社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ソースコード特徴量を用いた機械学習による ソースコード品質の評価手法

槇原　啓介† 　松下　　誠† 　井上　克郎†

† 大阪大学

あらまし　本研究は，機械学習を用いてソースコードの品質を定量的かつ自動で評価する手法を提案する．本手法では，プログラミングコンテストサイトから収集したレーティング上位と下位のソースコードを上級者と初級者とし、両者で差異が見られたソースコード特徴量を用いて学習モデルを作成する．本手法の実装を行い，評価実験として上級者，初級者のソースコードを分類した結果，およそ 90% の精度で適切な評価を行うことができた．

キーワード　プログラミングコンテスト，レーティング，機械学習

Source Code Evaluation Method by Machine Learning using Source Code Feature Metrics

Keisuke MAKIHARA†, Makoto MATSUSHITA†, and Katsuro INOUE†

† Osaka University.

Abstract　We propose a method to evaluate source codes quantitively and automatically using machine learning. In this method, we collected source codes from the site of programming contest. We define the higher rank rating as seniors, and the lower rank rating as beginners. We created a learning model using source code features that showed differences between the two. As a result of implementing this method and classifying source codes as seniors or beginners, it was possible to judge source code of senior or beginner with 90% accuracy.

Key words　Programming Contest, Rating, Machine Learning

1. ま え が き

ソースコードを編集することは，ソフトウェアを開発する上で必要不可欠な作業であり，ソフトウェア工学の研究分野として開発者がソースコードに対して行う編集作業の内容に関する研究は数多く行われている [1]〜[3]．開発の過程においては，ソースコードの実装，テスト，修正を繰り返し行うため，これらの作業にかかる労力は大きい．特に，初級者は時間的コストが多くかかるため，行われた編集が適切であるかどうか，判断できることが望ましい．2020 年度から小学校においてもプログラミング教育が必修化される(注1)など，プログラミングを学習し始める人の数は今後ますます増加していくことが予想される．プログラミングにおける基本的な作業の 1 つである編集作業を適切に支援することは今後重要となると考えられる．

ソースコードが与えられた時，仮にその内容の良否を判定できるならば，プログラミングの学習に役立つと考える．否と判定されたソースコードを修正し，その結果良と判定されるようになれば，修正した編集は適切であったと言える．また，結果が良好になることによって、プログラミングを行った作業者は自分の成長を実感できる．さらに良否の判定に用いた基準を利用し，より良い判定を得られるような修正内容を支援することができれば，初級者はそれらを参考にプログラミング学習を進めることができるようになり，効率的な技術力の向上を期待できる．

良いソースコードとはどのようなものかを考えた場合，構造や読みやすさ，実行時間など様々な基準が考えられる．しかしながら人によって良いソースコードの基準が異なるため，広く合意できる判断を行うことは難しい．広く受け入れられる基準を得るためには，多くのソースコードを用いて定量的にその良否を判定し、その判断に基づく基準が重要である．ソースコードの編集作業に関して，上級者と初級者間にどのような差異が存在するのかという研究は既に行われている [4], [5]．その研究によると，両者の間には，予約語利用頻度やメトリクスといった定量的なソースコード特徴量の差異が確認されている．この特徴の差異を利用すれば，書かれたソースコードが上級者に近

(注1)：http://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1375607.htm

いか，それとも初級者に近いかということを定量的に判断し，修正の指針を与えることができるのではないかと考えた．

本研究では，ソースコード特徴量を用いたソースコードの品質を評価する手法として，ソースコードの良否の判定を行う．ソースコードの良否を定量的かつ自動で判断する手法として機械学習を用いる．プログラミングコンテスト内で提出されたソースコードに対し，上級者・初級者間で差異が見られたソースコード特徴量の値と，ソースコードの「良」「否」を機械学習プログラムに与えることによって学習モデルを作成する．良否に関しては，上級者が書いたソースコードを「良」，初級者が書いたソースコードを「否」と定義した．作成した学習モデルに対し，学習に用いられていないソースコード特徴量の値を入力すると，ソースコードの良否を自動で出力する．出力結果が否の場合は，ソースコード特徴量を用いて修正の指針を提示する．

