組合せ 1 と同じ）とした．

2.2 目視評価の方法

第 2 章で述べた方法で目視評価の対象となるモジュールを選択する．目視評価は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

表 3.1 目視評価の観点 (1)

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.

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判別しようとしており，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得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得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 3.1 に，観点 (2) の例を表 3.2 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 $(m_1^t, m_2^t, \cdots, m_k^t)$ 全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

表 3.2 目視評価の観点 (2)

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.

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) は全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 3.1 に示す．図 3.1 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．

表 3.3 目視評価の観点 (1)

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 条件は有るが 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.

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) は全ての質問項目 $\{q_1^2, q_2^2, \cdots, q_{n_2}^2\}$ について目視評価を実施し，モジュール毎に “問題なし”，“問題あり”，“該当なし” を判定し，それらの数及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 $(j = 1, 2, \cdots, k,\ \ k = [\alpha n/100])$ 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

表 3.4 目視評価の観点 (2)

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無いか.
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無いか.
q_{15}^2			1.0	必要な部分で volatile の付加漏れが無いか.
q_{16}^2			1.0	必要でない割り込みの利用が無いか.
q_{17}^2			1.0	演算結果がオーバーフローする可能性が無いか.
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.

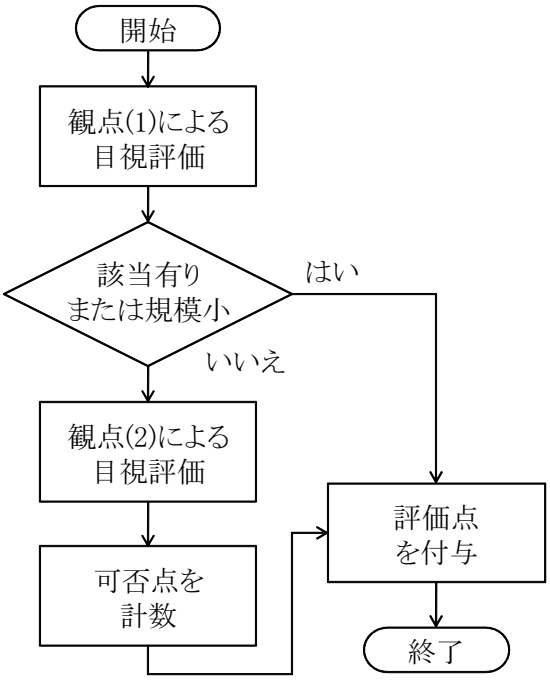

図 3.1 目視評価の手順

ル m_j の目視評価点 $I(m_j)$ を求める．

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

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，目視評価を組合せて評価する．それぞれのカテゴリにおいて，そもそもランク付け精度の向上があるか，向上する場合，どのカテゴリとの組合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 Alberg diagram の AUC(Area Under the Curve)[48] の値を用いる．

3.1 準備

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 Y と奇数番目の群 X 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-prone の度合い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 A．$F(m)$ として判別モデル (サポートベクタマシン) の判別結果を用いた結果
- 試行 B．$F(m)$ として対象モジュールの規模を用いた結果
- 試行 C．$F(m)$ をランダムとしたときの結果

試行 A では，文献 [21][36] で比較されている判別精度の大きい 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大きいとされ，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大きさが優先されるからである．試行 B 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 [2][55] 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 C 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%(目視無し), 25%，50%，75%，100% の 5 通りとした．ランク付けの評価には Alberg diagram [48] ，及び，AUC を用いる．Alberg diagram は，fault-prone モジュールの判別精度を可視化することを目的とした記法であり，判別手法による fault-prone の判別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，横軸を判別手法による 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程，判別精度が大きいことを示す．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 AUC は大きな値となる．

Alberg diagram，及び，AUC を用いた fault-prone モジュール判別の評価は，テストのためのリソースが不足し，全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 [24][36].

対象ソフトウェア

対象ソフトウェアは，第 2 章で述べたものと同じものを用いた.

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踏まえ，以下に示す方針で表 3.3(観点 (1)), 表 3.4(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.

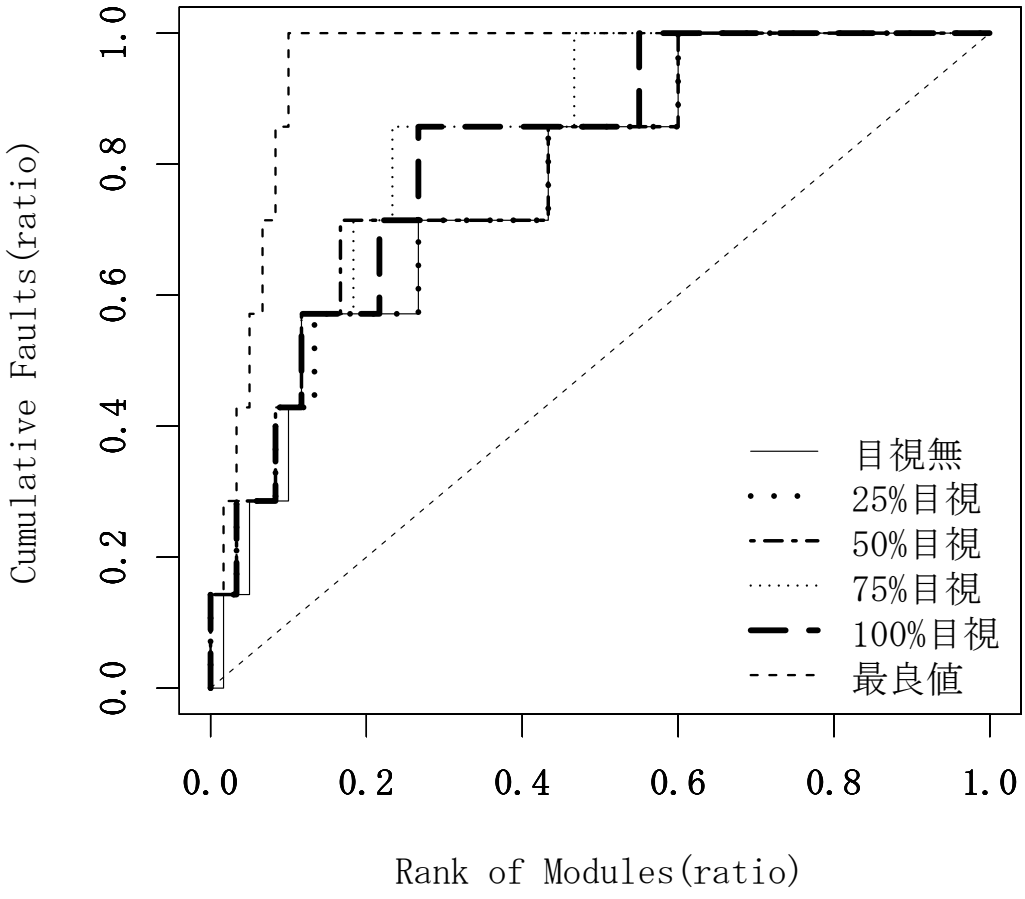

図 3.2 試行 A-X サポートベクタマシン，予測データ: モジュール群 X

- 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，スクロール無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．

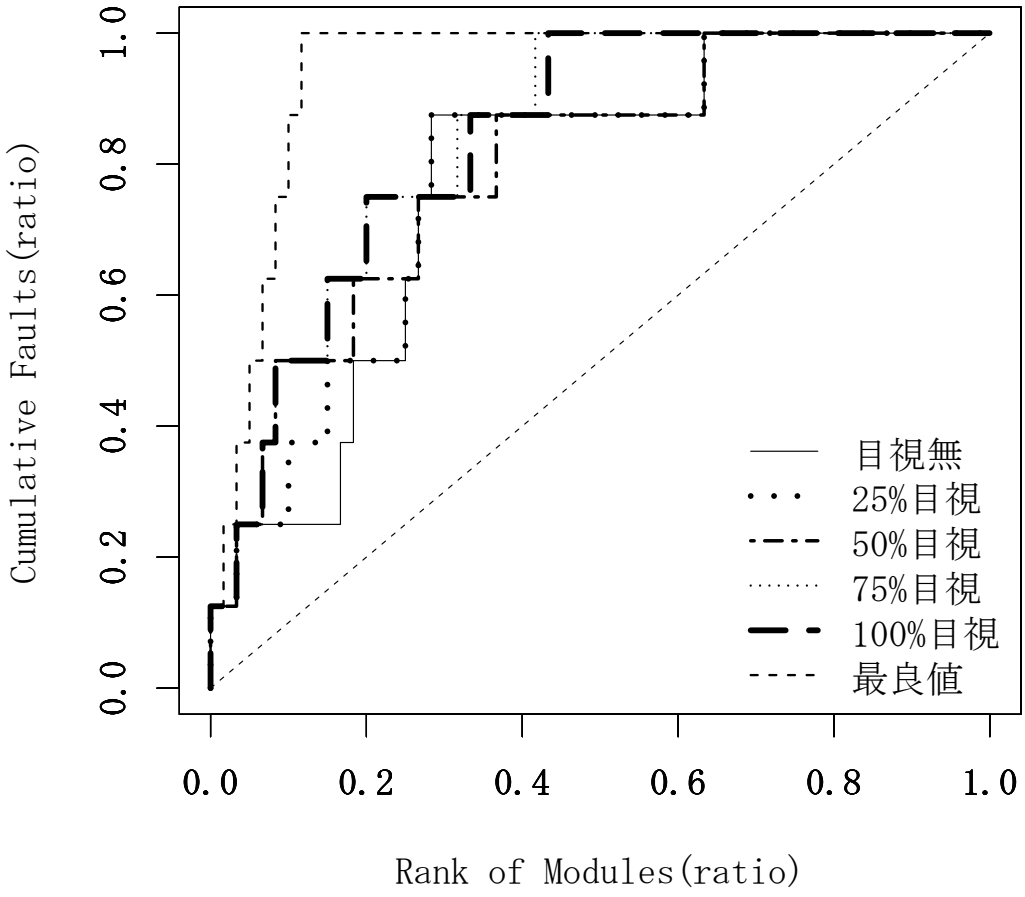

図 3.3 試行 A-Y サポートベクタマシン，予測データ: モジュール群 Y

3.2 適用結果

試行 A の Alberg diagram を図 3.2(試行 A-X)，図 3.3 (試行 A-Y) に示す．図 3.2 は学習データをモジュール群 Y とし，予測データをモジュール群 X とした場合，図 3.3 は予測データをモジュール群 Y とした場合である．全ての図において，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 ($\alpha = 0\%$) をあらわしたもの，細い一点鎖線は最良値 (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%，細い一点鎖線は 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 3.6 に後述する各試行を含めた AUC，及び，AUC の最良値に対する割合 (%)

