卒 業 研 究 報 告 書

題　目　　StackOverFlow における
質問と回答に含まれるソースコードの分析

指導教員　　水野　修　准教授

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　12122030

氏　　名　　田中　健太郎

平成28年2月15日提出

StackOverFlow における質問と回答に含まれるソースコードの分析

平成 28 年 2 月 15 日　　12122030　田中　健太郎

概　要

プログラミング技術に関するナレッジコミュニティとして最も有名なものの一つに Stack Overflow がある．Stack Overflow の主な機能として質問と回答，及び複数の回答の中から最も良い解答を，質問者が「承認」する機能が存在する．

Stack Overflow の質問文中に記述されているソースコードは，文法エラーや機能的な不具合（バグ）を含んでいると考えられる．また，承認された回答に含まれるソースコードは，それらの問題を修正されていると考えられる．

本研究の目的は，Stack Overflow に投稿された質問と回答に含まれるソースコードそれぞれの特徴を用いることで，一般のソフトウェアに含まれる不具合をソースコードから予測することにある．

本研究では質問と回答に含まれるソースコードを Stack Overflow のダンプデータから取得し，テキスト分類フィルタの学習データ・評価データに用いて交差検証を行うことで，質問に含まれるソースコードと承認された回答に含まれるソースコードの間に異なる特徴が存在し，識別可能であるのかを検証する．また，これらのソースコードをテキスト分類フィルタの学習データに用いた上で，オープンソースソフトウェアの不具合を予測することが可能であるかを検証する．テキスト分類フィルタは過去に発表されたモジュールを用いる．

本研究により，質問文と「承認」された解答文それぞれに存在するソースコードの識別をすることができた．

目　次

1. 緒言

プログラミング技術に関するナレッジコミュニティとして最も有名なものの一つにStack Overflowがある．Stack Overflowの主な機能として質問と回答，及び複数の回答の中から最も良い解答を，質問者が「承認」する機能が存在する．

Stack Overflowには，質問者が記述，回答者が記述・修正したソースコードが多量に含まれている．質問内容は多岐に渡り，以下の様な質問が多数存在する．

- 文法エラーが含まれているが直し方が分からない.
- 意図した動作をしない．
- ある機能を実装したいがどのようにコーディングすれば良いかが分からない．

このような質問文中に記述されているソースコードは，文法エラーや機能的な不具合（バグ）を含んでいると考えられる．また，承認された回答に含まれるソースコードは，それらの問題を修正されていると考えられる．

Stack Overflowの質問や回答に含まれるソースコードに関する研究について言及すると，BuseとWeimerによるコードの可読性を計算するツール[1]を用いて，未解決の質問に含まれるソースコードの可読性を調べたRahmanとRoyによる研究[2]や，DiamantopoulosとSymeonidisによる質問に含まれるソースコードを用いて類似した質問を検索する研究[3]などが挙げられる．しかし，ソースコードそのものの特徴に着目して別のソフトウェアにおける不具合予測に応用する事例は未だ存在しない．

本研究の目的は，Stack Overflowに投稿された質問と回答に含まれるソースコードそれぞれの特徴を用いることで，一般のソフトウェアに含まれる不具合をソースコードから予測することにある．

本研究では質問と回答に含まれるソースコードをStack Overflowのダンプデータから取得し，テキスト分類フィルタの学習データ・評価データに用いて交差検証を行うことで，質問に含まれるソースコードと承認された回答に含まれるソースコードの間に異なる特徴が存在し，識別可能であるのかを検証する．また，これらのソースコードをテキスト分類フィルタの学習データに用いた上で，オープンソースソフトウェアの不具合を予測することが可能であるかを検証する．

2. 準備

2.1 Stack Exchange Data Dump

Stack Overflow を運営している Stack Overflow 社（旧 Stack Exchange 社）は．コミュニティサイト Stack Overflow に寄せられた投稿のダンプデータを公開している(注 1)．XML(Extensible Markup Language) 形式で配布されているが，本研究ではリレーショナルデータベースへと形式を変換して使用している．

本研究では 2015 年 2 月にダンプされたデータを用いている．

2.2 CRM114

2.2.1 概要

CRM114 は Yerazunis らによって開発されたテキスト分類フィルタである [4]．スパムフィルタ(注 2) としての使用を目的として作成されたが，その汎用性からシステムログやネットワークトラフィックの監視，また Fault-Prone モジュール予測等にも使用することが可能となっている．

CRM114 は基本的にはベイズ認識を利用したテキスト分類フィルタである．学習・分類の単位として利用する最小単位をトークンと呼び，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．それに対し，CRM114 は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 の標準の分類手法である”Othogonal Sparce Bigrams Marcov-model (OSB)”を使用する．OSB は任意の連続する 5 単語の組み合わせのうち，2 単語の組のみをトークンとする手法である．

2.2.2 閾値

対象となるモジュールに CRM114 を適用すると，0 以上 1 以下の値が得られる．この値はベイズ識別によって得られた，そのモジュールが質問に含まれるソースコードである確率である．閾値は，質問に含まれるソースコードか回答に含まれるソー

(注 1)：https://archive.org/details/stackexchange
(注 2)：迷惑メールとそれ以外のメールを振り分ける機能を持つフィルタのこと．

スコードかの予測を決定するための境界地であり，質問に含まれるソースコードである確率が閾値より大きければ質問に含まれるソースコードと予測され．閾値以下であれば回答に含まれるソースコードと予測されるということである．

CRM114 から出力される値は，学習データの量に偏りがあると，より多くの学習データが存在するクラスである確率が高いと出力する．例えば，質問に含まれるソースコードの量が回答に含まれるソースコードの量よりも多い場合，出力される値は 1 に近くなる．そのため，学習データ毎に，適切な閾値を設定する必要がある．

2.2.3 評価

表 2.1 および表 2.2 は，実験で得られる結果の凡例である．N_1,N_2,N_3,N_4 は横に記す予測と縦に記す実測にそれぞれ該当する例数を表す.

本実験では以下の評価尺度を用いる．

(1) 正確度 (Accuracy)

正確度は，全モジュールの内，予測が正しかった割合を示す.

$$Accuracy = \frac{N_1 + N_4}{N_1 + N_2 + N_3 + N_4} \tag{2.1}$$

正確度は，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．本研究で使用するデータは，質問と回答に含まれるソースコード，Fault-Prone と Non-Fault-Prone のソースコード，いずれの場合も同数のモジュールを評価に用いるため，この点に関しては考慮しなくて良い．ただし，閾値の設定にも大きく影響を受けるため，注意が必要である．

(2) 適合率 (Precision)

適合率は，予測が正であるモジュールの内，実測が正である者の割合を示す.

$$Precision = \frac{N_1}{N_1 + N_3} \tag{2.2}$$

適合率は，質問に含まれるソースコードを 1 つ見つけるのに，回答に含まれるソースコードを質問に含まれるソースコードとどの程度誤判定するかを示す指標である．

表 2.1　結果の例 (1)

		予測	
		質問	回答
実測	質問	N_1	N_2
	回答	N_3	N_4

表 2.2　結果の例 (2)

		予測	
		質問	回答
実測	Fault-Prone	N_1	N_2
	Non-Fault-Prone	N_3	N_4

(3) 再現率 (Recall)

再現率は，実際に正であるもののうち，正であると予測されたものの割合を示す.

$$Recall = \frac{N_1}{N_1 + N_2} \tag{2.3}$$

再現率は，予測によって実際の質問に含まれるソースコード (Fault-Prone モジュール予測の場合は不具合) をどの程度網羅できるかを示す指標である．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重視すべき指標といえる.

(4) F 値

F 値は精度と再現率の調和平均である.

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{2.4}$$

適合率か再現率，どちらか一方のみが良い値を記録しても，良い識別結果とは言えない．適合率と再現率が共に高い時にのみ，F 値も高い値となる.

2.3 difflib

difflib は，シーケンス間の差異を算出するための Python 言語のライブラリである．特にテキストの比較に便利であり，テキスト行単位の差異を diff フォーマットで出力することが可能である.

本研究では，質問と回答に含まれるソースコードの類似度を算出する為に使用する.

2.4 Eclipse BIRT

Eclipse BIRT とは，米 Actuate 社が開発したオープンソースレポート作成ツールである．JAVA 言語で記述されており，ソースコードはバージョン管理システム Git によって管理されている.

