信州大学審査学位論文

Method Mining に基づく error-prone モジュールの予測

2015 年 3 月

内宮　秀明

図表目次

第一章

まえがき

第1章　まえがき

1.1. 研究の背景と目的

ソフトウェア開発を行うにあたり，プロジェクトチーム(以降はプロジェクトと称す)が結成される．プロジェクトは，限られた期間内に運用に耐えうるだけの品質を満たすソフトウェアを開発することが求められる．しかしながら，開発対象のソフトウェアは独自性があり，必ずしも，既存開発における各種成果物(設計書，ソースコード，テストツール)を流用することによる効率化だけでは十分であるとは言えない．従って，効率的なソフトウェア開発を行うためには，既存開発で培った成果物を再利用する他に，別の手段を適用することが必要である．

ソフトウェアの新規開発において最も工数のかかる工程は，製作工程であり，その次は，結合テストと総合テストからなる試験工程と言われている[1]．表 1 に文献[1]に掲載されていた「工程別の実績工数の比率の基本統計量(新規開発)」(図表 8-1-13)の内容を示す．本表によれば，中央値として，製作工程における実績工数の比率は 34.2%あり，試験工程の比率(結合テストと総合テストの合計比率)は 28.1%に及ぶ．

表 1　工程別の実績工数の比率の基本統計量(新規開発)(文献[1]より抜粋)

工程	件数	最小	P25	中央	P75	最大	平均	標準偏差
基本設計	679	0.001	0.104	0.154	0.210	0.589	0.167	0.092
詳細設計	679	0.014	0.120	0.171	0.221	0.533	0.175	0.077
製作	679	0.018	0.267	0.342	0.432	0.847	0.356	0.135
結合テスト	679	0.002	0.115	0.164	0.219	0.588	0.173	0.087
総合テスト	679	0.000	0.067	0.117	0.175	0.564	0.130	0.085

ソフトウェア開発では，限られた期間内にある程度の品質を確保するためにソフトウェアテスト等の効率化が求められており[2]，これらの工程における品質の確保と効率化は必要不可欠である．例えば，これらの工程における品質確保策としては，製作工程ではソースコードを対象としたインスペクションを行うことで品質の向上が期待できる．また，試験工程においては，網羅的な試験項目を消化することにより，納入までの間に誤りを摘出することで品質を向上させることが期待できる．しかし，インスペクション及びソフトウェアテストは開発規模に応じて，工数も必要となるため，開発の効率化を図るためには，品質を劣化させないことを前提として，実施対象範囲の絞りみが必要となる．実施対象範囲を絞り込むためには，まだ発見されていない誤りを含んでいる可能性の高いモジュール(error-prone モジュール)の予測を行うことは有効である．Error-prone モジュールと判定したモジュール或いは，error-prone モジュールが関係する機能部全体に対してインスペクションもく

しはソフトウェアテストを集中的に行うことで，早期に誤りを摘出するだけでなく，インスペクションやデバッグに費やす工数も削減することが出来るからである．一般的に，早期の誤りの摘出は工数削減に貢献できると言われている．例えば，1981 年に IBM 社が行った，ソフトウェア開発における誤りの早期検出がコストに与える影響の調査[28]によれば，設計中に発見した誤りを修正するためのコストを 1.0 としてコストを比較すると，テストの開始直前に検出された同じ誤りのコストは 6.5 倍，テスト中は 15 倍，ソフトウェア出荷後は 60～100 倍かかる結果となることが報告されている．この調査は，大規模なプロジェクトで収集したコストデータに基づく相対的なコストを算出した結果である．理想的には設計工程で誤りを摘出することが望ましいが，ソースコードに対するインスペクションを行うことでテスト前に誤りを検出することや，ソフトウェアテストの段階で誤りを摘出することでソフトウェア出荷後において誤りを含んだソフトウェアを流出させることを予防することでも，コスト削減を行うことが出来るため，ソフトウェア開発の効率化に寄与していると言える．

上述した背景から，これまで，ソフトウェア開発の効率化を目的として，数多くの error-prone モジュールの予測手法が提案されているが，現状，汎用的でかつ最適な error-prone モジュール予測器は存在しない[3].また，訓練のためのデータセットに関しては一般的には自プロジェクトの過去データセットを使用するが，新規開発においては適用できないケースが存在することや，必ずしも自プロジェクトの過去データが最適ではないという事例もある[4]．また，検査対象プロジェクトに対して，最適（もしくは適切）な error-prone モジュール予測器をどのようにして選ぶかについては，ほとんど研究が行われていない．

HE はメトリクスの集約値と予測結果の精度の関係を学習させることで，検査対象プロジェクトに対して適切な error-prone モジュール予測器を見つけ出す手法を提案している[4]．本論文では HE が提唱した手法を応用して，より適切にマイニングを行うためにプロジェクトの特徴量を導入し，method mining に基づく error-prone モジュールの予測を行うために，対象プロジェクト毎に適切な error-prone モジュール予測器を識別するマイニング手法を提案するとともに，複数のデータを用いて，マイニング手法の可能性を実証する．

1.2. 諸定義

モジュールとは，ソフトウェアを構成する最小となる品質評価の単位である．通常，ソフトウェアは複数のモジュールから構成される．本論文では，ソースコードの 1 ファイルを 1 モジュールとして定義する．

Error とは，モジュールに含まれる正しくない処理のことである．具体的には，仕様書に定められた機能を遂行できない事象の直接の原因となったプログラムの記述誤りのことである．一般的にバグもプログラムの欠陥を意味するが，本論文では，誤りと表記する．

Error-prone モジュールとは，まだ発見されていない誤りを含んでいる可能性の高いモジュールのことである．

目的変数とは，予測や識別を行った結果を格納する変数である．

説明変数とは，ある現象の原因と考えられる要素を意味し，目的変数の関係性を説明する要素である．本論文では，error-prone モジュールの予測や，error-prone モジュール予測器の識別で使用する学習アルゴリズム(予測アルゴリズム)や訓練データを識別する際，原因や要因と考えられる変数を意味する．

特徴量とは，プロジェクトの特性を示す要素であり，プロジェクト単位に存在し，かつ利用するメトリクスには現れない要素である．3.2 にて詳述する．

予測器とは，本論文では error-prone モジュール予測器を意味する．なお，予測アルゴリズムと訓練データを基に error-prone モジュール予測器は構築されることから，本論文において，予測アルゴリズムと訓練データの総称として予測器と呼ぶことがある．

識別器とは，検査対象プロジェクト毎に最適な error-prone モジュール予測器を識別するための識別器を意味する．なお，予測アルゴリズムと訓練データを基に error-prone モジュール予測器は構築されることから，本論文において，予測アルゴリズムと訓練データを識別することと，error-prone モジュール予測器を識別することは同意となる．

プロジェクトとは，ソフトウェアを開発するために期間限定で結成された組織もしくはチームのことである．

予測閾値とは，error-prone モジュールの予測を行う際に，どれだけの予測精度を求めているのかを示す基準値のことである．本閾値は，検査対象のモジュールが誤りを含むか否かを予測する error-prone モジュール予測器が最適かどうかを判断する基準となる．本閾値は，識別器を構築する際の目的変数に true/false を指定する基準となる．

予測精度とは，予測された値と真の値の近さの度合いを意味する．本論文では，error-prone モジュール予測結果と実際における各モジュールの誤り有無の実績とを対比させることで明確にした，予測の正確さを表した指標のことである．

モジュール毎のメトリクスデータとは，モジュール単位にソースコードを計測したメトリクスデータと誤りの有無をベクトル化したデータである．

プロジェクトデータとは，モジュール毎のメトリクスデータをプロジェクト単位に束ねたデータである．

予測のためのプロジェクトデータセットとは，プロジェクトデータを束ねたデータである．

集約したプロジェクトデータとは，予測のためのプロジェクトデータセットを構成するプロジェクトデータに含まれる，モジュール毎に計測したメトリクスデータから平均値，標準偏差，最小値，最大値をそれぞれ算出することでモジュール単位からプロジェクト単位に集約させたデータである．

参照データセットとは，集約したプロジェクトデータを束ねたデータである．

識別訓練データとは，(プロジェクトデータ X，プロジェクト X の特徴量，予測アルゴリズム，プロジェクトデータ X，プロジェクト X の特徴量，目的変数)の 6 つ組のデータである．

識別訓練データセットとは，識別訓練データを束ねたデータである．

1.3. 本研究の貢献と本論文の構成

本研究の貢献は，以下の点である．

(1) 検査対象プロジェクト及びそれに課された条件に基づいて適切な error-prone モジュール予測器を識別する，マイニングに基づく識別手法(本論文では「method mining に基づく識別手法」と定義する)の提案を行った．

(2) Method mining に基づく識別手法の識別精度を改善するために，プロジェクトの特性をあらわす特徴量を使用することを提案し，その可能性を示した．

(3) 同一プロジェクトの過去データと他プロジェクトデータそれぞれを訓練データとして用意して提案手法の有効性を確認する実証実験を行った．その結果，提案手法の有効性を実証した．

本論文では，2 章で error-prone モジュール予測手法の課題と関連研究について述べる． 3 章で HE が提案する手法及び，本論文で提案する，HE の提案手法を改良した検査対象プロジェクトに最適な予測器を識別し，識別した予測器を用いて error-prone モジュール予測を行うまでの手法(method mining に基づく error-prone モジュールの予測手法)について説明する．4 章で提案手法の有効性を実証するための仮説及び実験の内容を説明し，5 章で実験の結果と評価を述べる．そして，6 章で妥当性への脅威について述べ，7 章を結論とする．

第 2 章

Error-prone モジュール予測手法の課題と関連研究

第2章　Error-prone モジュール予測手法の課題と関連研究

2.1. 緒言

本章ではメトリクスや学習アルゴリズム(予測アルゴリズム)の概要, error-prone モジュール予測が可能な根拠, error-prone モジュール予測手法の概要, error-prone モジュール予測手法における課題及び, 関連研究を述べる.

ソフトウェア開発において誤りが混入する要因としては, 仕様が複雑であったり, プログラム構造が複雑であったり, 作業者のスキル不足等により, 不十分な理解に基づくコーディングや修正, 追加開発を行うことが考えられる. 誤り混入要因それぞれに直接的に対応したメトリクス(2.3 にて詳述する)もあれば, 誤り混入要因と間接的に関係があると想定されるメトリクスもある. 従って, これらのメトリクスに基づいて誤りの有無を予測することは可能である. 一般的な例としては, プログラム構造が複雑である度合いは, ソースコードメトリクスの場合, サイクロマチック数で計測可能である. これに対して, 作業者のスキル不足については直接的に計測できるメトリクスは存在しないが, 担当機能毎の誤り発生件数や日々の作業時間等を計測し, これらの値が相対的に大きければスキル不足であることの類推は可能である. 但し, 本論文では, 事例としてソースコードメトリクスを対象とするため, 担当機能毎の誤り発生件数や日々の作業時間のメトリクスは本論文で使用するメトリクスには含めていない. 本論文では, メトリクスが比較的容易に計測可能であることや, 提案手法を実用化するにあたり, ソースコードを用意すればソフトウェア開発の現場に導入できることを考慮して, 数あるメトリクスのうち, ソースコードから計測可能なソースコードメトリクスメトリクスを利用した.

これまで error-prone モジュールの予測手法については数多くの研究が行われているが, 検査対象プロジェクト毎の特徴を考慮した error-prone モジュール予測器の識別を行う手法の提案は行われていない. また, 予測器は, 予測アルゴリズムとメトリクスを利用した訓練データセットに基づいて構築されるが, これまでの研究では, 検査対象プロジェクトにとって最も適した予測器を構築するための予測アルゴリズムやメトリクスの提案も行われていない.

これらの内容を踏まえ, プロジェクト毎にどのような error-prone モジュール予測器(予測アルゴリズム, 訓練データ)を識別すればよいのかが本論文の主たるテーマとなる. なお, error-prone モジュール予測器は, 予測アルゴリズムと訓練データから構築されるため, 本論文において, 予測アルゴリズムと訓練データを識別することと, error-prone モジュール予測器を識別することは同意となる.

2.2. Error-prone モジュールの予測

Error-prone モジュールの予測に関しては，これまで，数多くの研究が行われ，それぞれ評価されている[3,5-11]．一般的に，error-prone モジュールの予測では，まず，学習アルゴリズム群から予測アルゴリズムを選択する．次に，利用するメトリクス群から， error-prone モジュール予測で説明変数として利用するメトリクスの種別を選択しておく．そして，プロジェクトデータを用意する．プロジェクトデータは，メトリクスを基にした説明変数と誤りの有無を示す目的変数から構成され，以下に示す手順で用意する．

手順1. これまでの開発実績に基づいて誤りの出現が判明しているプロジェクトのソースコード等の成果物を入力としてメトリクス(ここでメトリクスとはソースコード等の諸情報を定量化したものである)を計測する．これら計測したメトリクスデータを説明変数とする．

手順2. バグ管理システムの情報に基づいてモジュール単位に抽出した誤りの有無を目的変数とする．

手順3. これらの説明変数と目的変数を基にプロジェクトデータを用意する．

上記により用意したプロジェクトデータと予測アルゴリズムを用いて error-prone モジュール予測器を構築する．この構築した予測器に検査対象のプロジェクトから計測したメトリクスデータを検査データとして与えることで error-prone モジュールの予測を行う．図 1 に error-prone モジュール予測器を構築するまでの流れを示し，図 2 に error-prone モジュール予測の概要を示す．

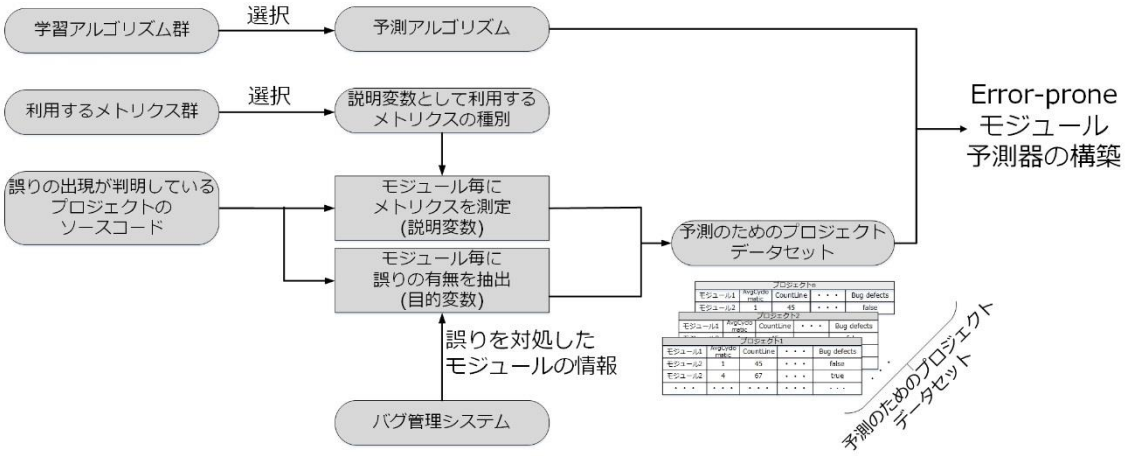

図 1　Error-prone モジュール予測器の構築までの流れ

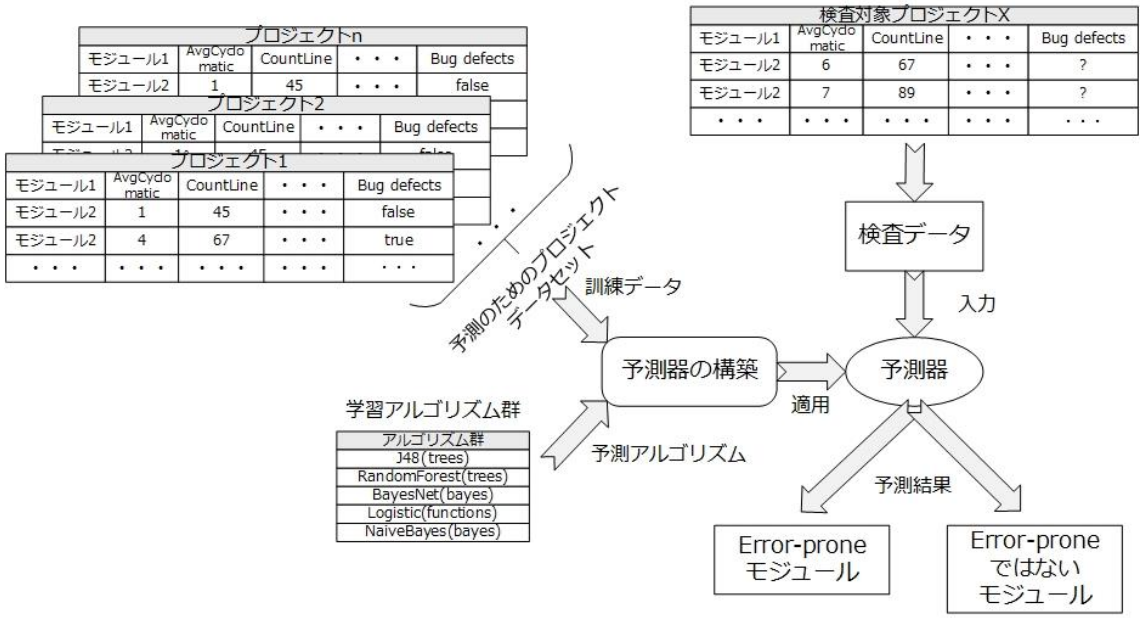

図 2　Error-prone モジュール予測の概要

2.3. メトリクスについて

メトリクスとは，ソフトウェアメトリクスとも言われ，ソフトウェアの数値化指標のことである．例えば，IEEE Standard Glossary[29]によると「システム，コンポーネント，プロセスにおける特定の属性の定量的な測定値」と定義される．主に，ソフトウェア開発工程のいくつかの特質や成果物を計測する標準方法として定義される．なお，ここで述べた特質に対する具体例としては，サイズ，コスト，誤り，困難性などがあげられる．また，成果物の代表例としては，基本設計書，詳細設計書，ソースコード，試験項目表などがある．図 3 に開発工程ごとの作業成果物メトリクス例を示す．

メトリクスは，ソースコードのライン数(行数)の様に，直接計測できるものもあれば，人月(見積もりの単位)のように，成果物から直接計測出来ないものであっても構わない．Fenton[38]は，「ソフトウェアメトリクスは非常に広い意味で使用されており，少なくとも以下に示す 4 種類に分類される．」と指摘している．

(1) 数値(Number)

計測対象から直接計測することで得られる具体的な数値．具体的として，ソースコード行数，リリース後に摘出した誤りの数がある．

(2) 尺度(Scale)

計測のものさしや，計測対象を分類するための基準．具体例としては，混入工程に基づく誤り分類や，重大さに基づく故障分類[39]がある．

(3) 属性(Attribute)

計測対象の特性，あるいは，計測対象を識別できる性質．具体例としては，プログラム複雑度，モジュール間結合度や信頼性(一定期間内に故障を起こさない確率)がある．

(4) モデル(Model)

成果物，開発プロセス，あるいはソフトウェア開発に要する資源における関係を表す論理的モデル，あるいは計測したデータに基づいて構築されたモデル．具体例としては，ソフトウェア信頼度成長モデル[40]，COCOMO[41]モデルがある．

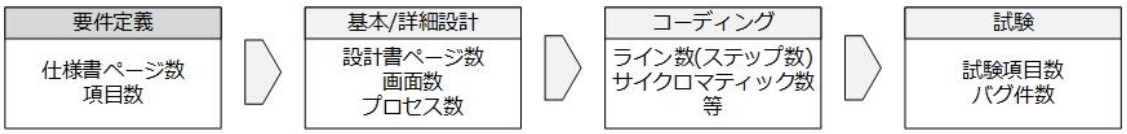

図 3　開発工程ごとの作業成果物メトリクス例

図 3 は，ソフトウェア開発工程毎に分けて，代表的なメトリクス例を示した．メトリクスの主な計測対象は，「成果物」，「(ソフトウェアの開発)プロセス」，「組織」の 3 つがあり，具体例として，以下に示す 3 種類のメトリクスが存在する．

(1) 成果物メトリクス

成果物メトリクスは，ソフトウェア開発における各工程で生産された成果物を対象としたメトリクスである．ソースコードを対象としたコードメトリクスも含まれる．具体例としては，ソースコード行数やファンクションポイント[42]がある．

(2) プロセスメトリクス

ソフトウェア開発は上流工程から下流工程に至る各工程において，多様なプロセスで構成される．プロセスメトリクスは，プロセス全体の他，各作業の状態を計測したメトリクスである．具体例としては，ソースコード 1000 行あたりの開発作業工数やテスト時間あたりの発見誤り数がある．また，change メトリクスも本メトリクスに該当する．Change メトリクス[44]は開発に関わるある属性の時系列に伴う変化を数量化したものである．

(3) 組織メトリクス

組織メトリクスは，組織全体の状態や構成員の状態を計測したメトリクスである．具体例として，作業者の作業経験年数や IT スキル標準[43]がある．

Error-prone モジュールの予測で使用されるメトリクスは，主に，成果物のメトリクスに属する，ソースコードメトリクスが使用される．具体例としては McCabe が提案した循環的複雑度(サイクロマチック数)[23]がある．特に，オブジェクト指向言語を対象としたソースコードメトリクスは，OO(Object Oriented)メトリクスと呼ばれ，Chidamber と Kemerer が提案した CK メトリクス[24]等，数多くのメ

トリクスが提案されている. 以下に代表例として, 循環的複雑度と CK メトリクスの詳細を示す.