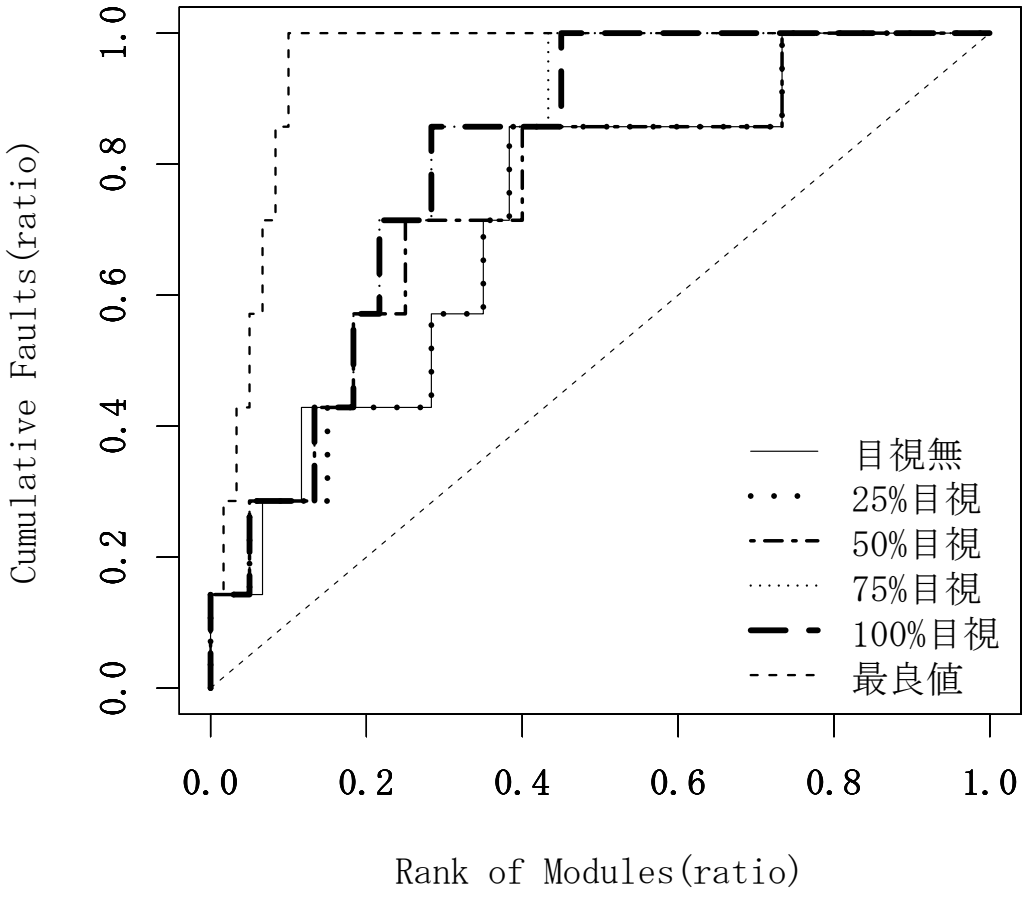

図 3.4 試行 B-X 規模順，予測データ: モジュール群 X

を示す．AUC の最良値は，全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 X のときは 0.950，Y のときは 0.942 となる．

試行 A 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，大きくなる結果が得られた．$\alpha = 25\%$ の場合，全ての試行において，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 3.6 において AUC が判別モデル単体より上回った．試行 A-X 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，

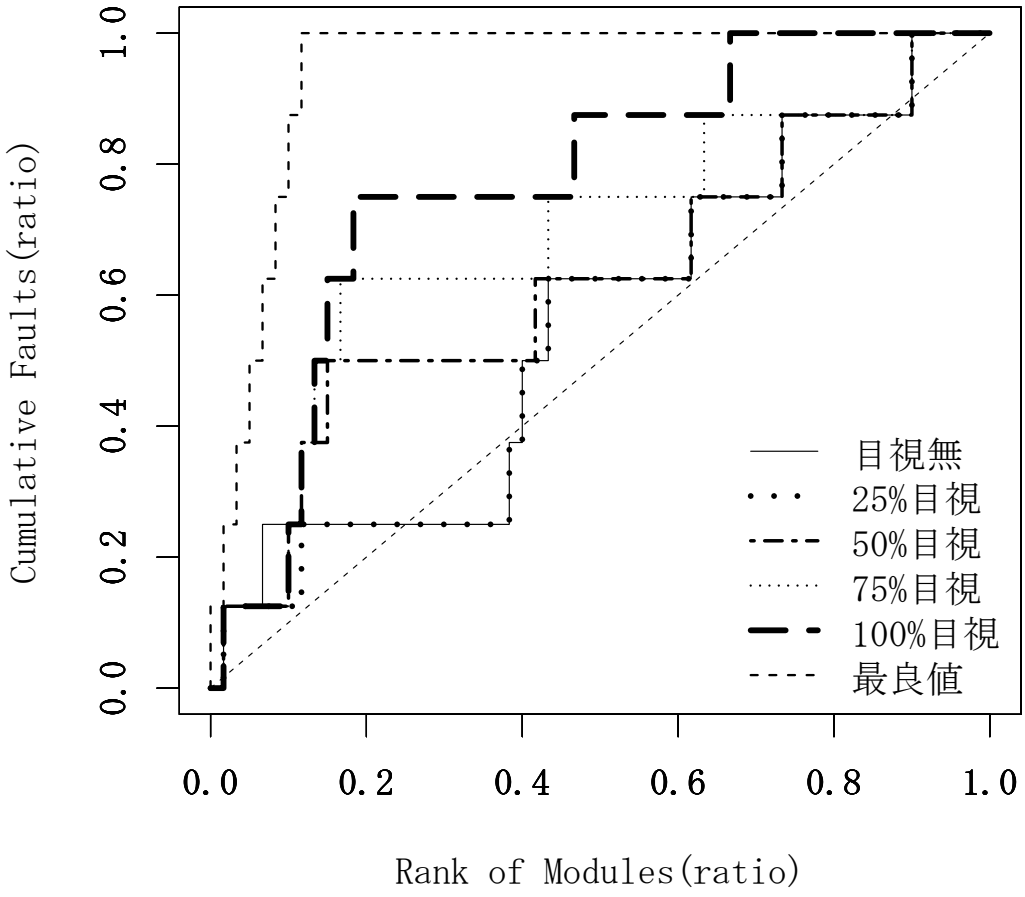

図 3.5 試行 B-Y 規模順，予測データ: モジュール群 Y

試行 A-Y 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，概ね全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 B における Alberg diagram を図 3.4, 図 3.5 に示す．表 3.6 に示すように，α= 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 C における Alberg diagram を図 3.6, 図 3.7 に示す．全ての 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 が大きくなる結果が得られた．観点に該当するモジュールの件数を表 3.5 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．