本研究ではこのプロジェクトのソースコードから，不具合の含まれるソースコードと修正されたソースコードを取得し，Fault-Prone モジュール予測の評価データに用いる．

2.5 GitPython

GitPython は，Git レポジトリを Python 言語で扱うためのライブラリである．本研究では，Eclipse BIRT の Git レポジトリから，不具合を含むソースコードと修正したソースコードを取得するために用いる．

3. 研究の目的

3.1 研究の目的

本研究の目的は，Stack Overflow に投稿された質問と回答に含まれるソースコードそれぞれの特徴を用いることで，一般のソフトウェアに含まれる不具合をソースコードから予測することにある．

3.2 研究設問

本研究では次に示す研究設問（Research Question）について，検証を行う

RQ1: 質問と承認された回答に含まれるコードの行数に差はあるか．また，プロラミング言語別にはどのような傾向になるか．

RQ2: 質問と承認された回答に含まれるコードは類似しているか．また，プロラミング言語別にはどのような傾向になるか．

RQ3: 質問と承認された回答に含まれるコードをさらに細かく分類できる様な特徴は存在するか．

RQ4: Stack Overflow の質問と承認された回答に含まれるコードを，テキスト分類フィルタを用いて分類することは可能か．

RQ5: Stack Overflow の質問と承認された回答に含まれるコードを学習データに用いて，実際のバグに対して Fault-Prone モジュール予測が可能であるか

4. 実験

本研究では JAVA, C, C++, C#の 4 種類のプログラミング言語を対象に実験を行う.

4.1 実験準備

4.1.1 ダンプデータの変換

Stack Overflow のダンプデータは複数の XML ファイルから構成されているが，本研究では質問文と回答文しか必要としないため，Posts.xml ファイルのみを用いる.

本研究ではこのファイルを，リレーショナルデータベース管理システム MySQL のデータベースへと変換して，データの参照を行った．Posts.xml のデータを反映したテーブル”posts”の構造を表 4.1 に示す.

4.1.2 Stack Overflow に投稿された質問と回答に含まれるソースコードの取得

質問・回答ともに，ソースコードは Body フィールドに含まれている．内容は HTML(HyperText Markup Language) で記述されており，ソースコードは <pre> タグ及び <code> タグで囲われている.

本研究で取得するソースコードには，以下の制限を設けた

(1) 質問に対し，承認された回答が存在する
(2) <pre>・<code> タグで囲われた内容が，質問・承認された回答共に，1 箇所のみ存在する.

(1) は，質問文中の小ソースコードと回答文中の小ソースコードを比較するために設けた．(2) は，<pre>・<code> タグで囲われた内容がソースコードであるとは限らず，複数の内容を扱うことが難しいためである.

4.1.3 トークナイザの作成

フィルタリングを行うにあたって，テキスト情報を単語に分割するトークナイザが必要となる．本研究では，正規表現を用いて，ソースコードから変数名・演算子・キーワードを取得するプログラムをトークナイザとする．ただし，本研究では区切り文字 ”;” や格好　”( ){ } ” は取得しない.

表 4.1　Posts のテーブル構造

フィールド	内容
Id	
PostTypeId	投稿が質問か回答か，あるいはそれ以外か
AcceptedAnswerId	PostTypeId が質問であるとき，その質問に対して承認された回答の Id が記録されている．
ParentID	PostTypeId が回答であり，かつ他の回答の補足や訂正である場合，対象の投稿の Id が記録される．
CreationDate	
DeletionDate	
Score	投稿の貢献度が点数で表される．この点数は，質問者だけでなく閲覧者による評価も含まれる．
ViewCount	
Body	投稿内容の本文
OwnerUserId	
OwnerDisplayName	
LastEditorUserId	
LastEditorDisplayName	
LastEditDate	
LastActivityDate	
Title	
Tags	投稿内容を分類するためのキーワードやラベルが複数記載されている．多くの場合，対象としているプログラミング言語名が含まれている．
AnswerCount	
CommentCount	
FavoriteCount	
ClosedDate	
CommunityOwnedDate	

4.1.4　単語の抽象化モジュールの作成

トークナイザによって分割された単語には，識別子の名前，文字列，変数の値など，各ソースコード特有のものが存在する．本研究では，対象とする言語のキーワード・オペレーターを除いたものを以下の様に置換する抽象化スクリプトを作成した．

(1) 文字列は'STRING' に置換
(2) 文字は'CHAR' に置換
(3) 数値は'NUMBER' に置換
(4) (1) - (3)，及びキーワード・オペレーター以外の単語は'IDENTIFIER' に置換

4.1.5　Git レポジトリに存在するソースコードの取得

Eclipse BIRT に関する Git レポジトリ中にある，不具合を含むソースコードと修正されたソースコードを取得する．

(1)　ソースファイルの取得

以下の手順でソースファイルの取得を行う．

ただしプロジェクトには予め，不具合が混入したコミットに BUG タグ，不具合を修正したコミットに FIX タグが設定されており，BUG タグ及び FIX タグには変更のあったファイル名がメッセージに記載されている．

1. git tag コマンドを用いて，タグ一覧を表示する．
2. git tag で得た FIX タグを git checkout コマンドで指定することにより，不具合を修正したコミットが適用された状態のソースファイルを取得することができる．また FIX タグの付与されたコミットの１つ前の状態をチェックアウトすることで，不具合が含まれた状態のソースファイルを取得することができる．
3. FIX タグのメッセージに記載されているファイルの内，拡張子が java であるファイルのみを取得する．

(2)　差分の取得

以下の手順でソースコードの差分の取得を行う．

1. git tag コマンドは，タグ一覧を表示する．
2. git tag で得た FIX タグを git show コマンドで指定することで以下の内容を取得する．
   - 変更のあったファイル名
   - テキストファイルに関して，変更のあった行とその前後 3 行
3. git show で得たファイル名と変更内容に対し，拡張子が java であるファイルに関する変更内容のみを取得する．

4.2 実験 1 (RQ1)

質問と承認された回答に含まれるソースコードそれぞれの行数 (LOC : Lines Of Code) を求め，プログラミング言語毎に比較する．ただし，空行は除くものとする．

4.3 実験 2 (RQ2)

プログラミング言語毎に，質問と承認された回答に含まれるソースコードそれぞれの類似度を求める．

類似度は，以下の手順で算出する．

1. 質問と承認された回答に含まれるソースコードそれぞれを，トークナイザによって単語ごとに分割する．この時抽象化は行わない．
2. difflib を用いて，単語の並びの差分を取得する．
3. 変更のない単語の数を L_e，追加された単語の数を L_a，削除された単語の数を L_d とし，式 4.1 に従って類似度 S を計算する．類似度 S は 0 以上 1 以下の値となる．

$$S = \frac{L_e}{L_e + L_a + L_d} \tag{4.1}$$

また，JAVA に関して，類似度が 0 である取得したテキストと類似度が 0.05 以上の取得したテキストをそれぞれ 100 件ランダムに取得し，それらがソースコードであるかを手作業で確認する．

4.4 実験 3 (RQ3)

質問に含まれるソースコードの行数を L_q，承認された回答に含まれるソースコードの行数を L_a とし，$L_q > L_a$ となるソースコードの組と $L_q \leqq L_a$ となるソースコードの組，それぞれの類似度の頻度分布を求める．

4.5 実験 4 (RQ4)

CRM114 を用いて，各言語ごとに質問に含まれるソースコードと回答に含まれるソースコードを分類可能であるかを検証する．学習データ・評価データは 10 分割交差検証を用いる．

ここで注意しなければならないことは，Stack Overflow から取得した内容は <pre>・<code> タグで囲われたテキストであり，ソースコードとは限らないことである．質問者が記述したソースコードに対して回答者が追加・修正したものでなければ，テキストの類似度は低くなる．また，質問に含まれるソースコードと回答に含まれるソースコードが同一である，つまり類似度が 1 となるものは，本実験には適さない．

以上を踏まえ，本実験，及び以降の実験では，類似度に関して式 4.2 を満たすテキストの対をデータセットとし，学習・評価データに用いる．

$$0.05 \leqq S < 1 \tag{4.2}$$

4.6 実験 5 (RQ3, 4)

JAVA 言語に関して，$L_q > L_a$ となるソースコードの組と $L_q \leqq L_a$ となるソースコードの組それぞれに対し，実験 4 と同様の方法で分類を試みる．

4.7 実験 6 (RQ5)

質問に含まれるソースコードは不具合を含んでいるものであり，回答に含まれるソースコードは不具合を修正したものであると仮定したとき，Stack Overflow から取得した JAVA 言語のソースコードを学習データに用いた CRM114 によって，Eclipse

BIRT から取得したソースファイルに対して Fault-Prone モジュール予測が可能であるかを検証する.

