# 先 端 科 学 技 術 演 習 報 告 書

題　目　　テキストフィルタに基づく不具合検出手法の
細粒度リポジトリへの適用と評価

指導教員　　水野 修　准教授

京都工芸繊維大学 工芸科学部 先端科学技術課程

学生番号　　08171736

氏　　名　　松村 好剛

平成24年2月15日提出

# テキストフィルタに基づく不具合検出手法の細粒度リポジトリへの適用と評価

平成 24 年 2 月 15 日　　08171736　松村 好剛

## 概　要

開発の早期段階でソースコード中の fault-prone モジュールを特定することはプロダクトの品質向上につながる．これまでにも fault-prone モジュールを予測する多くの研究が行われてきたが，それらは全てメトリクスベースによるもので，ソフトウェアメトリクスの測定に余分な工数やコストがかかってしまう場合もある．我々は，簡単に適用でき，かつ効果の高い fault-prone モジュールの検出手法として，spam フィルタに基づく fault-prone モジュール検出法「Fault-prone フィルタリング」を提案している．この手法は全く事前の知識がない状態からでも開発プロジェクトに適用できるという特徴を持っている．

こうした fault-prone モジュール予測手法は，ソフトウェアのモジュールを単位として予測を実施する．しかし，このモジュールの単位が大きすぎるとせっかく不具合があると予測できても、そのモジュール内のどこにあるのかを特定できない．そのため，細粒度のモジュールを利用した不具合予測が必要とされている．

本研究では，細粒度モジュールを扱えるリポジトリを利用して，Fault-prone フィルタリング法を細粒度モジュールに対して適用する．その際，研究設問として，細粒度モジュールによる予測は従来の予測精度よりも良くなるか否か，そして，それは実用的な値が得られるかどうかを調査する．

実験の結果，予測精度はやや低くなるものの，その値は高い水準を維持しており，実用的に意味のある値であることを確認した．この結果，細粒度モジュールによる fault-prone モジュール予測は，早期の不具合発見に有効であることを確認した．

# 目　次

# 1. 緒言

ソフトウェアの社会的位置づけがますます重要性を帯びてくると共に，その品質を保証することが重要な目標となっている．高品質なソースコードの作成がプロダクトの品質向上につながるため，早期段階での，不具合を含みやすい (fault-prone, FP) モジュールの特定が求められている．これまでにも fault-prone モジュールを予測する多くの研究が行われてきた [1–10]．これらの手法では，主にモジュールの複雑さや変更頻度などのソフトウェアメトリクスを用いて予測モデルを構築している．しかし，こうしたソフトウェアメトリクスは測定に余分なコストがかかってしまう場合もある．また，構築された予測モデルが別の環境で利用できるという保証も無かった．

そこで，我々はソースコードのみを入力として与え，事前知識が無くても予測モデルが構築可能な手法を提案した．(以降，Fault-prone フィルタリングと呼ぶ．) この手法では，迷惑メール検出に利用される SPAM フィルタをソフトウェアのソースコードに対して適用し，純粋にテキスト情報のみから FP モジュールを予測する [11–13]．

こうして多数の研究が実施されている fault-prone モジュール予測であるが，これまでに提案された多くの手法で,「クラスまたはファイルを粒度の単位として利用している」という問題が存在する．このため，現実に不具合予測手法として利用する場合に,「このファイル（クラス）に不具合がある」と指摘されても，不具合の存在する範囲が広すぎて活用できない恐れがある．

そこで，ソフトウェアの「モジュール」を細かく扱うための手法「Historage」が模索されている [14]．Historage [14] では，Git で管理されるソフトウェアのリポジトリを再スキャンすることで，クラス，メソッド，コンストラクタ，フィールド定義を別々の実体として管理できる．そのため，従来のファイル単位よりも細かいレベルでのソフトウェア構成単位の管理が可能になっている．本研究では Historage を用いたソフトウェアリポジトリに対する Fault-prone フィルタリング法の適用を試みる．

一方で，細粒度でのモジュール管理であれば不具合の修正には役立つが，それで不具合の予測精度が向上するとは限らない．そこで本研究では不具合モジュールの予測精度が細粒度リポジトリを採用することでどの程度変化するかを実験を通じて調べる．