以降，2. 節でプログラミングコンテストについて説明する．3. 節で先行研究での調査内容について説明する．4. 節で提案手法について説明し，5. 節で提案手法の実装を用いた実験について説明する．最後に， 6. 節でまとめを述べる．

2. プログラミングコンテスト

本節では，プログラミングコンテストについて述べる．以下ではまず，プログラミングコンテストに用いられるオンラインジャッジシステムと，プログラミングコンテストの概要について説明する．

2.1 オンラインジャッジシステム

プログラミングコンテストにおいて，その採点に利用されるオンラインジャッジシステムについて説明する． オンラインジャッジシステムは，利用者に問題を提示し，問題に対する利用者の回答を受け取る． そして，受け取った回答を採点し，結果を利用者に通知する． オンラインジャッジシステムの一例として，国内の AIZU ONLINE JUDGE(注2)，アメリカの Topcoder(注3)などが挙げられる．

2.2 プログラミングコンテストの概要

プログラミングコンテストとは，プログラミングの能力や技術を競い合うコンテストのことである．オンラインジャッジシステムを用いて複数の参加者が同じ問題セットを同じ時間に解く方式で行われる．参加者の正解問題数や回答時間に応じて参加者の順位が決定し，コンテスト終了後に順位に応じて参加者のレーティング [6], [7] が変動する．

2.2.1 ルールと流れ

プログラミングコンテストの実際の流れについて，本研究でデータセットとして用いる Codeforces(注4)を例として説明する．Codeforces における一般的なコンテストでは，指定の時間になるとコンテスト参加者に複数の問題が公開される． 参加者は自由な順序・言語で問題を解くことができ，解いた問題の難易度と回答までにかかった時間に応じて参加者に得点が加算される．

コンテスト時間終了後に提出ソースコードに対して最終テストが実施され，参加者の最終的な得点が決定する．得点に応じて各参加者の順位が決定し，レーティングが変動する．

2.2.2 レーティングシステム

Codeforces は参加者の熟練度を示す指標としてレーティングシステム [6] を用いている． 主にチェスやサッカーで用いられるイロレーティング [7] に似た計算方式を採用しており，コンテスト毎に参加者の順位に応じてレーティングを計算する．レーティングが高い参加者ほどコンテストで高い成績を取っているということができる． 本研究においても，このレーティングが高い参加者を熟練度が高い参加者とし，アルゴリズムの問題に対して適切にコーディングを行う能力が高いと考える．

3. ソースコード編集における上級者・初級者間の比較調査

先行研究として，上級者と初級者が行う編集作業に見られる差異について調査が行われている [4], [5]．本節では，先行研究の調査内容と調査結果のうち，本研究に関連する部分について述べる．

3.1 調 査 内 容

先行研究では，世界最大規模のプログラミングコンテストである Codeforces の参加者を対象に，参加者がコンテストの問題へ回答として提出したソースコードの編集作業について調査を行った．

ソースコード編集作業の特徴調査を行うために，2016/5/19〜2016/11/15 の 6 ヶ月間に Codeforces に対して提出された 1,644,636 のソースコードと，それに付随した提出履歴情報を収集しており， このうち，収集ソースコードの 9 割を占める C++ でのソースコードが調査対象となった．収集したソースコードデータと提出履歴情報はデータベースに保管しており，オンラインで公開してある(注5)．以下，この研究に用いられたデータセットの説明と調査結果の説明を行う．

3.2 データセット

データセットは，参加者が問題への回答として提出したソースコードと，言語やタイムスタンプ等の提出履歴情報データベースの 2 種類からなる．参加者数は，2016/5/19〜2016/11/15 の期間に 1 度以上 Codeforces のコンテストに参加したユーザーの総数である．