4.8 実験 7 (RQ5)

Eclipse BIRT から取得するソースコードを，不具合を修正したコミットにおいて変更があった行とその前後 3 行のみとし，実験 6 と同様に CRM114 を用いて Fault-Prone モジュール予測を試みる.

4.9 実験 8 (RQ3, 5)

学習データを $L_q > L_a$ となるソースコードの組と $L_q \leqq L_a$ となるソースコードの組に分け，それぞれ実験 7 と同様に CRM114 を用いて Fault-Prone モジュール予測を試みる.

5. 実験結果

5.1 実験 1

JAVA の結果を図 5.1，C の結果を図 5.2，C++の結果を図 5.3，C#の結果を図 5.4 に示す.

C, C++, C#, JAVA, いずれの言語に関しても，以下の性質が見られた.

- 行数が短いものの方がより多く存在する.
- 質問に含まれるテキストの方が，回答に含まれるテキストよりも行数が長いものが多い.

5.2 実験 2

JAVA の結果を図 5.5，C の結果を図 5.6，C++の結果を図 5.7，C#の結果を図 5.8 に示す.

C, C++, C#, JAVA, いずれの言語に関しても，類似度が 0 のテキストの組が最も多く，類似度 0 を除くと 0.03 付近がピークとなり，それ以上の類似度は下降傾向にある.

JAVA に関して，類似度が 0 である取得したテキストを 100 件ランダムに取得した結果，64 件が JAVA のソースコードであった．一方，類似度が 0.05 以上の取得したテキストを 100 件ランダムに取得した結果，90 件が JAVA のソースコードであった．以上の結果より．類似度が著しく低い組にはソースコードでないテキストが含まれている可能性が高いことが分かる.

5.3 実験 3

JAVA の結果を図 5.9，C の結果を図 5.10，C++の結果を図 5.11，C#の結果を図 5.12 に示す.

いずれの言語に関しても，$L_q \leqq L_a$ となる質問と回答に含まれるソースコードの方が類似度が高いものの割合が多い結果となった.

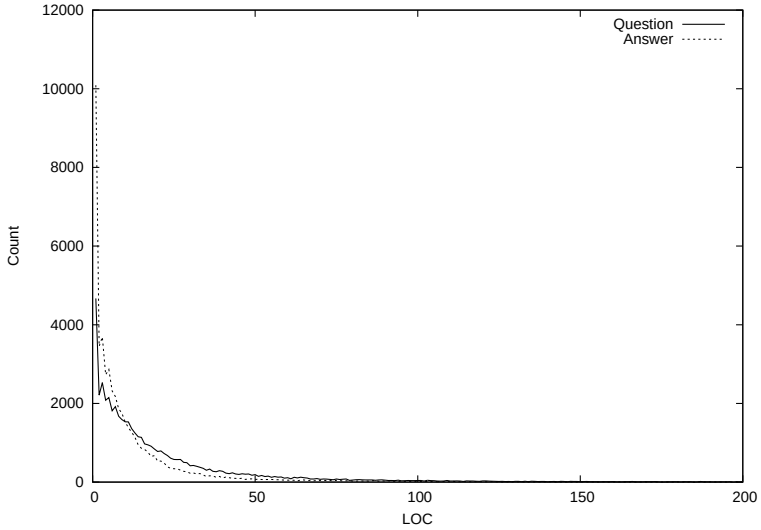

図 5.1　LOC のヒストグラム (JAVA)

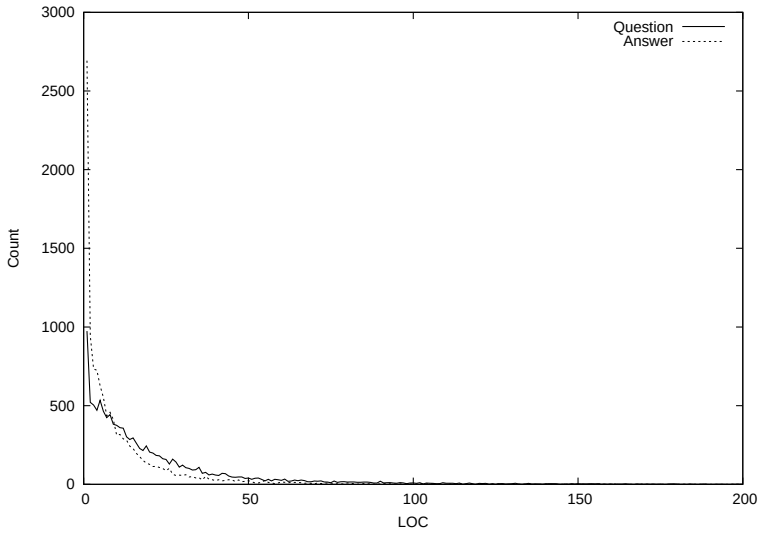

図 5.2　LOC のヒストグラム (C)

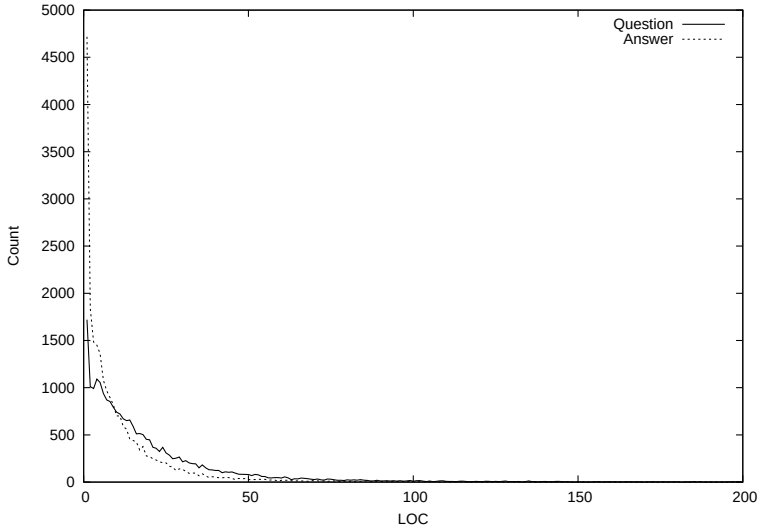

図 5.3　LOC のヒストグラム (C++)

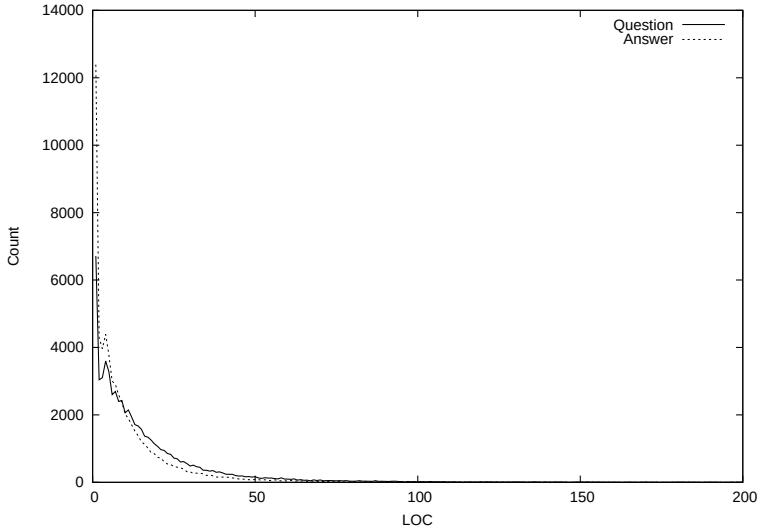

図 5.4　LOC のヒストグラム (C#)