- 循環的複雑度(サイクロマチック数)

  プログラムの制御構造に着目したメトリクスとして, サイクロマチック数がある. これは, if 文や while 文などを基にしたソースコードのロジックの複雑さを表したものである. 以下に示す式(1)で求めることが出来る.

$$M = E - N + 2P \tag{1}$$

  M=循環的複雑度(サイクロマチック数)
  E=グラフのエッジ数
  N=グラフのノード数
  P=連結されたコンポーネントの数

  具体例を図 4 に示す. 図 4 は, 左のソースコードに対して右はグラフ化したものである. 図 4 において, 灰色の四角は分岐を伴わない命令文の集まり(基本ブロックと称し, グラフのエッジを意味する)であり, 白抜きの円は分岐を意味する. 図 4 に示した例において, 基本ブロックの数(グラフのエッジ数)は 7 となり, 分岐の数(グラフのノード数)は 6 となる. その結果, 連結されたコンポーネントの数は 1 となるため, 式(1)にこれらの値を代入すると, サイクロマチック数は 3 となる.

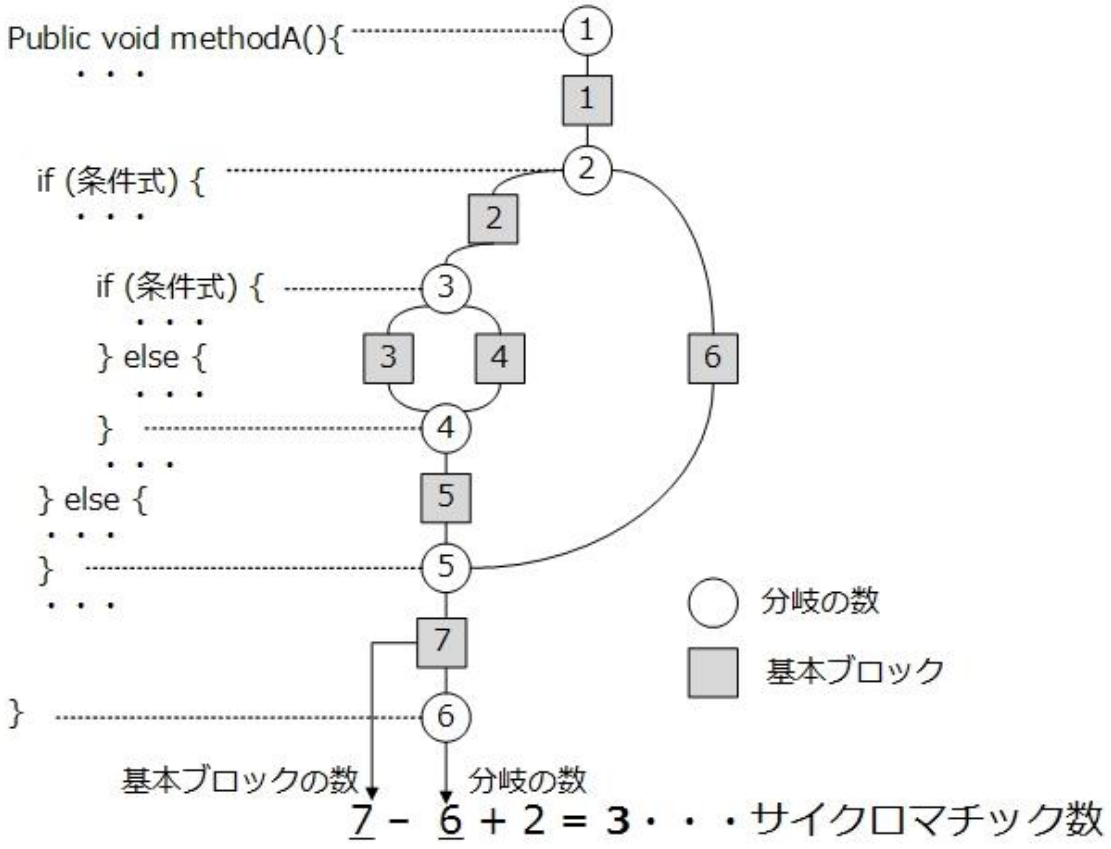

図 4　サイクロマチック数の具体例

- Chidamber と Kemerer[24]のメトリクス(CK Metrics)

オブジェクト指向ソフトウェアに対する複雑度メトリクスとして，Chidamber と Kemerer は表 2 に示す CK メトリクスを提案している．表 2 に示した CK メトリクスの内容のうち，DIT と NOC の具体例を図 5 に，また，RFC と CBO の具体例を図 6 に示す．図 5 において，矢印は継承関係を示し，継承度の深さは 3 となるため DIT は 3 となる．また，クラス C から派生しているクラスは C1 クラス～C3 クラスの 3 つであるため NOC は 3 となる．図 6 において，クラス B は，クラス A から導出され，インスタンス変数 ddd(変数の型はクラス D)，メソッド method_a と method_b をそれぞれ具備する．また，クラス B の method_a の定義において，クラス A のメソッド method_a とクラス D のメソッド method_d をそれぞれ呼び出している．さらに，method_b の定義において，クラス C のメソッド method_c を呼び出している．クラス B は，2 つのメソッドを持っており，クラス A,C,D が持つメソッドをそれぞれ参照しているため，RFC は 5 になる．また，クラス B は，A,C,D の 3 つのクラスをそれぞれ参照しているため，CBO は 3 となる．

表 2　CK メトリクスの内容

メトリクス	説明
WMC (Weight Methods per Class)	各クラスの全メソッドの複雑さの総和．複雑性をすべて同じとした場合はメソッド数となる．
DIT (Depth of Inheritance Tree)	ルートまでのパス長．それぞれのクラスにおける継承度の深さ．
NOC (Number of Children)	それぞれのクラス直下のサブ・クラス(チャイルド・クラス)の数．
CBO (Coupling between Object Classes)	それぞれのクラスと結合しているクラスの総数，すなわちそのメソッドや属性を利用しているクラスの総数．
RFC (Response for a Class)	クラスが外部からのメッセージに対応して実行可能なメソッドの数，すなわち継承も含めてクラスが持つ public なメソッドの数．
LCOM (Lack of Cohesion in Methods)	インスタンス変数を共有していないメソッドの組の数からインスタンス変数を共有しているメソッドの組の数を引いた数．値が負の場合はゼロとする．

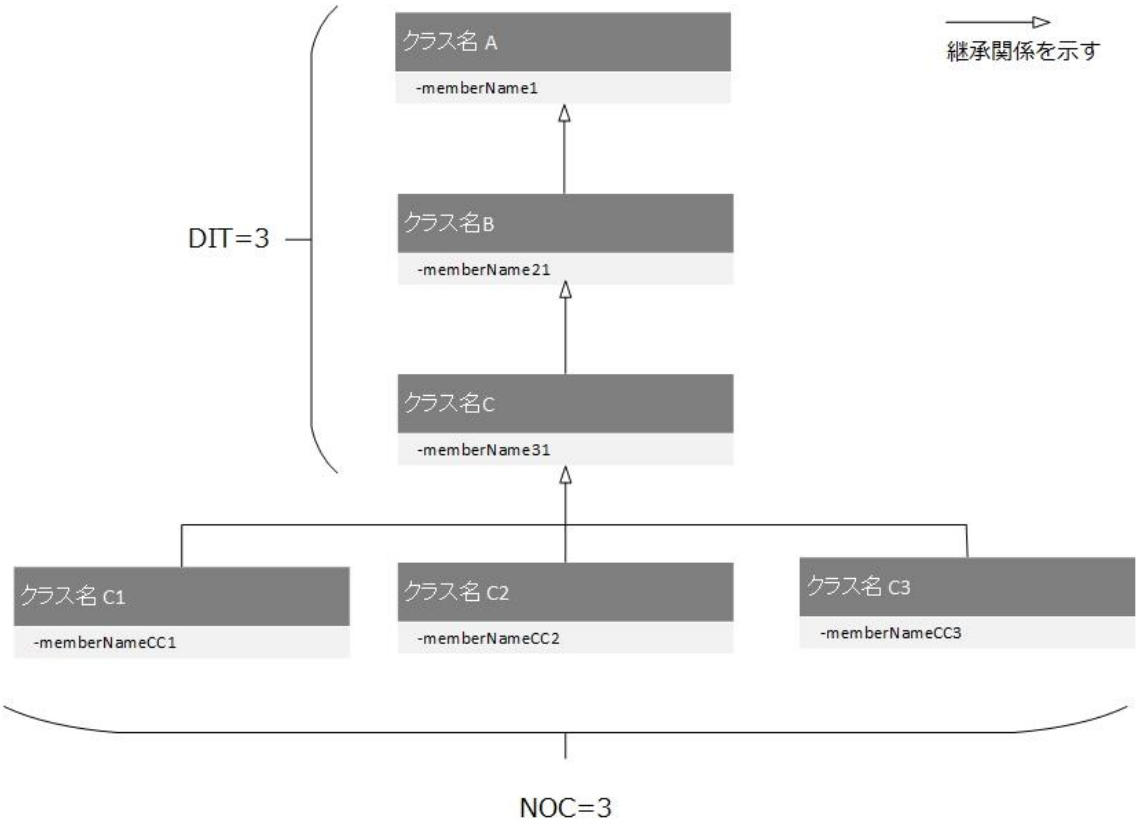

図 5　DIT と NOC の具体例

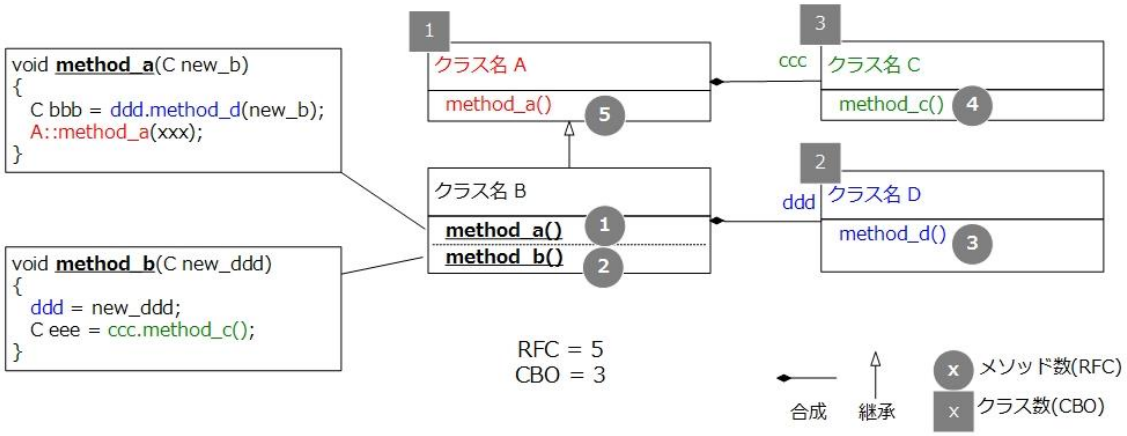

図 6　RFC と CBO の具体例

本論文では，ソースコードを用意すればメトリクスが容易に計測可能であることや，提案手法を実用化するにあたり，ソースコードがあれば容易に導入できることを考慮して，ソースコードメトリクスを採用した．

2.4. 学習アルゴリズム(予測アルゴリズム)

Error-prone モジュールの予測を行うためには，訓練データの他，学習アルゴリズム(予測アルゴリズム)が必要である．学習アルゴリズムの種類は多数あるが，一般的に error-prone モジュールの予測では，以下に示すアルゴリズムが用いられる．

- J48 と Random Forest(決定木による分類を行うアルゴリズム)

  決定木を生成して分類を行うアルゴリズムとして，J48 や Random Forest がある．J48 は C4.5(ロス・キンランが開発した決定木を生成するためのアルゴリズム)を使用して決定木を生成して分類を行う．Random Forest は，ランダムに生成した複数の決定木による分類を行う．

  決定木は，説明変数と目的変数の関係を木構造で表現したモデルである．決定木の各ノードは 2 つ以上の子ノードを持っており，説明変数の値によっていずれかの子ノードへと分岐する． 図 7 に決定木のイメージを示す．

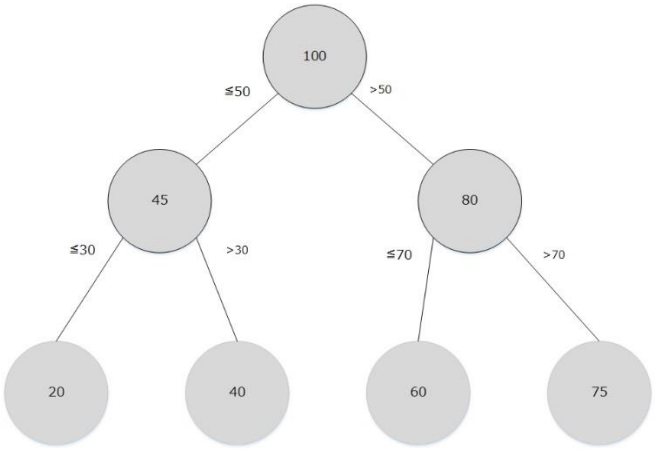

図 7　決定木のイメージ

- ベイズの定理を用いたアルゴリズム

  アルゴリズムの例としては，ベイジアンネットワークやナイーブベイズがある．ベイジアンネットワークは，因果的な特徴を有向グラフによる重み付けグラフとして表現し，その上で確率推論を行うことで，不確実で絶えず変化するために数式で表現が困難なものを予測することができる．これまで蓄積された情報をもとに，起こりうる確率をそれぞれの場合について求め，それらを起こる経路に従って計算することで，複雑な経路を伴った因果関係の発生確率を定量的に表すことが可能となる．ナイーブベイズは，単純ベイズ分類器とも言う．各説

明変数同士にまったく影響が無いという仮定に基づいて比較的少数の訓練事例で学習が可能である. なお, ベイズの定理は式(2)の形となる.

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{2}$$

- 線形判別分析

  線形判別分析では, 判別条件の特性を表す説明変数の線形結合により表される. 各条件においてn個の説明変数の値が計測されているとき, 線形判別式は式(3)の形となる.

$$Z = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n \tag{3}$$

- ロジスティック回帰分析

  ロジスティック回帰分析では, 説明変数の変動によって目的変数が2値のどちらを取る確立が高いかを予測する手法である. 具体的には, n個の説明変数の値が計測されているとき, 判別式は式(4)の形となる.

$$P(y|x_{1,\cdots,x_n}) = \frac{1}{1 + e^{-(\alpha 1 x 1 + \cdots + \alpha n x n + \beta)}} \tag{4}$$

  ここで, $y \in \{0,1\}$は目的変数, x_i は説明変数, α_i は判別係数であり, $P(y|\ x_1, \cdots, x_n)$は, 説明変数の値の組 $x_1, \cdots, x_n$ に対して, y が 1 の値をとる条件付き確率である.

2.5. プロジェクトデータのメトリクス計測と誤り有無の判定について

一般的に, プロジェクトデータを構築するためのメトリクス計測は誤りが改修されたリビジョンよりも1つ前のリビジョン(誤りを含んだ世代)におけるソースコードを対象に行われる. また, モジュール毎における誤り有無の判定は, バグ管理システムに記録された誤り改修情報とバージョン管理システム(Subversion 等)に記録されているソースコードの編集履歴を対応付けることで, 誤りの改修モジュールを特定し, 誤り改修時期及び対応リビジョンを特定する. 誤り改修時のリビジョンが特定できれば, その一つ前のリビジョンがメトリクス計測の対象リビジョンとなる.

2.6. Error-prone モジュール予測の根拠

Error-prone モジュール予測では，前述した様に，学習アルゴリズム群から選択した予測アルゴリズムと，過去に開発されたモジュールのソースコードから計測したメトリクスを説明変数とし，そして各モジュールにおける誤りの有無を目的変数とした予測器を構築する．構築した予測器に対して，検査対象プロジェクトのモジュールから計測したメトリクスデータを入力することで，該当モジュールにおける誤りの有無が予測される．本節では，メトリクスに基づく誤り有無の予測が可能であると考える根拠を述べておく．

ソフトウェア開発において誤りが混入する要因は 1 つのときもあれば，複数の要因が絡み合うこともある．誤りの混入に至る要因として考えられる要素を，以下に「要因 1」～「要因 5」として示すとともに，誤りと諸要因との因果関係を体系的に整理した特性要因図を図 8 に示す．

要因1. コメントに関する要因

ソースコードに記述したコメント内容が抽象的もくしは難解であるケースや，コメント内容とソースコードの不一致により，作業者が仕様を誤解してしまい，その結果，誤りの混入を犯してしまうケースが該当する．

要因2. コーディングに関する要因

プログラミング言語は，日常，我々が使用する言語に近いものが大半であるが，一部の言語では記号や簡略表記が可能であり，プログラミング言語に疎い作業者が改造を行う場合，コーディング誤りを誘発する懸念がある．また，プログラム構造が良形ではないケースや，複雑である場合においても，仕様誤解やプログラム言語の理解不足に起因した誤りの混入を犯すことがある．

要因3. プログラム仕様に関する要因

仕様書の品質が悪いために書くべき仕様が明記されていないケースや，詳細仕様まで十分に具体的な検討が出来ていないケース，或いは，複雑な仕様であることに起因して，仕様誤解や考慮不足による誤りの混入を犯すケース等が該当する．

要因4. 作業者に起因する要因

短期開発を顧客から要望されることや，上流工程の作業遅延に起因して下流工程の期間が圧迫されることがある．この場合，作業者は過重な負担が強いられることになり，誤りの混入を犯すことがある．また，ソフトウェア開発の経験が浅いことによるスキル不足のために誤りの混入を犯すこともある．

要因5. 構成管理面における要因

該当モジュールやクラスに対して，仕様変更等による変更が頻繁に行われたために，プログラム構造が複雑になり，理解容易性が損なわれたことに起因して誤りが混入するケースや，複数の担当者で変更を行ったことにより，本来持つべき機能が不完全となり，仕様通りに動作しないケースが該当する．

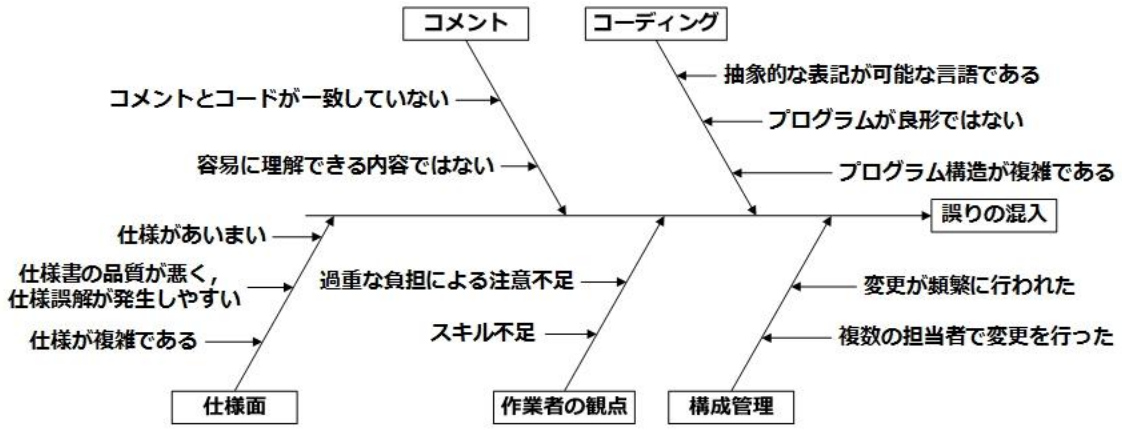

図 8　誤りと混入要因との因果関係

図 8 に示す因果関係が存在し，それぞれの要因をメトリクスで計測できることが出来れば，各要因に関係するメトリクスから誤りの有無は予測可能となる．これが，メトリクスに基づく誤りの有無の予測が可能となると考える根拠である．

本論文では，ソースコードからメトリクスを比較的容易に計測することができることや，提案手法を実用化するにあたり，ソースコードを用意すれば導入できることを考慮して，ソースコードメトリクスを利用する．ソースコードメトリクスは，図 8 において，「コーディング」に関する誤り混入要因を計測することが可能である．特に，プログラム構造の複雑度は優れた予測因子であり得ることを示す調査研究[36,37]もあり，error-prone モジュールの予測においてソースコードメトリクスを利用することは有意であると考える．

但し，これまでメトリクスはこれまで色々と提案されているが，必ずしも，すべての誤り混入要因を直接的に計測できる訳ではなく，各要因に対して間接的に関与すると考えられるメトリクスも存在する．例えば，「プログラム構造が複雑である」という要因はソースコードメトリクスのひとつである，「Cyclomatic 複雑度」で計測可能である．しかし，複雑であることを比例的に定量化できているとは限らない．また，「過重な負担による注意不足」を計測するためのメトリクスとしては，月当たりの超勤時間が考えられるが，注意不足か否かまで計測できているとは言い難い．従って，間接的ながら注意不足に陥っていたものと仮定するのが限界である．

2.7. Error-prone モジュールの予測における評価方法

Error-prone モジュール予測において，誤りを含む可能性がある場合は目的変数を true とし，誤りを含まないと判断できる場合は目的変数を false として，error-prone モジュールの予測を行い，結果として得られた混同行列[1]から予測精度を算出し，その値を評価する．混同行列の詳細を表 3 に示す．予測精度自体については，recall，precision，f-measure，AUC(Area Under the Curve の略であり，0 から 1 の値をとり，1 に近いほど精度が高いことを示す)等，いろいろな評価基準が提案されている[3,4,7,8]．参考として，表 3 を用いて recall, precision, f-measure の定義を以下に示す．

予測精度として求めた precision の値が高くかつ，recall の値が低い場合，error-prone モジュールの予測結果として無駄は少ないが取りこぼしが多い状態であると言える．また，precision の値が低く，recall が高い場合は，error-prone モジュールの予測結果は期待通りの内容を網羅できているものの，無関係な予測結果も大量に存在する状態と言える．つまり，precision を高くしようとすると recall は低くなり，recall を高くしようとすると precision が低くなるため，precision と recall はトレードオフの関係となり，単純にどちらかを高くするともう一方が低くなりがちとなる．評価基準が precision と recall のどちらを採用するのか決まっていない場合，precision と recall の調和平均を取った値を f-measure(F 値)として採用することもある．F-measure が 1 に近ければ予測性能が良いことを示す．但し，f-measure は後述する式(7)にて precision と recall の調和平均として求められ，1 に近いことを理想値とするが，recall と precision が共に 1 となることはないため f-measure が 1 になることも無い．

表 3　混同行列(予測と実際の関係マトリクス)

	True と予測	False と予測
実際は True	True positive(TP)	False negative(FN)
実際は False	False positive (FP)	True negative(TN)

- precision(適合率)

  precision は，true(誤りを含むモジュール)と予測したモジュールのうち，実際に誤りを含むモジュールである割合を示す．表 3 で示す記号を用いて式(5)で定義される．

$$precision = \frac{TP}{(TP + FP)} \tag{5}$$

[1] 予測した全てのケースについて，予測値が実際の値と一致した件数と一致しなかった件数を分類した表を混合行列という．

- recall(再現率)

  recall は，誤りを含む全てのモジュールのうち，true(誤りを含むモジュール)と予測した割合を示す．表 3 で示す記号を用いて式(6)で定義される．

$$precision = \frac{TP}{(TP + FN)} \tag{6}$$

- f-measure(F 値)

  F 値(f-measure)は，適合率(precision)と再現率(recall)の調和平均によって求めることができる．再現率を上げるには，数多く error-prone モジュールの予測を行えばよいが，正しく予測されない結果も増加するため，適合率は下がる．つまり，適合率と再現率はトレードオフの関係があると言える．そこで，適合率と再現率のバランスを評価する目的で f-measure を確認することがある．F-measure は，式(7)として定義され，0 から 1 までの値をとり，1 のときが最も精度が高く，0 のとき最も低い．

$$F = \frac{2 \times precision \times recall}{(precision + recall)} \tag{7}$$

2.8. Error-prone モジュールの予測の問題点

Error-prone モジュール予測はこれまで，数多くの手法が提案され，評価されている．しかし，これまで検査対象プロジェクトに適した error-prone モジュール予測器は，検査対象プロジェクト毎に異なり，いかなる検査対象プロジェクトにおいても適用できる汎用的に最適な予測器は存在しない[3].