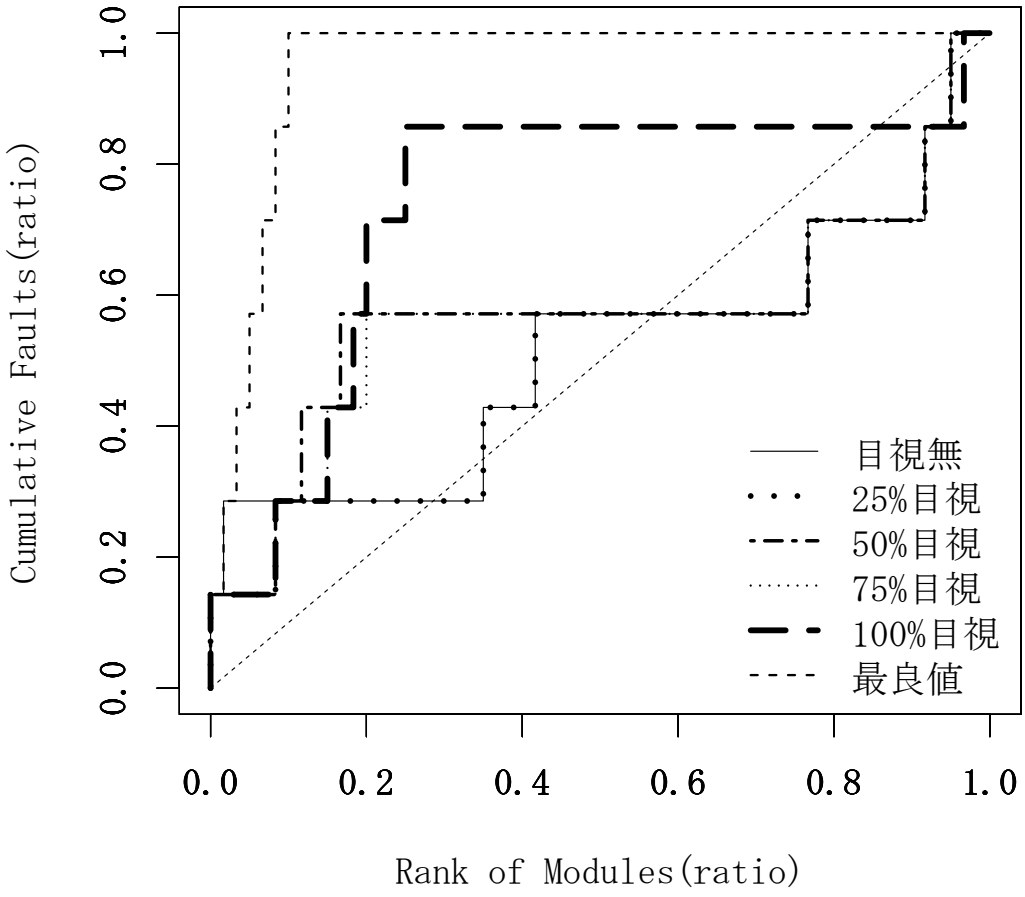

図 3.6 試行 C-X ランダム順，予測データ: モジュール群 X

表 3.5 観点 (1)，観点 (2) とケーススタディの該当件数

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

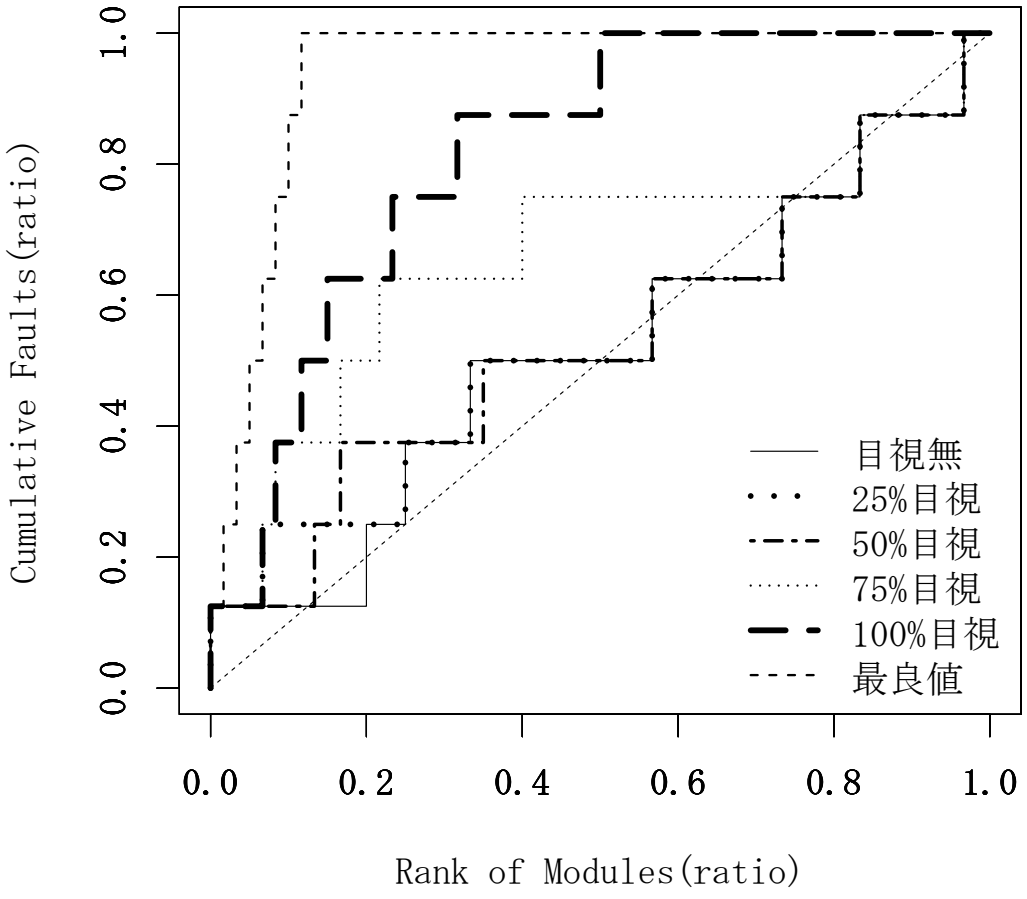

図 3.7 試行 C-Y ランダム順，予測データ: モジュール群 Y

表 3.8 に試行 A, B, C の目視評価において必要となったコストを示す．全ての α において試行 A, B の間に大きな目視コストの差はなかった．試行 C は α が 50%以下で試行 A, B と比較して最大で 1 人時程度小さい値となった．

4. 考察

4.1 判別精度

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 A では，$\alpha = 50\%$

表 3.6 各試行における AUC と最良値に対する割合 [%]

	判別モジュール群	試行 A(SVM)		試行 B(規模順)		試行 C(ランダム順)	
		AUC	ratio	AUC	ratio	AUC	ratio
目視無し	X	0.774	(81.5)	0.724	(76.2)	0.512	(53.9)
	Y	0.773	(82.1)	0.556	(59.0)	0.515	(54.6)
$\alpha = 25\%$	X	0.779	(82.0)	0.721	(75.9)	0.502	(52.9)
	Y	0.785	(83.4)	0.550	(58.4)	0.531	(56.4)
$\alpha = 50\%$	X	0.795	(83.7)	0.750	(78.9)	0.571	(60.2)
	Y	0.796	(84.5)	0.619	(65.7)	0.531	(56.4)
$\alpha = 75\%$	X	0.840	(88.4)	0.814	(85.7)	0.562	(59.1)
	Y	0.842	(89.4)	0.688	(73.1)	0.658	(69.9)
$\alpha = 100\%$	X	0.819	(86.2)	0.812	(85.5)	0.738	(77.7)
	Y	0.838	(89.0)	0.771	(81.9)	0.817	(86.7)

において，モジュール群 X, Y ともに AUC が 0.8 に近い値となり，ソースコードメトリクスのカテゴリを用いた試行 B，ランダムの試行 C 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合せにおいても 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合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 の最良値に対する割合も 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合せた提案手法の優位性を示せた． 試行 A では，判別得点だけで X 群，Y 群とも AUC の最良値に対して約 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大きい判別モデルとの組合せにおいても精度の向上があった．

表 3.6 に示した 25%おきの区間で AUC の増加を比較すると，試行 A-X, A-Y, B-X では，50%から 75%の間で，増加が最も大きくなった．試行 B-Y, C-X, C-Y 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 C 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．

目視評価を組合せた判別精度を第 2 章の結果と比較するために，目視評価対象のモジュールを選択する 4 つの組合せと 5%毎の α の値について判別精度を評価した．AUC の最大値とそのときの α の値を表 3.7 に示す．AUC と目視評価のモジュールの割合の関係を示すグラフは付録に収録した．表 2.5 と比較して 3-A-X で同点となっている以外は全て第 2 章の結果の方が判別精度が上回っている．

表 3.7 最大の AUC と α

組合せと	判別	$\alpha(\%)$	AUC	判別	$\alpha(\%)$	AUC
モデル	モジュール群			モジュール群		
1-A	X	65	0.840	Y	65	0.850
1-B	X	75	0.814	Y	95	0.771
1-C	X	95	0.740	Y	100	0.817
2-A	X	100	0.819	Y	95	0.854
2-B	X	95	0.812	Y	100	0.771
2-C	X	95	0.748	Y	95	0.819
3-A	X	75	0.831	Y	90	0.854
3-B	X	90	0.812	Y	95	0.771
3-C	X	95	0.760	Y	100	0.817
4-A	X	100	0.819	Y	100	0.838
4-B	X	100	0.812	Y	100	0.771
4-C	X	100	0.738	Y	100	0.817

4.2 目視評価コスト

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することができる．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 上での判別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

表 3.8 各試行における目視コスト [人時]

α	判別モジュール群	試行 A	試行 B	試行 C	第 2 章
25%	X	3.59	3.76	2.74	5.12
	Y	3.59	3.59	2.57	5.12
50%	X	5.99	6.16	5.31	10.23
	Y	5.99	5.99	5.48	10.23
75%	X	8.56	8.56	7.88	15.35
	Y	8.39	8.39	8.05	15.35
100%	X	10.96	10.96	10.96	20.46
	Y	10.96	10.96	10.96	20.46

(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いた部分を前倒ししてテストを実施し，この部分に潜在する不具合を先に洗い出すことで，手戻りコストを低減できる．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 3.8 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無いと考えられる．また，第 2 章 4.2 節に示した目視コストを比較として同表に示す．いずれの試行においても目視評価コストは削減でき，α が 100% の場合で目視評価コストは約 46%削減した．

4.3 目視観点

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 A' を実施した．試行 A' 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 A' の α の値を他の試行と同様に 5 段階に変化させて試行 A 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 B 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．

目視評価に使用した観点は，ケーススタディで使用したアプリケーションに特化したものではなく，対象とするアプリケーションの特性に合わせて一般化したものを用いている．対象とするアプリケーションは災害用システムのバックアップ回線を提供したり，24 時間 365 日の連続運転を要求される，ミッションクリティカルなシステムに搭載されるものであり，高い稼働率と信頼性を要求される．ケーススタディで使用した目視評価観点はこれらのシステムに適用するアプリケーションに必要とされる要件を一般的に適用した．この結果として一部の観点において偽陽性が発生している．

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毎に 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得るもの，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，両観点の系統的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．

4.4 判別モデル

本研究のアプローチで用いた判別モデルには，ケーススタディで用いたソースコードの静的解析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 [28]．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．

ケーススタディで使用したアプリケーションが搭載されるシステムはミッションクリティカルであり，不具合を市場に流出させないことが最優先であるが，仮に軽微な不具合が市場に流出した場合，納入先への補償などを除いた修正工数は，(i) 現地調査と客先への状況説明に 4 名で 2 日（64 人時），(ii) 再現及び確認のための試験環境の構築に 1 名で 1 日（8 人時），(iii) 改修と確認試験に 2 名で 1 日（16 人時），(iv) 確認試験のための仕様書の改訂やリリースのための手続き書類の作成に 1 名で 1 日（8 人時），(v) 品質管理部門での確認試験に 2 名で 1 日（16 人時），(vi) 納入先へのリリース，客先環境での確認試験，客先への報告に 5 名で 1 日（40 人時），合計で 150 人時以上の工数が必要となる．第 3 章の全モジュールの目視評価コストは 21.92（人時）であり，十分にコストメリットがあると言

える．

ケーススタディではサポートベクタマシンと規模順をランダムと比較して評価した．サポートベクタマシンは規模順と比較してモデル構築のための学習データが必要であったり，使用するソースコードメトリクスの選択や計測方法，カーネルの選択とそのパラメータの調整など適用するためのスキルが必要となる．一方で，判別モデルに規模を用いることができればこれらのスキルのための教育が不要であることや，特に学習データが不要であるため現場への導入が直ぐに可能であり，このメリットは大きいと考えられる．

5. まとめ

判別モデルにより得られたfault-proneの度合いと目視評価得点を組合せてfault-proneモジュールの判別精度を向上させる手法において，fault-proneの度合いの大きいモジュールから順に目視評価を行い，目視評価のコストを低減する方法を提案した．具体的には，判別モデルから得た判別得点によってfault-proneの度合い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．目視評価のコストを低減するため，観点は，全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，目視評価を組合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合せにおいても，目

視評価を組合せることにより，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大きい fault-prone モジュール判別モデルであっても，目視評価との組合せにより判別精度が大きくなり，3 つのカテゴリの中で最も大きな AUC 値が得られた．

第4章　関連研究

fault-prone モジュール判別では入力として様々なメトリクスを用い，出力として判別結果を得る．文献 [33] によると，メトリクスはプロダクトメトリクス，プロセスメトリクス，実行メトリクスに大別される．プロダクトメトリクスにはソースコードの行数やサイクロマチック数のようなソースコードメトリクス，プロセスメトリクスには設計者が発見した問題点の数，異なるバージョン間でのソースコード行数の差異，開発者の数，設計レビューで発見された欠陥数 [60] など，実行メトリクスにはアプリケーションの実行時間や OS からの割当て時間などがある．

ソースコードメトリクスを用いて fault-prone モジュールを判別する研究は多岐にわたり，ソースコードメトリクスの分布や性質についての研究や考察 [2] [35] [37] [39] [41] [48] [55] ，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 [15] [16] [32] [40] [51] がある．文献 [51] では決定木を利用，文献 [16]，文献 [32] では文献 [48] で提示された新規/変更モジュールに注目，文献 [15] ではロジスティック回帰分析を用いた判別，文献 [40] 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 モジュールを判別する手法として，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 [3]，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 [65]，改版情報，欠陥履歴の併用 [19]，構成管理システムのログから得たデータの併用 [50]，構成管理システムから得たプロセスメトリクスの併用 [45]，設計メトリクスの併用 [23]，自動検査 (Automated Software Inspection) ツールによる fault-prone モジュールの判別結果とソースコードの改変規模を併用 [47]，がある．Csallner ら [14] は入力としてのソースコードメトリ