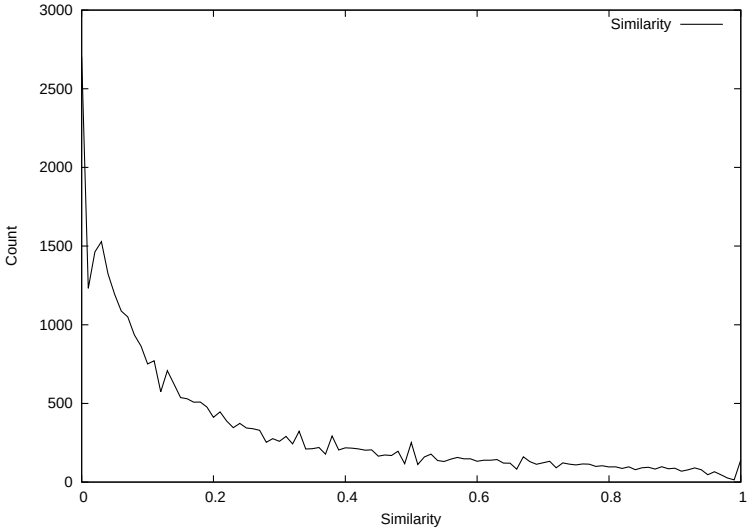

図 5.5　類似度のヒストグラム (JAVA)

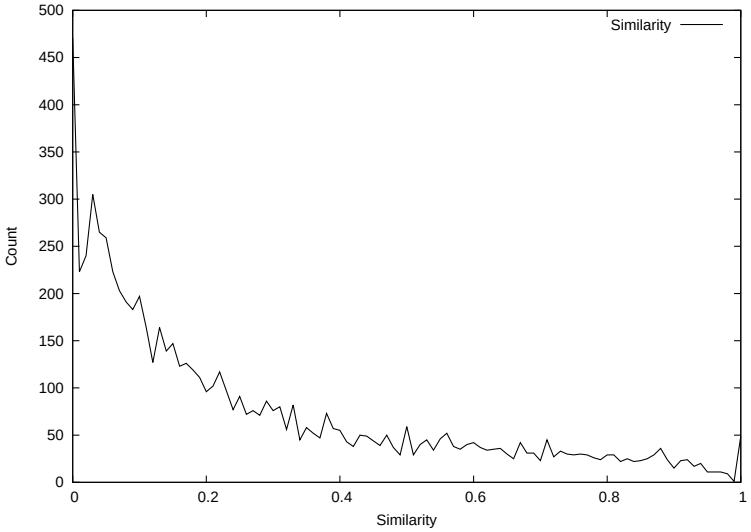

図 5.6　類似度のヒストグラム (C)

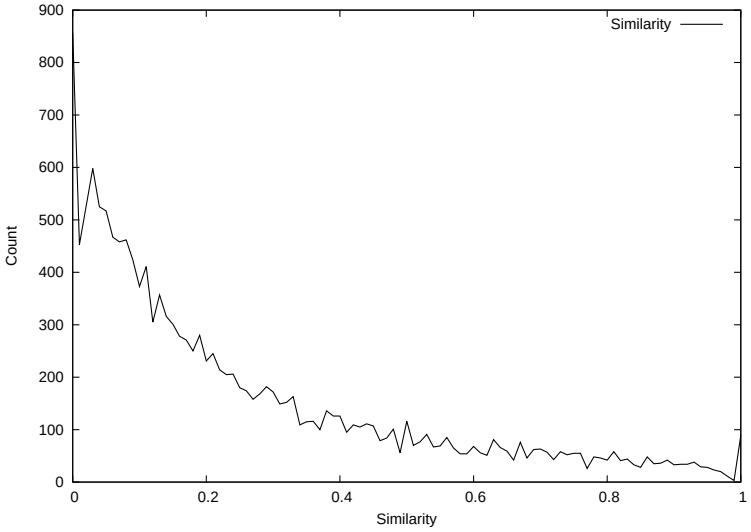

図 5.7　類似度のヒストグラム (C++)

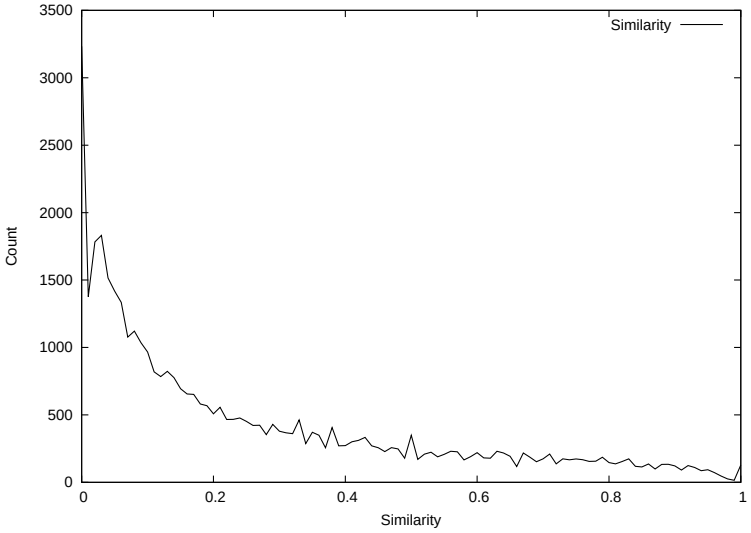

図 5.8　類似度のヒストグラム (C#)

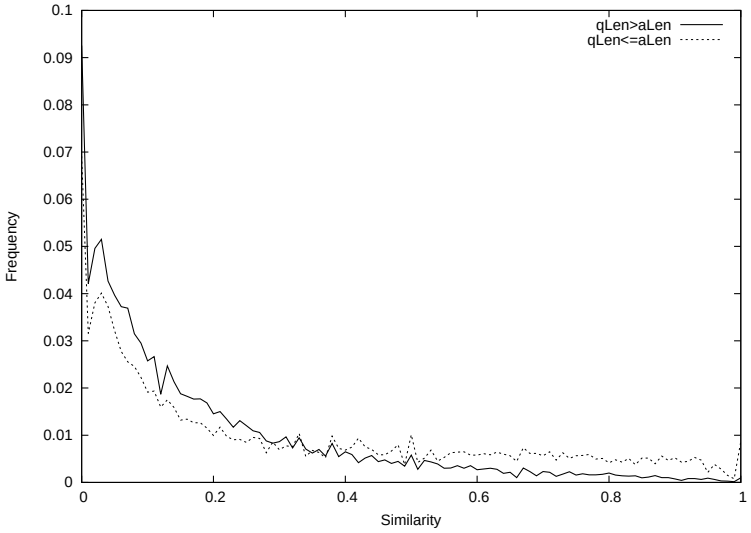

図 5.9　$L_q > L_a$ と $L_q \leqq L_a$ で分割した類似度のヒストグラム (JAVA)

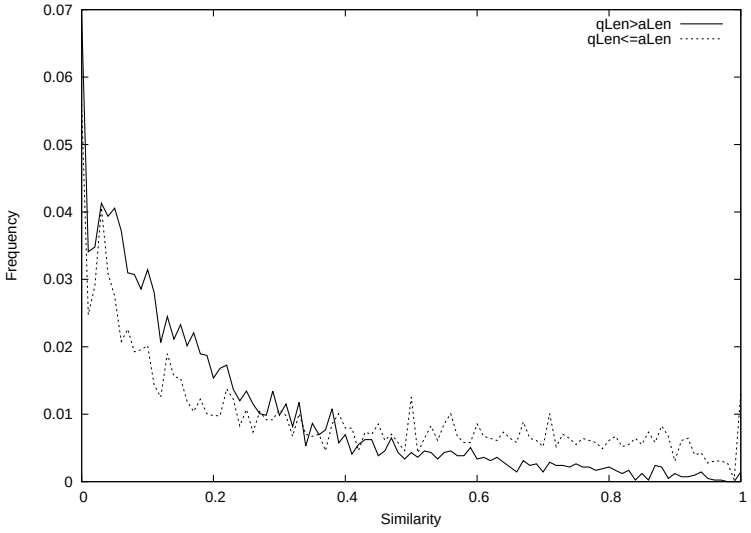

図 5.10　$L_q > L_a$ と $L_q \leqq L_a$ で分割した類似度のヒストグラム (C)

