様式 C-19

# 科学研究費補助金研究成果報告書

平成２３年　5　月　20　日現在

機関番号：１４３０３
研究種目：若手研究(B)
研究期間：2008　～　2010
課題番号：20700025
研究課題名（和文）　スパムフィルタを応用したソフトウェア不具合の検出手法の開発

研究課題名（英文）　Development of fault-prone module detection technique using spam filter

研究代表者
水野　修（MIZUNO OSAMU）
京都工芸繊維大学・工芸科学研究科・准教授
研究者番号：60314407

研究成果の概要（和文）：
本研究ではスパムフィルタを利用したソフトウェアのバグ検出手法の理論を提案した．提案法では，ソースコード中に出現する単語の頻度に着目し，バグ出現の履歴と単語の頻度を関連づけることで新規コード中に不具合が含まれる確率を計算した．提案法の有効性を実証するためにオープンソースソフトウェアを用いた実験を行った．実験の結果，提案手法は既存の検出手法と同等か優れた能力を持つことを確認できた．
研究成果の概要（英文）：
In this research, we have proposed an approach to detect software bugs in source code. Our approach uses a text filtering technique to detect bugs. In order to show the usefulness of our approach, we conducted experiments.
交付決定額

（金額単位：円）

| | 直接経費 | 間接経費 | 合　計 |
|---|---|---|---|
| ２００８年度 | 1,300,000 | 390,000 | 1,690,000 |
| ２００９年度 | 1,100,000 | 330,000 | 1,430,000 |
| ２０１０年度 | 800,000 | 240,000 | 1,040,000 |
| 年度 | | | |
| 年度 | | | |
| 総　計 | 3,200,000 | 960,000 | 4,160,000 |

研究分野：ソフトウエア工学
科研費の分科・細目：情報学　・　ソフトウエア
キーワード：ソフトウエア工学、不具合検出、スパムフィルタ

## １．研究開始当初の背景

高品質なソースコードの作成はプロダクトの品質向上だけでなくコストの削減にもつながる．コードを作成した時点で Fault-prone モジュールを特定できれば，早期にバグを除去できるだけでなく，レビュー，デバッグに費やす工数の削減も可能となる．そのため，これまでにも Fault-prone モジュールを予測すべく，多くの研究が行われてきた[1]．従来の手法では，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を測定するためには，メトリクスの測定環境が必要となる．

そこで，我々はソースコードのみを入力として与え，メトリクスなどの測定無しに Fault-prone モジュールの予測が可能となる手法を提案する．この手法では，迷惑メール検出に利用されるスパムフィルタで利用される技術をソフトウェアのソースコードに対して適用し，純粋にコードのテキスト情報のみから FP モジュールを予測する．この手法を「Fault-prone フィルタリング」と呼ぶ．

Postini 社の調査によると，2006 年 11 月の時点で世界中を流れる電子メールの 94%はスパムメールであるとされている．そのた

め，スパムメールをブロックする技術の開発が進められてきている．Graham はベイズ識別器によりスパムメールの分別が可能であることを示した[2].

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．

## ２．研究の目的

我々は，スパムフィルタ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．すなわち，ソースコードを入力することでその中に不具合が存在するか否かを何かしらのフィルタリングを行うことで判定できるのではないかと考えた．そこで，本研究ではこの考えに基づいた Fault-prone モジュールフィルタの開発を行う．

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．

本研究での期間内に目指したことは，次の通りである．

（１）提案する Fault-prone フィルタリング手法の理論的な妥当性，および，実証的な妥当性を確認する．理論的な面からは本手法がうまく適用できる条件をできる限り定性的に明らかにすることが必要となる．例えば，本手法で検出できる不具合の種類を明確にすること，自然言語とソースコードでのテキストフィルタリング手法の有効性調査などが，これに相当する．実証的な面からは，本手法が有効である開発環境などを明らかにする必要がある．現在はオープンソースソフトウェアへの適用を視野に入れているが，産業的なソフトウェアの開発への適用によって，こうした実証的な部分を明らかにできると考えている．