収集されたソースコードは，Codeforces のプログラミングコンテスト参加者が，コンテストの問題に対する回答として提出したものである．ソースコードは，Codeforces がコンパイル環境を用意している任意の言語で記述することができる．本データセットのソースコードのうち，90% は C++ によって記述されており，それ以外は Java が約 4%，C が約 2%，Python が約 2%，その他が約 2% となっている．

提出履歴情報としては，参加者，対象となったコンテストとその問題，参加者が提出したソースコード，ソースコードの提

(注2)：http://judge.u-aizu.ac.jp/onlinejudge/
(注3)：https://www.topcoder.com/
(注4)：http://codeforces.com/
(注5)：https://sites.google.com/site/miningprogcodeforces/

出結果，連続した提出間の編集距離などが記録されている．編集距離としては，ソースコードのテキストとしてはレーベンシュタイン距離 [9]，抽象構文木上の編集距離は GumtreeDiff [10] を用いている．

3.3 上級者・初級者の定義

ソースコード編集作業について分析を行う上で，プログラミングコンテスト参加者を上級者と初級者の 2 群に分割して比較調査を行った．具体的には次のように参加者を分割している．

（1） 参加者をレーティングでソート

（2） ソートした参加者を人数が等しくなるように 4 分割

（3） 4 分割された参加者のうち，最もレーティングが高いグループを上級者，最もレーティングが低いグループを初級者とする

4 分割された参加者のうち，レーティングが中央値付近の 2 群については考慮しない．このとき，上級者・初級者の分布は表 1 のようになった．人数が 2 群で異なるのは，同じレーティングを持つ参加者が境界に複数分布するためである．

表 1 上級者・初級者の 2 群におけるレーティングの統計情報

		初級者	上級者
レーティング	平均	1171.12	1824.824
	分散	113.62	226.54
	最小値	-39	1573
	最大値	1299	3367
人数		3634	3622

3.4 調 査 結 果

プログラミングコンテスト上級者・初級者間におけるソースコードの編集作業における特徴調査のうち，予約語利用頻度の調査とメトリクスの値に関する調査について以下で述べる．Codeforces 内の同一の問題に対して提出されたソースコードの集合に対し，ソースコード特徴量を標準化するなどの処理を施し，調査を行った．

3.4.1 予約語利用頻度の調査

上級者・初級者間で統計的に有意差が見られた予約語をそれぞれ表 2，表 3 に示す．表 2 に示した上級者の上位予約語に着目すると，プログラムの構造化を促進する予約語を多く用いている傾向が確認された．上級者は初級者と比較して，ソースコードの抽象度が高いのではないかと考えられる．また，表 3 に示した初級者の上位予約語に着目すると，分岐に関する予約語を多く用いる傾向にあることが確認された．初級者は分岐文を多用し，ソースコードが複雑化する傾向があるのではないかと考えられる．

3.4.2 メトリクスの値に関する調査

調査対象となったメトリクスの一覧と説明を表 4 に示す．SourceMonitor(注6) で計測できる 11 種類のメトリクスである．

(注6)：http://www.campwoodsw.com/sourcemonitor.html

表 2 上級者の利用頻度が高い予約語

t	初級者分散	上級者分散
typedef	0.896	1.082
template	0.778	1.162
return	0.881	1.042
typename	0.602	1.225
while	0.863	1.088
operator	0.793	1.156
class	0.883	1.100
using	0.720	1.270
continue	0.914	1.043
struct	0.800	1.061
do	0.564	0.925
public	0.633	1.016
namespace	0.885	1.106
enum	0.179	1.012
asm	0.049	0.772
private	0.390	0.814
typeid	0.010	0.489
friend	0.650	0.844
decltype	0.180	0.657
try	0.011	0.410
catch	0.011	0.408

表 3 初級者の利用頻度が高い予約語

t	初級者分散	上級者分散
else	1.061e+00	0.915
break	1.074e+00	0.942
if	9.862e-01	1.003
for	1.005e+00	1.023
goto	8.934e-01	0.657
case	6.454e-01	0.436
switch	6.406e-01	0.463
extern	8.354e-04	0.001