Lessmann [3] では 10 種類のデータセットと 22 種のアルゴリズムによる実験を行い，結果を AUC 尺度を使って評価を行っている． Briand [12] では「Logistic regression」と「Multivariate adaptive regression Splines」による2つのアルゴリズムの比較を行っている．これらの研究では，学習アルゴリズムの比較検討を行っているが，いずれの研究においても，データセットによって最適なアルゴリズムは異なる結果となっている．また，Tim [18]では J48 と naive bayes の比較を行っているが，これらの学習アルゴリズムにおいて明確な相違は見出していない．Erik [7], [8] ではアルゴリズムのみならず，メトリクス，そして評価尺度の影響も考察している．しかし，最適なアルゴリズム，メトリクス，評価尺度については見出していない．

Zimmermann [19]によると，12 個の実アプリケーションに対して，622 種のクロスプロジェクトで

の予測実験を行ったところ，十分な精度で予測できた組み合わせは僅か 3.4%であった．クロスプロジェクト間での予測において高い予測精度が確保できなかった理由として，Zimmermann ではプロジェクトの特性の違いやデータセット中におけるメトリクスの分布が異なる点を挙げている．

予測器の評価方法に関しても，2.7 で述べた通り，種々の考え方が提案されており，どのような評価方法を使用するのかによっても，どのような予測器が適しているかは異なるといわれている．

Ekanayake[20] では予測精度に影響を与えるいくつかの要因について考察したところ，予測器の正確さに関して，開発の段階によって相違があり，この相違が予測精度に影響を与えていると述べている．そして，予測器を評価する手法を提案している．しかしながら，この研究では評価は行えるが，実際に最適な手法を探索する事はできない．

また，D'Ambros[16]は「我々の実験結果が示すように，あるアプローチは他のアプローチよりも統計的に優位であると示されているが，欠陥予測における外的妥当性はまだ研究途上である．従って得られた結果を他のドメインや学習機に適用するには更なる研究が必要である．」と述べ，検査対象プロジェクト毎に適切な予測器の探索が必要であることを示している． Ma[17]では，予測器の正確な定義と評価，そして予測器を作るためのソフトウェアデータベースの特徴の考察が必要であるとしている．

このように検査対象プロジェクトに適した error-prone モジュール予測を行うための予測器すなわち，予測器を構築するために必要となる訓練データと予測アルゴリズムは，検査対象プロジェクトによって異なるといわれている．しかしなから，検査対象プロジェクトに対して，最適(もしくは適切)な訓練データと予測アルゴリズムをどのようにして選ぶかについては，ほとんど研究が行われていない．

2.9. 予測のためのプロジェクトデータセットに関する補足事項

予測のためのプロジェクトデータセットに関しては，一般的には自プロジェクトにおける過去バージョンに対して計測したメトリクスデータとモジュール毎における誤りの出現有無を用いる[45,46,47,48]．しかし，必ずしも自プロジェクトの過去バージョンにおけるデータが最適ではないという研究事例[4]もある．

2.10. 結言

本章では，メトリクス，学習アルゴリズム及び error-prone モジュール予測の概要を述べるとともに，error-prone モジュール予測が可能となる根拠として，誤りの混入要因をメトリクスとして計測することで誤りの有無を予測することが出来る可能性を示した．また，error-prone モジュール予測器について，これまで数多くの研究にて色々な手法が提案され評価されているものの，全ての検査対象プロジェクトに対して一意に最適な予測器は実在しないことを述べた．それに加えて，error-prone モジュール予測を行うための予測器を構築するために必要となる訓練データと予測アルゴリズムについては，検査対象プロジェクトデータに対してどのようにして選ぶかの研究は，ほとんど行われていないことを指摘した．

上記を踏まえ，実際の開発現場において，検査対象プロジェクト毎に最適な error-prone モジュール予測器(検査対象プロジェクト毎に適切な error-prone モジュール予測器を構築するための予測アルゴリズムと訓練データ)を選択するためにはどのような手法を適用すればよいのか．これが本論文で解決しようと取り組む課題である．以下，3 章では，本論文で提案する手法の基になった HE[4]が提案する手法及び，HE が提案する手法に改良を加えた，本論文が提案する手法について説明する．

第 3 章

Error-prone モジュール予測器のマイニング

第3章　Error-prone モジュール予測器のマイニング

3.1. 緒言

3 章では，前提知識の説明及び，2 章で述べた error-prone モジュール予測手法の問題点を解決するための提案手法について説明を行う．

前章で述べた通り，これまでの研究において，検査対象プロジェクト毎に最適な error-prone モジュール予測器は異なるが，個々の検査対象プロジェクトに適した error-prone モジュール予測器を識別する手法を研究することはほとんど行われていない．検査対象プロジェクトに対して最適な予測器を識別するにあたり，検査対象プロジェクトの特徴を考慮した要素を説明変数として加えることにより，最適な予測器を識別できると考える．よって，本論文では，検査対象プロジェクトの特徴を考慮した要素を特徴量として定義し，検査対象プロジェクト毎に最適な error-prone モジュール予測器を識別する際に，特徴量を説明変数として加えることにした．

本論文では，いかなる検査対象プロジェクトに対して最適となる error-prone モジュール予測器は存在せず，最適となる error-prone モジュール予測器は，検査対象プロジェクト毎に異なることを前提として，特徴量を考慮したマイニング手法[2]による予測器の探索手法(識別手法)を提案する．なお，error-prone モジュール予測器は，訓練データと予測アルゴリズムから構築されることから，提案手法の識別対象は，訓練データと予測アルゴリズムとなる．すなわち，検査対象プロジェクトに最適な error-prone モジュール予測器構築するための訓練データと予測アルゴリズムをマイニング手法により探索（本論文では method mining と称する)し，探索した訓練データと予測アルゴリズムを用いて error-prone モジュール予測器を構築することで，検査対象プロジェクトに最適な error-prone モジュール予測器を導出することが可能となる．これが，「method mining に基づく error-prone モジュールの予測」の概要である．

3.2. 特徴量について

2.6 で述べた通り，error-prone モジュール予測は，選ばれたメトリクスと誤りとの間にある相関性に基づいている．具体的には，メトリクスは対象の複雑さを定量化しており，この複雑さが誤りの有無に関係するという推論に基づいている．しかしながら，この相関性には検査対象プロジェクトの種々の特性が影響すると考えられる．すなわち，必ずしもメトリクスだけに依存するものではなく，他の要素として，検査対象プロジェクトの特性が絡むと考えられる．検査対象プロジェクトの特性の具体例として，誤り率が考えられる．開発途中のプロジェクトと，既に開発は完了したソフトウェアのリリースを終えたプロジェクトでは，誤りの摘出数は異なる．一般的に，開発完了プロジェクトの場合，

[2] 大量のデータから価値ある情報を探索する意味で用いられる．

一通りの試験は完了しており，品質は安定しているため，開発途中のプロジェクトと比較して，誤りの摘出件数は少なくなる傾向がある．また，プロジェクトの規模や開発対象のアプリケーションの分野・種別によっても，プロジェクトの特性は変わると考えられる．

Ekanayake[20]は，予測精度に影響を与える要因として，編集を行った SE の数，ファイルに対するこれまでの誤り修正の数などをあげている．しかしながら，error-prone モジュール予測はモジュール単位であるので，プロジェクト単位のプロジェクト固有となるデータは使われない．一般的には error-prone モジュールの予測ではプロジェクト単位の固有な値は考慮されていない．これら考慮されない要素を本論文では特徴量として定義し，プロジェクト単位に存在し，かつ利用するメトリクスには現れない要素とする．

なお，本論文では特徴量として，以下述べる 3 つを選定した．

- モジュール数

  モジュールの数が多くなれば，プログラムの構造も複雑になると考えられる．プログラムの構造が複雑であれば，2.6 で述べたように，誤りが混入する可能性が高くなり，モジュール数と誤り件数は相関関係があると考え，特徴量として選定した．
- 誤り率(誤り密度[3]が 10%以上，10%未満かつ 1%以上，1%未満の 3 種類を 0～2 の数値で表現する)

  開発初期段階であれば，誤りは多発し誤り件数は増加する傾向となり，納入後(もしくはファイルリリース後)から数ヶ月間は比較的，少ない数の誤りが発生し，納入後(もしくはファイルリリース後)から半年経過した以降では，品質は安定しており，誤りの摘出件数は低い傾向となると考える．これらの傾向はプロジェクトの特徴と捕らえることができることや，error-prone モジュール予測を行う必要が生じた段階で，その時点の時期で 3 区分のうち，どの区分に属するのか判断可能である．また，誤り摘出数との因果関係があり，error-prone モジュール予測の一助になると考え特徴量として選定した．なお，誤り率の段階を判断するため基準とした，「10%以上，10%未満かつ 1%以上，1%未満」は経験に基づいた予測値である．なお，3 段階による値ではなく，誤り率そのものを使用する方法も案としては考えられる．しかし，検査対象プロジェクトは誤りの出現が不明な状態で error-prone モジュールの予測を行うため，誤りの摘出数を求めることはできず，誤り率も算出することは困難である．従って，誤り率そのものを特徴量として使用することはできない．これに対して，三段階の表記では，検査対象プロジェクトの現時点の開発状況から，これら 3 つのうち，ど

[3] 誤り密度(単位は%)は，(摘出した誤り件数÷該当プロジェクトの規模)×100 で求めることが出来る．

れに該当するのか想定することは可能である. 従って, 今回は三段階表記とする方法を採用した.

- 開発言語

  開発言語の特性により, 誤りの発生傾向は異なると想定し, 開発言語も検査対象プロジェクトの特徴を現す要素と考え特徴量として採用した. 例えば, Java 言語にはポインターの概念は無いが, C/C++にはポインターの概念がある. ポインター操作はメモリ上のアドレスを操る操作であり, 操作を誤るとソフトウェア割り込みが発生することもあり, 経験上, 誤りの混入が発生しやすい箇所である.

3.3. N 重交差検証(N fold cross validation)

N 重交差検証では, 一般的に N=10 が採用され, 10 重交差検証として実施される. 10 重交差検証は, 訓練データを 10 分割し, 9 個のデータを使用して予測器を構築し, 残りの 1 つのデータで検証を行う方法である. これを 10 回繰り返し, 10 回の検証で得られた予測精度のうち, その平均を結果とする. 検査データを用意することなく, 検証を行うことが出来る. 図 9 に 10 重交差検証の概要を示す.

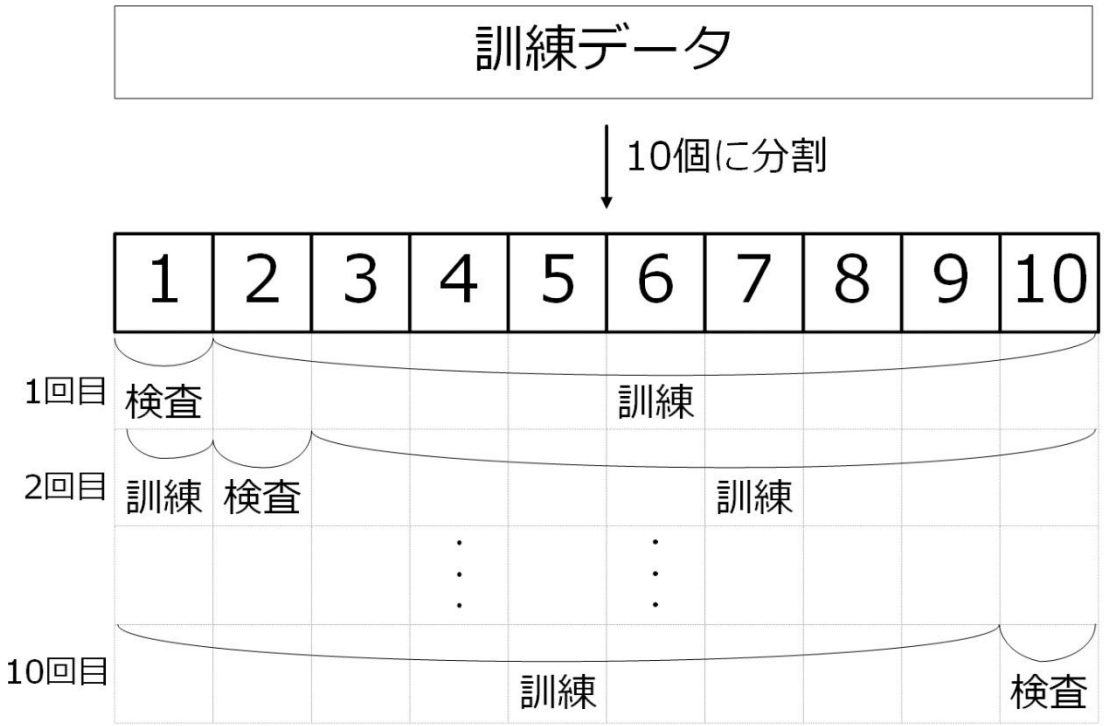

図 9　10 重交差検証の概要

3.4. HE が提案する手法

3.2 で述べたように, 一般的に, error-prone モジュール予測において, プロジェクト単位の特徴量は考慮されていない. 従って, 検査対象プロジェクトに対して適切な error-prone モジュール予測器を構築するためには, プロジェクトの特性を考慮した特徴量を使用することが不可欠である. これまで, プロジェクト単位の特性をあらわした要素を間接的に取り入れる手法として, 開発手法や開発規模等で類似しているプロジェクトのメトリクスデータを訓練データとする試み[33]や, 似ているように補正するという考え方[34]は存在したが, 予測器の識別に特徴量を適用するという考え方はなかった.

HE[4]は, メトリクスをプロジェクト単位に集約した, メトリクスの集約値と error-prone モジュールの予測結果の予測精度の関係を学習させることで, 検査対象プロジェクトに対して適切な予測器を見つけ出す手法を提案している. 以下に HE が提案した手法の手順を示すと共に, 図 10 から図 14 に各手順を図示した内容を示す.

手順1. 複数のプロジェクトから, 誤りの出現有無が判明しているモジュール毎にメトリクスを計測し, 予測のためのプロジェクトデータセットとして用意する (図 10 の手順 1).

手順2. 予測のためのプロジェクトデータセットを構成するプロジェクト毎に総当りで予測アルゴリズムを用いて, error-prone モジュールの予測実験を行い, 予測結果として得られた混同行列を基に precision と recall を算出する(図 10 の手順 2).

手順3. 予測のためのプロジェクトデータセットを構成する各プロジェクトのモジュール単位に計測したメトリクスの値から, 最頻値, 中央値(メジアン), 平均値等の値を算出し, モジュール単位からプロジェクト単位にメトリクスデータを集約させる(図 11).

手順4. 手順 2 の結果から,「recall≧0.7 and precision≧0.5」の条件を満たす場合は, 目的変数を true とし, 満たさない場合は false を設定する(図 12 の手順 4).

手順5. 手順 3 でプロジェクト単位に集約したメトリクスデータ及び予測アルゴリズムを使用して, (プロジェクト A のメトリクスデータ, 予測アルゴリズム, プロジェクト B のメトリクスデータ, 予測結果)の 4 つ組のデータの総当りとなる組み合わせを用意する(A と B は, プロジェクトの総数を N とすると, N から 1 までの任意の整数となる. 但し, A と B は同じ値は指定しない). これらのデータは, 予測のためのプロジェクトデータセットを構成するプロジェクト数分, 総当りで用意し, 識別器を構築するための訓練データとする(図 12 の手順 5).

手順6. 検査データとして, (手順 3 で集約したデータセットから取り出したプロジェクト X のメトリクスデータ, 予測アルゴリズム, 検査対象プロジェクトのメトリクスデータ,予測結果はダミー値)の 4 つ組のデータを用意する(図 13 の手順 6).

手順7. 手順 5 で用意した訓練データを基に識別器を構築し，検査データを入力して，最適な予測器を構築するための訓練データと予測アルゴリズムを識別する(図 13 の手順 7).

手順8. 識別した訓練データと予測アルゴリズムを用いて，error-prone モジュール予測器を構築する. そして，検査対象プロジェクトからモジュール単位に計測したメトリクスデータを入力し，それぞれのモジュールが error-prone モジュールかどうか予測する(図 14).

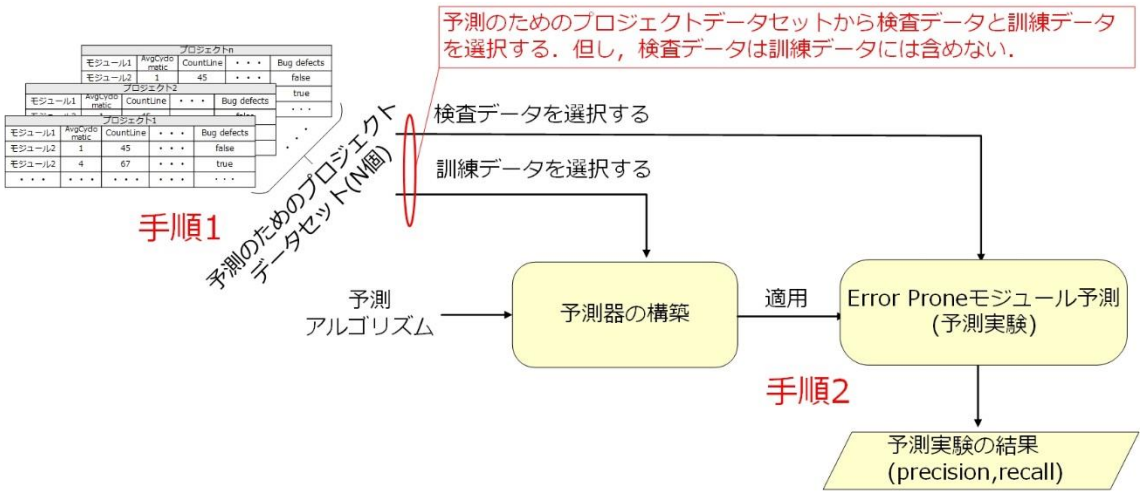

図 10　HE の提案手法(手順 1～手順 2)

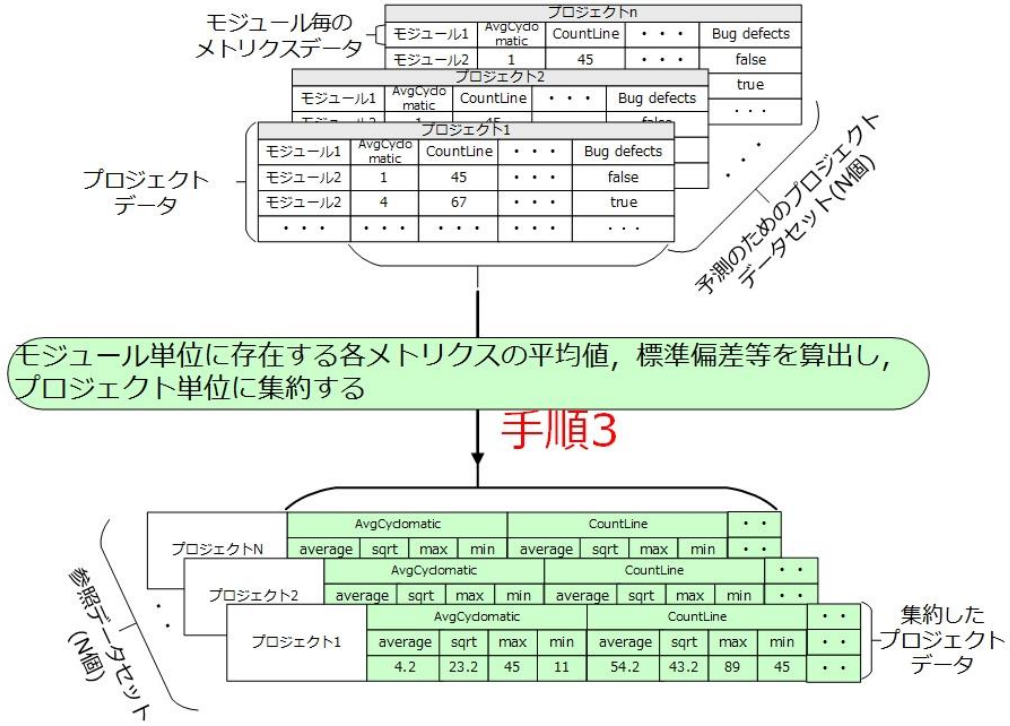

図 11　HE の提案手法(手順 3)