（２） Fault-prone フィルタリング手法を実開発環境でも使えるようにツールなどを整備する．本手法をソフトウェアの開発で活用するためには，ツール化が必要となる．現在申請者はプロトタイプを試作し，実験環境での適用を行っているが，ツールの開発により実環境でのデータ収集が可能になると考えている．本研究期間の後半にこうしたツールの作成および実証実験を行う予定である．

## ３．研究の方法

この節では Fault-prone フィルタリングがどのように不具合のあるソフトウェアモジュールを検出するのかを，単純な例を用いて説明する．

```
1: public int fact(int x) {
2:     return(x==1?1:x*fact(++x));
3: }
```

(a) 不具合を含む (FP) モジュール $m_{FP}$

```
1: public int fact(int x) {
2:     return(x==1?1:x*fact(--x));
3: }
```

(b) 不具合を含まない (NFP) モジュール $m_{NFP}$

図 1: 不具合ありと無しのモジュール例

図 1(a) と (b) はそれぞれ，不具合を含む (FP) モジュールと含まない (NFP) モジュールの例である．以降ではそれぞれを m_FP，mNFP と表記する．fact() は与えられた自然数 x に対してその階乗を返すことを意図しているが，図 1(a) の実装では--x とすべきところを++x と誤記しているため，正常に動作しない．図 1(b) はその不具合を取り除いた状態である．この 2 つのモジュールのみが辞書に学習された時点で，図 2 に示すモジュール m_new が新たに作成されたとし，このモジュールが不具合を含む確率を計算することを考える．なお，モジュール m_new は与えられた正整数 y に対してその総和を求めるつもりであるが，m_FP と同様に本来--とすべきところを++としている．

```
1: public int sigma(int y) {
2:     return(y==1?1:y+sigma(++y));
3: }
```

図 2: 新しいモジュール

まず，m_FP と m_NFP をそれぞれ FP，NFP として学習する．この時，CRM114 によってそれぞれのモジュールについて図 3(a)，(b) に示すようなトークンの集合 T^FP, T^NFP が生成される．m_FP から生成されたトークンの集合 T^FP は Fault-prone モジュールの特徴として FP 辞書に格納される．同様に，m_NFP のトークンの集合 T^NFP は, NFP 辞書に格納される．

新たなモジュールが与えられると，その時点で辞書に学習されている全てのトークンとのマッチングがとられ，確率の計算が行わ

れる． 図 3 は，m_FP，m_NFP，m_newについて生成されるトークンの集合，T^FP，T^NFP，T^new を列挙したものである．図 3(a)と(b)が現時点でそれぞれ FP と NFP の辞書に格納されている全てのトークンであり，図 3(c)は新たに生成されたトークンである．下線を引いた部分はモジュール m_new と同一のトークンであることを表す．この図から，m_FP と m_new の間で同一なトークンの数は 14 であり，m_NFP と m_new の間で同一なトークンの数は 13 であることが分かる．

| public | int |
|---|---|
| public | fact |
| public | x |
| int | fact |
| int | int |
| int | x |
| int | return |
| fact | int |
| fact | x |
| fact | return |
| int | == |
| x | return |
| x | x |
| x | == |
| x | 1 |
| return | x |
| return | == |
| return | 1 |
| return | ? |
| x | ? |
| x | : |
| == | 1 |
| == | ? |
| == | : |
| == | x |
| 1 | ? |
| 1 | : |
| 1 | x |
| 1 | * |
| ? | : |
| ? | x |
| ? | * |
| ? | fact |
| : | x |
| : | * |
| : | fact |
| : | ++ |
| x | * |
| x | fact |
| x | ++ |
| * | fact |
| * | ++ |
| * | x |
| fact | ++ |
| ++ | x |