クスと 3 段階の動的分析の結果を組合せたハイブリッドモデルを提案した．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合せて fault-prone モジュールを判別する手法として，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合せた手法 [43]，ロジスティック回帰分析と相関ルール分析を組合せた手法 [25][27]，ロジスティック回帰分析と決定木を組合せた手法 [56]，ロジスティック回帰分析と分類木を組合せた手法 [44]，ロジスティック回帰分析と動的リスクモデルを組合せた手法 [64]，探索木とニューラルネットワークを組合せた手法 [49]，OSR(Optimized Set Reduction) を組合せた手法 [7] がある．これらの手法は判別方法 (モデル) を組合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合せた手法ではない．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が大きい判別や予測が可能となる．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，[59] では決定木を用いて，より正確な情報を取得するために必要なコストを学習によるトレードオフによって最適化しており，文献 [4] では情報取得コストと誤判別コストの総和を最小化している．誤判別コストとのトレードオフの研究では文献 [34] や [45] があり，不具合が下流工程へ流出する第二種の過誤に対して，第一種の過誤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 [5][22][63] などがあり，文献 [22] 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 [5] 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 [18] 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．

第5章　結論

本研究では，fault-prone モジュールの判別精度を向上させることを目的として，モジュール毎の fault-prone の度合いと，ソースコードを直接目視して評価することで得られる評価得点によって fault-prone 判別精度を向上させる方法を提案した．

fault-prone 判別モデルに目視評価を加えることで判別精度が向上するかどうかを確認するために，3 種類の fault-prone モジュール判別モデル，(A) サポートベクタマシン，(B)LOC，(C) ランダム，によって得られたモジュールの fault-prone の度合いと目視評価を行うモジュールの選択順位付けについて，fault-prone の度合いの (1) 大きい順序，(2) 小さい順序，(3) 中間，(4) 上位と下位交互，の 4 種類の組合せ，及び，目視評価点の重み係数 β を変化させ，目視評価対象の選択順に $\alpha\%$ のモジュールを目視評価したときの判別精度を評価した．ソースコードの目視評価は事前に設定したチェックリストによって行い，fault-prone 判別精度の評価には Alberg diagram の AUC を用いた．この結果，目視評価を行うことで fault-prone モジュール判別モデルのみのときよりも fault-prone モジュール判別精度が増加することが解った．

詳細には，判別モデルがランダムの場合で AUC が 0.267 向上し，判別モデル単体で最も判別精度が大きかったサポートベクタマシンの場合で目視評価点を加えることで AUC が 0.069 向上した．試行した組合せにおいて，AUC が最も大きくなるものは，判別モデルが規模，目視評価対象のモジュールの選択優先順位付けは fault-prone の度合いの上位と下位から交互に取得，目視評価の割合 α が 70%，加える目視評価点の重み係数 β が 1.8 のときで 0.871 であった．次に AUC が大きくなる組合せは，判別モデルがサポートベクタマシン，目視評価対象の選択優先順位付けは上位から取得，α が 40%，β が 1.6 のときで 0.860 であった．また，判

別モデルが規模，目視評価対象のモジュールの選択優先順位付けは fault-prone の度合いの下位から取得，目視評価点の重み係数 β が 5.2 のとき，目視評価を行うモジュール数が増加すると AUC の値が単調増加した．AUC の値が単調増加することは，事前に評価を行うモジュール数を調整することが可能であることを示唆している．目視評価を行う観点のチェックリストについて，ケーススタディでは偽陽性が多いものがあった．

次に，目視評価にかかる工数を小さく抑えることを目的とし，評価対象のモジュールを規模によって大小に分け，ソースコード全体を見通せる小規模なモジュールに対してはより直接的に不具合を見つける観点，詳細な処理の流れを追うことが困難な規模の大きなモジュールに対しては潜在的な不具合を見つけるような観点，を設定して評価した．この結果，目視評価を加えた判別精度をほぼ保ったまま，目視評価工数は全モジュールの目視の場合で 46%を削減することができた．

今後の課題は，目視評価を行うチェックリストを改善して偽陽性を減らし，AUC が評価を行うモジュール数に対して単調増加するパラメータを増やすこと，目視評価工数の低減に向けた系統的な観点の設定方法を明らかにすることである．

謝辞

本研究を進めるにあたって，多くの方々から御指導，御協力，御支援を賜りました．ここに感謝の意を表します．

本研究全般，及び，本論文の執筆にあたって，貴重な御指導，御助言を頂きました奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授に心より感謝致します．研究のマイルストーン毎に要所要所を押さえた御助言，御指導を賜りました．誠にありがとうございます．

名古屋大学 大学院情報科学研究科 関 浩之 教授には，本質的な御指摘を頂き，研究全般に渡ってに大きな示唆を頂きました．ここに感謝の意を表します．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 准教授には，特に本論文の第 3 章に関して貴重な御助言，御指導を賜りました．ここに御礼を申し上げます．

名古屋大学 大学院情報科学研究科 森崎 修司 准教授には，博士後期課程全般に渡り基本的な考え方を含む数多くの御助言，御指導を賜りました．ここに感謝の意を表します．

九州大学大学院 システム情報科学研究院 情報知能工学部門 亀井 靖高 助教には，本論文の執筆にあたり御指導を賜りました．真に有難うございます．

奈良先端科学技術大学院大学 情報科学研究科 畑 秀明 助教には，登校する時間が非常に少ない中，学内での手続き等煩雑な処理を行って頂き，真に有難うございます．

奈良先端科学技術大学院大学 情報科学研究科 伊原 彰紀 助教には，学内申請に関する手続きなど便宜を図って頂き，大変助かりました．ここに感謝の意を申し上げます．

奈良先端科学技術大学院大学 学生課学務係 の方々には，休学，復学の手続き

や学生証の発行手続き等，色々ご迷惑をお掛けしました．丁寧な対応を頂きまして真に有難うございます．

三菱電機株式会社 通信機製作所 通信情報システム部 品質管理技術第二課 原田 良尚 課長には社外発表時の申請稟議など御対応頂きまして真に有難うございます．ここに感謝の意を申し上げます．

三菱電機株式会社 通信機製作所 情報技術部 ソフトウェア技術第二課 小崎 仁 課長には論文執筆にあたり御指摘，御指導を賜りました．ありがとうございます．

三菱電機株式会社 通信機製作所 生産管理部 中島正信部長，通信情報システム部 技術発表委員 堀江 延佳 プロジェクト部長には，論文の投稿に際しセキュリティ審査を実施頂きました．ここに御礼を申し上げます．

三菱電機マイコン機器ソフトウェア株式会社 第１事業部 通信制御技術部 森田 直樹 専門課長には，ケーススタディで使用したアプリケーションについてタイムリーな情報提供を頂きました．真に有難うございます．

三菱電機マイコン機器ソフトウェア株式会社 第１事業部 プロセス支援グループ 井平 博久 主査には，ケーススタディで使用したアプリケーションの歴史的経緯や開発当初の状況など，参考になる情報を提供頂きました．ここに御礼を申し上げます．

ノボノルディスクファーマ株式会社 開発本部　臨床統計解析グループ 清見 文明 上級統計家 には，研究を進めていくにあたり統計的な確認方法について検定の基礎から適用する手法とその手順等，懇切丁寧な御指導を頂きました．ここに厚く御礼を申し上げます．

最後に，研究に際して休日の家族サービスなど不自由を掛けました．御協力頂いた家族に感謝します．

参考文献

[1] Sillitti. Alberto and Succi. Giancarlo. Requirements engineering for agile methods. In Aybüke. Aurum and Claes. Wohlin, editors, *Engineering and Managing Software Requirements*, pp. 309–326. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[2] Victor. R. Basili and David. H. Hutchens. An empirical study of a syntactic complexity family. *IEEE Transactions on Software Engineering*, Vol. 9, No. 6, pp. 664–672, 1983.

[3] Victor R. Basili and Barry T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, Vol. 27, No. 1, pp. 42–52, 1984.

[4] Mustafa Bilgic and Lise Getoor. Voila: Efficient feature-value acquisition for classification. In *In Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, 2007.

[5] Lars Borner and Barbara Paech. Integration test order strategies to consider test focus and simulation effort. *Advances in System Testing and Validation Lifecycle, International Conference on*, Vol. 0, pp. 80–85, 2009.

[6] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal mrgin classifiers. In *Proceedings of the 5th annual ACM workshop on computational learning theory*, pp. 144–152. ACM Press, 1992.

[7] Lionel C. Briand, Victor R. Basili, and William M. Thomas. A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering*, Vol. 18, No. 11, pp. 931–942, 1992.

[8] Lionel C. Briand, John W. Daly, and Jürgen Wüst. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering*, Vol. 3, No. 1, pp. 65–117, July 1998.

[9] Lionel C. Briand, Sandro Morasca, and Victor R. Basili. Property-based software engineering measurement. *Software Engineering*, Vol. 22, No. 1, pp. 68–86, 1996.

[10] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, Vol. 2, No. 2, pp. 121–167, June 1998.

[11] Christopher J. C. Burges and David J. Crisp. Uniqueness of the svm solution. In *NIPS*, pp. 223–229, 1999.

[12] Cagatay Catal and Banu Diri. Review: A systematic review of software fault prediction studies. *Expert Systems with Applications*, Vol. 36, No. 4, pp. 7346–7354, May 2009.

[13] Ram. Chillarege, Inderpal.S. Bhandari, Jarir.K. Chaar, Michael.J. Halliday, Diane.S. Moebus, Bonnie.K. Ray, and Man-Yuen. Wong. Orthogonal defect classification-a concept for in-process measurements. *IEEE Transactions on Software Engineering*, Vol. 18, pp. 943–956, 1992.

[14] Christoph Csallner, Yannis Smaragdakis, and Tao Xie. Dsd-crasher: A hybrid analysis tool for bug finding. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 17, No. 2, pp. 1–37, 2008.

[15] Giovanni Denaro, Sandro Morasca, and Mauro Pezzè. Deriving models of software fault-proneness. In *SEKE '02: Proceedings of the 14th International*

Conference on Software Engineering and Knowledge Engineering, pp. 361–368, New York, NY, USA, 2002. ACM.

[16] Norman E. Fenton and Niclas Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, Vol. 26, No. 8, pp. 797–814, 2000.

[17] Martin Fowler. *Refactoring: Improving the design of existing code.* Addison-Wesley, Boston, MA, USA, 1999.

[18] Vijay Gangaram, Deepa Bhan, and James K. Caldwell. Functional test selection for high volume manufacturing. In *Proceedings of the Seventh International Workshop on Microprocessor Test and Verification*, pp. 15–19, Washington, DC, USA, 2006. IEEE Computer Society.