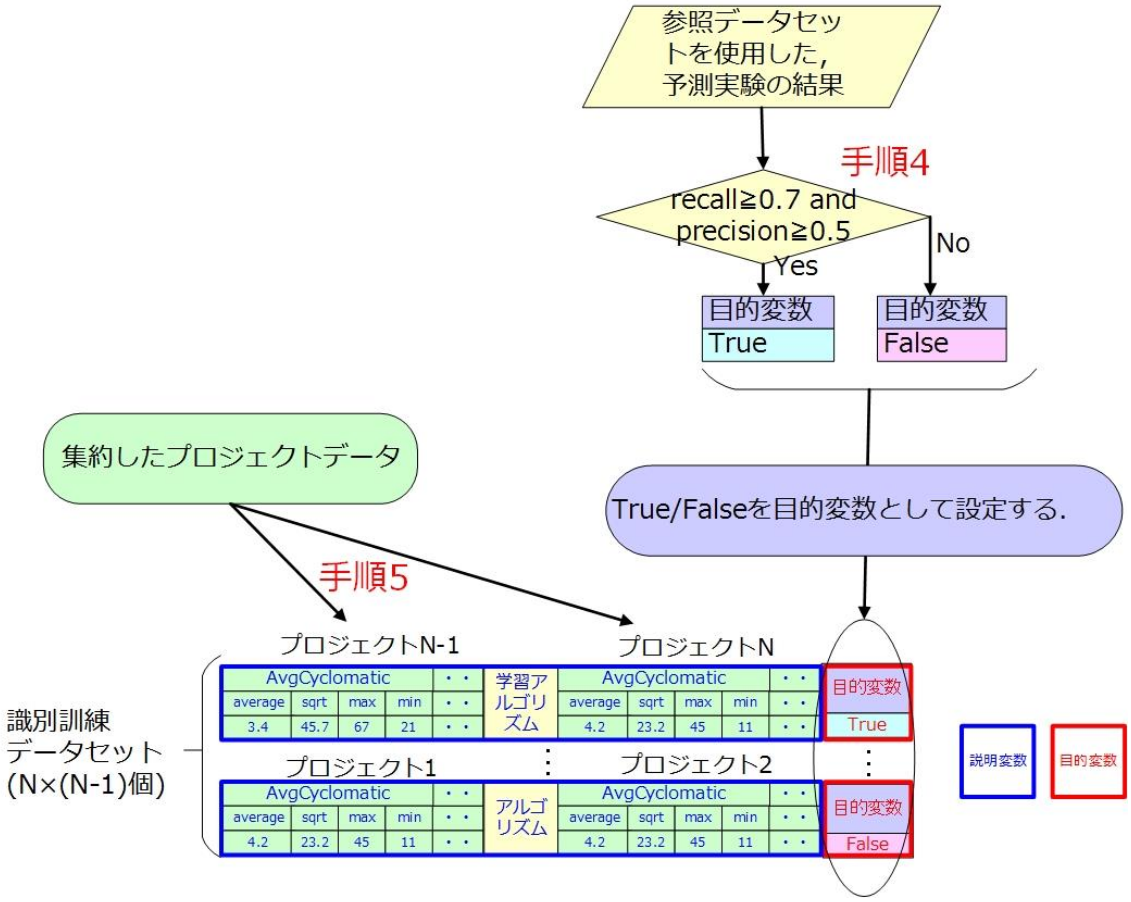

図 12　HE の提案手法(手順 4～手順 5)

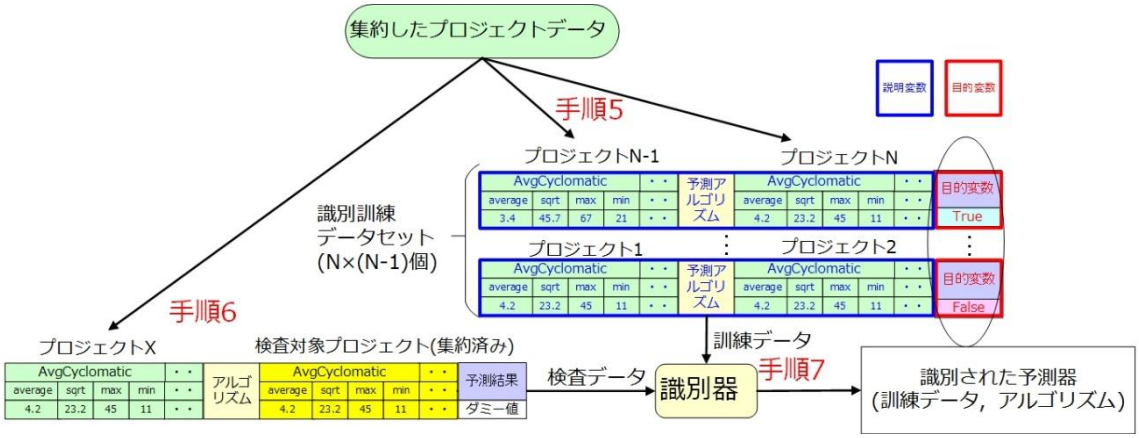

図 13　HE の提案手法(手順 5～手順 7)

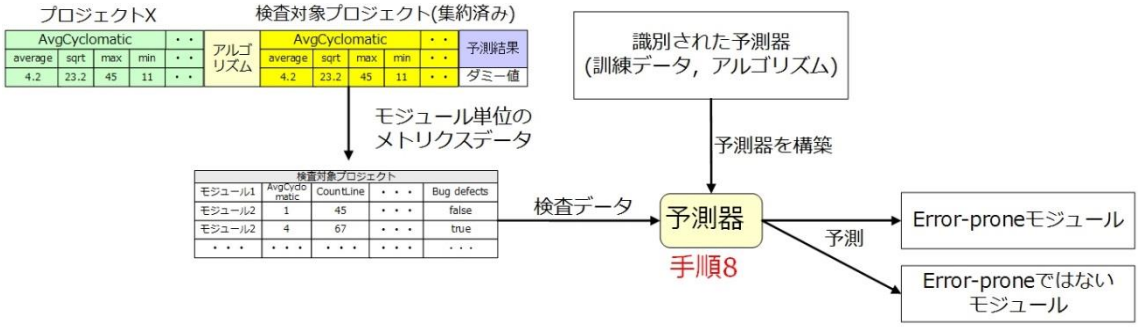

図　14 HE の提案手法(手順 8)

3.5. 本論文の提案手法

HE の提案手法では，error-prone モジュールの予測結果が良好となる可能性の高い予測アルゴリズムや訓練データを決定する過程において，メトリクスに依存しない外部要因(特徴量)については考慮されていない．また，予測結果の true/false[4]を決定する予測閾値についても recall ≧ 0.7 and precision≧ 0.5 という単一の基準のみでのマイニングであるため，現場が求める error-prone モジュールの予測精度を満たす必要性を考慮すると，十分な手法とは言えない．現場では，いろいろな利用シーンにおいて求められる error-prone モジュールの予測精度は異なる．例えば，納入まで残り少ない時期であれば，厳密な予測精度を求めることで，インスペクションやテストの範囲を現実的な日程で対応できる分量の結果が得られるようにするであろう．また，コーディング作業が完了した直後であれば，インスペクションやテストに費やすことの出来る日程は比較的余裕があると考えられ，厳密な予測精度でなくともよいと考えられる．このように，その時々の状況に応じて弾力的に予測閾値を変更できる対応は必要である．また，予測閾値を固定化した場合，最適な予測器が識別できないことも想定されるため，弾力的に予測閾値を変更できる機構は必要と考える．

上記を踏まえ，HE が提案した手法を改良し，特徴量を導入して，かつ，検査対象プロジェクト毎に適切な基準(以降，予測閾値と称す) を決めて，検査対象プロジェクト毎に最適な error-prone モジュール予測器を構築できる訓練データと予測アルゴリズムを探索しながら識別できるマイニング手法を提案する．

3.5.1. 一般的な予測のためのプロジェクトデータセットの作成について

提案手法を述べる前に，本項では，提案手法を実運用で適用したと想定した場合における，予測のためのプロジェクトデータセットの一般的な作成手順を述べる．なお，予測のためのプロジェクトデータセットは，過去の開発実績からモジュール毎に誤りの有無が判明しているソースコードから計測したメトリクスデータを説明変数とし，誤りの有無を目的変数としたデータである．Error-prone モジュールの予測対象となる，検査対象プロジェクトにおいて，過去リリースファイルにおける実績が取得できる場合は，過去の開発におけるデータを予測のためのプロジェクトデータセットに含める必要がある．

[4] true は error-prone モジュールを意味し， false は error-prone モジュールではないモジュールを示す．

- 予測のためのプロジェクトデータセット

バグ管理システムから，誤り改修の時期，ソースコードリポジトリ(CVS/SVN)にコミットしたときのリビジョン，モジュール名をそれぞれ抽出する．ソースコードリポジトリから，誤り改修前のリビジョンにおけるソースコードを取得し，プロジェクトにおける全ソースコードに対してソースコードメトリクスを計測する．そして，バグ管理システムから抽出した誤りを改修したモジュール名に基づき，誤りを含むモジュールには目的変数に true を設定し，誤りを含まないモジュールには目的変数に false を設定する．本手順は，複数のプロジェクトを対象に，プロジェクト単位に行う． 図 15 にプロジェクトデータの準備概要を示す．

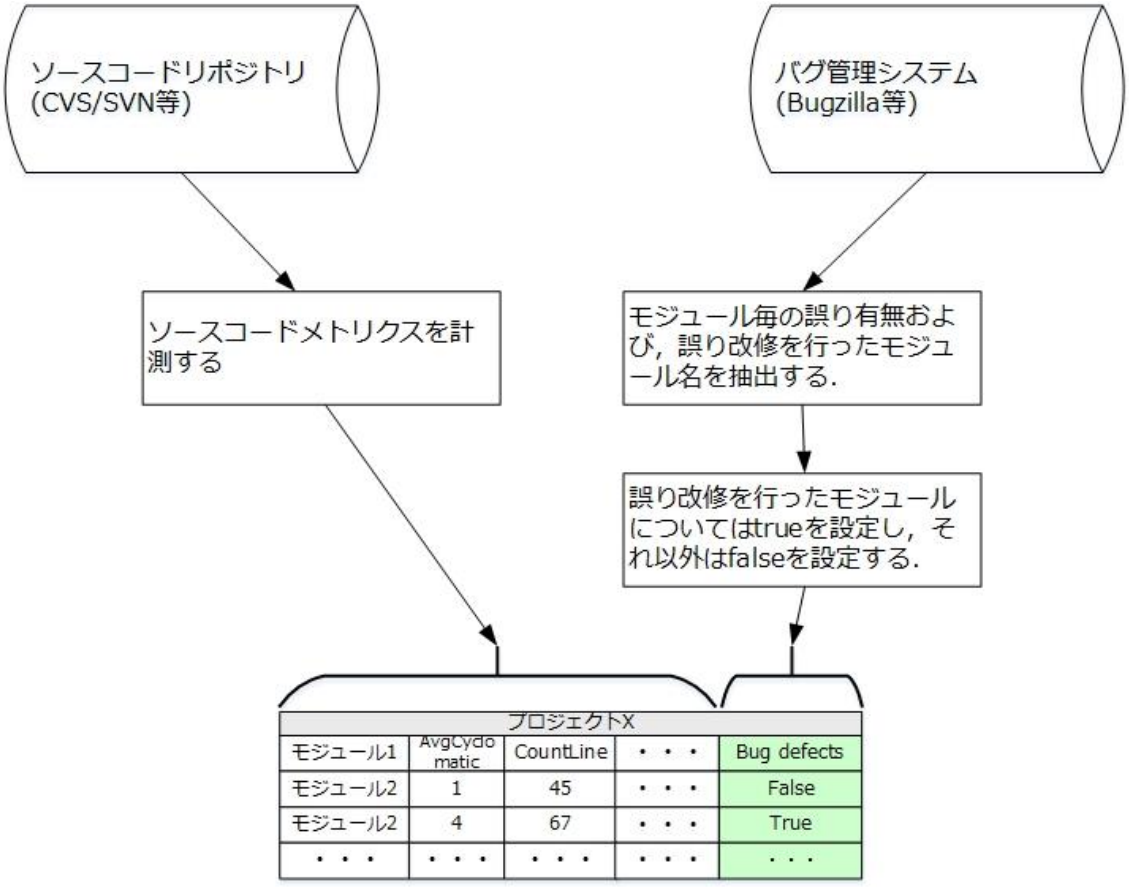

図 15　プロジェクトデータの準備

3.5.2. 提案手法の詳細

提案手法は，図 16に示す通り，大きく分けて3つのステップから構成される．予測のためのプロジェクトデータセットと検査対象プロジェクトのソースコードを入力とし，検査対象プロジェクトに対する error-prone モジュールの予測結果を出力とする．

Step1. 識別器の構築

Step2. 識別器による予測器の識別

Step3. 予測器による(error-prone モジュールの)予測

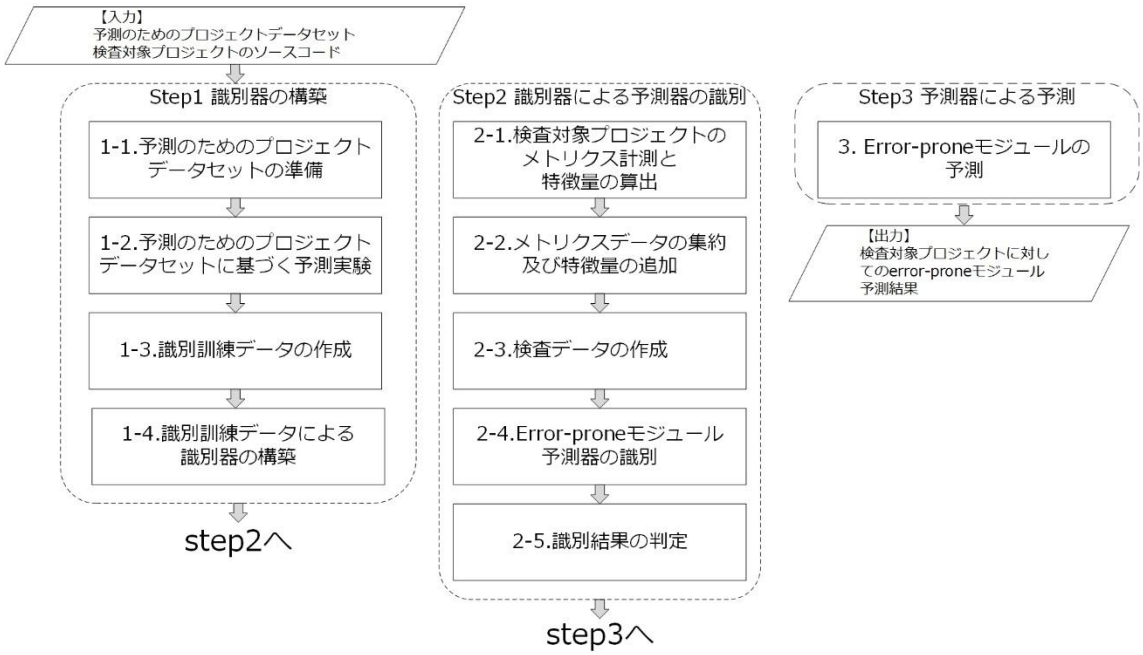

図 16　提案手法の概要フロー

上記を踏まえ，以下に各手順の詳細を説明する．

Step1.識別器の構築

本ステップでは，検査対象プロジェクトに対して最適な error-prone モジュール予測器を識別するための識別器を構築する．主に後述する 1-1 から 1-4 までの手順で構成される．

1-1.　予測のためのプロジェクトデータセットの準備

複数のプロジェクトから，誤りの出現有無が判明しているモジュール毎に計測したメトリクスデータを収集する．これらの収集したメトリクスデータと誤りの有無を含めて予測のためのプロジェクトデータセットとして用意する．

1-2.　予測のためのプロジェクトデータセットに基づく予測実験

予測のためのプロジェクトデータセットを構成するプロジェクト毎に総当りで予測アルゴリズムを用いて，error-prone モジュールの予測実験を行い，予測結果として得られた混同行列を基に precision と recall を算出する(図 17).

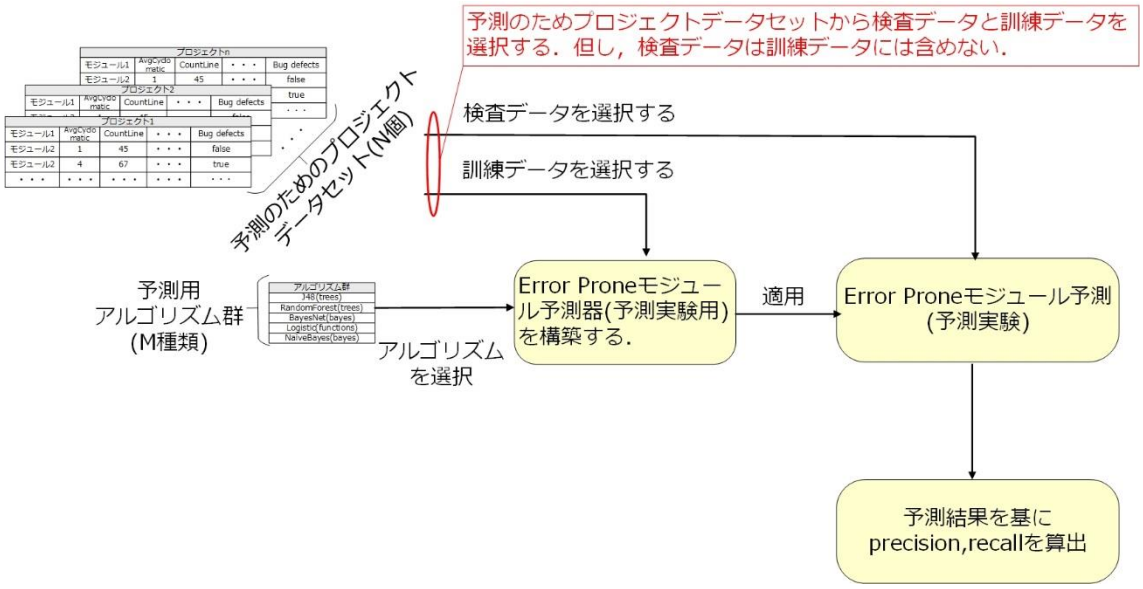

図 17　予測のためのプロジェクトデータセットに基づく予測実験

1-3.　識別訓練データの作成

(A)　予測のためのプロジェクトデータセットを構成する各プロジェクトにおけるモジュール単位に計測したメトリクスの値から，平均値，標準偏差，最小値，最大値を算出し，モジュール単位のメトリクスデータからプロジェクト単位に集約する(図 18).

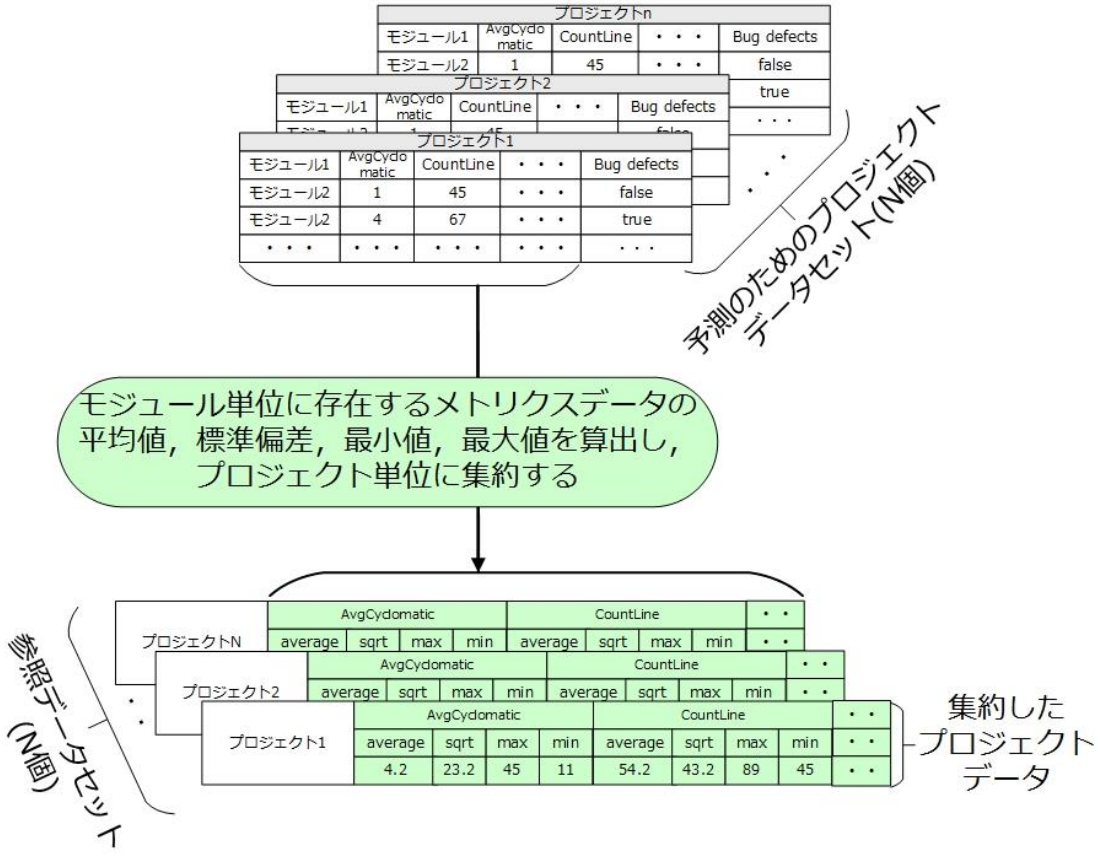

図 18　モジュール単位からプロジェクト単位への集約

(B) プロジェクト毎に特徴量を算出する(図 19 の 1-3(B)).