(a) $m_{FP}$ から生成されるトークン $T^{FP}$

| public | int |
|---|---|
| public | fact |
| public | x |
| int | fact |
| int | int |
| int | x |
| int | return |
| fact | int |
| fact | x |
| fact | return |
| int | == |
| x | return |
| x | x |
| x | == |
| x | 1 |
| return | x |
| return | == |
| return | 1 |
| return | ? |
| x | ? |
| x | : |
| == | 1 |
| == | ? |
| == | : |
| == | x |
| 1 | ? |
| 1 | : |
| 1 | x |
| 1 | * |
| ? | : |
| ? | x |
| ? | * |
| ? | fact |
| : | x |
| : | * |
| : | fact |
| : | -- |
| x | * |
| x | fact |
| x | -- |
| * | fact |
| * | -- |
| * | x |
| fact | -- |
| -- | x |

(b) $m_{NFP}$ から生成されるトークン $T^{NFP}$

| public | int |
|---|---|
| public | sigma |
| public | y |
| int | sigma |
| int | int |
| int | y |
| int | return |
| sigma | int |
| sigma | y |
| sigma | return |
| int | == |
| y | return |
| y | y |
| y | == |
| y | 1 |
| return | y |
| return | == |
| return | 1 |
| return | ? |
| y | ? |
| y | : |
| == | 1 |
| == | ? |
| == | : |
| == | y |
| 1 | ? |
| 1 | : |
| 1 | y |
| 1 | + |
| ? | : |
| ? | y |
| ? | + |
| ? | sigma |
| : | y |
| : | + |
| : | sigma |
| : | ++ |
| y | + |
| y | sigma |
| y | ++ |
| + | sigma |
| + | ++ |
| + | y |
| sigma | ++ |
| ++ | y |

(c) $m_{new}$ から生成されるトークン $T^{new}$

図 3:生成されるトークン

この情報から新規モジュール m_new が不具合を含む確率 P(T^FP|T^new)を算出する．ベイズの定理によって，確率 P(T^FP|T^new)は次のように求められる．

$$\frac{P(T^{new}|T^{FP})P(T^{FP})}{P(T^{new}|T^{FP})P(T^{FP}) + P(T^{new}|T^{NFP})P(T^{NFP})}$$

まず，学習されているのは T^FP と T^NFP だけなので，それぞれのトークンの存在する事前確率は P(T^FP) = P(T^NFP) = 1/2 となる．次に，m_FP のトークン T^FP 内に m_new のトークン T^new が存在する確率 P(T^new|T^FP)=14/45 である．また，m_NFP のトークン T^NFP 内に T^new が存在する確率 P(T^new|T^NFP)=13/45 である．よって，確率は次式で求められる：

$$P(T^{FP}|T^{new}) = \frac{\frac{14}{45} \times \frac{1}{2}}{\frac{14}{45} \times \frac{1}{2} + \frac{13}{45} \times \frac{1}{2}} = 0.519$$

この結果，新規モジュール m_new が不具合を含む確率は 0.519 となる．本研究では確率の閾値を 0.50 と定め，0.50 以上であれば FP モジュール，0.50 未満であれば NFP モジュールと判定する．よって，この例では FP モジュールと判定されることになる．

Fault-prone フィルタリングを適用するにあたって，我々は「誤判定時のみ学習(Training Only Errors: TOE)」という方式を採用する．これは，スパムフィルタにおいても利用される方式であり，メールを到着順に分類し，その分類が正しかったかどうかを判断できる時点で学習を行う手法である．Fault-prone フィルタリングの適用実験では，次のような手順になる．

(1)．当該プロジェクトのモジュール群を古いものから順番にソートする．
(2)．古いモジュールから順に 1 つずつ取り出し，そのモジュールが FP か NFP であるかを Fault-prone フィルタにより判定する．
(3)．モジュールに不具合が含まれていることが判明した時点で，予測が正しければ何もせずに (2) へ戻る．
(4)．予測が正しくなければ，正しい結果を学習させる．例えば，NFP と予測したのに実際の結果が FP であった場合，FP 辞書に該当するモジュールの内容を学習させる．その後，(2) へ戻る．

## ４．研究成果

ここでは，オープンソースソフトウェアである Eclipse Modeling Framework(EMF)，Eclipse Project(EP)の 2 種類の開発データについて TOE を用いた実験を行った結果を示す．