[19] Todd L. Graves, Alan F. Karr, J.S. Marron, and Harvey Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, Vol. 26, pp. 653–661, 2000.

[20] Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, Vol. 31, No. 10, pp. 897–910, October 2005.

[21] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, Vol. 19, pp. 167–199, 2012. 10.1007/s10515-011-0090-3.

[22] 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. 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, Vol. 101, No. 98, pp. 1–8, 2001-05-22.

[23] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In *PROMISE '08: Pro-*

ceedings of the 4th International Workshop on Predictor Models in Software Engineering, pp. 11–18, New York, NY, USA, 2008. ACM.

[24] Yue Jiang, Bojan Cukic, and Yan Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, Vol. 13, pp. 561–595, October 2008.

[25] 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, December 2008.

[26] Yasutaka Kamei, Akito Monden, and Ken-ichi Matsumoto. Empirical evaluation of an svm-based software reliability model. *Empirical Software Engineering*, Vol. 2, No. September, pp. 39–41, 2006.

[27] Yasutaka Kamei, Akito Monden, Shuji Morisaki, and Ken-ichi Matsumoto. A hybrid faulty module prediction using association rule mining and logistic regression analysis. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pp. 279–281, New York, NY, USA, 2008. ACM.

[28] 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, Sep 2009.

[29] 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, June 2005.

[30] 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, Sep 2009.

[31] 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 2012-2013. 日経 BP, Sep 2012.

[32] Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaichelvan, and Nishith Goel. Early quality prediction: A case study in telecommunications. *IEEE Software*, Vol. 13, No. 1, pp. 65–71, 1996.

[33] Taghi M. Khoshgoftaar, R. Shan, and Edward.B. Allen. Using product, process, and execution metrics to predict fault-prone software modules with classification trees. In *High Assurance Systems Engineering, 2000, Fifth IEEE International Symposim on. HASE 2000*, pp. 301–310, 2000.

[34] Taghi M. Khoshgoftaar, Xiaojing Yuan, and Edward B. Allen. Balancing misclassification rates in classification-tree models of software quality. *Empirical Software Engineering*, Vol. 5, pp. 313–330, December 2000.

[35] Meir M. Lehman, Dewayne E. Perry, and Juan F. Ramil. Implications of evolution metrics on software maintenance. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, p. 208, Washington, DC, USA, 1998. IEEE Computer Society.

[36] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, Vol. 34, No. 4, pp. 485–496, July 2008.

[37] H. F. Li and W. K. Cheung. An empirical study of software metrics. *IEEE Transactions on Software Engineering*, Vol. 13, No. 6, pp. 697–708, 1987.

[38] Mika Mäntylä and Casper Lassenius. Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering*, Vol. 11, No. 3, pp. 395–431, 2006.

[39] Thomas J. McCabe. A complexity measure. In *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, p. 407, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[40] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, Vol. 33, pp. 2–13, 2007.

[41] Tim Menzies, Justin S. Di Stefano, Mike Chapman, and Ken McGill. Metrics that matter. *Software Engineering Workshop, Annual IEEE/NASA Goddard*, Vol. 0, pp. 51–57, 2002.

[42] 株式会社三菱総合研究所. 国内及び海外におけるソフトウェアプロセス改善活動の状況に関する動向調査. 情報処理振興事業協会, March 2002.

[43] Sandro Morasca and Günther Ruhe. A hybrid approach to analyze empirical software engineering data and its application to predict module fault-proneness in maintenance. *The Journal of Systems and Software*, Vol. 53, No. 3, pp. 225–237, 2000.

[44] Sandro Morasca. A proposal for using continuous attributes in classification trees. In *SEKE '02: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pp. 417–424, New York, NY, USA, 2002. ACM.

[45] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, pp. 181–190, New York, NY, USA, 2008. ACM.

[46] John C. Munson and Taghi M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, Vol. 18, No. 5, pp. 423–433, 1992.

[47] Nachiappan Nagappan, Laurie Williams, John Hudepohl, Will Snipes, and Mladen Vouk. Preliminary results on using static analysis tools for software inspection. *Software Reliability Engineering, International Symposium on*, Vol. 0, pp. 429–439, 2004.

[48] Niclas Ohlsson and Hans Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, Vol. 22, No. 12, pp. 886–894, 1996.

[49] Parag C. Pendharkar. Exhaustive and heuristic search approaches for learning a software defect prediction model. *Engineering Applications of Artificial Intelligence*, Vol. 23, No. 1, pp. 34–40, 2010.

[50] Amit A. Phadke and Edward B. Allen. Predicting risky modules in open-source software for high-performance computing. In *SE-HPCS '05: Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications*, pp. 60–64, New York, NY, USA, 2005. ACM.

[51] Adam A. Porter and Richard W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, Vol. 7, No. 2, pp. 46–54, 1990.

[52] Bernhard. Schölkopf, Kah-Kay Sung, Chris.J.C. Burges, Federico. Girosi, Partha. Niyogi, Tomaso. Poggio, and Vladimir. 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.

[53] Jan Schumacher, Nico Zazworka, Forrest Shull, Carolyn Seaman, and Michele Shaw. Building empirical support for automated code smell detection. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pp. 8:1–8:10, New York, NY, USA, 2010. ACM.

[54] Daniel J. Sebald and James A. Bucklew. Support vector machine techniques for nonlinear equalization. *IEEE Transactions on Signal Processing*, Vol. 48, No. 11, pp. 3217–3226, 2000.

[55] Richard W. Selby and Victor R. Basili. Analyzing error-prone system structure. *IEEE Transactions on Software Engineering*, Vol. 17, No. 2, pp. 141–152, 1991.

[56] Richard W. Selby and Adam A. Porter. Learning from examples: Generation and evaluation of decision trees for software resource analysis. *IEEE Transactions on Software Engineering*, Vol. 14, No. 12, pp. 1743–1757, 1988.

[57] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, Vol. 32, No. 2, pp. 69–82, 2006.

[58] 田村晃一, 亀井靖高, 上野秀剛, 森崎修司, 松本健一. 修正確認テスト規模の低減を目的としたコードレビュー手法. 情報処理学会論文誌, Vol. 50, No. 12, pp. 3074–3083, dec 2009.

[59] Ming Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, Vol. 13, pp. 7–33, October 1993.

[60] 角田雅照, 大杉直樹, 門田暁人, 松本健一, 佐藤慎一. 協調フィルタリングを用いたソフトウェア開発工数予測方法 (<特集>産学連携論文). 情報処理学会論文誌, Vol. 46, No. 5, pp. 1155–1164, may 2005.

[61] 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, April 2006.

[62] Vladimir Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, Vol. 10, No. 5, pp. 988–999, 1999.

[63] Qiang Wang, Shuai Wang, and Yindong Ji. A test sequences optimization method for improving fault coverage. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pp. 80 –84, april 2010.

[64] W. Eric Wong, Yu Qi, and Kendra Cooper. Source code-based software risk assessing. In *SAC '05: Proceedings of the 2005 ACM Symposium on Applied Computing*, pp. 1485–1490, New York, NY, USA, 2005. ACM.

[65] Tze-Jie. Yu, Vincent. Y. Shen, and Hubert. E. Dunsmore. An analysis of several software defect models. *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, pp. 1261–1270, 1988.

[66] 財団法人日本情報システム・ユーザー協会. ソフトウェア開発管理基準に関する調査報告書. 経済産業省, Feb 2012.

付録

付録

A.1 ケーススタディで使用したソースコードメトリクスの性質

第2章，第3章のケーススタディで使用した商用アプリケーションのソースコードメトリクスについて確認した結果を示す.

ソースコードメトリクスの正規性の確認

ケーススタディで使用した商用アプリケーションから計測したソースコードメトリクスの評価，分析にあたり，測定したメトリクス値が，正規分布に従っているとみなして良いかを確認する目的で，標準化したメトリクス値の経験分布関数と標準正規分布の累積密度関数を重ねて描画したグラフ（P-P プロット）を示す. 図 A.1 と図 A.2 はサイクロマチック数，図 A.3 と図 A.4 はコメントを除いた実行数，図 A.5 と図 A.6 はループや条件分岐のネスト数，図 A.7 と図 A.8 は呼び出し数を示す．それぞれ測定した値を標準化したものと対数（常用対数）を取った値について，累積経験分布と標準正規分布関数のグラフを重ねて描画した.

図に示すように，サイクロマチック数，実行数，呼び出し数については対数を取った値を正規分布とみなすこととする．尚，本評価に先立って，正規化し常用対数を取ったサイクロマチック数と標準正規分布にてコルモゴロフ－スミルノフ検定を実施したが，標本数が多いことの影響で帰無仮説が棄却できず，第一種の過誤となったため採用しなかった（尚，標本数を減らすと非常に小さい p 値で帰無仮説を棄却できた）．ネスト数については定義域が狭く両者のグラフで明確な

表 A.1 ケンドールの順位相関係数

	Cmplx	Stat	Depth	Calls
Cmplx	1.000			
Stat	0.797	1.000		
Depth	0.751	0.722	1.000	
Calls	0.585	0.703	0.548	1.000

表 A.2 スピアマンの順位相関係数

	Cmplx	Stat	Depth	Calls
Cmplx	1.000			
Stat	0.931	1.000		
Depth	0.864	0.857	1.000	
Calls	0.755	0.869	0.698	1.000

違いは判明しなかった．本論文では，測定したソースコードメトリクスはすべて常用対数を取り，その値で評価した．

ソースコードメトリクス間の相関

ケーススタディで使用した商用アプリケーションから計測したソースコードメトリクスについて，相関が有るか否かを測定した．測定したソースコードメトリクスはサイクロマチック数，実行数，ループや条件判定のネスト数，関数呼出し数の 4 種類である．これらのうち，サイクロマチック数，実行数，呼出し数は対数正規分布すると見なせるので，ケンドールの順位相関係数行列（表 A.1），スピアマンの順位相関係数行列（表 A.2）を計測した．表より，ケーススタディで使用したソースコードメトリクスについてはメトリクス間に強い相関があることが判る．