(C) 1-2 で得られた予測実験結果から予測閾値を基に true/false を判断し，目的変数として設定する(図 19 の 1-3(C)).

(D) (A)でプロジェクト単位に集約したメトリクスデータ及び学習アルゴリズム群から選択した予測アルゴリズムを訓練データして，(プロジェクト N のメトリクス集約値，プロジェクト N の特徴量，予測アルゴリズム，プロジェクト N-1 のメトリクス集約値，プロジェクト N-1 の特徴量，目的変数)の 6 つ組のデータを用意する(N は予測のためのプロジェクトデータセットを構成するプロジェクトの総数から 2 までの整数値). これらのデータは，識別訓練データと呼ぶ(図 19 の 1-3(D)).

1-4.　識別訓練データによる識別器の構築

識別訓練データに基づいて，識別器を構築する(図 19 の 1-4).

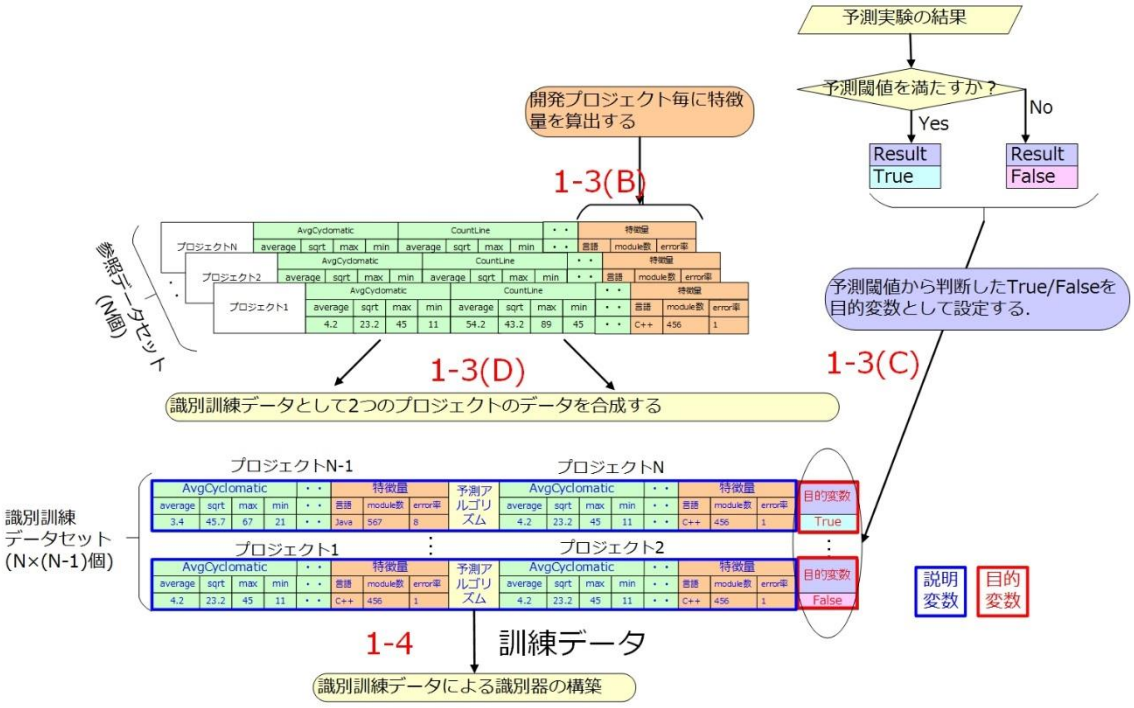

図 19　識別訓練データの生成と構築

Step2.識別器による error-prone モジュール予測器の識別

本ステップは，識別器を用いて検査対象プロジェクトに対して適切な error-prone モジュール予測器を識別するまでの手順で構成される.

2-1.　検査対象プロジェクトのメトリクス計測と特徴量の算出(図 20 の 2-1)

検査対象プロジェクトのソースコードに対して，メトリクスの計測を行う．また，検査対象プロジェクトに対する特徴量も算出する.

2-2.　メトリクスデータのプロジェクト単位への集約及び特徴量の追加(図 20 の 2-2)

(A)　1-3 の(A)と同様の手順で，検査対象プロジェクトにおいてモジュール単位のメトリクスデータをプロジェクト単位に集約する.

(B)　検査対象プロジェクトにおける特徴量を追加する.

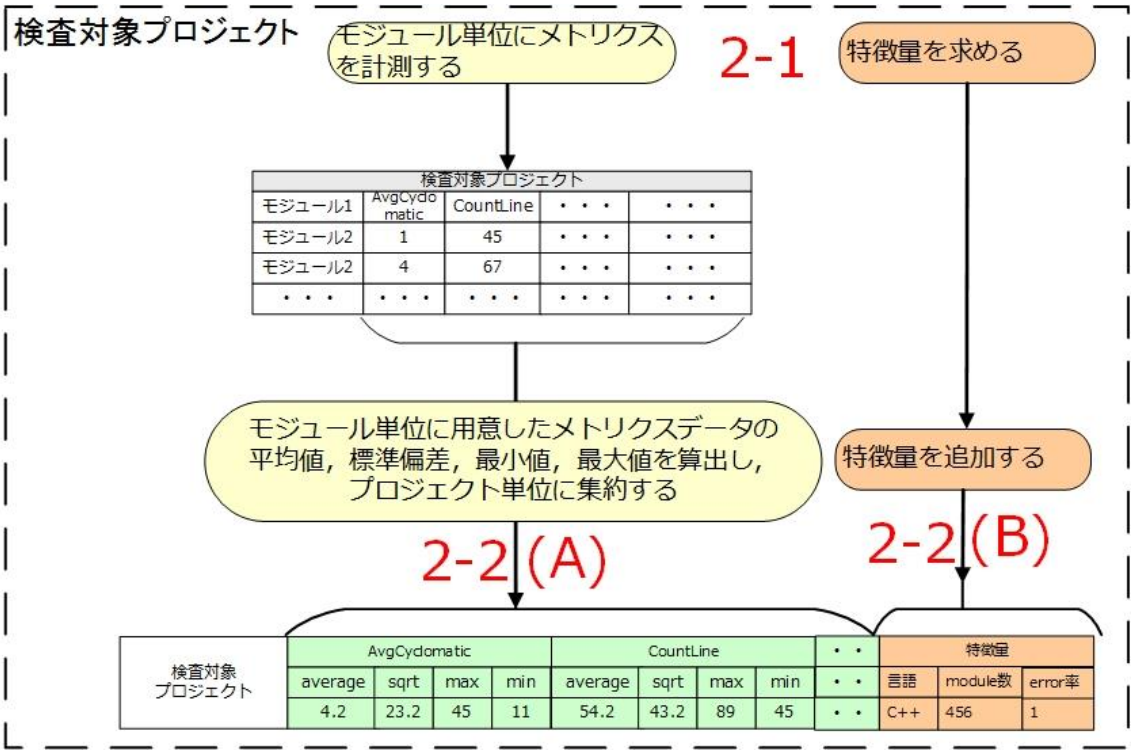

図 20　検査対象プロジェクトのメトリクス集約と特徴量の追加

2-3.　検査データの作成

(A)　予測結果(目的変数)にはダミー値を設定する[5]. (図 21 の(A))

(B)　1-3 で用意した識別訓練データセット, 学習アルゴリズム群から選択した予測アルゴリズムと検査対象プロジェクトのメトリクスを集約した値を使用して, (プロジェクト N のメトリクスデータ, プロジェクト N の特徴量, 予測アルゴリズム, 検査対象プロジェクトのメトリクスの集約値, 検査対象プロジェクトの特徴量, 予測結果)の 6 つ組のデータを用意する(N は識別訓練データを構成するプロジェクトの総数から 1 までの整数値). これらのデータは, 検査データと呼ぶ. (図 21 の(B))

[5] データマイニングツール(WEKA)(3.7 参照)を使用する場合は, 予測結果としてダミー値の設定が必要となる. 一般的にはダミー値の設定は不要である.

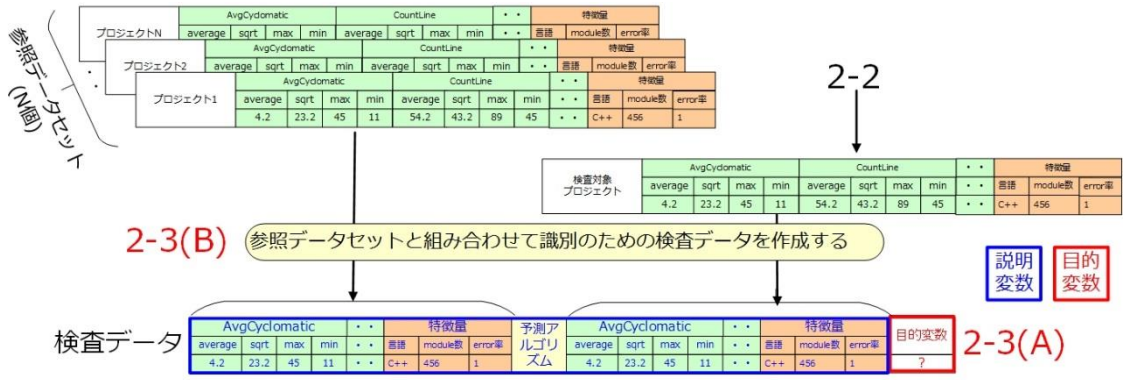

図 21　検査データの作成

2-4.　Error-prone モジュール予測器の識別

1-4 で構築した識別器に対して，検査データを与えることで，error-prone モジュール予測器を識別する(図 22 の 2-4).

2-5.　識別結果の判定

識別器からの出力された結果として，true となる予測器が存在しない場合は，予測閾値を 1 ステップずつ下げて 1-3 の(C) から繰り返す．予測閾値が下げうる下限になっていれば最適な予測器は識別できないと判断して終了する(図 22 の 2-5).

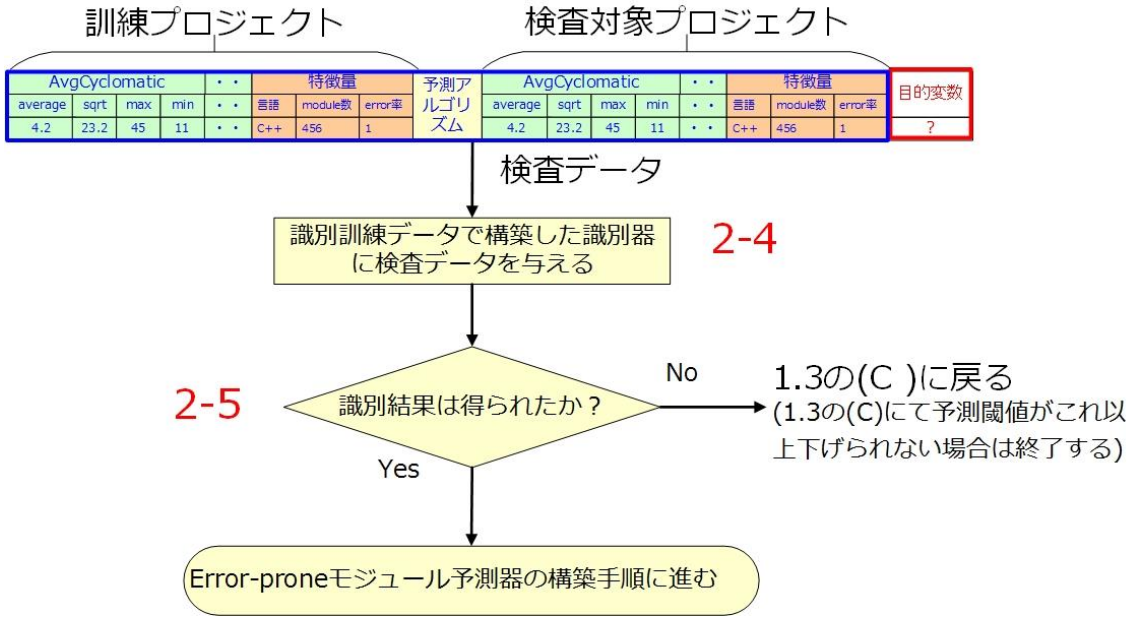

図 22　Error-prone モジュール予測器の識別

Step3. 予測器による error-prone モジュールの予測

2-4 の識別で使用した検査データに含まれる訓練プロジェクトに対応したプロジェクトデータ(モジュール毎に計測したメトリクスデータと該当モジュールの誤りの有無から構成されるプロジェクト単位のデータ)と予測アルゴリズムから error-prone モジュール予測器を構築し，検査対象プロジェクトからモジュール単位に計測したメトリクスデータを入力し，それぞれのモジュールが error-prone モジュールかどうか予測する(図 23).

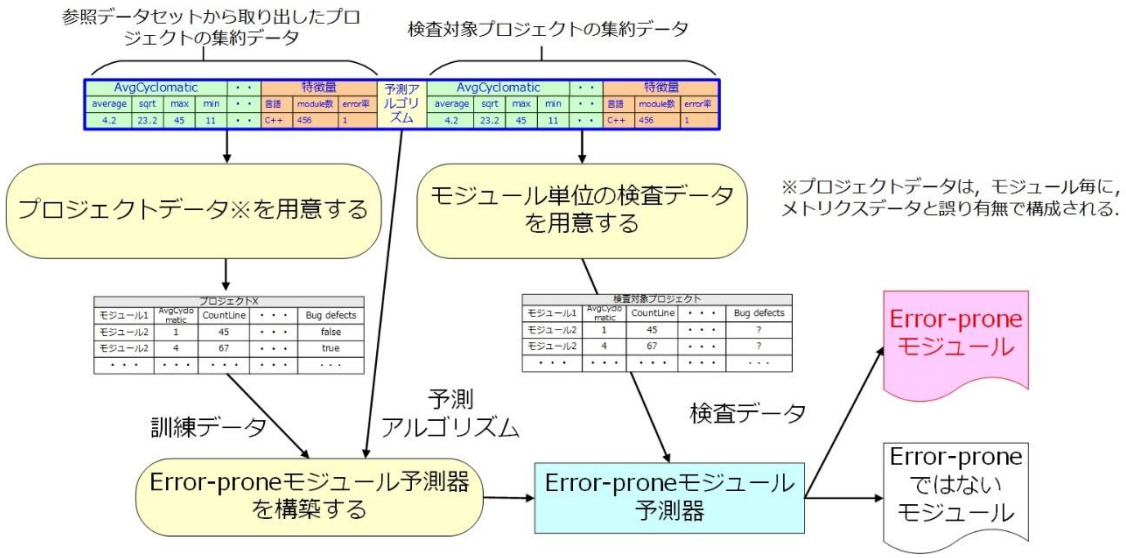

図 23　Error-prone モジュールの予測

我々が提案する手法では，予測閾値を調整することにより，検査対象プロジェクト毎に最適な予測器(予測器を構築するための訓練データと予測アルゴリズム)を識別することができる．なお，検査対象プロジェクト毎に最適であると識別された予測器は複数個，識別されることがある．識別された複数の予測器のいずれを使用しても有効に機能すると言える．

3.6. 本論文の提案手法で使用するメトリクスについて

2.3 でメトリクスの概要を述べた．その中で error-prone モジュールの予測で使用するメトリクスはソースコードメトリクスが使用されることを説明した．本節では，本論文で提案する手法において説明変数として使用するメトリクスはソースコードメトリクスを対象とする理由を述べておく．

本論文の目的は，インスペクションやソフトウェアテストの対象範囲を絞り込むことによる作業の効率化であり，実運用に適用できるための障壁を極力，なくしておく必要がある．ソースコードメトリクスの他にも，プロセスメトリクスや組織メトリクス等，他のメトリクスが存在する．しかし，他のメトリクスを計測する場合，計測が困難な場合に遭遇することが有り得る．例えば，プロセスメトリクスの場合，

その都度，継続的に計測する必要があるが，プロジェクトによっては取得する習慣がないことや，プロセスメトリクスの取得手順が共通化されていないため，プロジェクト間で共通的な尺度で計測したデータを取得することが困難なケースが該当する．また，プロジェクト毎に開発手順が異なるケースや過去におけるメトリクスを取得していないケースもありえる．これに対して，ソースコードメトリクスの場合，ソースコードがあれば，ソースコードリポジトリから過去世代のソースコードを入手することが可能であるため，任意の時点におけるメトリクスを容易に計測可能である．また，訓練データで計測した同一ツールを適用することで，検査対象プロジェクトにおいても同一条件で容易に計測が可能となる．従って，本論文ではソースコードメトリクスを採用することにした．

3.7. データマイニングツール(WEKA)

マイニングには， WEKA[21]を使用する． WEKA は，ニュージーランドの Waikato 大学が中心となって開発されたデータマイニングツールである．WEKA は，Java 言語で実装され，GPL(GNU Public License)の下に配布されているため，無料で入手することが可能である．WEKA は Java の実行環境があれば動作可能であるため，異なる OS 上でも同一の操作手順で実行することができる．

WEKA に入力するデータは，arff(Attribute Relationship File Format)ファイル形式のテキストファイルである．先頭行の”@relation”でデータセットの名称を定義する．その後，”@attribute”で属性を定義し，@attribute の最後が目的変数の定義となる．各属性は一意な属性名称と型が対応する．型としては，数値型(numeric)，名義型(属性値を{}内にカンマ区切りで列挙する)，文字列型がある．データ部は”@data”以降にコンマで区切って属性を列挙する形(CSV 形式)で 1 行に 1 データを記述する．WEKA に入力可能なデータ形式は，arff ファイル形式の他に，C4.5 形式や CSV 形式のファイルがある．本論文では arff ファイル形式を使用する．図 24 に arff ファイル形式の概要を示す．

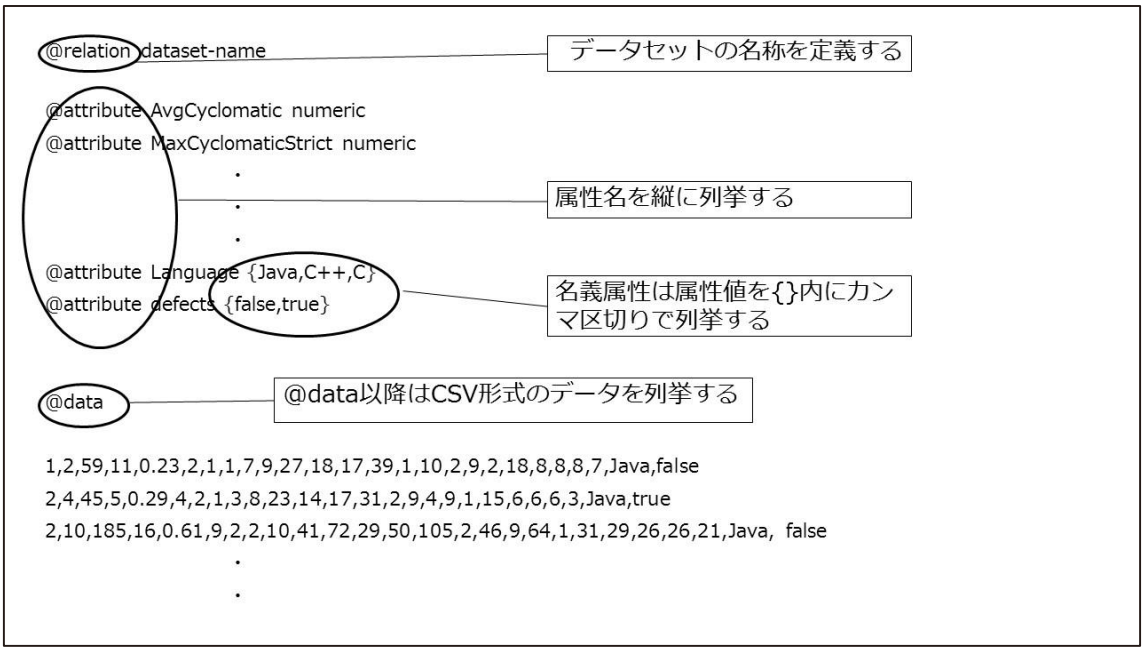

図 24　arff ファイル形式の概要

WEKA は GUI ベースの使用も可能であるが，大量データを扱う場合，GUI の場合は，その都度，手動による操作が伴うため，不適切である．本論文では，コマンドラインから WEKA を実行した．図 25 にコマンドラインからの WEKA の実行例を示すとともに，表 4 に一般的なコマンドラインオプションの一覧を，また，表 5 に決定木に関するオプションの一覧を示す．

```
$ java –cp weka.jar weka.classifiers.trees.j48 –t test.arff
```

図 25　コマンドラインからの WEKA の実行例

表 4　一般的なコマンドラインオプション

オプション	説明
-t file_name	訓練データセットの指定
-T file_name	検査データセットの指定
-x num	交差検定の fold 数を指定する．

表 5　決定木に関するオプション

オプション	説明
-v	決定木に対する評価(Error on training data)を表示しない.
-d file_name	生成した決定木を保存する.
-l file_name	保存した決定木を呼び出す.

3.8. 本論文で使用する学習アルゴリズム(予測アルゴリズム)

学習アルゴリズム(予測アルゴリズム)は, これまでの研究で利用実績があるものから代表的なものを選定した. 学習アルゴリズムは表 6 に示す 5 種類のアルゴリズムを使用する. いずれも WEKA に装備されているアルゴリズムである. なお, 表中の「No」は, 実証実験で使用するアルゴリズムの種別を意味する.