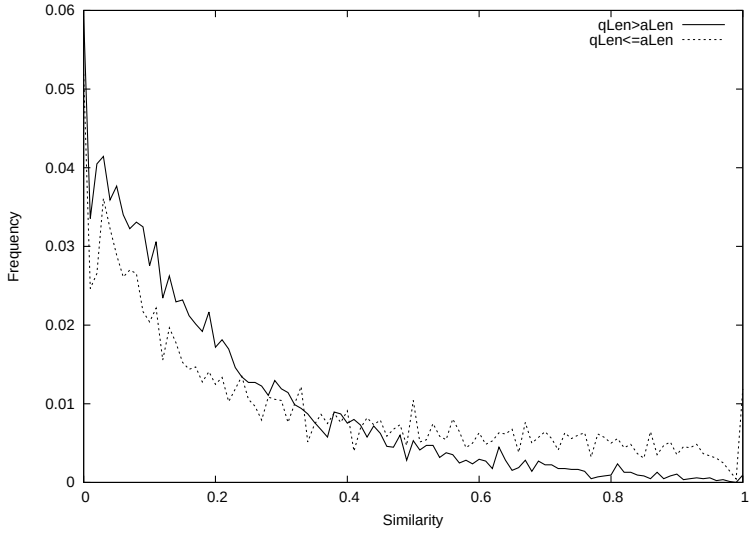

図 5.11　$L_q > L_a$ と $L_q \leqq L_a$ で分割した類似度のヒストグラム (C++)

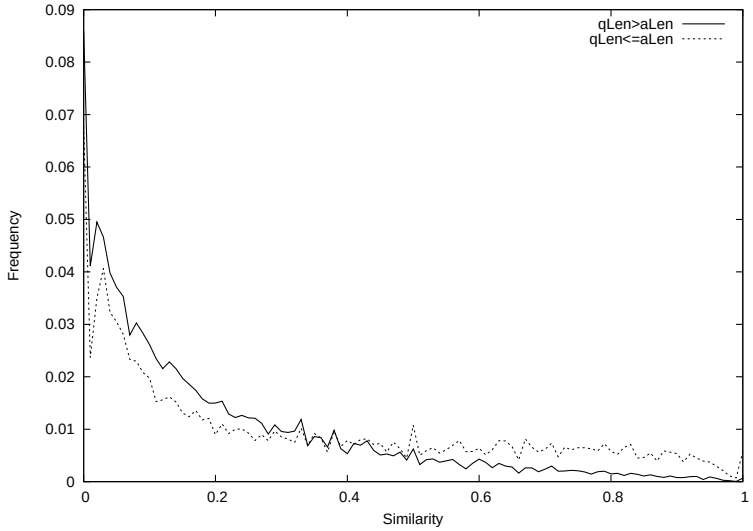

図 5.12　$L_q > L_a$ と $L_q \leqq L_a$ で分割した類似度のヒストグラム (C#)

5.4 実験 4

閾値を 0.6 とした時の，JAVA に関する識別結果を表 5.1，C に関する識別結果を表 5.2，C++に関する識別結果を表 5.3，C#に関する識別結果を表 5.4 に示す.

JAVA を対象とした時の CRM114 が出力した値のヒストグラムを図 5.13，C を対象とした時のヒストグラムを図 5.14，C++を対象とした時のヒストグラムを図 5.15，C#を対象とした時のヒストグラムを図 5.16 に示す.

いずれの言語に関しても約 60%の正確度で分類することが出来た．ただし，C 言語に関してはデータの数が少ないため注意が必要である.

5.5 実験 5

閾値を 0.9 とした時の，$L_q > L_a$ となるソースコードの組に関する識別結果を表 5.5，CRM114 が出力した値のヒストグラムを図 5.17 に示す.

閾値を 0.1 とした時の，$L_q \leqq L_a$ となるソースコードの組に関する識別結果を表 5.6，CRM114 が出力した値のヒストグラムを図 5.18 に示す.

$L_q > L_a$ となるソースコードの組に関しては，正確度，精度，再現率全てにおいて，実験 4 で行った結果よりも良い結果を得た．$L_q \leqq L_a$ となるソースコードの組に関しては，正確度，精度は同程度を維持し，再現率が高くなる結果となった．いずれにおいても，実験 4 よりも良い結果となった.

5.6 実験 6

閾値を 0.6 とした時の識別結果を表 5.7，CRM114 が出力した値のヒストグラムを図 5.19 に示す.

表 5.7 から，不具合を全く識別できていないことが分かる．また，表 5.19 を見ると，CRM114 の出力が 1 であるモジュールが 60%も存在する，

以上から，Eclipse BIRT のソースコード全体をファイル単位で CRM114 へ入力しても，良い結果は得られないと考えられる.

表 5.1　識別結果 (JAVA)

		予測	
		質問	回答
実測	質問	19236	13284
	回答	12167	20343

正確度	精度	再現率	F 値
0.6086	0.6126	0.5915	0.6018

表 5.2　識別結果 (C)

		予測	
		質問	回答
実測	質問	1738	1452
	回答	1072	2108

正確度	精度	再現率	F 値
0.6038	0.6185	0.5448	0.5793

表 5.3　識別結果 (C++)

		予測	
		質問	回答
実測	質問	11808	5322
	回答	7576	9546

正確度	精度	再現率	F 値
0.6234	0.6092	0.6893	0.6468

表 5.4　識別結果 (C#)

		予測	
		質問	回答
実測	質問	24785	18825
	回答	16849	26749

正確度	精度	再現率	F 値
0.5909	0.5953	0.5683	0.5815

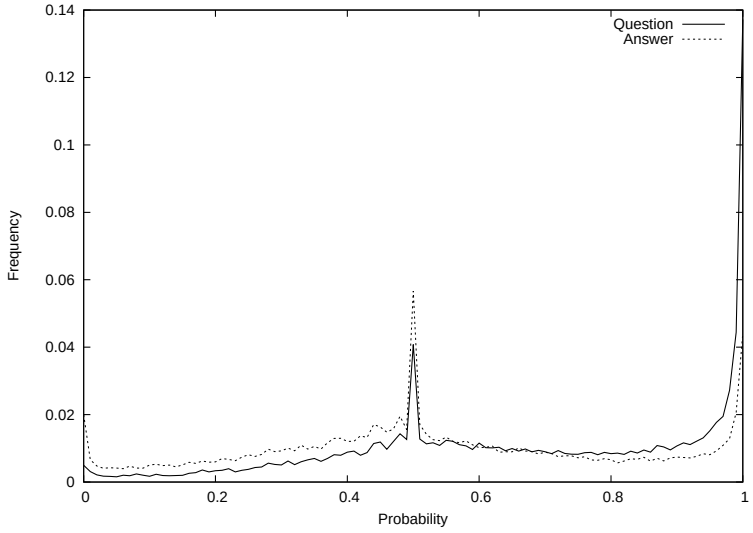

図 5.13　CRM114 から出力された値のヒストグラム (JAVA)

表 5.5　識別結果 ($L_q > L_a$)

		予測	
		質問	回答
実測	質問	13147	4963
	回答	6172	11936

正確度	精度	再現率	F 値
0.6926	0.6805	0.7260	0.7025

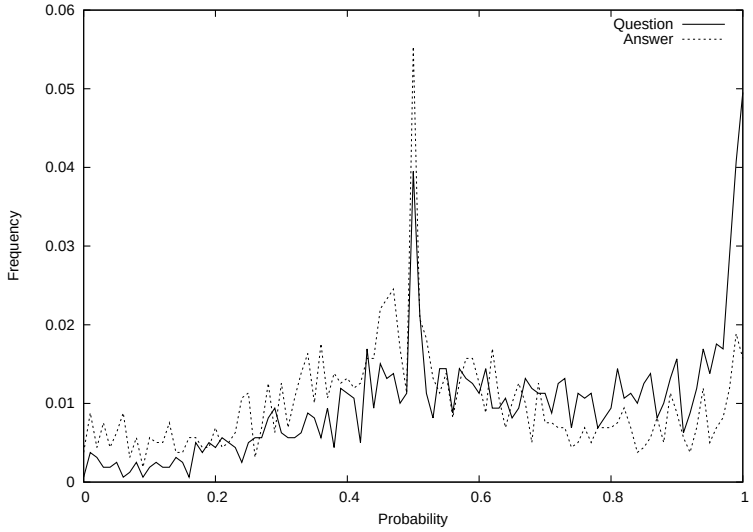

図 5.14　CRM114 から出力された値のヒストグラム (C)