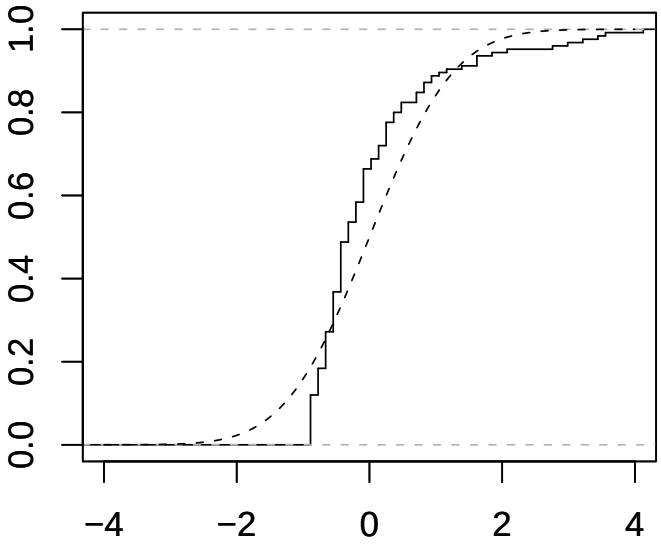

図 A.1 P-P プロット サイクロマチック数

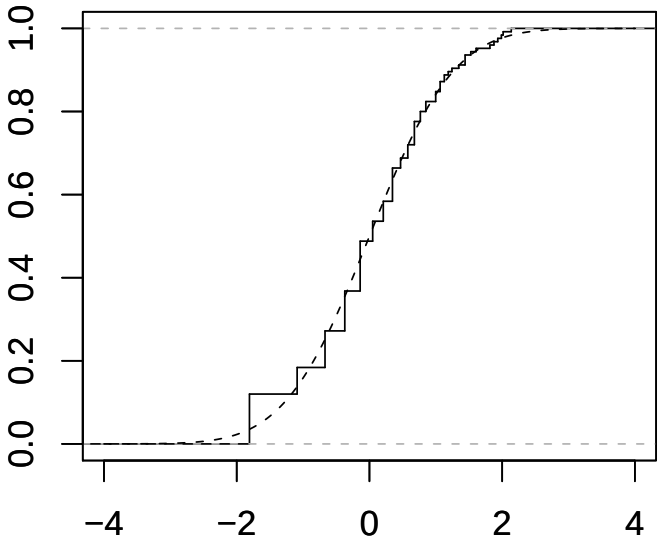

図 A.2 P-P プロット Log サイクロマチック数

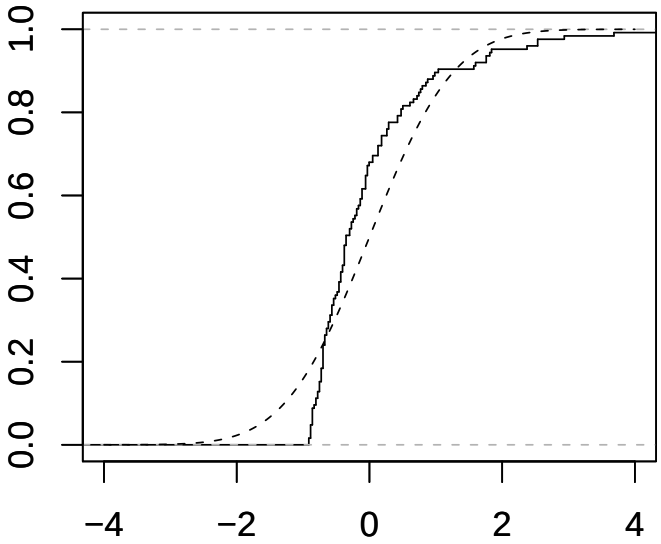

図 A.3 P-P プロット 実行数

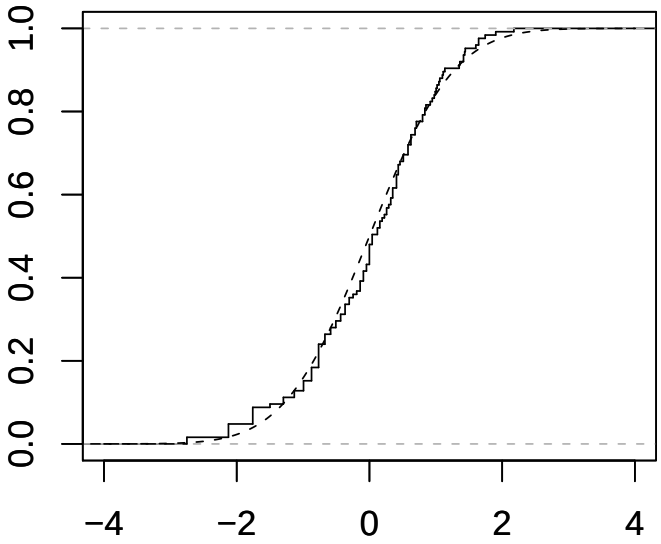

図 A.4 P-P プロット Log 実行数

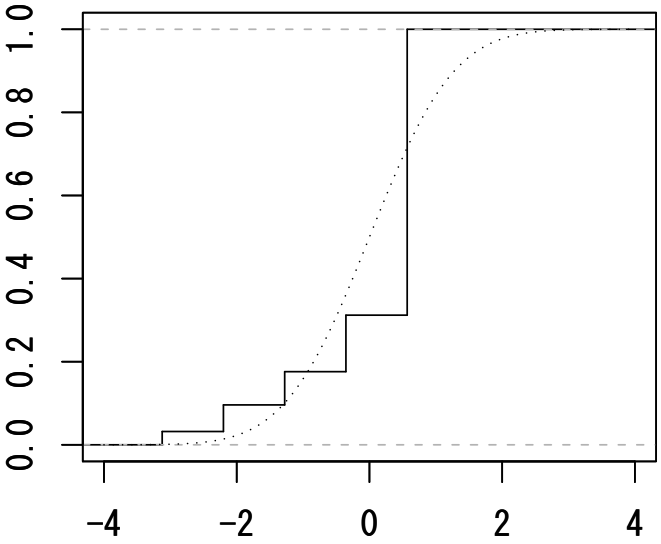

図 A.5 P-P プロット ネスト数

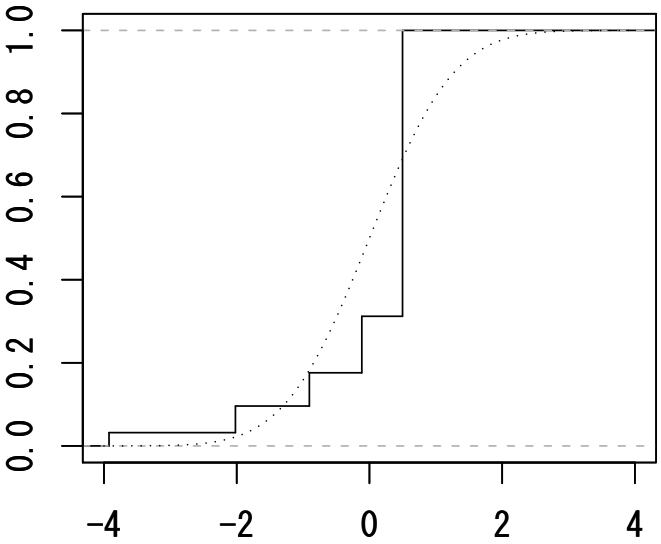

図 A.6 P-P プロット Log ネスト数

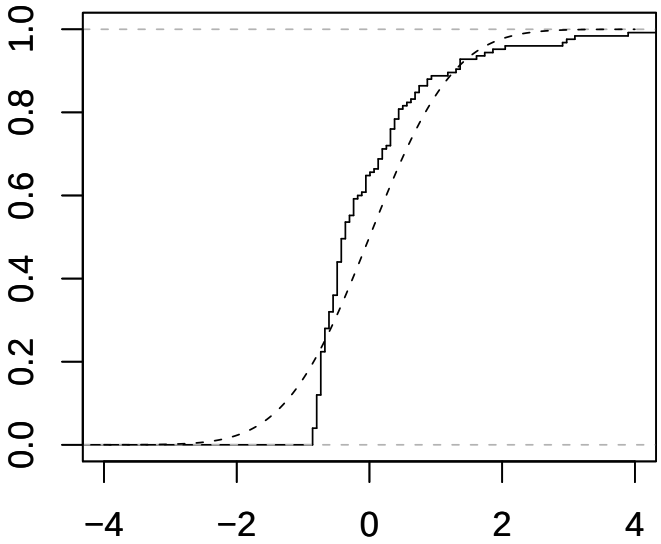

図 A.7 P-P プロット 呼出数

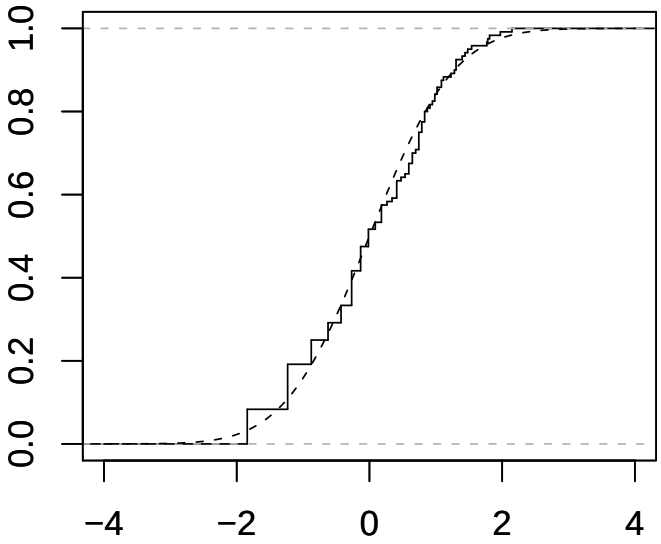

図 A.8 P-P プロット Log 呼出数

A.2 判別分析モデルの比較

ケーススタディで使用した商用アプリケーションについて，計測したソースコードメトリクスから判別分析モデルを構築して目視評価前の判別を実施した．モデルの採用にあたり，以下に示すモデルについて評価を実施した．尚，ロジスティック回帰分析について事前試行した判別結果では，ケーススタディで使用した商用アプリケーションの場合，判別成績が非常に悪く無作為の結果に近かったため検討対象から外した．

線形判別分析

線形判別分析では，判別関数は線形式の形で表現される．p 個の変量 $x_1, \cdots, x_p$ に対して

$$z = a_1 x_1 + \cdots + a_p x_p$$

のような線形結合量 z を考え，この値によって判別を行う．

平均が μ ，分散共分散行列が Σ の p 次元正規分布の密度関数は，

$$\phi(\mathbf{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}(\mathbf{x}-\mu)^t \Sigma^{-1}(\mathbf{x}-\mu)) \tag{A.1}$$

と表せる．ここで，A^t は行列 A の転置行列，$|A|$ は A の行列式，A^{-1} は A の逆行列を表す．

式（A.1）の対数をとると，

$$\log\phi(\mathbf{x}; \mu, \Sigma) = -\frac{p}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\mathbf{x}-\mu)^t \Sigma^{-1}(\mathbf{x}-\mu) \tag{A.2}$$

となる．

部分母集団 A，B の構成比率を π_{A} ，π_{B} とすると、全体の母集団の密度関数は，式（A.1）と π_{A} ，π_{B} の積和で表すことができる．部分母集団 A，B に属するデータ $\mathbf{x}_{\mathrm{A}}$，$\mathbf{x}_{\mathrm{B}}$ の平均を μ_{A}，μ_{B}，分散共分散行列を Σ_{A}，Σ_{B} とすると，或る観測デー