表 6　本論文で使用する学習アルゴリズム

No	名称	説明
0	J48(trees)	C4.5 アルゴリズムによる決定木を使用する.
1	RandomForest(trees)	ランダムに生成した複数の決定木による分類を行う.
2	BayesNet(bayes)	ベイジアンネットワークを生成した分類を行う.
3	Logistic(functions)	ロジスティック回帰分析による分類を行う.
4	NaiveBayes(bayes)	単純な確率モデルによる分類を行う.

各アルゴリズムの選定理由を以下に示す.

(1) J48(trees)

これまでの研究において精度の良い識別器を高速に構築できることが一般的に知られていることや, HE の論文でも C4.5 アルゴリズムによる予測器を構築していたことを踏まえ選定した.

(2) RandomForest

J48 と同様に, 決定木を使用したアルゴリズムであるが, bagging(訓練データの各々から作成した決定木による結果の多数決により決定する手法)と, random feature selection(各ノードで利用する最適な feature の部分集合を選択する手法)を合わせた手法[22]であり, J48 との比較検討ができることを目的として選定した.

(3) BayesNet(bayes)

複雑でかつ不確実な事象の起こりやすさやその可能性を有向グラフによるネットワークとし

て表すことで予測するアルゴリズムである. 決定木では予測困難なケースでも対応できる点を考慮して選定した.

(4) Logistic(functions)

ロジスティック回帰分析は与えられた説明変数の組に対する, 誤りが発生する条件付き確率を重回帰分析の原理を応用して求める手法である. 誤りの発生は事前に予見することはできないが, 誤りが発生する確率はある程度は想定できる場合, 誤りの発生は確率事象として捉えることが可能である. この場合, 本アルゴリズムの利点が発揮できると考えて選定した.

(5) NaiveBayes(bayes)

BayesNet と同様にベイズ定理を使用しているが, BayesNet とは異なり, 単純ベイズ分類器を用いたアルゴリズムである. BayesNet と同様の傾向が現れるのか等, ベイズ定理を用いたアルゴリズムとの比較検討や, 他のアルゴリズムとの比較検討を行うために選定した.

3.9. 評価基準

提案手法では予測閾値を使用する. 予測閾値の定義を以下に示す.

- 予測閾値

  識別訓練データセットの説明変数の値を決定する基準となる閾値である. この閾値は, 識別器を生成する学習データ(訓練データ)が妥当かどうかの判断基準となる.

提案手法により, 表 7に示す予測結果が得られた場合における, precision, recall, efficiency の計算式を以下に示す.

表 7 混同行列(Confusion Matrix)

	誤り含む(true)と予測	誤り含まない(false)と予測
実際は誤り含む(true)	True positive(TP)	False negative(FN)
実際は誤り含まない(false)	False positive(FP)	True negative(TN)

- precision(2.7 にて詳述)

  true と分類されたデータの中で, 実際に true である数の割合を示す. precision が大きければ, 分類誤りが少なく効率よく予測(識別)が行えていると判断できる.

- recall(2.7 にて詳述)
  実際に true であるデータの中で， true であると分類された数の割合を示す．recall が大きければ，網羅的に漏れが少ない予測（識別）が行えていると判断できる．

- efficiency(効率性)
  Efficiency は，表 7 の記号を用いて式(8)として定義する．Efficiency は， error-prone モジュールを対象にインスペクションやソフトウェアテストを実施する際，実施対象は全モジュールのうち，どれくらいの割合を占めるのかを示す指標となる． 従って， efficiency が小さい方が効率的にインスペクションソフトウェアテストができるため，値が小さい方が望まれる． 但し， 式(8)において「TP+FP」は error-prone モジュールの数を意味することから， efficiency の値は， 対象データに含まれる error-prone モジュールの数以下になることはない．

$$efficiency = \frac{(TP + FP)}{(TP + FP + FN + TN)} \tag{8}$$

HE の手法では予測閾値を recall および precision で与えている．しかし，我々の目的は作業効率の改善が目的であるから，インスペクションやソフトウェアテストの対象をどの程度，削減できるのかが焦点となる．表 7 の予測結果から全体は(TP+FN+FP+TN)であるから，インスペクションやソフトウェアテストを行う割合は(TP+FP)/(TP+FP+FN+TN)となる．これを efficiency と定義した．網羅性を考慮した場合，recall に着目する必要があるが，インスペクションやソフトウェアテストの作業の効率を考慮した場合，efficiency に着目することが妥当であると考える．なぜなら，precision は実測値が全 error-prone モジュールのうち，正しく error-prone モジュールと予測できたモジュールの割合を示す指標であるのに対して，efficiency はインスペクションやソフトウェアテストの作業対象が，いかに少ないかを示す指標として使用できるからである．また，閾値の調整が必要となった場合は，網羅性を下げるよりも効率性を下げるべきと考え， efficiency を 0.1 刻みで調整するものとする．但し，予測閾値の値は error-prone モジュール予測の適用分野に依存すると考えるため，適用するにあたっては適宜変更すべきであろう．また，我々は efficiency を指標として使用することを提案しているが，開発の効率化よりも誤りの摘出を重視する場合等，適用状況に応じて，recall や precision を使用することも有効と考える． なお，efficiency は，誤りの密度の高低に依存するため，ソフトウェア開発における経験的な内容を鑑みると，efficiency の上限値は 10 数%程度と考えられる．

3.10. 結言

本章では, HE が提案した手法では, メトリクスに依存しない外部要因(特徴量)については考慮されていない点や, 予測結果の true/false を決定する予測閾値についても recall ≧ 0.7 and precision≧ 0.5 という単一の基準のみでのマイニングである点を指摘した. HE が提案する手法は有力であるものの, そのまま適用しても, これらの指摘事項を解決しないまま実運用において適用した場合, 実運用においては十分な手法とは言えないと考えられる. これら HE が提案する手法における指摘事項を踏まえ, 本論文で提案する手法では, 検査対象プロジェクトに適した予測器を選定する際, 検査対象プロジェクトの特性を表した特徴量を導入する点及び, HE の提案手法では単一的な予測閾値であったのを可変に変更できる機構を設けた点を改善したことを説明した. 4 章では本論文の提案手法が有効であることを主張する仮説及びその実証実験について詳述する.

第 4 章

実験の設定

第4章　実験の設定

4.1. 緒言

本章では，本論文の提案手法の有効性を示すための実証実験を行うにあたり，検証すべき仮説や実験手順・目的を述べる．また，実証実験で使用するデータ及び，使用データに含まれる誤り件数の算出手順についても説明する．

4.2. 実験で使用するデータ

本論文の実験には，大西[25]によって sourceforge.net で公開されている 20 種類のオープンソースプロジェクトを対象として，テクマトリクス社のメトリクス計測ツール Understand[26]を用いて計測された OO メトリクスを中心としたデータセット(以下，OSS データと称す)および，PROMISE(Prediction Models In Software Engineering)[27]で公開されているデータセットのうち，NASAの公開データ(以下，Promise データと称す) を使用する．以下の項では，各データの詳細を述べる．

4.2.1. データの選定基準

OSSデータは，20種類のオープンソースプロジェクトから取得したデータで構成される．OSSデータは，以下に示す基準 1～基準 3 を満たしたプロジェクトを対象にデータを取得した． OSS データのみでは本論文の提案手法の有効性は十分に証明できないため，提案手法が一般化できることを確認するためにPromiseデータも用意することとした．なお，Promiseデータにおける誤り件数は OSS データとは異なり，公開された件数となる．Promise データについては，OSS データとは異なり，既に計測された誤り件数，メトリクスを使用するだけであり，特に基準は設けていない．但し，Promiseデータの場合，計測したメトリクスがプロジェクト毎に異なるものが存在した．このため，各プロジェクトで共通であるメトリクスのみに限定して使用した．

基準1. Understandが対象とする開発言語(Java/C++/Cのいずれか[6])で記述されたプロジェクトであること．

基準2. 開発期間が開発開始から一年以上，経過していること．(過去データと現在データの二種類を半年単位で収集することを考慮して，開発期間は一年以上とした．)

基準3. コミットログが保存されているプロジェクトであること．(コミットログから誤りの数を算出する

[6] Understand は Java/C++/C 以外の言語にも対応しているが，我々が所有するライセンスは Java/C++/C のみであるため，これら 3 つの言語に限定した．

ためにコミットログが保存されているプロジェクトを対象とした.)

本論文で利用したプロジェクトの名称と誤りの割合を 表 8 および表 9 に示す. また, それぞれのデータで使用したメトリクスを表 10 および表 11 に示す.

表 8　OSS データの概要

No	プロジェクト名		誤りを含むモジュール数	誤りを含まないモジュール数	開発言語
0	Ant	old	12	1183	Java
1		new	9	1186	Java
2	ant-ivy	old	4	585	Java
3		new	38	561	Java
4	ant-ivyde	old	10	133	Java
5		new	26	137	Java
6	DavMail	old	39	24	Java/C++
7		new	22	152	Java/C++
8	DeSmuMe	old	45	361	C++
9		new	33	378	C++
10	DrJava	old	43	1215	Java
11		new	84	1174	Java
12	JCS	old	1	521	Java
13		new	104	415	Java
14	SMPlayer	old	1	116	C++
15		new	2	115	C++
16	Saros	old	32	656	Java
17		new	195	851	Java
18	WinMerge	old	14	393	C++
19		new	5	411	C++
20	Ffdshow	old	232	714	C++
21		new	260	736	C++
22	Firebird	old	87	959	C++
23		new	50	996	C++
24	gnome-panel	old	18	57	C
25		new	50	39	C

No	プロジェクト名		誤りを含むモジュール数	誤りを含まないモジュール数	開発言語
26	gnome-session	old	11	24	C
27		new	4	30	C
28	jEdit	old	22	533	Java
29		new	38	516	Java
30	squirrel-sql	old	37	3732	Java
31		new	36	3893	Java
32	tomcat tomcat	old	282	1276	Java
33		new	568	1024	Java
34	VASSAL_Engine	old	1	887	Java
35		new	742	147	Java
36	Inkscape	old	168	1223	C
37		new	235	1151	C
38	Shareaza	old	83	473	Java
39		new	150	405	Java

表 9　Promise データの概要

No	プロジェクト名	誤りを含むモジュール数	誤りを含まないモジュール数	開発言語
0	ar3	8	55	C
1	ar4	20	87	C
2	ar5	8	28	C
3	kc1	326	1783	C++
4	kc2	107	415	C++
5	kc3	43	415	C++
6	cm1	49	449	C
7	mw1	31	372	C
8	pc1	77	1033	C
9	mc2	52	109	C

表 10　OSS データで使用するメトリクス

メトリクス名	説明
AvgCyclomatic	関数・メソッドの Cyclomatic 複雑度の平均
MaxCyclomaticStrict	Strict Cyclomatic 複雑度の最大値
CountLine	行数
CountLineBlank	空白行数
RatioCommentToCode	コード行に占めるコメント行の割合．コードとコメントの両方を含む行がある場合，コメント率は 1 を超える場合がある．
MaxCyclomaticModified	Modified Cyclomatic 複雑度の最大値
AvgCyclomaticModified	関数・メソッドの Modified Cyclomatic 複雑度の平均
AvgEssential	関数・メソッドの Essential 複雑度の平均
CountDeclFunction	関数の数
CountStmtExe	実行可能ステートメント数
CountStmt	宣言・実行可能ステートメント数
CountLineCodeDecl	宣言コード行数
CountSemicolon	セミコロン数
CountLineCode	コード行数．クラスの場合，クラスのメンバー関数の CouneLineCode の総和．
AvgCyclomaticStrict	関数・メソッドの Strict Cyclomatic 複雑度の平均
CountLineCodeExe	実行可能コード行数
MaxCyclomatic	Cyclomatic 複雑度の最大値
CountLineComment	コメント行数
CountDeclClass	クラス数
CountStmtDecl	宣言ステートメント数
SumCyclomaticStrict	Strict Cyclomatic 複雑度の総和
SumCyclomatic	Cyclomatic 複雑度の総和
SumCyclomaticModified	Modified Cyclomatic 複雑度の総和
SumEssential	Essential 複雑度の総和
Language	言語

表 11　Promise データで使用するメトリクス

メトリクス名	説明
blank_loc numeric	空行数
branch_count numeric	フローグラフの分岐の数
code_and_comment_loc numeric	コード行数とコメント行数の和
comment_loc numeric	コメント行数
cyclomatic_complexity numeric	循環複雑度
design_complexity numeric	設計複雑度
halstead_difficulty numeric	プログラム難易度レベル
halstead_effort numeric	プログラムに対する推定工数
halstead_length numeric	プログラム長測定基準
halstead_time numeric	アルゴリズム推定実行時間
halstead_volume numeric	プログラムをコード化するのに必要な最小ビット数
total_loc numeric	コードの総行数
total_operands numeric	オペランドの総数
total_operators numeric	演算子の総数
unique_operands numeric	オペランドの種類
unique_operators numeric	演算子の種類

4.2.2. 誤り有無の判定方法について(OSS データ)

OSS データは 2 つの期間に分けて収集したデータ(モジュール毎に計測したメトリクスデータとモジュール毎の誤りの有無から構成される)であり，それぞれ old と new とした．2 つの期間に分けて収集した理由は，検査対象プロジェクトに過去データ(old)が存在し，訓練データとして過去データを使用し，検査データとしてもう一方の期間に収集したデータ(new)を使用した場合，最も良好な予測精度が確保できるのか，或いは，そのような結果にはならないのかを確認できる利点があるからである．これに対して，Promise データは，期間毎に分けた提供は行われておらず，1つの期間における(メトリクス，誤りの数)のデータである．

OSS データにおいては，バグ管理システムで誤りの詳細は公開されているものの，モジュール(ファイル)毎の誤り改修の有無についての情報は含まれていない．従って，ファイル単位における誤りの存在有無は自前で計測する必要があった．計測にあたっては，SZZ アルゴリズム[30]を用

いるのが一般的であるが，Subversion(バージョン管理システム)から取得したコミットログには，バグ管理システム(Bugzilla 等)に記録された情報との対応付けを行うキー項目(バグ管理システムがユニークに払い出すID等)が記入されていないレコードも多数含まれていることから同アルゴリズムを使用することは出来なかった．このため，ファイル毎に，ある期間における Subversion のコミットログ を取得し，取得したログから”bug”，”fix”といった単語が含まれるか否かで判定した．

これに対して，Promise データについては，ファイル毎の誤りの有無は公開されていることや，ソースコード及びSubversion(バージョン管理システム)のコミットログは非公開であるため，OSSデータと同一の方法による誤り有無の収集は実施していない．

4.2.3. メトリクス計測のタイミングについて(OSS データ)

OSS データにおけるメトリクスの計測にあたっては，誤りの修正を行ったリビジョンから直前のリビジョンのソースコードに対してメトリクスを計測する方法もある．しかし，この方法では，メトリクスと誤りとの関係は厳密になるものの，ファイル毎にメトリクスの計測時期が異なり，プロジェクト全体をまとめたデータとしては利用できない．従って，本論文で使用した OSS データにおけるメトリクスの計測は，ある時点のソースコードから Understand を用いてメトリクスの計測を行い，その時点から半年間のコミットログを参考に誤りの有無を計測する方式を採用した．図 26 に誤り件数のカウントとメトリクス計測のタイミングの概要を示す．図 26 において，誤り件数のカウント期間にリビジョンが 2 回(n+1 と n+2)更新されている．この期間において，それぞれのリビジョンに対してメトリクスの計測を行うことになる．

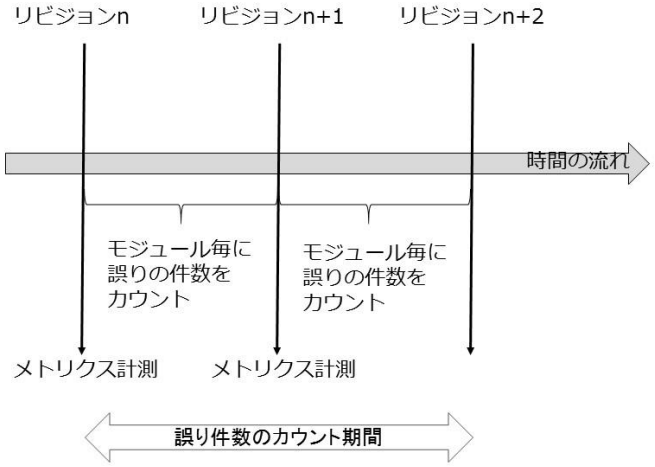

図 26　OSSデータにおける誤り件数のカウントとメトリクス計測のタイミング

4.3. 仮説の設定

提案手法の有効性を示すための実験を行うにあたり，以下に示す仮説を設定した．

仮説1: 予測閾値に基づいて適切と判定された予測器は一意ではなく，検査対象プロジェクト毎に適切な訓練データと予測アルゴリズムがそれぞれ異なる．

仮説2: 提案するマイニング手法を適用すれば予測閾値による識別の結果から得ることのできる precision が十分な値となり，適切な予測器(訓練データと予測アルゴリズム)を識別することができる．

仮説3: 特徴量はマイニング手法の識別精度向上に寄与する．

仮説 1 が成立しないのであれば，本論文で提案するマイニング手法は不要であることになる．仮説 1 は既存研究においても設定され実証されている内容であるが，本研究の前提条件を満たしていることを確認するために，検証を行う．仮説 2 は，本論文が提案する error-prone モジュール手法のマイニングを適用することで最適な error-prone モジュールを予測できる予測器を識別できることの可能性を示すために設定した．仮説 3 は，我々が提案する「特徴量」を導入することにより，error-prone モジュール予測器を識別するための識別精度が向上する事を主張するために設定した．

4.4. 仮説検証のための実験

上記で設定した仮説を検証するために，以下に示す実験を行った．

実験 1

OSS データおよび Promise データを用いて予測器(訓練データと予測アルゴリズム)の識別実験を行う．具体的には，各データのうち，1 つを検査データとし，検査データを除いた全てのデータを訓練データとして，データ毎に(訓練データ，検査データ)対を作成する．これらの対を 5 つの学習アルゴリズムを用いて，誤りの有無を目的変数，各モジュールから取得したソースコードメトリクスを説明変数としたデータマイニングツール WEKA の入力ファイル(arff ファイル) を作成し，WEKA を使用して予測器(訓練データと予測アルゴリズム)の識別実験を行う．この実験結果から，true とされる(訓練データ，検査データ，アルゴリズム)の 3 つ組集合において，唯一の訓練データとアルゴリズムが検査データに対して選択されないことを確認する．

OSS データの場合，プロジェクトは新旧合わせて 40 個であり，訓練データと検査データの組み合わせでは訓練データで使用したデータは検査データとしては使用しないので，全体で 39×40×5(アルゴリズム数)＝7800 通りの実験を行う．Promise データの場合，プロジェクト数は 10 である

ので, 全体で 9×10×5(アルゴリズム数)＝450 通りの実験を行う.

実験 2

以下に示す手順(手順 1～手順 5)で, 説明変数と目的変数を決めて arff ファイルを作成し, 10 重交差検証を行う. 本実験では, 表 12 に示した予測閾値の条件毎かつ, 表 13 に示す特徴量の組み合わせ(8 通り)にて, マイニングによる識別精度としての precision(以下, 識別の precision と称す) の変化を調べる. 変化を調べる理由は, 提案手法による予測器の識別がどれくらの精度で行われるのかを確認するとともに, 特徴量の組み合わせによる識別の precision への効果を確認するためである. 実験 2 では, OSS データ/Promise データともに, 12(予測閾値の種類)×8(特徴量種別の種類)×5(アルゴリズム数)＝480 通りの実験を行う.

手順1. データに含まれるメトリクスの値はソースファイル単位のため, プロジェクト単位に集約する必要がある. 集約するために, 各ソースコードから計測したメトリクスデータから, 平均値, 標準偏差, 最小値, 最大値を算出し, これらを説明変数とする.

手順2. 実験 1 の結果で得られた混同行列から precision, recall, efficiency を算出し, 表 12 に示す予測閾値の条件毎に, これらの条件を満たす場合は, 目的変数を true とし, 満たさない場合は false を設定する.

手順3. 個々の訓練データと検査データに対して特徴量として, 使用言語, モジュール数, 誤り率(error 密度が 10%以上, 10%未満かつ 1%以上, 1%未満の三段階に分類する)を説明変数として追加する.

手順4. 上記の各手順で求めた説明変数及び目的変数を arff ファイルに記述する.

手順5. arff ファイルを入力として WEKA による 10 重交差検証を行う.

表 12 に用意した予測閾値の条件は,「約 20%のモジュールに, 誤りの約 60%～80%が存在する」という考え方[31,32,49,50,51,52,53]に基づき, 実運用において許容できるであろうと想定される条件を複数用意した. 予測閾値の条件として, precision と recall は, error-prone モジュールの予測精度は期待値以上となる条件が true となるように設定した. これに対して, efficiency については, 本論文がインスペクションやソフトウェアテストの対象範囲の絞込みを行うことを目的としていることを踏まえ, 指定した割合以下となる条件が true となるように設定した. また, 表中の「閾値種別」は, 本実験で適用した予測閾値を識別するために用意している. 例えば, 後述する実験結果において, 閾値種別 0 は,「efficiency ≦ 0.3 and recall≧0.7」であることを意味する.