本報告の以降の構成は次の通りである．2 章では，研究の目的について述べる．3 章では本実験での準備として，細粒度リポジトリ (Historage) と Fault-prone フィルタリング法の概要を述べる．4 章では，オープンソースソフトウェアに対して実施した実験について述べる．実験で得られた結果を 5 章にまとめる．また，6 章では得られた結果について考察する．最後に，7 章で結言を述べる．

## 2. 研究の目的

本研究の目的は細粒度リポジトリ「Historage [14]」で管理されるソフトウェアのリポジトリに対して，ソフトウェア不具合混入モジュールを予測する手法「Fault-prone filtering [13]」を適用し，その予測精度を調べることである．具体的には，従来通りのファイル単位での管理リポジトリと比較して，Historage が提案するメソッド単位の管理リポジトリにおける不具合モジュールの予測精度がどう変化するかを分析する．

具体的には，細粒度モジュール，すなわち，クラス，メソッド，コンストラクタ，フィールド定義などの情報のみを用いた予測の結果と，従来通りにファイル全体（仮に粗粒度モジュールと呼ぶ）をモジュールと見なして行った予測結果との比較を行う．この目的のために，本研究では以下の研究設問を設定する．これらの研究設問に答えることで本研究で明らかにしたいことを示す．

RQ1 細粒度モジュールを学習させて細粒度モジュールを予測させることで，不具合の予測精度は向上するか．

RQ2 細粒度モジュールを学習させて粗粒度モジュールを予測させることで，不具合の予測精度は向上するか．

RQ3 粗粒度モジュールを学習させて細粒度モジュールを予測させることで，不具合の予測精度は向上するか．

# 3. 準備

## 3.1 細粒度リポジトリ

Hata らは Historage と名付けた細粒度リポジトリ構築手法を提案している [14]．この手法は，分散版管理システム Git をベースとし，各コミットでのファイルをクラス定義部，フィールド定義部，コンストラクタ部，メソッド定義部に分割し，それぞれの部分を別個にデータベースに格納し，その履歴を管理するものである．

Historage は元のリポジトリを破壊せず，元のリポジトリが保っていたファイルレベルでの履歴構造はそのまま残して新たに細粒度モジュールの履歴構造を構築する．そのため，本実験では従来法の適用環境はファイルレベルのモジュールを抽出したものを用い，提案法には細粒度モジュールを抽出したものを用いる．

## 3.2 Fault-prone フィルタリング

Fault-prone フィルタリングはスパムメール (迷惑メール) の判別をおこなうスパムフィルタで用いられるテキスト分類フィルタ技術を利用する．スパムフィルタは，過去に受信した電子メール内の単語群を利用して，スパムメールと通常のメールを判別するための辞書を作成する．そして，新たに受信した電子メールについては，ベイズ識別などの技術により，スパムか否かを判定する．学習は随時行われ，辞書は常にその時点の状況を反映したものになるため，新種のスパムメールなどにも柔軟に対応できるとされている．この考えは，スパムメールには特定の単語群や文章が頻繁に含まれている，という事実に基づいている．

我々は，この考え方がソースコード内の不具合についても適用できるのではないかと考えた．もちろん，元々悪意を持って作成されたスパムメールと，意図的ではないがバグが混入したソースコードを全く同じものと見なすのは無理があるかもしれない．しかし，一連のソフトウェア開発においては，同じ開発者が同じ文脈でバグを混入することや，類似の関数や API の呼び出しなどにおいてバグを混入してしまうことは良くあることだと考えられる．すなわち，スパムメールの中の特定の単語のように，バグが存在するところには特定のコード片が存在するのではないかと類推した．

### 3.2.1 スパムフィルタ: CRM114

本研究ではテキスト分類フィルタとして CRM114 を用いた [15]．主にスパムフィルタとして開発されているが，汎用的な用途，例えば，計算機のログ監視やネットワークのトラフィック監視などにも活用できるとされている．また，現在開発されているメールフィルタの中でも高い予測精度をあげているものの 1 つである．