タ x が集団 A に属すると判定される領域は，各部分母集団 A，B の密度関数に構成比率を乗じた項の差に式（A.2）を適用して，

$$\log\pi_{\mathrm{A}} - \frac{1}{2}\log|\Sigma_{\mathrm{A}}| - \frac{1}{2}(\mathbf{x}-\mu_{\mathrm{A}})^t\Sigma_{\mathrm{A}}^{-1}(\mathbf{x}-\mu_{\mathrm{A}}) > \log\pi_{\mathrm{B}} - \frac{1}{2}\log|\Sigma_{\mathrm{B}}| - \frac{1}{2}(\mathbf{x}-\mu_{\mathrm{B}})^t\Sigma_{\mathrm{B}}^{-1}(\mathbf{x}-\mu_{\mathrm{B}}) \tag{A.3}$$

を満たす $\mathbf{x}$ と表すことができる．ここで，集団 A，B が共通の分散共分散行列 Σ を持ち，構成比率 π_{A} ，π_{B} が等しいと仮定すると，式（A.3）は，

$$\frac{1}{2}(\mathbf{x}-\mu_{\mathrm{A}})^t\Sigma_{\mathrm{A}}^{-1}(\mathbf{x}-\mu_{\mathrm{A}}) < \frac{1}{2}(\mathbf{x}-\mu_{\mathrm{B}})^t\Sigma_{\mathrm{B}}^{-1}(\mathbf{x}-\mu_{\mathrm{B}}) \tag{A.4}$$

となる．

ここで，マハラノビスの距離を考える．

$$D_{\mathrm{A}}^2 = (\mathbf{x}-\mu_{\mathrm{A}})^t\Sigma_{\mathrm{A}}^{-1}(\mathbf{x}-\mu_{\mathrm{A}})$$

は部分母集団 A と $\mathbf{x}$ のマハラノビス距離である．部分母集団 A，B と $\mathbf{x}$ のマハラノビス距離 D_{A}^2，D_{B}^2 を用いて式（A.4）を書きなおすと，

$$\begin{aligned}
\frac{1}{2}(D_{\mathrm{B}}^2 - D_{\mathrm{A}}^2) &= \frac{1}{2}\{(\mathbf{x}-\mu_{\mathrm{B}})^t\Sigma_{\mathrm{B}}^{-1}(\mathbf{x}-\mu_{\mathrm{B}}) - (\mathbf{x}-\mu_{\mathrm{A}})^t\Sigma_{\mathrm{A}}^{-1}(\mathbf{x}-\mu_{\mathrm{A}})\} \\
&= -\mathbf{x}^t\Sigma^{-1}\mu_{\mathrm{B}} + \frac{1}{2}\mu_{\mathrm{B}}^t\Sigma^{-1}\mu_{\mathrm{B}} + \mathbf{x}^t\Sigma^{-1}\mu_{\mathrm{A}} + \frac{1}{2}\mu_{\mathrm{A}}^t\Sigma^{-1}\mu_{\mathrm{A}} \\
&= \mathbf{x}^t\Sigma^{-1}(\mu_{\mathrm{A}}-\mu_{\mathrm{B}}) - \frac{1}{2}(\mu_{\mathrm{A}}+\mu_{\mathrm{B}})^t\Sigma^{-1}(\mu_{\mathrm{A}}-\mu_{\mathrm{B}}) \\
&= \{\mathbf{x} - \frac{1}{2}(\mu_{\mathrm{A}}+\mu_{\mathrm{B}})\}^t\Sigma^{-1}(\mu_{\mathrm{A}}-\mu_{\mathrm{B}})
\end{aligned} \tag{A.5}$$

となり，式（A.5）が 0 以上の時，A 群に判別，負の時 B 群に判別される．

部分母集団 A，B の母数は通常未知であるため，トレーニングデータからこれらを推定する．部分母集団 A，B から n_{A}，n_{B} 個のトレーニングデータが得られたとすると，μ_{A}，μ_{B} の推定値 $\hat{\mu}_{\mathrm{A}}$，$\hat{\mu}_{\mathrm{B}}$ は，

$$\hat{\mu}_{\mathrm{A}} = \frac{1}{n_{\mathrm{A}}}\Sigma x_{\mathrm{A}}, \quad \hat{\mu}_{\mathrm{B}} = \frac{1}{n_{\mathrm{B}}}\Sigma x_{\mathrm{B}} \qquad x_{\mathrm{A}} \in A, x_{\mathrm{B}} \in B$$

となる．部分母集団 A，B の分散共分散行列は，群内標本分散共分散行列 $\mathbf{S}_{\mathrm{A}}$ ，$\mathbf{S}_{\mathrm{B}}$ が，

$$\mathbf{S}_{\mathrm{A}} = \frac{1}{na-1}\Sigma(xa-\hat{\mu}_{\mathrm{A}})(xa-\hat{\mu}_{\mathrm{A}})^t, \mathbf{S}_{\mathrm{B}} = \frac{1}{na-1}\Sigma(xa-\hat{\mu}_{\mathrm{B}})(xa-\hat{\mu}_{\mathrm{B}})^t$$

で推定されるが，BoxM 検定等の結果により，両者を等しいと考えてよい場合は，プールされた分散共分散行列として，

$$\mathbf{S} = \frac{(n_{\mathrm{A}}-1)\mathbf{S}_{\mathrm{A}} + (n_{\mathrm{A}}-1)\mathbf{S}_{\mathrm{B}}}{n_{\mathrm{A}}+n_{\mathrm{B}}-2}$$

を用いて，判別式は

$$z = \{\mathbf{x} - \frac{1}{2}(\hat{\mu}_{\mathrm{A}}+\hat{\mu}_{\mathrm{B}})\}^t\mathbf{S}^{-1}(\hat{\mu}_{\mathrm{A}}-\hat{\mu}_{\mathrm{B}})$$

となる．z が 0 以下のときバグを含む群，正のときバグを含まない群に判別する．

サポートベクタマシン

サポートベクタマシン（Support Vector Machine,SVM）は、現在知られているパターン認識手法の中でも最も認識精度が大きい学習モデルのひとつと考えられており，未知の学習データに対して識別精度が高いことが知られている．

2 つの部分母集団 A，B に属することが判っているトレーニングデータ

$$(x_1, y_1), \cdots, (x_l, y_l), \quad x \in \mathcal{R}^n, \quad y \in \{+1, -1\}$$

が与えられているとき，これらのデータが超平面で分離可能（線形分離可能）な場合とそうで無い場合が存在する．ここでは，後者について考える．この場合，二つの群を誤りなく分ける超平面

$$\langle w, x\rangle + b = 0$$

が存在しないため，このような w，b は存在しない．

ここで，入力ベクトル x を非線形写像によって $\phi(x)$ に拡張し，この超空間で

線形識別を行う．超空間で行った線形判別は元の空間では非線形な分離面となる．写像 ϕ に対して，

$$K(x,y) = \langle \phi(x), \phi(y) \rangle$$

となるようなカーネル関数 $K(x,y)$ を考える．ここで $\langle x,y \rangle$ は内積を表す．カーネル関数を用いた場合の目的関数は，

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

を最大化，制約条件は，

$$\alpha_i \geq 0 \ \ (i = 1, \cdots, l), \ \ \sum_{i=1}^{l} y_i \alpha_i = 0$$

となる．ここで，α はラグランジュ乗数である．カーネルトリックにより，実際に ϕ を求めることなくカーネル関数 K の計算のみで済む．このときの分離超曲面は，

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b = 0$$

となる．カーネル関数の例として，多項式カーネル

$$K(x,y) = (\langle x,y \rangle + 1)^p$$

や，ガウシアンカーネル

$$K(x,y) = exp(\frac{-\|x-y\|^2}{\sigma^2})$$

などが知られている．超空間で行った線形判別による判別得点を，0 以下のときバグを含む群，正のときバグを含まない群に判別する．

主成分分析

多種類のデータが得られたとき，これらのデータ（説明変数）を用いて或る値（目的変数）がどれくらい説明できるかを分析する手法として回帰分析があるが，説明変数間の相関が高い場合，説明変数間の相関関係（多重共線性）を避けるため，説明変数を除外する必要がある．主成分分析（Principal Component Analysis,PCA）は，多数の観測データを処理する場合処理速度などの都合によりデータの次元数を減らす目的で使用されることが多いが，ここではソフトウエアメトリクス間の相関の高さに注目し，多重共線性の問題を避け，観測データの次元を落とさない目的として主成分分析を用いる．

n 個 p 次元のデータ $\mathbf{x}_1, \cdots, \mathbf{x}_n$ が得られたとき，このデータの相関行列を Σ とすると，$\Sigma\mathbf{A} = \lambda\mathbf{A}$ を満たす $\lambda \in \mathcal{R}$ を Σ の固有値，$\mathbf{A}$ を λ に属する固有ベクトルという．固有方程式 $|\Sigma\mathbf{A} - \lambda\mathbf{I}| = 0$（但し $\mathbf{I}$ は単位行列）を解いて得られる固有値 $\lambda_1, \cdots, \lambda_p$ のうち最大のものを λ_1 とすると，λ_1 に属する固有ベクトル $(a_1 \cdots a_p)$ を用いて，第一主成分の主成分得点 Z_1 は，

$$Z_1 = a_1 x_1' + a_2 x_2' + \cdots + a_p x_p'$$

但し，x_i' は x_i を平均が 0，分散が 1 となるように x を標準化した値とする．以下，同様に第 p 主成分得点まで求めることができる．主成分負荷量は固有ベクトルに固有値の正の平方根を乗じて得ることができ，寄与率は固有値を変量の次数 p で割って得ることができる．固有値の大きい順に得た寄与率を累積して得られる累積寄与率が 8 割を超えるところまでの主成分を採用することが多い．第 1 主成分，第 2 主成分，等は互いに独立であり，それぞれの主成分が何を意味するかは分析者が決める必要がある．このとき，主成分負荷量の符号を確認してその意味を考える．後述のように，本実施例では累積負荷量が第 1 主成分のみで 70% を超え，かつ第二主成分の寄与率が 0.2 未満であったため，第 1 主成分を採用した．負荷量の符号は負であったので，0 以下のときバグを含む群，正のときバグを含まない群に判別する．

表 A.3 各判別モデルの AUC と最良値に対する割合 [%]