本実験より，以下の点を確認する．

(1) 実験結果から得られた precision を降順でソートを行い，高い値を記録した識別の precision がどれだけの値になっているのか確認する．

(2) 実験結果から得られた識別の precision の上位 100 位における特徴量種別毎の内訳を明らかにし，特徴量を利用しているケース(特徴量種別が 0 以外)はどれぐらい占めるのか確認する．本手順により，特徴量を利用することで識別精度が向上することの効果を確認する．

(3) 実験結果から特徴量種別(特徴量の組み合わせ)，学習アルゴリズムが同一である場合における識別の precision の値を比較し，どの特徴量が優位に機能しているのか調査する．

表 12　実験 2 で適用した予測閾値の条件

予測閾値 （閾値種別）	範囲 （precision を P，recall を R， efficiency を E と称す）
0	E≦0.3 and R ≧ 0.7
1	P≧0.4 and R ≧ 0.7
2	P≧0.4 and R ≧ 0.6
3	P≧0.4 and R ≧ 0.5
4	P≧0.3 and R ≧ 0.7
5	P≧0.3 and R ≧ 0.6
6	P≧0.3 and R ≧ 0.5
7	E≦0.3 and R ≧ 0.6
8	E≦0.3 and R ≧ 0.5
9	E≦0.2 and R ≧ 0.7
10	E≦0.2 and R ≧ 0.6
11	E≦0.2 and R ≧ 0.5

表 13　特徴量の組み合わせ

特徴量種別	言語種別	モジュール数	error 密度
0	-	-	-
1	使用する	-	-
2	-	使用する	-
3	-	-	使用する
12	使用する	使用する	-
13	使用する	-	使用する
23	-	使用する	使用する
123	使用する	使用する	使用する

実験 3

実験 2 で実施した 10 重交差検証は，3.3 で述べた通り，与えられた入力データを 10 分割し，そのうち 9 個を訓練データとして利用し，残りの 1 つを検査データとして使用する検証を 10 回実施する．この手順では，予測器の構築にて検査データと同一データが訓練データとして使用される可能性があり，予測精度が比較的高くなる傾向がある．より厳しくマイニング精度を評価するために，以下の手順で訓練データと検査データを独立させた arff ファイルを作成して識別実験を行い，precision の変化を調べる．

- 実験 2 と同様の手順で，説明変数と目的変数を決める．
- 訓練データには検査データを含めない形で arff ファイルを作成する．

実験 3 では，OSS データの場合，19200 通り＝ 12(予測閾値の種類)×8(特徴量種別の種類)×5(アルゴリズム数) ×40(対象プロジェクト数)の実験を行う．また，Promise データの場合，4800 通り＝12(予測閾値の種類)×8(特徴量種別の種類)×5(アルゴリズム数) ×10(対象プロジェクト数)の実験を行う．

4.5. 結言

本章では, まず, 実験で使用するデータとして用意した, OSS データと Promise データの 2 つについて詳細を説明すると共に, データの選定基準や誤り件数の算出手順について説明した. 次に, 本論文が提案する手法の有効性を実証するにあたり, 予め設定した 3 つの仮説について, その内容及び設定理由を述べるとともに, 各実験における目的と実験手順の詳細を説明した. 次章では, 実験結果を示すとともに実験結果に基づいた考察と評価を行う.

第 5 章

実験の評価

第5章　実験の評価

5.1. 緒言

4 章では，実証実験で使用する 2 種類のデータの詳細を説明すると共に，本論文が提案する手法の有効性を実証するための仮説及び実験内容を説明した．本章では，4 章で述べた各種実験の結果を明示しながら，評価・考察を行う．なお，各実験の結果を一覧で表記する際，「訓練」と「検査」の欄に表記する数字は，表 8 や表 9 の「No」に対応し，アルゴリズムの番号は，表 6 の「No」に対応する．

5.2. 実験 1 の評価

OSS データを用いた実験結果のうち 比較的，良好な結果が得られた，(precision ≧ 0.5 and recall ≧ 0.8)を満たす識別結果を表 14 に示す．この条件に合致する件数は 42 件であった．この内，訓練データとアルゴリズムの対が重複するものは計 14 件(表中の「PT」欄に●を記載したものが該当する)あるが，検査データに渡って同じものは無かった． なお，precision と recall で見た場合，比較的，良好な結果が得られているが，efficiency で見た場合，0.8 もくしは 1 のものは，確認対象のほとんどを確認することになり，開発作業の効率化を考慮すると必ずしも良好とは言えない．

表 14　OSS データを用いた実験 1 の結果
(precision ≧ 0.5 and recall ≧ 0.8)

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	22	0	15	1	1	0.009
	36	1	15	1	1	0.009
	36	3	15	1	1	0.009
●	39	4	35	0.877	0.849	0.808
	18	4	35	0.87	0.857	0.823
●	6	2	35	0.835	1	1
●	18	2	35	0.835	1	1
●	25	2	35	0.835	1	1
	13	4	25	0.645	0.816	0.713
●	18	2	6	0.644	1	0.967
●	35	0	6	0.643	0.947	0.918
●	35	3	6	0.638	0.974	0.951

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	28	2	25	0.635	0.816	0.724
	11	2	25	0.629	0.898	0.805
	31	2	25	0.625	0.918	0.828
●	25	2	6	0.623	1	1
	37	2	25	0.623	0.878	0.793
	13	2	25	0.622	0.939	0.851
●	35	1	6	0.62	0.816	0.82
	5	4	25	0.618	0.857	0.782
	36	2	25	0.611	0.898	0.828
	5	2	25	0.605	0.939	0.874
	17	2	25	0.603	0.959	0.897
	30	2	25	0.597	0.939	0.885
	17	3	25	0.592	0.918	0.874
●	35	0	25	0.592	0.857	0.816
	35	4	6	0.589	0.868	0.918
	3	2	25	0.578	0.98	0.954
●	35	3	25	0.576	1	0.977
	16	2	25	0.573	0.959	0.943
	35	2	25	0.57	1	0.989
	33	2	25	0.57	0.918	0.908
●	35	1	25	0.568	0.939	0.931
●	6	2	25	0.563	1	1
	38	4	25	0.554	0.939	0.954
●	39	4	25	0.554	0.939	0.954
	9	4	15	0.5	1	0.017
	20	1	15	0.5	1	0.017
	23	0	15	0.5	1	0.017
	23	4	15	0.5	1	0.017
	24	1	15	0.5	1	0.017
	29	0	15	0.5	1	0.017

我々はさらに以下に示す条件に基づいて適切と判定された予測器は一意ではないことを確認した.

・ precision ≧ 0.4 and recall ≧ 0.8

本条件は,「precision ≧ 0.5 and recall ≧ 0.8」の条件よりも precision の値を 0.1 ポイント下げることで識別条件を緩和させた. その結果, 44 件が選択されたが, ほとんどが上記条件と同じ結果であった. 本条件における識別結果を表 15 に示す.

・recall ≧ 0.8 and efficiency ≦0.2

20%のモジュールが全体の 80%の誤りを見つけられるべきであるとした Weyuker[31]の研究報告及び, 80%の誤りが 20%のコードに含まれているとした Glass[32]の研究内容等を踏まえ, 実運用において全体の 20%モジュールに対するインスペクションやソフトウェアテストにより, 全体の 80%以上の誤りを摘出することを目的とした条件となる. 本条件では, 71 件が選択された. 選択されたデータのうち, 9 件は(precision ≧ 0.5 and recall ≧ 0.8) の条件を満たすものと重複していたが, いずれも検査データ, アルゴリズムは異なっていることを確認した. 本条件における識別結果を表 16 に示す.

表 15 OSS データを用いた実験 1 の結果

(precision ≧ 0.4 and recall ≧ 0.8)

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	22	0	15	1	1	0.009
	36	1	15	1	1	0.009
	36	3	15	1	1	0.009
●	39	4	35	0.877	0.849	0.808
	18	4	35	0.87	0.857	0.823
●	6	2	35	0.835	1	1
●	18	2	35	0.835	1	1
●	25	2	35	0.835	1	1
	13	4	25	0.645	0.816	0.713
●	18	2	6	0.644	1	0.967
●	35	0	6	0.643	0.947	0.918
●	35	3	6	0.638	0.974	0.951
	28	2	25	0.635	0.816	0.724
	11	2	25	0.629	0.898	0.805

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	31	2	25	0.625	0.918	0.828
●	25	2	6	0.623	1	1
	37	2	25	0.623	0.878	0.793
	13	2	25	0.622	0.939	0.851
●	35	1	6	0.62	0.816	0.82
	5	4	25	0.618	0.857	0.782
	36	2	25	0.611	0.898	0.828
	5	2	25	0.605	0.939	0.874
	17	2	25	0.603	0.959	0.897
	30	2	25	0.597	0.939	0.885
	17	3	25	0.592	0.918	0.874
●	35	0	25	0.592	0.857	0.816
	35	4	6	0.589	0.868	0.918
	3	2	25	0.578	0.98	0.954
●	35	3	25	0.576	1	0.977
	16	2	25	0.573	0.959	0.943
	35	2	25	0.57	1	0.989
	33	2	25	0.57	0.918	0.908
●	35	1	25	0.568	0.939	0.931
●	6	2	25	0.563	1	1
	38	4	25	0.554	0.939	0.954
●	39	4	25	0.554	0.939	0.954
	9	4	15	0.5	1	0.017
	20	1	15	0.5	1	0.017
	23	0	15	0.5	1	0.017
	23	4	15	0.5	1	0.017
	24	1	15	0.5	1	0.017
	29	0	15	0.5	1	0.017
	25	3	26	0.444	0.8	0.545
●	39	4	33	0.418	0.832	0.71

表 16　OSS データを用いた実験 1 の結果
(recall ≧ 0.8 and efficiency≦ 0.2)

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	22	0	15	1	1	0.009
	36	1	15	1	1	0.009
	36	3	15	1	1	0.009
	9	4	15	0.5	1	0.017
	20	1	15	0.5	1	0.017
	23	0	15	0.5	1	0.017
	23	4	15	0.5	1	0.017
	24	1	15	0.5	1	0.017
	29	0	15	0.5	1	0.017
	9	0	15	0.333	1	0.026
	10	0	15	0.333	1	0.026
	10	1	15	0.333	1	0.026
	20	3	15	0.333	1	0.026
	22	4	15	0.333	1	0.026
	39	3	15	0.333	1	0.026
	7	3	15	0.25	1	0.035
	9	1	15	0.25	1	0.035
	9	3	15	0.25	1	0.035
	37	3	15	0.25	1	0.035
	38	0	15	0.25	1	0.035
	8	4	15	0.2	1	0.043
	21	4	15	0.2	1	0.043
	36	0	15	0.2	1	0.043
	37	0	15	0.2	1	0.043
	37	1	15	0.2	1	0.043
	8	3	15	0.167	1	0.052
	20	4	15	0.167	1	0.052
	21	2	15	0.167	1	0.052
	39	1	15	0.167	1	0.052
	7	1	15	0.143	1	0.061

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	8	2	15	0.143	1	0.061
	9	2	15	0.143	1	0.061
	21	0	15	0.143	1	0.061
	23	2	15	0.143	1	0.061
	24	4	15	0.143	1	0.061
	29	4	15	0.143	1	0.061
	4	0	15	0.125	1	0.07
	26	0	15	0.125	1	0.07
	28	4	15	0.125	1	0.07
	38	3	15	0.125	1	0.07
	10	4	15	0.111	1	0.078
	19	4	15	0.111	1	0.078
	20	2	15	0.111	1	0.078
	28	3	15	0.111	1	0.078
	32	1	15	0.111	1	0.078
	37	4	15	0.111	1	0.078
	22	2	15	0.1	1	0.087
	24	3	15	0.1	1	0.087
	36	4	15	0.1	1	0.087
	2	4	15	0.090909091	1	0.095652174
	20	0	15	0.083	1	0.104
	25	4	15	0.077	1	0.113
	1	4	15	0.076923077	1	0.113043478
	1	2	15	0.071428571	1	0.12173913
	11	4	15	0.071	1	0.122
	29	3	15	0.071	1	0.122
	39	0	15	0.071	1	0.122
	32	0	15	0.067	1	0.13
	10	3	15	0.063	1	0.139

PT	訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
	32	4	15	0.063	1	0.139
	33	4	15	0.063	1	0.139
	38	2	15	0.063	1	0.139
	6	4	15	0.059	1	0.148
	33	3	15	0.059	1	0.148
	5	1	15	0.05	1	0.174
	7	4	15	0.048	1	0.183
	13	1	15	0.045	1	0.191
	32	3	15	0.045	1	0.191
	7	2	15	0.043	1	0.2
	16	4	15	0.043	1	0.2
	31	4	15	0.043	1	0.2

我々は, Promise データを用いて, 上記と同様の実験を行った. Promise データを用いた実験結果のうち (precision ≧ 0.5 and recall ≧ 0.8)を満たす識別結果を表 17 に示す. 選択されたデータの数は異なるが, OSS データの場合と同様の傾向を確認することができた.

表 17　Promise データを用いた実験 1 の結果
(precision ≧ 0.5 and recall ≧ 0.8)

訓練	ｱﾙｺﾞﾘｽﾞﾑ	検査	precision	recall	efficiency
1	1	2	0.636	0.875	0.306
1	2	2	0.636	0.875	0.306
1	4	2	0.636	0.875	0.306
4	4	2	0.500	1	0.444
5	4	2	0.500	1	0.444
6	4	2	0.636	0.875	0.306
8	4	2	0.700	0.875	0.278
9	2	2	0.500	1	0.444
9	4	2	0.538	0.875	0.361

これらの結果から，以下の点が判断できる．

(1) 最適な予測器は一意ではなく，対象プロジェクト毎に適切な訓練データと学習アルゴリズムがそれぞれ異なる．従って，仮説 1 は検証できたと判断する．

(2) OSS データでは偶数番号が旧のデータ(old)，＋1した奇数番号が新のデータ(new)である．表 14 を見ると訓練データと検査データで旧，新の対は存在しない．従って，同一プロジェクトの過去データが最適な訓練データであると判断することはできず，プロジェクト毎に最適な訓練データは異なるといえる．

(3) 学習アルゴリズムの優位性は無い．

5.3. 実験 2 の評価

5.3.1. 識別の可能性(仮説 2)への考察

実験 2 の結果より，識別の precision が 0.7 以上のデータを表 18 および表 19 に示す．これらの結果より，識別の precision の値は 0.8 を超えるケースは無かったが，OSS データ/Promise データともに 0.7 以上のケースは存在した．よって，比較的良い精度で各プロジェクトに対して適切な error-prone モジュール予測器を識別できる可能性はあると言える．なお，それぞれの表において，「特徴量種別」のみが異なるエントリが複数存在し，「特徴量種別」が異なっても識別の precision には影響を受けないケースもあるが，これは一部の事例であり，必ずしも全ての組み合わせにおいて該当する訳ではない．特徴量を利用した効果への考察は 5.3.2 で行う．

表 18　実験 2 の結果(OSS データ)(precision>0.7)

閾値番号	特徴量種別	アルゴリズム	TP	FN	FP	TN	precision
3	3	0	50	117	13	7620	0.794
3	23	0	50	117	13	7620	0.794
3	13	0	51	116	14	7619	0.785
3	123	0	51	116	14	7619	0.785
2	3	0	39	80	11	7670	0.780
2	23	0	39	80	11	7670	0.780
1	3	0	42	42	12	7704	0.778
1	13	0	42	42	12	7704	0.778
1	23	0	42	42	12	7704	0.778
1	123	0	42	42	12	7704	0.778

閾値番号	特徴量種別	アルゴリズム	TP	FN	FP	TN	precision
5	1	0	77	165	23	7535	0.770
5	12	0	77	165	23	7535	0.770
3	1	0	53	114	16	7617	0.768
3	12	0	53	114	16	7617	0.768
2	0	0	42	77	13	7668	0.764
2	2	0	42	77	13	7668	0.764
5	0	0	77	165	24	7534	0.762
5	2	0	77	165	24	7534	0.762
3	0	0	53	114	17	7616	0.757
3	2	0	53	114	17	7616	0.757
2	1	0	40	79	13	7668	0.755
2	12	0	40	79	13	7668	0.755
2	13	0	39	80	13	7668	0.750
2	123	0	39	80	13	7668	0.750
1	1	0	36	48	14	7702	0.720
1	12	0	36	48	14	7702	0.720
1	0	0	35	49	14	7702	0.714
1	2	0	35	49	14	7702	0.714
3	123	2	22	145	9	7624	0.710
4	3	0	53	117	22	7608	0.707
4	23	0	53	117	22	7608	0.707

表 19　実験 2 の結果(Promise データ)(precision>0.7)

閾値番号	特徴量種別	アルゴリズム	TP	FN	FP	TN	precision
6	0	0	44	24	14	368	0.759
6	1	0	44	24	14	368	0.759
6	2	0	44	24	14	368	0.759
6	12	0	44	24	14	368	0.759
6	3	0	42	26	16	366	0.724
6	13	0	42	26	16	366	0.724
6	23	0	42	26	16	366	0.724
6	123	0	42	26	16	366	0.724

5. 3. 2. 特徴量を利用した効果(仮説 3)への考察

実験 2 の結果より, OSS データ/Promise データそれぞれにおいて, 特徴量を利用することで識別の precision が改善したケースが存在した. OSS データにおける改善例を表 20 に示す. また, Promise データにおける改善例を表 21 に示す. OSS データの場合, 特徴量種別 0 の precision は 0.583 に対して, 特徴量種別 123(表 13 に示した 3 種類の特徴量をすべて説明変数として使用したケース)の precision は 0.710 に向上している. また, Promise データの場合, 特徴量種別 0 の precision は 0.273 に対して, 特徴量種別 2 の precision は 0.323 に向上している.

表 20 識別精度が改善した例(OSS データ)

閾値番号	特徴量種別	ｱﾙｺﾞﾘｽﾞﾑ	precision
3	123	2	0.710
3	13	2	0.667
3	23	2	0.667
3	3	2	0.636
3	12	2	0.625
3	1	2	0.600
3	0	2	0.583

表 21 識別精度が改善した例(Promise データ)

閾値番号	特徴量種別	ｱﾙｺﾞﾘｽﾞﾑ	precision
2	2	1	0.323
2	3	1	0.303
2	13	1	0.286
2	1	1	0.281
2	123	1	0.281
2	0	1	0.273
2	23	1	0.267

これらの結果だけでは, 特徴量を利用することによる効果は有効とは断定できないため, 次に, 実験 2 の結果から得られた識別の precision の上位 100 位のうち, OSS データ/Promise データそれぞれについて特徴量種別毎の内訳を確認することとした. 特徴量種別毎の内訳を表 22 に示す. 表 22 より, OSS データを用いた実験結果では, 特徴量種別が 0 のケース(特徴量を説明変数として使用しないケース)は 11 件あるのに対し, 特徴量種別が 0 以外のケース(何らかの特徴量を説明変数として使用するケース)は 89 件であった. これに対して, Promise データを用いた実験結

果では，特徴量種別が 0 のケースは 12 件あるのに対し，特徴量種別が 0 以外のケースは 88 件であった．表 22 より，有意な特徴量はみあたらないこと，及び，特徴量を利用しない実験よりも特徴量を利用した実験件数が多いことを踏まえ，特徴量を利用することにより識別精度が向上することは断定できないが，特徴量を利用することにより識別精度が向上する可能性はあると言える．

表 22　上位 100 位までの特徴量種別毎の内訳

特徴量種別	件数	
	OSS データ	Promise データ
0	11	12
1	12	12
2	8	13
3	14	12
12	12	14
13	16	12
23	14	13
123	13	12

今回の実験結果では，一意に優位な特徴量(もしくは，特徴量の組み合わせ)は見つからず，また特徴量を使用しても改善されないケースもあるため，特徴量と識別精度との因果関係や新たな特徴量を見出す等が，今後の検討課題である．また，実験 2 で得られた全データにおいて，予測閾値と学習アルゴリズムが同一である場合における特徴量の利用有無による識別の precision の精度を比較したところ，特徴量の組み合わせには顕著な傾向は見られなかった．しかしながら，特徴量を利用することにより識別精度が改善した例も複数あることから，特徴量を利用することで識別精度が改善する可能性はあると言える．

5.4. 実験 3 の評価

実験 3 の結果，微増であるが特徴量を利用することで向上するケースは存在した．増加件数は，OSS データにおいて，特徴量を使用した総件数 16800 件のうち 725 件(全体の 4.32%)にて，識別の precision が増加した．また，Promise データにおいて，特徴量を使用した総件数 4200 件のうち 227 件(全体の 5.4%)にて，識別の precision が増加した．OSS データ/Promise データそれぞれにおける，特徴量種別毎の増加件数の内訳を表 23 と表 24 に示す．これらの結果から，増加ポイント数は微増であるものの，特徴量を利用し，かつ，マイニングを行うことで識別精度の改善

が図られる可能性は確認できたと判断する.

なお, 特徴量を利用しても識別の precision が改善しないケースについての要因分析及び, さらなる効果的な特徴量の探索は今後の検討課題としたい.

表 23　特徴量種別毎(0 は除く)の内訳(OSS データ)

特徴量種別	precision の増加件数	総件数	増加比率
1	67	2400	2.79%
2	74	2400	3.08%
3	83	2400	3.46%
12	109	2400	4.54%
13	116	2400	4.83%
23	130	2400	5.42%
123	146	2400	6.08%
合計	725	16800	4.32%

表 24　特徴量種別毎(0 は除く)の内訳(Promise データ)

特徴量種別	precision の増加件数	総件数	増加比率
1	20	600	3.3%
2	17	600	2.8%
3	32	600	5.3%
12	32	600	5.3%
13	34	600	5.7%
23	43	600	7.2%
123	49	600	8.2%
合計	227	4200	5.4%

5.5. 結言

本章では, 4 章で述べた提案手法の有効性を実証するために行った各実験の結果を示しながら, それぞれの結果に対する考察及び評価を行った. その結果, 特徴量を加味すること及びマイニング手法により, 識別精度が向上する可能性があることを示した. 但し, 今回採用した特徴量では, 有意な識別精度は確保できなかったため, 他の特徴量を探索する必要があるが, これば, 次回の課題としたい. 次章では, 本論文の妥当性への脅威について考察を行う.