CRM114 は基本的にはベイズ識別を利用したテキスト分類フィルタであるが，複数の単語を組み合わせたものをトークンと呼び，学習・分類の単位として利用することが大きな特徴である．従来のテキスト分類フィルタは 1 単語をトークンとしているのに対し，複数単語の組をトークンとすることで，より複雑な学習が可能となっている．本研究では，CRM114 のデフォルトの分類手法である “Othogonal Sparce Bigrams Marcov model (OSB)” を使用する．OSB は任意の連続する 5 単語の組合せのうち，2 単語からなるものだけをトークンとする手法である．

OSB によるテキスト処理について以下に簡単に示す．表 3.1 は “`a = b + 1 ; return a;`” という文について，トークンを生成した様子である．OSB ではある単語を起点として 5 単語からなる単語列に対し，正確に 2 単語のみを含むもののみを学習・分類の対象として抽出する．なお，本研究ではプログラム言語中の区切り文字をあらかじめ排除するため，“;” は単語群に含まれていない．

### 3.2.2 利用したツール

本研究では，研究室において開発・保守されている Fault-prone フィルタリング用ライブラリ (`FPFilter.pm`) を利用した．FPFilter.pm は perl 言語で書かれたライブラリで，上記 CRM114 をソースコードの予測に特化した形で利用できるようになっている．

**表 3.1　OSB で生成されるトークン例**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | = | | | | | | | | |
| a | | b | | | | | | | |
| a | | | + | | | | | | |
| a | | | | 1 | | | | | |
| | = | b | | | | | | | |
| | = | | + | | | | | | |
| | = | | | 1 | | | | | |
| | = | | | | return | | | | |
| | | b | + | | | | | | |
| | | b | | 1 | | | | | |
| | | b | | | return | | | | |
| | | b | | | | | | a | |
| | | | + | 1 | | | | | |
| | | | + | | return | | | | |
| | | | + | | | | | a | |
| | | | | 1 | return | | | | |
| | | | | 1 | | | | a | |
| | | | | | return | a | | | |

# 4. 実験

## 4.1 実験対象

本実験では，オープンソースソフトウェアである 4 つのプロジェクト，Eclipse ECF, Eclipse EMF compare, Eclipse Xpand, Eclipse Webtools incubator を対象とする．

## 4.2 実験の準備

### 4.2.1 Historage への変換

まず，4 つのプロジェクトそれぞれに対して Hata らの提供する `mkHistorage` ツールを適用し，それぞれを細粒度リポジトリに変換する．

### 4.2.2 バグ情報との統合

次に，同じく Hata らが提供するツール `szz_tools` を用い，Historage で管理された 4 つのリポジトリに対して，バグデータベースの情報とリポジトリの更新箇所を結びつける．実際には Git の tag コマンドを利用して，バグの混入時点と除去時点に対してタグを添付する．ここでは Sliwerski らが提案した SZZ アルゴリズム [16] を利用して，混入時点と除去時点の特定を行っている．

**利用したツール**: `szz_tools`

> Historage に SZZ アルゴリズム [16] を適用し，不具合の混入時点と除去時点をリポジトリ内に書き込むツール．perl 言語で書かれており，コマンドラインから実行する．

### 4.2.3 不具合を含むモジュールと含まないモジュールの抽出

前節でタグを付けた Historage が管理するリポジトリから，不具合有りモジュールと無しモジュールを判別しなくてはならない．ここでの判別は予測ではなく，実測の不具合の有無となる．

**作成したツール**: `getModules.pl`

Historage で処理された各プロジェクトの Git リポジトリから，不具合有りモジュールと不具合なしモジュールをより分けるプログラムである．

利用方法は次の通りである．

```
$ getModules.pl 対象リポジトリ 結果を納めるディレクトリ
```

「対象リポジトリ」には.git ディレクトリが存在するパスを渡す.「結果を納めるディレクトリ」以下には “ft”, “nf” の 2 つのディレクトリが生成され，それぞれが「不具合有り」,「不具合無し」と判断されたモジュールを格納する．このディレクトリ内には，Git が管理するハッシュ値をファイル名に持つファイルが生成される．