	判別モジュール群	SVM		LDA		PCA	
		AUC	ratio	AUC	ratio	AUC	ratio
目視無し	X	0.774	(81.5)	0.643	(67.7)	0.741	(77.9)
	Y	0.773	(82.1)	0.554	(58.8)	0.599	(63.5)
$\alpha = 25\%$	X	0.779	(82.0)	0.655	(68.9)	0.741	(77.9)
	Y	0.785	(83.4)	0.581	(61.7)	0.593	(62.8)
$\alpha = 50\%$	X	0.795	(83.7)	0.736	(77.4)	0.769	(80.7)
	Y	0.796	(84.5)	0.577	(61.3)	0.639	(67.7)
$\alpha = 75\%$	X	0.840	(88.4)	0.717	(75.4)	0.823	(86.4)
	Y	0.842	(89.4)	0.569	(60.4)	0.698	(73.9)
$\alpha = 100\%$	X	0.819	(86.2)	0.805	(84.7)	0.821	(86.2)
	Y	0.838	(89.0)	0.794	(84.3)	0.782	(82.9)

A.3 AUC の評価

他の判別モデルの評価

第 3 章において，fault-prone 判別モデルの候補として線形判別分析，サポートベクタマシン，主成分分析について事前評価した．その結果を表 A.3 に示す．

表に示すように，判別モデル単体としての性能はサポートベクタマシンが最も良好で有ったため，本論文ではこの判別モデルを採用した．主成分分析を判別分析に使用する例はあまり多くない [32][50] が，メトリクス値間の相関が高いことから試行した結果，線形判別分析より（前述の通りロジスティック回帰分析を含む）も AUC の値は大きかった．

AUC と目視評価モジュールの割合の関係（第 3 章）

第 3 章では目視評価を行ったモジュールの割合は四分位数に絞って評価したが，第 2 章と同様に α を 5%刻みで AUC を評価したグラフを図 A.9，A.10，A.11，A.12，A.13，A.14 に示す．

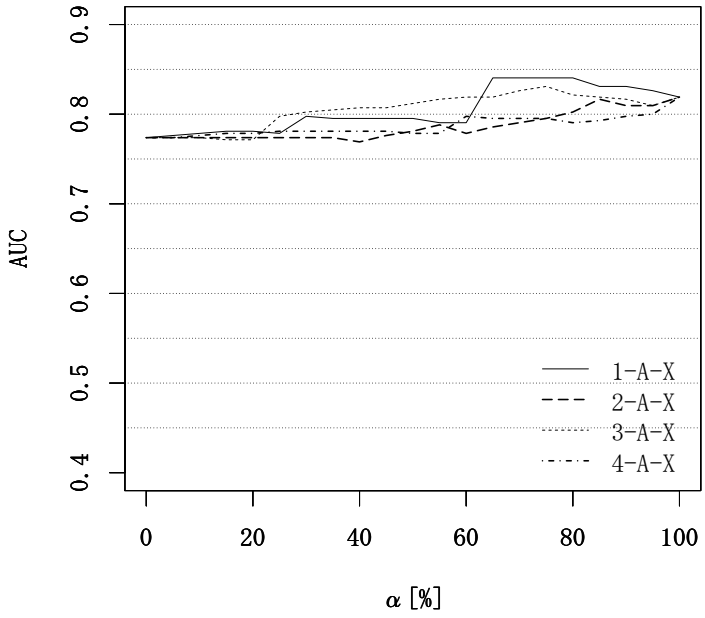

図 A.9 サポートベクタマシンと目視評価の組合せ データセット X（第 3 章）

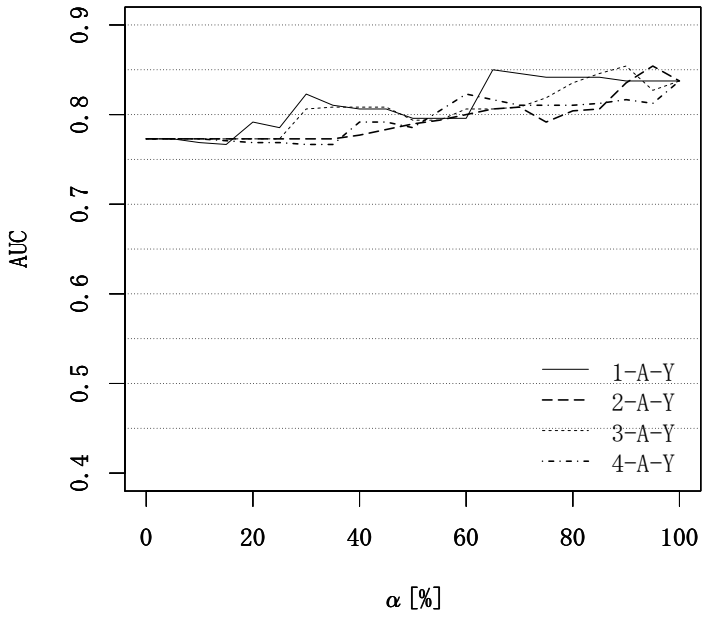

図 A.10 サポートベクタマシンと目視評価の組合せ データセット Y（第 3 章）

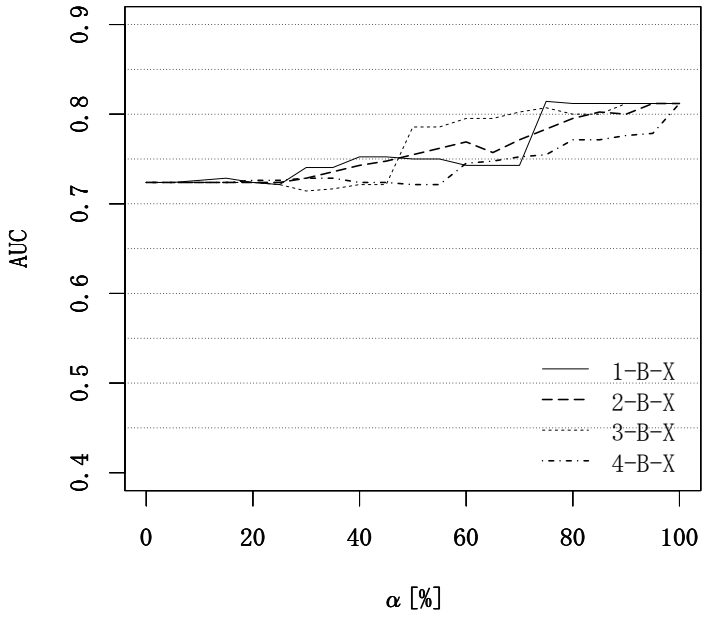

図 A.11 LOC と目視評価の組合せ データセット X（第 3 章）

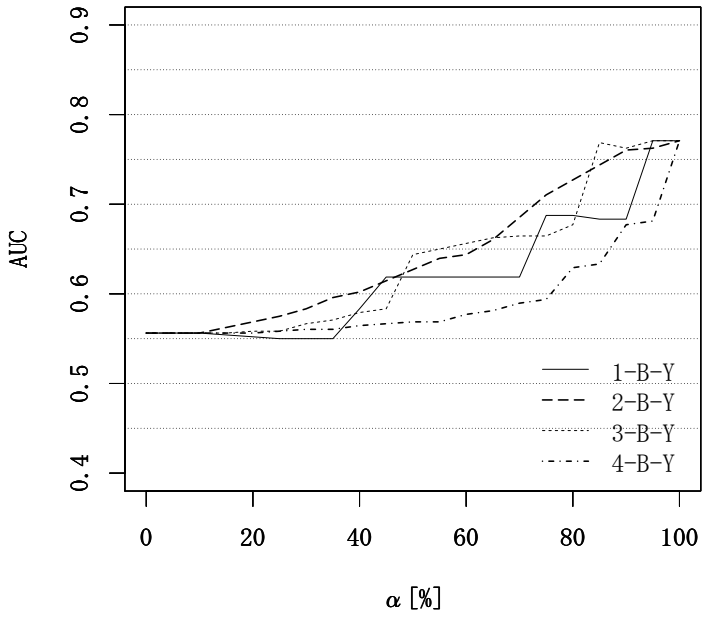

図 A.12 LOC と目視評価の組合せ データセット Y（第 3 章）

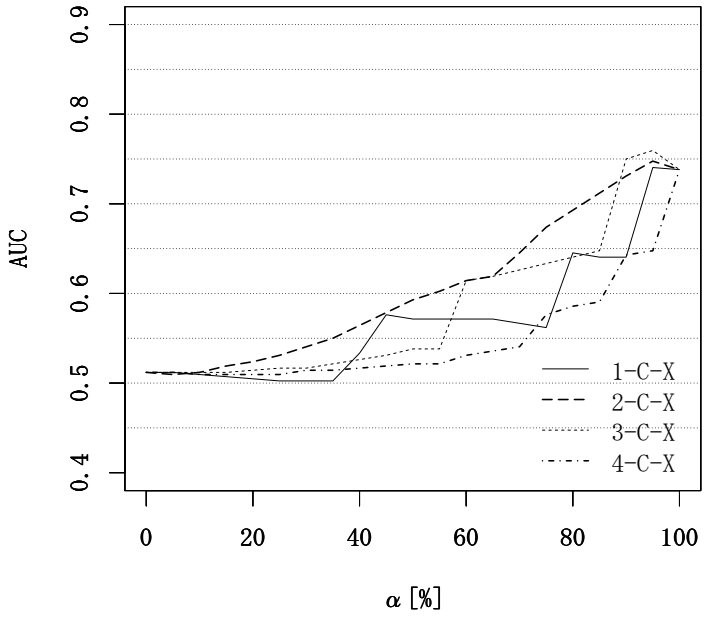

図 A.13 ランダムと目視評価の組合せ データセット X（第 3 章）

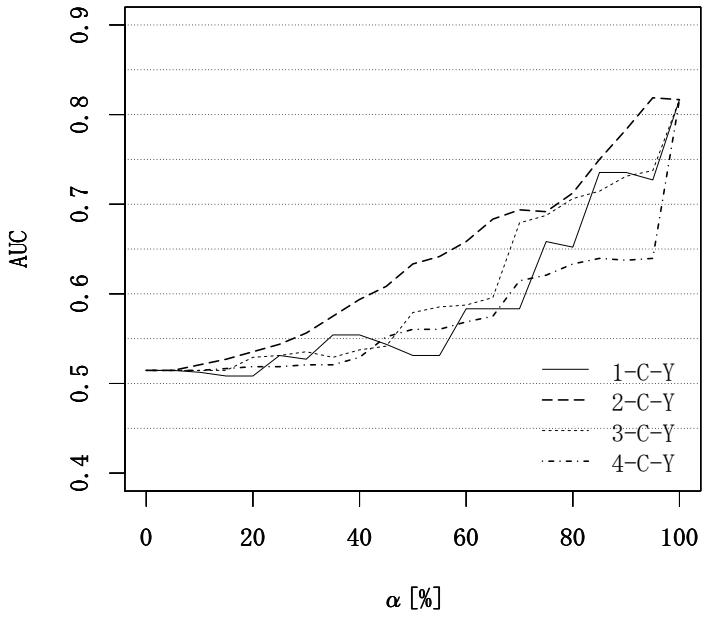

図 A.14 ランダムと目視評価の組合せ データセット Y（第 3 章）