いずれのメトリクスについても上級者・初級者間で統計的に有意差が見られた．上級者の計測値が大きかったメトリクスに着目すると，上級者はソースコードの行数が増える傾向が伺える．初級者の計測値が大きかったメトリクスとして，ネストの深さや複雑度の高さが確認された．

3.5 本研究への利用

調査結果から，上級者は構造に関する記述が，初級者は分岐に関する記述が多いという差異が見られた．

本研究では，これらのソースコード特徴量をベクトル化したデータを用いて，定量的にソースコードの良否の判定及び修正の指針の提示を行う．また機械学習を用いて，自動で判定を行う．

4. 提 案 手 法

本節では，ソースコード品質を評価する手法として，ソースコード特徴量を用いた機械学習によるソースコードの良否の判定方法及び修正の指針を提示する手法について説明する．

表 4　調査対象メトリクス

メトリクス	説明
avg_complexity	各関数の循環的複雑度 [8] の平均値
max_complexity	各関数の循環的複雑度の最大値
avg_depth	各関数のネスト深さの平均値
max_depth	各関数のネスト深さの最大値
methods_per_class	クラス当たりのメソッド数
n_classes	クラス数
n_func	関数の数
n_lines	物理行数
n_statements	セミコロンで区切られた論理行数
percent_branch_statements	全体の論理行数に占める分岐文の割合
percent_comments	全体の物理行数に占めるコメントの割合

4.1　目　　的

ソースコードの良否を判定できれば，プログラミングの学習に役立つ．否と判定されたソースコードを修正し，良と判定されるようになれば，書き手は自分の成長を実感できる．また，ソースコード特徴量を用いた修正のヒントが得ることができれば，効率的に編集を行うことができる．

4.2　判 定 方 法

ソースコードの良否の判定を行う流れを，図 1 に示すように大きく 3 つの段階に分けた．それぞれの段階について，以下で詳細を述べる．

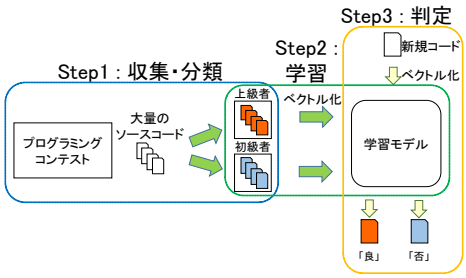

図 1　ソースコードの良否の判定の流れ

Step1: 収集・分類

本研究では，3. 2 節で述べたオンラインで公開しているデータセットを MySQL 上に保存し，データベースを管理した．ソースコードを上級者と初級者に分類した結果，上級者は 544,290 個，初級者は 332,863 個であった．

Step2: 学習

説明変数はソースコード特徴量，目的変数はソースコードの良否である．

ソースコードに対して字句解析を行うことで表 2，表 3 で示した 29 種類の予約語の利用頻度を，SourceMonitor を用いることで表 4 で示した 11 種類のメトリクスの値を取得する．これらを組み合わせて，1 つのソースコードにつき 40 次元のベクトルを作成する．

次に，上級者が書いたソースコードを「良」，初級者が書いたソースコードを「否」と定義し，それぞれ値として 1，-1 を与える．この値を先ほどのベクトルに追加し，41 次元のベクトルを作成する．図 2 にベクトルの一部を掲載する．"skill"という変数名で目的変数を追加している．

学習の対象となる全てのソースコードに対して図 2 のようなベクトル化を行い，ソースコード特徴量に対する良否判定の学習モデルを構築する．

asm	break	・・・	avg_complexity	n_func	skill
0	0		0	0	1
0	0		0	0	1
0	0		0	0	1
0	0		0	0	-1
0	2		0	0	-1
0	4		0	0	-1