## 4.3 実験の構築

先に挙げた研究設問を調べるために，ここでは以下の 3 種類の実験を実施する．

e1 学習に**粗粒度**モジュールを利用し，予測に**粗粒度**モジュールを用いる．

e2 学習に**細粒度**モジュールを利用し，予測に**細粒度**モジュールを用いる．

e3 学習に**細粒度**モジュールを利用し，予測に**粗粒度**モジュールを用いる．

e4 学習に**粗粒度**モジュールを利用し，予測に**細粒度**モジュールを用いる．

e1 の実験は，従来法の実験である．今回はこれを比較のベースとする．e2, e3, e4 の実験は Historage の細粒度モジュールを利用した実験となる．e2 は全てを細粒度モジュールとしたときの予測精度を測る実験である．e3 は学習単位のみを細かくすることで，予測精度に影響が出るかどうかを調べる．また，e4 は学習単位を粗くして，細かい単位での不具合予測がきちんと行えるかどうかを確認する．

## 4.4 実験の手順

### 4.4.1 実験 e1 と e2

実験 e1 と e2 は交差検証を用いて実験を行う．交差検証とは，対象となるデータの一部分を予測用に確保し，残りの部分を学習したモデルで予測を実施する実験を

データの全体が予測を網羅できるように実行する手法である．今回の実験では，全データを十等分した実験を行う．これを十重交差検証と呼ぶ．

**作成したツール**: `exp_cv.pl`

Fault-prone フィルタを適用させつつ，$n$ 重交差検証を実施するプログラムである．比較のため，Java ファイルのみを利用した予測 (e1) と細粒度モジュールを利用した予測 (e2) の 2 種類の実験を同じプログラムで行えるように設計されている．

利用方法は次の通りである．

```
$ exp_cv.pl -config configfile -exp experiment_name -target ext -iter n -p
```

-config オプションには設定ファイル名を示す．-exp オプションには設定ファイル中の実験名を示す．（設定ファイル内には複数の実験を記すことができる．）-target オプションには実験の対象とするモジュールの単位を記す．-iter オプションは交差検証の重数を設定する．また，-p オプションを指定することで，交差検証の繰り返しを全て並列実行させることができる．これにより，実験を 6〜7 倍高速に動作させることに成功している．

### 4.4.2 実験 e3 と e4

実験 e3 と e4 は学習データと予測データを分離した実験を行う．これは学習・予測データが別なので交差検証をする意味がほとんど無いからである．そのために，以下のツールを作成している．

**作成したツール**: `exp_one.pl`

Fault-prone フィルタを与えられた学習データと予測データに適用させるプログラムである．利用方法は次の通りである．

```
$ exp_one.pl -config configfile -exp experiment_name -learn ext -test ext
```

-config オプションには設定ファイル名を示す．-exp オプションには設定ファイル中の実験名を示す．-learn オプションには学習の対象とするモジュールの単

位を記す．-test オプションには予測の対象となるモジュールの単位を表す．なお，-learn と-test は複数設定することができる．

これらのツールを用いて，実験対象とした 4 つのプロジェクトに対して実験を実施する．次章では実験結果について述べる．

# 5. 実験結果

実験対象とした 4 つのプロジェクトに対し，実験 e1 から e4 を実施した．

## 5.1 評価指標

表 5.1 は今回の実験で得られる結果の凡例である．$N_1$，$N_2$，$N_3$，$N_4$ は横に示す予測と縦に示す実測にそれぞれ該当する例数を表す．この得られた結果の評価指標として正答率 (Accuracy)，再現率 (Recall)，適合率 (Precision) を用いる．

正答率 (Accuracy) は全モジュールのうち，実測が not faulty(NFT) のモジュールを non-fault-prone(NFP)，実測が faulty(FT) のモジュールを fault-prone(FP) と正しく予測した割合を示す．よって正答率は凡例の表 5.1 を用いると以下のように定義される．

$$Accuracy = \frac{N_1 + N_4}{N_1 + N_2 + N_3 + N_4}$$

正答率は予測の全体的な傾向を把握するには便利であるが，実測値の偏りなどに大きく影響を受ける指標であるため，この値のみで予測の良さを判断するのは危険である (注 1)．そのため，本研究では以下の 2 つの指標に重点を置く．

再現率 (Recall) は実測が FT である全てのモジュールのうち，正しく FP と予測できたものの割合を示す．よって再現率は以下のように定義される．

$$Recall = \frac{N_4}{N_3 + N_4}$$

直感的には，再現率は「予測によって実際の不具合をどれだけ網羅できるか」を示している．そのため，Fault-prone モジュール予測にあっては，不具合を未然に防ぐという観点から最も重視すべき指標と言える．