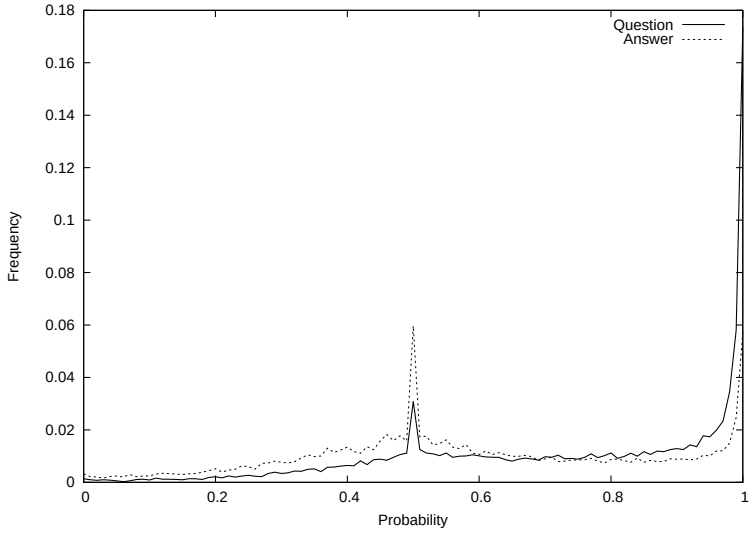

図 5.15　CRM114 から出力された値のヒストグラム (C++)

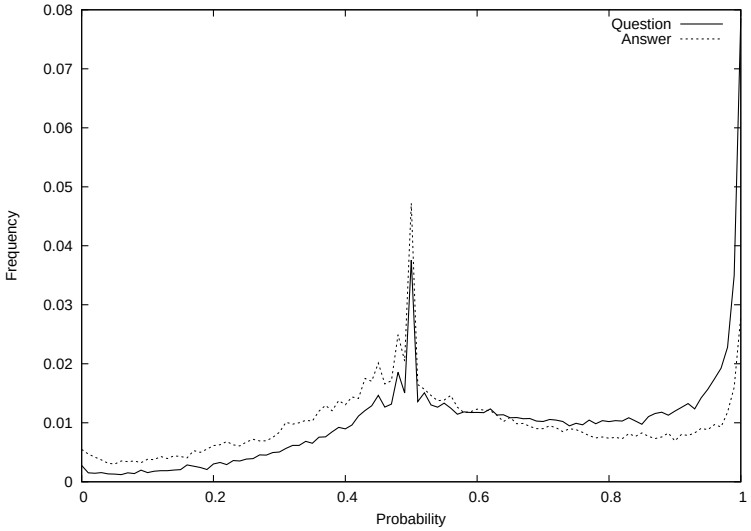

図 5.16　CRM114 から出力された値のヒストグラム (C#)

表 5.6　識別結果 ($L_q \leqq L_a$)

		予測	
		質問	回答
実測	質問	10165	4245
	回答	6735	7667

正確度	精度	再現率	F 値
0.6189	0.6015	0.7054	0.6493

表 5.7　Eclipse BIRT に対する Fault-Prone モジュール予測結果

		予測	
		質問	回答
実測	Fault-Prone	9921	1394
	Non-Fault-Prone	9914	1400

正確度	精度	再現率	F 値
0.5003	0.5002	0.8768	0.6370

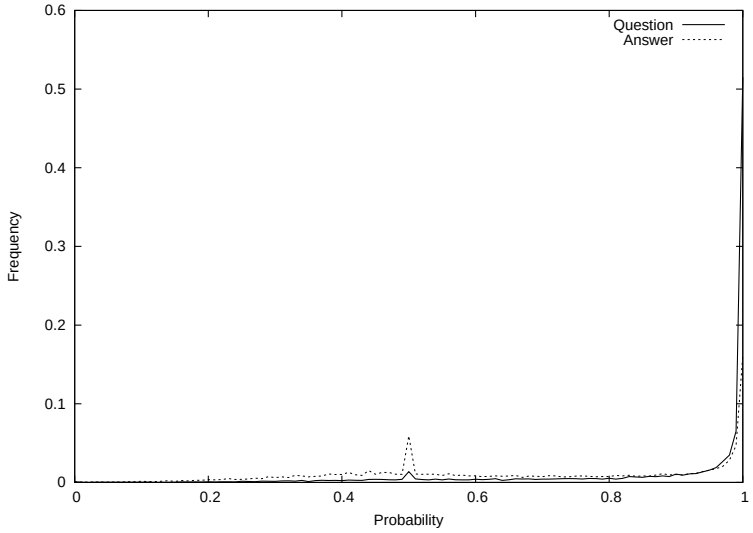

図 5.17　CRM114 から出力された値のヒストグラム ($L_q > L_a$)

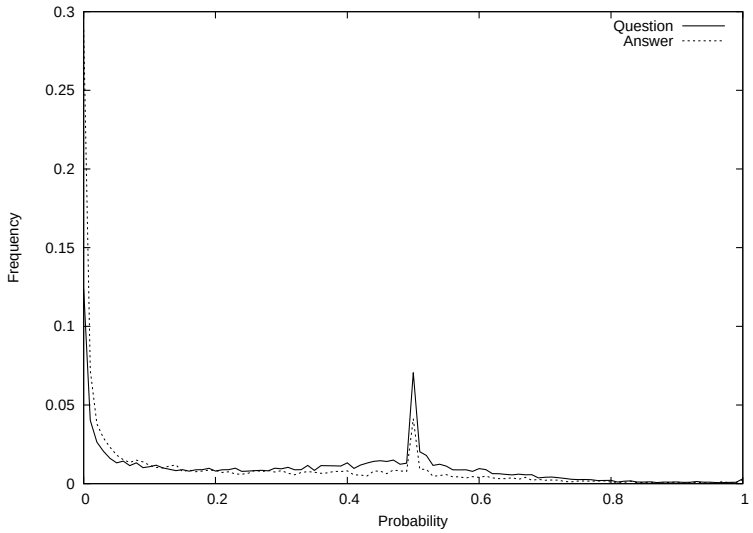

図 5.18　CRM114 から出力された値のヒストグラム ($L_q \leqq L_a$)

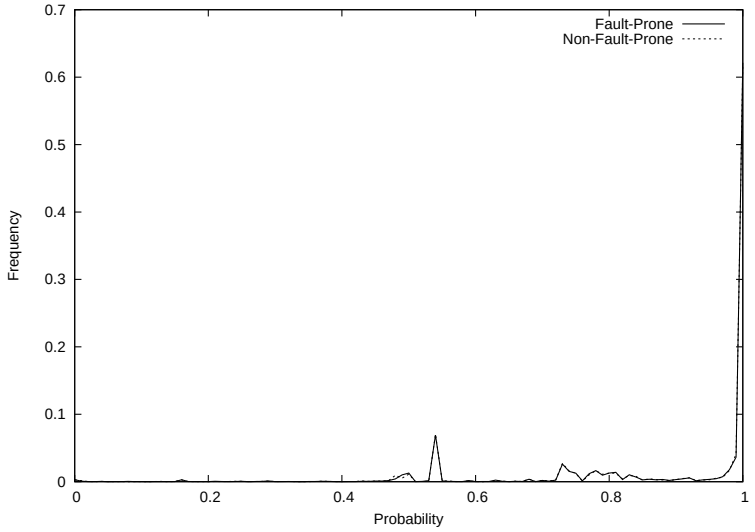

図 5.19　CRM114 出力値のヒストグラム (Eclipse BIRT)

5.7 実験 7

閾値を 0.6 とした時の識別結果を表 5.8，CRM114 が出力した値のヒストグラムを図 5.20 に示す.

表 5.8 を見ると，正確度が 0.4769 と，0.5 を下回っている．また，図 5.20 を見ると，不具合を含む Eclipse BIRT のソースコードの方が 0 に近い値が出力される事が多く，更に，不具合を含まないソースコードの方が 1 に近い値が出力される事が多い事が分かる．1 に近い値ほど質問のコードに類似していると CRM114 が識別しているということであるため，CRM114 は以下のように識別していることになる.

- Eclipse BIRT のプロジェクトにおいて不具合を含むモジュールは，CRM114 の識別では回答に含まれるソースコードに類似している.
- Eclipse BIRT のプロジェクトにおいての不具合を含むモジュールは，CRM114 の識別では質問に含まれるソースコードに類似している.

ただし，正確度，精度，図 5.20 から分かるように，この差は軽微なものである.