図 2　目的変数"skill"を追加したベクトル

Step3: 判定

良否の判定を行いたいソースコードに対し 40 次元のベクトルを作成する．学習モデルに対し，ベクトルを入力すると，「良」または「否」のいずれかが出力される．

4.3　修正の指針の提示

修正の指針を提示するにあたって，単一のソースコード内における構造の特徴に関する調査を行い，上級者・初級者の差が大きい特徴量を利用することを考えた．初級者と比較して上級者の利用頻度が高い予約語がソースコード内で使用されていなければ，使用するよう促す．

本研究では，単一のソースコード内での予約語利用頻度を-1 から 1 までの値に正規化し，上級者と初級者の値の差が 0.02 以上となる予約語を初級者と比較して上級者の利用頻度が高い予約語と定めた．その内容を表 5 に示す．

表 5　単一ソースコード内において上級者の利用頻度が高い予約語

t	上級者・初級者間の利用頻度の差
return	0.1298554
while	0.0758102
template	0.0560683
for	0.0469827
continue	0.0457610
typename	0.0390510
struct	0.0313462
class	0.0311679
operator	0.0274451

3. 節では，ある予約語 t に対し，同一の問題に提出されたソースコードの集合における相対的な予約語利用頻度を調査していた．それに対し，表 5 に示した値は，単一のソースコード内において，ある予約語 t の，利用された全ての予約語におけ

る相対的な予約語利用頻度を調査した結果である．他のソースコードでの利用数は使用せず，単一のソースコード内のみで利用数の正規化を行った．これは，単一のソースコードを独立して見た場合の構造の特徴を調べるためである．例えば"for"のように，表 3 にある場合でも，表 5 における上級者と初級者の値の差が正になる場合がある．これは，同一の処理を行うソースコードにおける"for"の利用回数は，上級者は初級者と比較して少ないが，単一のソースコード内で利用されているすべての予約語に対する"for"の利用回数は，上級者は初級者と比較して多いということを意味している．この結果からも，表 2 において表の上部の予約語を使用し，下部の予約語は使用しないという単純な修正では良いソースコードにすることが難しいことが分かる．

入力ソースコード内で該当する予約語が使用されていなければ，良否の判定結果と共に予約語の利用を促す．

5. 評 価 実 験

4. 節で述べた手法を実装し，それを用いて実験を行った．本節では，その実験結果を説明する．具体的には，4. 2 節で述べたソースコードの良否判定を行った学習モデルの精度と，4. 3 節で示した特徴量を参考に編集を行った後の判定結果の変化について述べる．

5.1 学習モデルの精度

5.1.1 実 験 手 順

4. 2 節で構築した学習モデルの精度を求めるため，本研究において対象となった合計 877,153 のソースコードに対し，以下のような手順で実験を行った．

（1） 上級者・初級者全てのソースコードに対し，4. 2 節で述べた 41 次元のベクトルを作成する

（2） 1 で作成したベクトルの中からランダムで 1 割抽出し，テストデータとする

（3） 残りの 9 割のベクトルを学習データとし，学習モデルを作成する

（4） テストデータの説明変数を学習モデルに入力する

（5） 入力ソースコードが上級者の場合は「良」，初級者の場合は「否」と出力されれば正解，そうでなければ不正解とする

5.1.2 実 験 結 果

機械学習の精度の評価指標として，上級者・初級者それぞれについて適合率，再現率，F 値を算出した．その結果を表 6 に示す．使用したアルゴリズムは，決定木と SVM である．両アルゴリズム共に，評価指標の値はおおよそ 90% に近い値が多かった．

表 6 機械学習の評価指標

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.918	0.906	0.912	0.940	0.900	0.920
初級者	0.850	0.869	0.860	0.846	0.906	0.875

5.1.3 考 察

ソースコードを上級者・初級者に分類する際に用いたのはレーティングのみであり，ソースコードの構造は無視している．Codeforces においては，制限時間内に問題を解くことができれば得点がつき，終了時の得点が高いほどレーティングが高くなる．つまり，入力に対して正しい出力ができるソースコードを記述することさえできれば，レーティングは上がっていくため，レーティングが高い上級者が提出したソースコードでも，構造を見ると初級者の特徴に近い場合がある．上級者が提出した全てのソースコードの構造 3. 節で述べたような上級者の構造に近いわけではない．判定の能力が高いと，テストデータとして構造が初級者に近いソースコードが入力された場合は，提出者が上級者であっても「否」という判定が出力される．初級者に関しても同様のことが言える．したがって，テストデータの判定においてはある程度の誤りが許容できる．実際には，90% という値以上の分類性能を持っている可能性があり，判定の能力は十分にあると言える．