適合率 (*Precision*) は予測が FP であるモジュールのうち，実測が FT であったものの割合を示す．すなわち，適合率は以下のように定義される．

$$Precision = \frac{N_4}{N_2 + N_4}$$

(注 1)：例えば，全ての予測を NFP とした場合でも，正答率は $\frac{N_1}{N_1+N_3}$ となるので，実測での FP と NFP の比を表す値を示す．

**表 5.1　実験結果の凡例**

| | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | $N_1$ | $N_2$ |
| | FT | $N_3$ | $N_4$ |

適合率は，直感的には 1 つの不具合を見つけるのにどのぐらい無駄なモジュールを調べる必要があるか，すなわちテストのためのコストを表している．

最後に，F 値 ($F_1$) を定義する．これは，再現率と適合率の調和平均で表される指標である．

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

一般的に，F 値の値が高いと予測精度が高いと考えることができる．そのため，本研究では単に予測精度が高いといった場合には，F 値を利用している．

## 5.2 実験の結果

### 5.2.1 Eclipse EMF Compare

表 5.2 に EMF プロジェクトに対する実験 e1 と e2 の結果を示す．なお，e1 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.842, 適合率 = 0.723, 再現率 = 0.967, F 値 = 0.831.

また，e2 については次のように計算される．

正答率 = 0.868, 適合率 = 0.498, 再現率 = 0.997, F 値 = 0.665.

この結果からは実験 e1 ですでにかなり高い評価指標を示しているものの，実験 e2 においては再現率が極めて高くなっていることが確認できる．

表 5.3 に EMF プロジェクトに対する実験 e3 と e4 の結果を示す．e3 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.729, 適合率 = 0.601, 再現率 = 0.966, F 値 = 0.741.

また，e4 については次のように計算される．

正答率 = 0.730, 適合率 = 0.341, 再現率 = 0.944, F 値 = 0.501.

この結果からは，粗粒度での学習と細粒度での予測がやや予測精度を落としていると読み取れる．ただ，どちらにしても再現率が非常に高いのは特徴である．

### 5.2.2 Eclipse Xpand

表 5.4 に Xpand プロジェクトに対する実験 e1 と e2 の結果を示す．e1 についての正答率，再現率，適合率，F 値は次のように計算される．

**表 5.2　実験 e1 と e2 (Eclipse EMF compare)**

| 実験 e1 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 1174 | 376 |
| | FT | 34 | 1006 |

| 実験 e2 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 6247 | 1113 |
| | FT | 3 | 1107 |

**表 5.3　実験 e3 と e4 (Eclipse EMF compare)**

| 実験 e3 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 884 | 667 |
| | FT | 35 | 1007 |

| 実験 e4 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 9140 | 4035 |
| | FT | 125 | 2086 |

正答率 = 0.823, 適合率 = 0.432, 再現率 = 1.000, F 値 = 0.604.

また，e2 については次のように計算される．

正答率 = 0.900, 適合率 = 0.293, 再現率 = 1.000, F 値 = 0.453.

特筆すべきはどちらも再現率が 1 であることである．つまり，予測で不具合なしとしたもので実際には不具合があったものは無かったということであり，これは非常に特徴的である．

表 5.5 に Xpand プロジェクトに対する実験 e3 と e4 の結果を示す．e3 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.762, 適合率 = 0.354, 再現率 = 0.897, F 値 = 0.508.

また，e4 については次のように計算される．

正答率 = 0.789, 適合率 = 0.146, 再現率 = 0.816, F 値 = 0.247.

### 5.2.3 Eclipse ECF

表 5.6 に ECF プロジェクトに対する実験 e1 と e2 の結果を示す．e1 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.670, 適合率 = 0.474, 再現率 = 0.949, F 値 = 0.633.

また，e2 については次のように計算される．

正答率 = 0.768, 適合率 = 0.266, 再現率 = 0.811, F 値 = 0.401.

ECF プロジェクトでは，正答率以外の指標で e2 が良くない結果となった．

表 5.7 に実験 e3 と e4 の結果を示す．e3 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.513, 適合率 = 0.379, 再現率 = 0.978, F 値 = 0.546.