5.8 実験 8

閾値を 0.9 とし，$L_q > L_a$ となる質問と回答に含まれるソースコードの組を学習データに用いた際の識別結果を表 5.9，CRM114 が出力した値のヒストグラムを図 5.21 に示す.

また，閾値を 0.1 とし，$L_q \leqq L_a$ となる質問と回答に含まれるソースコードの組を学習データに用いた際の識別結果を表 5.10，CRM114 が出力した値のヒストグラムを図 5.22 に示す.

$L_q > L_a$ となるソースコードの組を学習データに用いた場合，実験 7 と類似した結果を示している．一方，$L_q \leqq L_a$ となるソースコードの組を学習データに用いた場合は，正確度が 0.5321 と 0.5 を上回っている.

実験 7 と同様，正確度，精度，図 5.21, 5.22 から分かるように，この差は軽微なものである．しかし，$L_q > L_a$ となるソースコードの組を学習データに用いた場合は正確度が 0.5 を下回り，$L_q \leqq L_a$ となるソースコードの組を学習データに用いた場合で正確度がが 0.5 を上回るという結果は重要である.

表 5.8　Eclipse BIRT の差分データに対する Fault-Prone モジュール予測結果

		予測	
		質問	回答
実測	Fault-Prone	11641	15724
	Non-Fault-Prone	12907	14459

正確度	精度	再現率	F 値
0.4769	0.4742	0.4254	0.4485

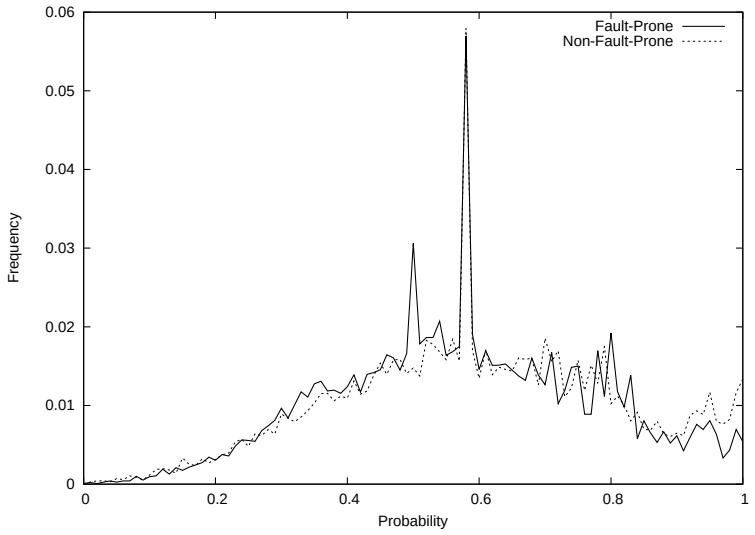

図 5.20　CRM114 出力値のヒストグラム (Eclipse BIRT・差分)

表 5.9　Fault-Prone モジュール予測結果 ($L_q > L_a$)

		予測	
		質問	回答
実測	質問	10769	16596
	回答	12675	14691

正確度	精度	再現率	F 値
0.4652	0.4593	0.3935	0.4239

表 5.10　Fault-Prone モジュール予測結果 ($L_q \leqq L_a$)

		予測	
		質問	回答
実測	質問	17150	10215
	回答	15396	11970

正確度	精度	再現率	F 値
0.5321	0.5269	0.6267	0.5725

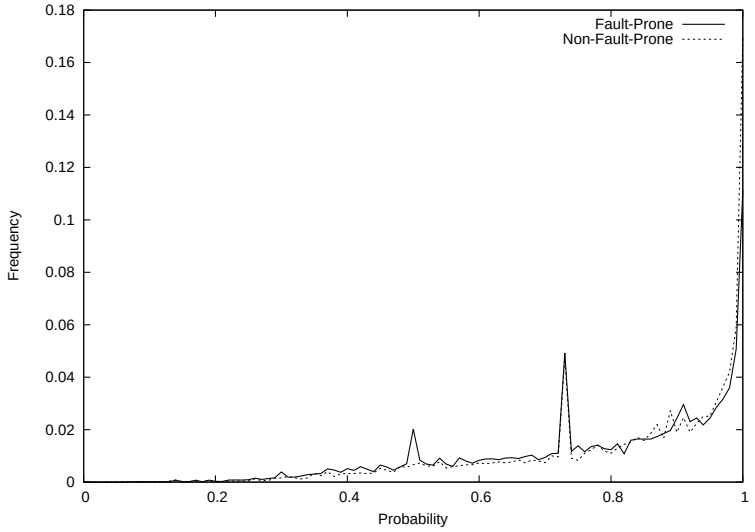

図 5.21　CRM114 から出力された値のヒストグラム ($L_q > L_a$)

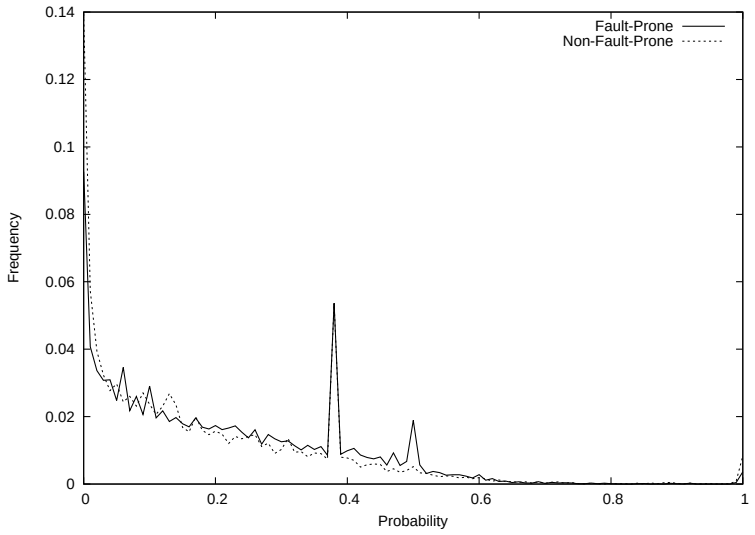

図 5.22　CRM114 から出力された値のヒストグラム ($L_q \leqq L_a$)

6. 考察

6.1 実験結果に関して

実験 1, 2, 3 では各言語に関する質問と回答から取得したテキストに対して，LOC と類似度の観点から比較を行ったが，いずれの実験でもプログラミング言語による差は見られなかった．

実験 4 では交差検証を用いて，約 60%の正確度で分類することが出来た．よって，質問に含まれるソースコードと回答に含まれるソースコードには CRM114 で識別できる特徴が含まれていると考えられる．

実験 6, 7 では Eclipse BIRT の不具合に対して Fault-Prone モジュール予測を適用したが，良い結果を得ることは出来なかった．ただし，Eclipse BIRT に存在する不具合と Stack Overflow で質問される内容が異なる種類のものである可能性がある．Eclipse BIRT で過去に存在した不具合は，機能的なものが殆どであり，文法エラー等により，コンパイルが通らないファイルは存在しない．一方，Stack Overflow から取得したソースコードには，文法エラーを含むものが多数存在する．また，Eclipse BIRT を記述しているプログラマーは JAVA 言語に対する成熟度が高いと思われる一方，Stack Overflow の質問・回答者には成熟度が低いプログラマーも多数存在する．

この様に，Eclipse BIRT に存在する不具合と Stack Overflow に存在する質問・不具合は別種のものである可能性がある．

また，実験 6 で使用した Eclipse BIRT のソースファイルは，1 つのファイルに対し数百行も記述されている大規模なソースファイルが多い．不具合の修正によって変更された部分はそれらのファイルの内，数行でしかないため，CRM114 から出力される値がが殆ど変化しなかったと考えられる．

次に，$L_q > L_a$ となるソースコードの組と $L_q \leqq L_a$ となるソースコードの組に関して考察を行う．実験 5 では，$L_q > L_a$ と $L_q \leqq L_a$ のクラスそれぞれで，分割しない場合より良い結果を得た．このことから，$L_q > L_a$ となるソースコードの組と $L_q \leqq L_a$ となるソースコードの組にはそれぞれ違う特徴が存在すると考えられる．また，実験 3 において，$L_q \leqq L_a$ となるソースコードの組では類似度が高いソースコードの組が多い結果となった．JAVA 言語に関して，$L_q \leqq L_a$ となるソースコードの組で類似