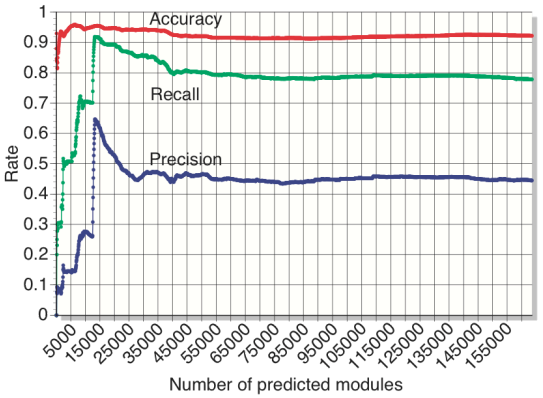
TOE においては，本来事前にソースコードを準備する必要は無い．しかし，本実験に当たっては事前にソースコードリポジトリからモジュールを取得しておき，そのモジュールにバグが含まれていたか否かの真値をあらかじめ把握しておく必要がある．そこで，本研究では Sliwerski らによって提案されているアルゴリズムを用いてオープンソースのソフトウェアリポジトリから FP モジュールと NFP モジュールを抽出した．

実験対象である Eclipse とその関連プロジェクトにおける FP モジュール抽出の結果を表 1 に示す．なお，モジュールの抽出に要す

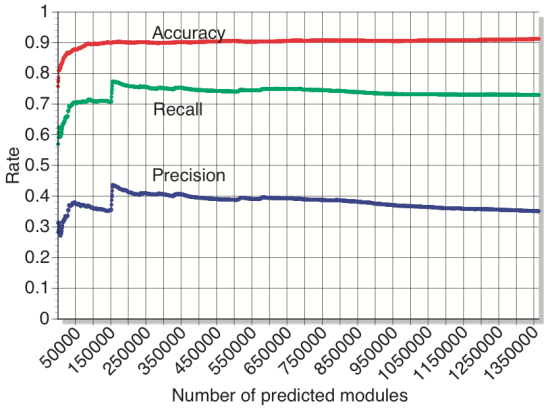
る時間はプロジェクトの規模によって異なるが，Eclipse Project(EP)については約 17 時間であった．

表 1:モジュール抽出の結果

| 名称 | EMF | EP |
|---|---|---|
| 開発言語 | Java | |
| リポジトリのサイズ | 962MB | 15.6GB |
| 収集した不具合の状態 | Resolved, Verified, Closed | |
| 不具合の解決状況 | Fixed | |
| 不具合の重大度 | blocker, critical, major, normal | |
| 上記条件での不具合の数 | 4,042 | 44,600 |
| CVS のログから発見した不具合数 | 2,832 (70.1%) | 24,344 (54.6%) |
| FP モジュールの数 | 10,636 | 73,902 |
| NFP モジュールの数 | 152,821 | 1,289,463 |




(a) EMF の結果




(b) EP の結果

図 4: TOE 実験の結果

図 4(a)，(b)に TOE 実験の経過を示すグラフを，表 2(a)，(b)に最終的な予測結果を示す．図 4(a)，(b)のグラフについて説明する．このグラフの縦軸は精度，再現率，適合率の値を表す．また，横軸は作成時刻の時系列順にソートした全モジュールの通し番号を表している．TOE においては，古いものから順に分類・学習を行うため，一度全てのモジュールを作成時刻でソートしておき，古いものから若い番号を付けてある．例えば横軸の 10000 はこの開発において 10000 番目に作成されたモジュールであることを表し，その時点までのモジュールを予測した結果としての 3 つの指標の値をプロットしたものがグラフになっている．

表 2: EMF，および，EP の結果

| | | 予測 NFP | 予測 FP |
|---|---|---|---|
| 実測 | NFP | 142,482 | 10,339 |
| | FP | 2,358 | 8,278 |
| 精度 (accuracy) | | 0.922 | |
| 再現率 (recall) | | 0.778 | |
| 適合率 (precision) | | 0.445 | |

| | | 予測 NFP | 予測 FP |
|---|---|---|---|
| 実測 | NFP | 1,189,740 | 99,723 |
| | FP | 19,985 | 53,917 |
| 精度 (accuracy) | | 0.912 | |
| 再現率 (recall) | | 0.730 | |
| 適合率 (precision) | | 0.351 | |