また，e4 については次のように計算される．

正答率 = 0.499, 適合率 = 0.185, 再現率 = 0.972, F 値 = 0.311.

### 5.2.4 Eclipse Webtools incubator

表 5.8 に Webtools incubator プロジェクトに対する実験 e1 と e2 の結果を示す．e1 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.794, 適合率 = 0.533, 再現率 = 0.995, F 値 = 0.695.

また，e2 については次のように計算される．

**表 5.4　実験 e1 と e2 (Eclipse Xpand)**

| 実験 e1 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 1884 | 486 |
| | FT | 0 | 370 |

| 実験 e2 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 8224 | 966 |
| | FT | 0 | 400 |

**表 5.5　実験 e3 と e4 (Eclipse Xpand)**

| 実験 e3 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 1760 | 618 |
| | FT | 39 | 339 |

| 実験 e4 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 14181 | 3810 |
| | FT | 147 | 652 |

**表 5.6　実験 e1 と e2 (Eclipse ECF)**

| 実験 e1 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 3097 | 2523 |
| | FT | 121 | 2279 |

| 実験 e2 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 14720 | 4550 |
| | FT | 384 | 1656 |

**表 5.7　実験 e3 と e4 (Eclipse ECF)**

| 実験 e3 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 1773 | 3854 |
| | FT | 53 | 2356 |

| 実験 e4 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 15060 | 19372 |
| | FT | 126 | 4413 |

正答率 = 0.845, 適合率 = 0.343, 再現率 = 1.000, F 値 = 0.510.

Webtools プロジェクトでは，適合率以外の指標において，e2 の方が良いという結果が得られている．

表 5.7 に実験 e3 と e4 の結果を示す．e3 についての正答率，再現率，適合率，F 値は次のように計算される．

正答率 = 0.635, 適合率 = 0.390, 再現率 = 0.972, F 値 = 0.557.

また，e4 については次のように計算される．

正答率 = 0.542, 適合率 = 0.128, 再現率 = 0.975, F 値 = 0.226.

**表 5.8　実験 e1 と e2 (Eclipse Webtools incubator)**

| 実験 e1 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 1841 | 669 |
| | FT | 4 | 766 |

| 実験 e2 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 10615 | 2145 |
| | FT | 0 | 1120 |

**表 5.9　実験 e3 と e4 (Eclipse Webtools incubator)**

| 実験 e3 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 1335 | 1180 |
| | FT | 22 | 256 |

| 実験 e4 | | 予測 | |
|---|---|---|---|
| | | NFP | FP |
| 実測 | NFT | 14338 | 13754 |
| | FT | 52 | 2016 |

# 6. 考察

本章では，実験結果についての考察を行い，先に定めた研究設問への答えを導く．

まず，プロジェクトを横断して俯瞰したときの傾向について考える．今回実験した 4 つのプロジェクトでは，予測の傾向は比較的一致点が多い．すなわち，多くの場合極めて高い再現率を持ち，その割には低い適合率を持つ，というものである．これは，粗粒度，細粒度を問わず見られる傾向でもある．これは，今回対象とした全てのプロジェクトが Eclipse の関連プロジェクトであるため，予測の傾向が似たものと推察できる．

一方，細粒度モジュールを利用した実験での結果について考える．各プロジェクトの e1 と e2 を比べた時，多くのプロジェクトでは e2 での評価指標は適合率，再現率共にやや下がっている．粒度を小さくしたことにより，予測するための情報が減るためであり，これ自体にさほど違和感はない．だが，下がったとは言え，例えば再現率は 0.8 以上の値を示している場合が多く，これは不具合予測を行う上では非常に有益であることを示唆する．一般的に，不具合予測においてはバグを見逃さないという視点からは再現率を重視すべきと考えられるからである．

始めに挙げた 3 つの研究設問は次の通りで会った．

RQ1 細粒度モジュールを学習させて細粒度モジュールを予測させることで，不具合の予測精度は向上するか．また，その精度は実用的か．

RQ2 細粒度モジュールを学習させて粗粒度モジュールを予測させることで，不具合の予測精度は向上するか．また，その精度は実用的か．

RQ3 粗粒度モジュールを学習させて細粒度モジュールを予測させることで，不具合の予測精度は向上するか．また，その精度は実用的か．

本実験の結果から導かれるこの 3 つの問いに対する答えは全て同じものである．すなわち,「予測精度は向上するとは言えないものの，そこで得られる予測精度は実用的なものである.」とするものである．