度 0.3 以上のものの例を表 6.1 に示す．このように，$L_q \leqq L_a$ となるソースコードでは，質問に含まれるソースコードに回答者が修正・機能追加を行っている可能性が高く，それによって，CRM114 による識別率の向上へ繋がった可能性がある．実験 8 において $L_q \leqq L_a$ で識別率が 50%を超えたことも，この特徴に関係すると考えられる．

一方，実験 8 において $L_q > L_a$ となるソースコードの組を学習データとして用いた場合，識別率が 50%を切る結果となった．$L_q > L_a$ となる時，回答に含まれるソースコードは質問に含まれるソースコードの一部を切り出している可能性が高い．その場合，質問文のソースコードに存在する不具合を含まない部分は，回答にあるソースコードに含まれない．その結果，CRM114 に入力された際に不具合を含まないソースコードの大部分が質問に含まれるソースコードとして学習されてしまうことが，原因の 1 つとして考えられる．

6.2 妥当性の検証

6.2.1 テキスト分類フィルタの選定

本研究ではテキスト分類フィルタとして CRM114 を標準の分類手法である OSB で用いたが，この手法がソースコードの分類に有効であるかが問題となる．OSB では任意の連続する 5 単語の組み合わせのうち 2 単語の組を使用するが，この手法がソースコードの分類に適しているかは検証が必要である．

6.2.2 Stack Overflow から取得したソースコード

本研究で使用した質問と回答に含まれるソースコードは，以下の制限を設けている．

(1) 質問に対し，承認された回答が存在する
(2) <pre>・<code> タグで囲われた内容が，質問・承認された回答共に 1 箇所のみ存在する．
(3) 質問と回答に含まれるソースコードの類似度が 0.05 以上 1 未満である．

制限 (2) に関して，以下の様な場合，ソースコードは取得されない．

表 6.1　$L_q \leqq L_a$ となるソースコードの例 (JAVA)

質問に含まれるソースコード	回答に含まれるソースコード

public int foo() {
  Resource f = new Resource();
  DoSomething(f);
  f.Release();
}
	public int foo() {
	  Resource f = new Resource();
	  try {
	  DoSomething(f);
	  }
	  finally {
	  f.Release();
	  }
	}

public static void main(String[] args){
  ...
}
	public class JavaClass{
	  protected JavaClass(int x){}
	  public void main(String[] args){
	  }
	}

for (int i = 65; i < 91; i++){
  alphabetPanel.add(new JButton(
  "<html> " + (char)i));
}
	ActionListener listener = something;
	
	for (int i = 65; i < 91; i++){
	  JButton button = new JButton(
	  "<html> " + (char)i);
	  button.addActionListener(listener);
	  alphabetPanel.add(button);
	}

- C や C++に関し，ソースファイルとヘッダファイルをそれぞれ記述している場合
- C#や JAVA に関し，クラスごとにファイル分割している場合

また，上記の条件で取得したテキストがソースコードではない可能性も存在する．

この様な制約や危険性によって，取得したソースコードに何らかの偏りが生じている可能性が存在する．

6.2.3 適用データの選定

本研究では適用データとして Eclipse BIRT のソースコードを用いた．

6.3 研究設問への回答

結果・考察を踏まえ，研究設問への回答を以下に示す．

RQ1: 質問と承認された回答に含まれるコードの行数に差はあるか．また，プロラミング言語別にはどのような傾向になるか．

質問に含まれるソースコードの方が，承認された回答に含まれるソースコードよりも行数が多い傾向がある．プログラミング言語による差は見られなかった．

RQ2: 質問と承認された回答に含まれるコードは類似しているか．また，プロラミング言語別にはどのような傾向になるか．

類似していないソースコードの組が多い傾向にあった．プログラミング言語による差は見られなかった．

RQ3: 質問と承認された回答に含まれるコードをさらに細かく分類できる様な特徴は存在するか．

質問に含まれるコードの行数 L_q と承認された回答に含まれるコードの行数 L_a を比較した時，$L_q > L_a$ となる組と $L_q \leqq L_a$ となる組で異なる特徴を持つ可能性がある．ただし，これに関しては詳しく検証する必要がある．

RQ4: Stack Overflow の質問と承認された回答に含まれるコードを，テキスト分類フィルタを用いて分類することは可能か．

約 60%の正確度で分類することができた．

RQ5: Stack Overflow の質問と承認された回答に含まれるコードを学習データに用いて，実際のバグに対して Fault-Prone モジュール予測が可能であるか

Eclipse BIRT に存在する不具合に適用したが，予測は出来なかった．ただし，様々なソースコードに適用し検証する必要がある．

6.4 今後の課題

6.4.1 ソースコードの取得方法の改善

<pre>・<code> タグで囲われた内容が，質問・回答共に 1 箇所のみ存在するソースコードを本研究では用いた．しかし，複数のソースコードを記載した質問・回答も Stack Overflow には多数存在し，これらのソースコードも用いるべきである．

また，<pre>・<code> タグで囲われた内容がソースコードではない可能性があり，それらを適切に取り除く必要もある．本研究では質問と回答から取得したテキストの類似度を用いることで一部を取り除くことが出来たが，他の方法に関しても検討する必要がある．

6.4.2 質問内容の把握

本研究では <pre>・<code> タグで囲われた内容のみを扱ったが，表 4.1 にあるように，質問や回答には他にも様々な情報が存在する．特に，質問のタイトルや本文には，質問内容が書かれており，ソースコードの目的や，不具合の内容などが記載されている．それらの内容を把握することができれば，不具合を含んだソースコードであるか，<pre>・<code> タグで囲われた内容がソースコードであるかどうか等，有用な情報が得られる可能性がある．

6.4.3 様々なソースコードへの適用

今回は Eclipse BIRT の不具合に対して適用実験を行ったが，他のプロジェクトの不具合に対しても適用する必要がある．Stack Overflow 等の質問サイトへ投稿される質問にはプログラミングに不慣れな者が記述したコードも存在する．そのため，情報工学コースの大学生が授業で記述したソースコード等，熟練度の低い者が書いたソースコードに対して適応実験を行うと，Eclipse BIRT で行った実験とは異なる結果が得られる可能性がある．

6.4.4 高級言語への適用

本研究ではトークナイザの都合上，JAVA，C 言語，C++，C#と文法が近い言語を対象とした．

Python や Haskell 等，文法が JAVA や C 言語と大きく異なるプログラミング言語に適用した場合，異なる結果が得られる可能性がある．

7. 結言

本研究では，Stack Overflow から質問と承認された回答それぞれに含まれるソースコードを取得し，テキスト分類フィルタ CRM114 を用いたソースコードの識別手法を提案し，JAVA，C 言語，C++，C#のプログラミング言語において，質問と解答それぞれに含まれるソースコードの識別を約 60%の正確度で行うことを可能にした．

今後の課題として，高級言語への適用や，Fault-Prone フィルタリングへの応用がある．

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・人間科学系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，本学ソフトウェア工学研究室の皆さん，学生生活を通じて著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] R.P.L. Buse and W.R. Weimer., “Learning a metric for code readability,” TSE, vol.36, no.4, pp.546–558, 2010.

[2] M.M. Rahman and C.K. Roy, “An insight into the unresolved questions at stack overflow,” Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on, pp.426–429, March 2015.

[3] Themistoklis Diamantopoulos and Andreas L. Symeonidis, Employing Source Code Information to Improve Question-Answering in Stack Overflow, MSR 2015（オンライン）, 入手先〈http://2015.msrconf.org/program.php〉（参照 2015-02-12）.

[4] William S.Yerazunis, CRM114 - the Controllable Regex Mutilator, CRM114（オンライン）, 入手先〈http://crm114.sourceforge.net/〉（参照 2015-01-29）.

[5] O. Mizuno and T. Kikuno, “Prediction of fault-prone software modules using a generic text discriminator,” IEICE Trans. on Information and Systems, vol.E91-D, no.4, pp.888–896, April 2008.

[6] O. Mizuno and T. Kikuno, “Training on errors experiment to detect fault-prone software modules by spam filter,” ESEC/FSE2007, vol.E91-D, pp.405–414, Sept. 2007.

[7] H. Hata, O. Mizuno, and T. Kikuno, “Fault-prone module detection using large-scale text features based on spam filtering,” Empirical Software Engineering, vol.15, no.4, pp.147–165, April 2010.