表 2 は，各プロジェクトについて全てのモジュールを予測し終わった時点での予測と実測をクロス集計したものである．表 2 からは，Fault-prone フィルタリングによって精度と再現率に関してはそれぞれ高い値が得られたことを確認できる．再現率に関しては共に 0.75 前後の値を示しており，高い不具合予測能力が確認できる．また，適合率に関しても 0.40 前後と比較的高い値を示していることから，不具合の予測に必要なコストもそれほど高くないものと考えられる．このことから，これら 2 つのプロジェクトにおける実験では Fault-prone フィルタリングがうまく適用できていることが確認できた．

なお，TOE 実験に要する時間は単純にモジュールの数に比例する．最大の規模をもつ EP においては約 20 時間を要したが，1 つのモジュールの予測・学習に要する時間は 0.5 秒前後である．

## ５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 4 件）


[1] 水野 修，畑 秀明，″スパムフィルタを用いた Fault-prone モジュール検出法の予測精度に関する従来法との比較評価，″ 電子情報通信学会論文誌 D，J94-D(1)，pp. 409-412，2011 年 1 月．（査読有）

[2] Osamu Mizuno and Hideaki Hata, ″A Hybrid Fault-Proneness Detection Approach Using Text Filtering and Static Code Analysis,″ International Journal of Advancements in Computing Technology, 2(5), pp. 1-12, December 2010.（査読有）

[3] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno, ″Fault -Prone Module Detection Using Large-Scale Text Features Based on Spam Filtering,″ Empirical Software Engineering, 15(2), pp. 147 -165, April 2010.（査読有）

[4] Osamu Mizuno and Hideaki Hata,

″Prediction of Fault-Prone Modules Using a Text Filtering Based Metric,″ International Journal of Software Engineering and Its Application, 4(1), pp. 43-52, January 2010. (査読有)

〔学会発表〕(計５件)
[1] Osamu Mizuno and Yukinao Hirata, ″Fault-Prone Module Prediction Using Contents of Comment Lines,″ In Proc. of International Workshop on Empirical Software Engineering in Practice 2010 (IWESEP2010), pp. 39-44, December7, 2010. (NAIST, Nara, Japan)
[2] Osamu Mizuno and Hideaki Hata, ″An Empirical Comparison of Fault -Prone Module Detection Approaches: Complexity Metrics and Text Feature Metrics,″ In Proc. of 34th Annual IEEE Computer Software and Applications Conference (COMPSAC2010), pp. 248-249, July 20, 2010. (Seoul, Korea)
[3] Osamu Mizuno and Hideaki Hata, ″An Integrated Approach to Detect Fault-Prone Modules Using Complexity and Text Feature Metrics,″ In Proc. of 2010 International Conference on Advanced Science and Technology (AST2010), LNCS 6059, pp. 457-468, June 24, 2010. (Miyazaki, Japan)
[4] Osamu Mizuno and Hideaki Hata, ″Yet Another Metric for Predicting Fault-Prone Modules,″ In Proc. of 2009 International Conference on Advanced Software Engineering & Its Applications (ASEA2009), CCIS 59, pp. 296-304, December 12, 2009. (Cheju, Korea)
[5] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno, ″Comparative Study of Fault-Proneness Filtering with PMD,″ In Proc. of 19th International Symposium on Software Reliability Engineering (ISSRE2008), pp. 317-318, November 12, 2008. (Seattle/Redmond, WA, USA)

６．研究組織
(1)研究代表者
水野　修 (MIZUNO　OSAMU)
京都工芸繊維大学・工芸科学研究科・
准教授
研究者番号：60314407

(2)研究分担者
(　　)

研究者番号：

(3)連携研究者
(　　)

研究者番号：