# 7. 結言

本研究では，細粒度モジュールを用いた Fault-prone フィルタリングの適用実験を通じて，細粒度モジュールの利用が不具合予測手法において有効かどうかを調べた．適用実験の結果，細粒度モジュールを利用した予測においても，それほど大きな精度の低下を招かずに適用可能であることが示された．特に，再現率を高いままで適用可能であることは，ソフトウェアの信頼性向上に寄与できる可能性を示している．

今後の課題としては，実際にどのようなモジュールの断片が不具合として判定されるのか，といった情報を開発者に提示できるような手法へと拡張することが考えられる．

# 謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学部門水野 修准教授に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，本学情報工学専攻平田幸直先輩をはじめとする，ソフトウェア工学研究室の皆さん，学生生活を通じて筆者の支えとなった家族や友人に深く感謝致します．

# 参考文献


[1] V.R. Basili, L.C. Briand, and W.L. Melo, “A validation of object oriented metrics as quality indicators,” IEEE Trans. on Software Engineering, vol.22, no.10, pp.751–761, 1996.

[2] J.C. Munson and T.M. Khoshgoftaar, “The detection of fault-prone programs,” IEEE Trans. on Software Engineering, vol.18, no.5, pp.423–433, 1992.

[3] N. Ohlsson and H. Alberg, “Predicting fault-prone software modules in telephone switches,” IEEE Trans. on Software Engineering, vol.22, no.12, pp.886–894, 1996.

[4] S. Takabayashi, A. Monden, S. Sato, K. Matsumoto, K. Inoue, and K. Torii, “The detection of fault-prone program using a neural network,” Proc. of International Symposium on Future Software Technology, pp.81–86, Oct. 1999. Nanjing.

[5] N.E. Fenton and M. Neil, “A critique of software defect prediction models,” IEEE Trans. on Software Engineering, vol.25, no.5, pp.675–689, 1999.

[6] A.R. Gray and S.G. McDonell, “Software metrics data analysis - exploring the relative performance of some commonly used modeling techniques,” Empirical Software Engineering, vol.4, pp.297–316, 1999.

[7] T.M. Khoshgoftaar and E.B. Allen, “Modeling software quality with classification trees,” Recent Advances in Reliability and Quality Engineering, pp.247–270, 1999.

[8] T.M. Khoshgoftaar, K. Gao, and R.M. Szabo, “An application of zero-inflated poisson regression for software fault prediction,” Proc. of 12th International Symposium on Software Reliability Engineering, pp.66–73, 1999.

[9] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Trans. on Software Engineering, vol.33, no.1, pp.2–13, Jan. 2007.

[10] M. Stoerzer, B.G. Ryder, X. Ren, and F. Tip, “Finding failure-inducing changes in java programs using change classification,” Proc. of 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp.57–68, ACM Press,

New York, NY, USA, 2006.

[11] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, "Spam filter based approach for finding fault-prone software modules," Proc. of 4th International workshop on Mining software repositories, pp. 4, 2007.

[12] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, "Fault-prone filtering: Detection of fault-prone modules using spam filtering technique," Proc. of 1st International Symposium on Empirical Software Engineering and Measurement, pp.374–383, 2007. Madrid, Spain.

[13] O. Mizuno and T. Kikuno, "Training on errors experiment to detect fault-prone software modules by spam filter," Proc. of 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp.405–414, 2007.

[14] H. Hata, O. Mizuno, and T. Kikuno, "Historage: Fine-grained version control system for java," Proc. of 12th International Workshop on Principles on Software Evolution and 7th ERCIM Workshop on Software Evolution (IWPSE-EVOL2011), pp.96–100, Sept. 2011. Szeged, Hungary.

[15] W.S. Yerazunis "Crm114 – the controllable regex mutilator,", http://crm114.sourceforge.net/.

[16] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes? (on Fridays.)," Proc. of 2nd International workshop on Mining software repositories, pp.24–28, 2005.