第 6 章

妥当性への脅威

第6章　妥当性への脅威

6.1. 緒言

提案手法の適用可能性を実証するために 4 章で述べた実験を行い，5 章で，それぞれの実験結果を示すと共に，それぞれの実験結果に対する考察及び評価を述べた．これらの実験は提案手法の適用可能性を実証するものであり，実験により導いた結果そのものの妥当性や，結論の推論プロセスで利用した手法の適用妥当性を考察する必要がある．Barbara[35]によれば，最低限，次の 2 つの考察は必要であるとしている．

- 内的妥当性(Internal validity)
- 外的妥当性(External validity)

内的妥当性とは，目的変数と説明変数の因果関係や，実験手順と実験結果との関係において，確かに整合しているということを考察することを言う．これに対して，外的妥当性は「得られた知見は一般化可能なのか」を考察することを言う．

本章では，4 章で示した実証実験の内容及び，5 章で結論付けた内容を対象に，その妥当性に影響を及ぼす可能性のある問題について考察する．　内的妥当性への脅威では，実証実験で得られた結果及び考察が妥当なのかを考察する．また，外的妥当性では，得られた知見は一般化可能かどうかについて考察する．

6.2. 内的妥当性への脅威

6.2.1. メトリクスの計測時点と誤り有無の判定時期の関係(OSS データ)

Promise データは，モジュール毎に計測したメトリクスデータが公開されているが，OSS データは自前でモジュール毎にメトリクスデータを計測する必要があった．計測のタイミングは 4.2.3 で述べた通り，下記(1)と(2)の方法があるが，本論文では(2)を採用した．理由は，ファイル毎にメトリクスの計測時期が異なるため，プロジェクト全体をまとめたデータとしては利用できないからである．

(1) 誤りの修正を行ったリビジョンから直前のリビジョンのソースコードに対してメトリクスを測定する．

(2) ある時点のプロジェクト全体のソースコードからメトリクスを計測し，その時点から一定期間のコミットログを基に，誤りの有無を確認する．

本論文では(2)を(1)の方法の近似値として採用したが，最も妥当な誤りの出現有無の判別とメトリクス計測タイミングはどうあるべきなのかについては今後の課題としたい．

6.2.2. 誤り有無の判定方法について(OSS データ)

4.2.2 で述べた通り, OSS データにおいては, バグ管理システムで誤りの詳細は公開されているものの, モジュール(ファイル)毎の誤り改修の有無に関する情報は含まれていない. 従って, ファイル単位における誤りの存在有無は自前で計測する必要があった. 誤りの存在有無の計測にあたっては, ファイル毎に, ある期間における Subversion のコミットログ を取得し, 取得したログから”bug”, ”fix”といった単語が含まれるか否かで判定した.

本方式を採用するにあたり, コミットログの有効性は, サンプリングによる, コミットログとバグ管理システムに登録されている誤り情報を目視確認することにより担保している.

誤り有無の計測方法については SZZ アルゴリズム[30]等, 複数の手法が提案されているが, これらの手法の相違が実際にどのような影響があるのかについての考察は今後の課題としたい.

6.2.3. 誤り率(特徴量)の予測値の妥当性

3.2 にて, プロジェクトの特性を現す要素として特徴量を提案し, その 1 つとして, 誤り率(誤り密度が 10%以上, 10%未満かつ 1%以上, 1%未満の 3 種類を 0～2 の数値で表現する)を提示した. 誤り率を特徴量として採用した理由は, 各プロジェクトにおける誤りの出現傾向(開発期間中は誤りの発生は多く, リリース後は収束すると考えられる)もプロジェクトの特徴と考えたからである. 今回は, 「誤り密度が 10%以上, 10%未満かつ 1%以上, 1%未満の 3 種類」を検査対象プロジェクトの開発進捗にあわせて想定することにしたが, 誤り率を特徴量とすることの妥当性については今後, 考察が必要である.

6.2.4. 実験 2 と実験 3 の意義について

4.4 で述べた通り, 実験 2 で実施した 10 重交差検証は, 予測器の構築にて検査データと同一データが訓練データとして使用される可能性があり, 識別精度が比較的高くなる傾向がある. そこで, より厳しくマイニング精度を評価するために, 訓練データと検査データを独立させた識別実験として実験 3 を行った. その結果, 10 重交差検証による結果と検査対象プロジェクトと訓練対象プロジェクトを完全に独立させることで, より的確な検証条件であっても, 提案手法による error-prone モジュール予測器の識別精度が向上する可能性は示すことが出来た.

6.2.5. 実験で使用したデータ

4 章で述べた実験で使用したデータは, OSS データ及び Promise データの 2 種類である. OSS データは sourceforge から自前で取得したデータである. 取得する際は, 十分な検証を行ったツー

ルを用いて人手が介在することなく取得しているため，手順誤りの可能性は無いと考える．

また，OSS データにおける誤り件数は，Subversion のログから誤りの改修を行ったと判断できるキーワード(bug， fix)を基に算出を行っている．Bug というキーワードはともかく，fix という単語で検出された場合，Bug fix 以外にも利用用途があるため，果たして本当に Bug fix を意味するのか？という疑問は常に付随する．しかしながら，データを収集した当時，複数個所のサンプリングによりログを確認する作業を行っていることから，データの妥当性は保証できると考える．なお，Promise データについては，公開されているデータのうち，NASA が提供しているもののみを使用していることや，各プロジェクト間でメトリクスデータの種類が異なる点は，各プロジェクトで共通するメトリクスのみ使用する対応を行っていることから，信頼できるデータであると判断する．

6.2.6. 実験で使用したメトリクス種別と学習アルゴリズムの妥当性について

実験で使用したメトリクス種別と学習アルゴリズム群は，本論文で提案した手法の適用可能性を検証することを目的としており，十分なものを見つける事が目的ではない．従って，メトリクス種別や学習アルゴリズム群の十分性に対する対応は今後の課題である．

6.3. 外的妥当性への脅威

本論文で実施した各実験は，複数のプロジェクトから得られたデータを使用しているが，今回使用したプロジェクトは比較的，小規模であり，商用システムのような大規模プロジェクトのデータは使用していない．従って，本論文で使用したプロジェクトの範囲で，有意な識別精度で識別可能である事例を実験で示したが，実験 2 や実験 3 の結果において，特徴量を使用しても識別精度が向上しなかった結果も存在したように，本論文での提案手法による識別精度は検査対象プロジェクトと訓練プロジェクトとの相性で変わってくる．そのため，相性が悪い場合は本論文で主張したような有意な精度での識別が行えない場合がある．検査対象プロジェクトと訓練プロジェクトの相性を司る特徴量を特定する条件等については明らかにできていない．従って，さらに研究を行う必要があると考える．

6.4. 結言

本章では，内的妥当性への脅威及び外的妥当性への脅威について考察を行った．内的妥当性への脅威では，メトリクスの計測時点と誤り有無の判定時期の妥当性，誤り有無の判定方法についての妥当性，誤り率を検査対象プロジェクトにおいて予測値として使用することの妥当性，実験で使用したメトリクス種別と学習アルゴリズムの妥当性及び，使用データの妥当性について考察した．また，外的妥当性については，本論文が提案した手法に一般性があるのか否かについて考察した．

第 7 章

結論

第7章　結論

ソフトウェア開発では，限られた期間内にある程度の品質を確保する必要がある．品質確保策としては，製作工程ではソースコードを対象としたインスペクションを行うことで品質の向上が期待できる．また，試験工程においては，網羅的な試験観点で用意した試験項目を用いてソフトウェアテストを行うことにより，納入までの間に誤りを摘出することで品質を向上させることが期待できる．しかし，インスペクション及びソフトウェアテストは開発規模に応じて，工数も必要となるため，開発の効率化を図るためには，品質を劣化させないことを前提として，実施対象範囲の絞り込みが必要となる．実施対象範囲を絞り込むためには， error-prone モジュールの予測を行い，error-prone モジュールと予測されたモジュールに対してインスペクションやソフトウェアテストを集中実施することにより品質向上及び開発の効率化を図ることが期待できる． なぜなら，早期に誤りを摘出するだけでなく，インスペクションやデバッグに費やす工数も削減することが出来るからである．

これまで，ソフトウェア開発の効率化を目的に，数多くの error-prone モジュールの予測手法が提案されているが，現状，あらゆるプロジェクトに対して汎用的に使用でき，かつ最適な予測器は存在しない.また，予測器を構築するための訓練データに関して，一般的には自プロジェクトの過去データを使用するが，新規開発においては適用できないケースが存在することや，必ずしも自プロジェクトの過去データが最適ではないという事例もある．また，検査対象プロジェクトに対して，最適（もしくは適切）な予測器(訓練データと予測アルゴリズム)をどのようにして選ぶかについては，ほとんど研究が行われていない．

HE はメトリクスの集約値と予測結果の精度の関係を学習させることで，検査対象プロジェクトに対して適切な予測器を見つけ出す手法を提案している．本論文では HE が提唱した手法を応用して，より適切にマイニングを行うためにプロジェクトの特徴量を導入し，method mining に基づく error-prone モジュールの予測を行うために，検査対象プロジェクト毎に適切な error-prone モジュール予測器を構築できる訓練データと予測アルゴリズムを識別するマイニング手法を提案した．そして，複数のデータを用いて，マイニング手法の可能性を実証した．

実証の結果，特徴量を加味すること及びマイニング手法により，error-prone モジュール予測器の識別精度(識別の precision)が向上する可能性があることを示した．但し，実験 2 や実験 3 の結果において，特徴量を使用しても識別精度が向上しなかった結果も存在したように，本論文で提案した手法による識別精度は，検査対象プロジェクトと訓練データの取得元プロジェクトとの相性で変わってくると考えられる．そのため，相性が悪い場合は本論文で主張したような有意な精度での識別が行えない場合がある．現時点では，検査対象プロジェクトと訓練データの取得元プロジェクトとの相性を司る特徴量を特定する条件等については明らかにできていない．

上記を踏まえ，以下に示す課題について更なる研究が必要である．

(1) 適切な特徴量の考察が必要である. 各モジュールの error-proneness はソースコードのメトリクスだけで決まるものではない. プロジェクト全体が持っている種々の特徴と個々のモジュールの種々の特徴が合わさって誤りの混入を招く. 従ってどのようなメトリクスが error-proneness に影響するかはプロジェクトの特徴量に依存する. 本論文では特徴量が関わることは示せたが, 特徴量の適切性や, 適切な特徴量については考察していない. 今後, これらの問題を解決する事でより精度の高いマイニングを可能にしたい.

(2) 予測器の再利用においては検査対象プロジェクトと似通った訓練プロジェクトに基づく予測器が必要である. 同様に適切な手法のマイニングには, 似通ったプロジェクトに基づいて作られた知識ベースが必要である. どのような方法で「似通ったプロジェクト」を選ぶかについての考察が必要である.

(3) 本論文では, ソースコードメトリクスのみを利用した. ソフトウェアメトリクスには, ソースコード以外にも, 開発プロセスから抽出できる各種メトリクス等, いろいろなソフトウェアメトリクスが存在する. 適切なメトリクス種別をどのような方法で選択するのかについては, マイニング手法で解決できないため, 各種メトリクスの入手手順の検討及び, メトリクス種別の選択方法やマイニング手法による適用可否について考察が必要である.

(4) ある要因が決定的に誤り混入の要因として影響するわけではないため, 確率論を基にしたアルゴリズムを, error-prone モジュール予測に適用する予測アルゴリズムとして採用する案はある. しかし, error-prone モジュール予測における説明変数で使用するメトリクスと誤り要因とは相関関係が存在する部分もあることから, 決定木によるアルゴリズムを適用することも有意と考えられる. 今後, 最適な予測アルゴリズムの選択方法についての考察が必要である.

(5) 本論文では, OSS データに対してメトリクスの計測と誤り有無の計測タイミングはモジュール単位ではなく, プロジェクト単位に行った. また, Promise データは公開データのため, メトリクス測定のタイミングと誤りの計測方法については非公開である. 今後の研究において, プロジェクト単位による, 適切なメトリクス計測のタイミングと誤り有無の判別方法との整合性をどのようにとるのか考察が必要である.

(6) 誤り有無の計測方法については SZZ アルゴリズム[30]等, 複数の手法が提案されているが, これらの手法の相違が実際にどのような影響があるのかについては考察が必要である.

(7) 本論文で提案した手法はプロジェクト単位で error-prone モジュール予測器を識別するが，モジュール毎に最適な error-prone モジュール予測器を識別することはできるのか，また，識別する手法の評価はどのように行うのかについては考察が必要である．

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，御支援を受け賜りました．ここに感謝の意を表します．

本研究全般，及び，本論文を執筆するにあたり，指導教官の海尻賢二教授から，丁寧かつ熱心なご指導を賜りました．今から振り返ると，仕事と学業の両立のため，登校できる機会が限られる中，定期的に東京オフィスにて面談の場を設けていただきました．また，疑問点等，メールで質問させていただいたときは，迅速な回答をいだきました．その都度，研究を進めるにあたり，要所を押さえた御助言，御指導を受け賜りました．誠にありがとうございました．ここに感謝の意を表します．

参考文献

[1] ソフトウェア開発データ白書 2014～2015(独立行政法人情報処理推進機構(IPA) 技術本部ソフトウェア高信頼化センター(SEC)刊

[2] Glenford J.Myers: The Art of Software Testing, Weley and Sons, New York, 1979.

[3] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch: Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Tr. on S.E, vol.34, no.4, pp.485-496, 2008.

[4] Z. HE, F. Shu, Y. Yang, M. Li, and QingWang: An investigation on the feasibility of cross-project defect prediction, Automated Software Engineering, vol.19, no.2, pp.167-199, 2012.

[5] T. Menzies, J. Greenwald, and A. Frank: Data mining static code attributes to learn defect predictors, IEEE Tr. on S.E, vol.33, no.1, pp.2-13, 2007.

[6] M. D'Ambros, M. Lanza, and R. Robbes: An extensive comparison of bug prediction approaches, MSR, IEEE, pp.31-41, 2010.

[7] E. Arisholm, L.C. Briand, and E.B. Johannessen: A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, The Journal of Systems and Software, vol.83, no.1, pp.2-17, 2010.

[8] E. Arisholm, L.C. Briand, and M. Fuglerud.: Data mining techniques for building fault-proneness models in telecom java software, ISSRE, IEEE, pp.215-224, 2007.

[9] L.C.Briand, W.L. Melo, and J. Wust: Assessing the applicability of fault-proneness model across object-oriented software projects, IEEE Tr. on S.E, vol.28, no.7, pp.706-720, 2002.

[10] M. Jureczko and D.D. Spinellis: Using 12 object-oriented design metrics to predict software defects, RELCOMEX, pp.61-81, 2010.

[11] 小林寛武, 戸田航史, 亀井靖高, 門田暁人, 峯恒憲, 鵜林尚靖: 11 種類の fault 密度予測モデルの実証的評価," 電子情報通信学会論文誌, vol.J96-D, no.8, pp.1892-1902, 2013.

[12] S. Watanabe, H. Kaiya, and K. Kaijiri: Adapting a fault prediction model to allow inter language reuse, Promise, ACM, pp.19-24, 2008.

[13] B. Turhan, T. Menzies, and A.B. Bener: On the relative value of cross-company and within-company data for defect prediction, Empirical Software

En-gineering, vol.14, no.5, pp.540-578, 2009.

[14] B. Turhan, A. Bener, and T. Menzies: Regulari-ties in learning defect predictors, Profes, pp.116-130, 2010.

[15] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy: Cross-project defect prediction a large scale experiment on data vs. domain vs. process, FSE, pp.91-100, 2009.

[16] M. D'Ambros, M. Lanza, and R. Robbes: Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empirical Software Engineering, vol.17, no.4, pp.531-577, 2012.

[17] Y. Ma and B. Cukic: Adequate and precise evaluation of quality models in software engineering studies, Promise, IEEE, pp.1-9, 2007.

[18] T. Menzies, J. DiStefano, A. OrregoS, and R. Chap-man: Assessing predictors of software defects, PSM, IEEE, pp.1-5 2004.

[19] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy: Cross-project defect prediction a large scale experiment on data vs. domain vs. process, FSE, pp.91-100, 2009.

[20] J. Ekanayake, J. Tappolet, H.C. Gall, and A. Bern-stein: Time variance and defect prediction in soft-ware projects - towards an exploitation of periods of stability and change as well as a notion of concept drift in software projects, Empirical Software Engi-neering,vol.17, no.4,5, pp.348-389, 2012.

[21] I.H. Witten and E. Frank: Data Mining - Practical Machine Learning Tools and Techniques, Morgan Kaufman, 2005.

[22] Kaur, A., Malhotra, R.: Application of Random Forest in Predicting Fault-Prone Classes, Advanced Computer Theory and Engineering, ICACTE '08, 2008.

[23] Arthur H. Watson, Thomas J. McCabe: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, NIST Special Publication, Vol.500-235, Aug. 1996.

[24] Shyam R. Chidamber, Chris F. Kemerer: A Metrics Suite for Object Oriented Design, IEEE Tr. on S.E, Vol.20, No.6, pp.476-493, Jun. 1994.

[25] 大西達也: Error-prone モジュール予測のためのデータ収集, 信州大学工学部情報工学科卒業論文, 2011.

[26] Understand. http://www.techmatrix.co.jp/quality/understand/

[27] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J.Krall, F. Peters, and B. Turhan: The promise re-pository of empirical software engineering data,

June2012. http://promisedata.googlecode.com

[28] Implementing Software Inspections, course notes, IBM Systems Sciences Institute, IBM Corporation, 1981.

[29] IEEE Standard Glossary of Software engineering Terminoogy, IEEE,1993.

[30] Sliwerski J., Zimmermann T. and Zeller A.: When do changes induce fixes? , In Proc. of MSR 2005, pp.1-5, 2005.

[31] E.J. Weyuker, T.J. Ostrand, and R.M. Bell: Com-paring negative binomial and recursive partitioning models for fault prediction, Promise, ACM, pp.3-9, 2008.

[32] Robert L. Glass, Facts and Fallacies of Software Engineering, Addison Wesley, 2003.

[33] 藏本達也，亀井靖高，門田暁人，松本健一： ソフトウェアプロジェクトをまたがる fault-prone モジュール判別の試み～18プロジェクトの実験から得た教訓，電子情報通信学会論文誌 J95D, No.3 pp.425-436, 2012.

[34] 亀井靖高，柗下真佑，柿元 健，門田暁人，松本健一： Fault-prone モジュール判別におけるサンプリング法適用の効果，情報処理学会論文誌 48(8)，2651-2662, 2007-08-15.

[35] Barbara A. Kitchenham, et al: Preliminary Guidelines for Empirical Research in Software Engineering, IEEE Tr. S.E., 28, 8 2002.

[36] Nagappan, N., T. Ball, and A. Zeller. : Mining metrics to predict component failures, Proceedings of the 28th International Conference on Software Engineering, pp.452-461, 2006.

[37] Zimmermann, T., R. Premraj, and A. Zeller. : Predicting defects for Eclipse, Proceedings of the Third International Workshop on Predictor Models in Software Engineering, pp.9, 2007.

[38] N.E.Fenton: Software Metrics: A Rigorous Approach, Chapman & Hall, 1991.

[39] D.Lawson, G. Coleman.: Investigating software measures to improve product reliability, Proc. of the 2002 ACM symposium on Applied computing, pp.1031 - 1035, 2002.

[40] C.V. Ramamoorthy, F.B. Bastani: Software reliability-Status and perspectives, IEEE Trans. Software Eng, Vol.SE-8, No.4, pp.354-371, 1987.

[41] B.W. Boehm: Software Engineering Economics, Prentice-Hall, 1981.

[42] A. J. Albrecht.: Measuring Application Development Productivity, Proc. of the Joint SHARE, GUIDE, and IBM Application Development Symposium, pp.83-92, 1979.

[43] 情報処理推進機構 IT スキル標準センター. http://www.ipa.go.jp/jinzai/itss/

[44] Jacek Ratzinger, et al: Quality Assessment based on Attribute Series of Software Evolution, WCRE 2007.

[45] Andrew R. Gray and Stephen G. MacDonell.: Software Metrics Data Analysis – Exploring the Relative Performance of Some Commonly Used Modeling Techniques, Empirical Software Engineering, Vol.4, No.4, pp.297–316, 1999.

[46] Rumelhart, D. E., Hinton, G. E. and Williams,R. J.: Learning Representations by Backpropagating Errors, Nature, Vol.323, pp.533-536, 1986.

[47] N. Ohlsson and H. Alberg.: Predicting fault-prone software modules in telephone switches, IEEE Tr. on S E., 22(12):886–894, 1996.

[48] J. C. Munson and T. M. Khoshgoftaar.: The detection of fault-prone programs, IEEE Tr. on S E, 18(5):423–433, 1992.

[49] Albert Endres, Dieter Rombach, A Handbook of Software and Systems Engineering, Pearson Education, 2003.

[50] Albert Endres, An analysis of errors and their causes in system programs, ACM SIGPLAN Notices, vol.10, issue 6, June 1975.

[51] B. Boehm, V.R. Basili, Gaining intellectual control of software development, IEEE Computer, vol.33, issue 5, pp.27-33, May 2000.

[52] N.E. Fenton, N. Ohlsson, Quantitative Analysis of Faults and Failures in a Complex Software System, IEEE Tr. on S E., vol.26, no.8, pp.797-814, Aug.2000.

[53] G. Andersson, P.Runeson, A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems, IEEE Tr. on S E., vol.33, no.5, pp.273-286, May 2007.

関連論文

[1] 内宮秀明, 小形真平, 海尻賢二: Method Mining に基づく error-prone モジュールの予測, バイオメディカル・ファジィ・システム学会誌, Vol.16, No.2,pp.45-58(2014)

[2] Hideaki Uchimiya, Shinpei Ogata, Kenji Kaijiri: Method Mining in Experimental Software Engineering, The 2nd International Conference on Systems and Informatics (ICSAI 2014)

[3] 内宮秀明,小形真平, 海谷治彦,海尻賢二: 最適な Error-prone モジュール予測器を選択する手法の提案, 信学技報, vol. 113, No.414, KBSE2013-77(2014-01), pp.71-76, 2014 年 1 月