二値分類では，評価指標の値を正の場合のみ，つまり本研究における上級者のみで算出を行う場合もあるが，本研究においては上級者・初級者の両方で算出を行った．理由としては，学習モデルの精度をより適切に判断するためである．比較対象として，決定木において上級者のデータ数を初級者に揃えた時の結果を表 7 に示す．表 6 よりも判定精度の偏りは小さい．このことより，一方の学習データ数が多いと，それに対応する出力を学習するデータ数が増え，判定範囲が広くなり，出力に偏りが生じると考えられる．このように，入力データ数のパターンは様々考えられるが，表 6 で示した評価指標の値が検証した中で高かったため，最終的にこの結果を採用した．

表 7 上級者・初級者のデータ数を揃えた場合の評価指標

	決定木		
	適合率	再現率	F 値
上級者	0.781	0.818	0.799
初級者	0.809	0.770	0.789

5.2 修正の指針の有効性

本研究においては，ソースコードに対する修正が適切かどうかについて表 8 のように定め，判定が「否」から「良」に変化した場合についてのみ修正が適切であるとした．出力される良否は，入力されたソースコードのソースコード特徴量が上級者と初級者のどちらに近いか，という判定基準で定められる．しかし，どれだけ近いかということは出力結果から判断ができない．例えば，修正前後で判定が「否」のままの場合，修正によって上級者に近づきはしたが依然として初級者に近いのか，より上級者から遠ざかっているのかということは不明である．修正前後で「良」である場合も同様である．「良」から「否」に変化した場合は明らかに修正は不適切である．

そこで今回の実験では，「否」と判定されたソースコードに対

表 8　良否の判定の変化における修正の適切さ

判定	修正前	良		否	
	修正後	良	否	良	否
修正の適切さ		不明	不適切	適切	不明

して表 5 に示した予約語を用いて修正を行い，「良」と判定されるようになれば，修正は適切であったと判断することとした．

5.2.1　実 験 手 順

4.2 節の機械学習プログラムに入力されたソースコードに対し，良否の判定結果とともに，表 5 に示した予約語が存在しない場合，4.3 節に従って，該当する予約語の使用を促す出力をする．実験対象は，学習モデルの構築に用いられていないソースコードである．例として，あるソースコードを入力すると，「否」の判定と共に，図 3 のように出力される．

```
"class" is unused. why don't you use it?
"continue" is unused. why don't you use it?
"for" is unused. why don't you use it?
"operator" is unused. why don't you use it?
"struct" is unused. why don't you use it?
"template" is unused. why don't you use it?
"typename" is unused. why don't you use it?
"while" is unused. why don't you use it?
```

図 3　修正の指針の提示

予約語を用いて行った修正が適切かどうかについて，以下の手順で実験を行い，修正前後における良否の判定の変化を調査した．構造的な修正による判定の変化を確認したいため，アルゴリズムはできるだけ変えず，入力に対する出力結果も変わらないように修正を行った．

（1） 4.2 節で作成した機械学習プログラムにソースコードを入力した結果，「否」と判定されたソースコードを修正対象とする

（2） 図 3 において出力された予約語を用いてソースコードの修正を行う

（3） 再び 4.2 節で作成した機械学習プログラムにソースコードを入力する

（4） 判定が「良」であれば，2 で行った修正は適切であると判断する

5.2.2　実 験 結 果

今回は，実験を行ったソースコードのうち，図 3 の出力がされたソースコードを用いて説明を行う．

図 3 で示された予約語のうち，新たに"class"と"while"を用いることにしてソースコードの修正を行った．その結果，SVM を用いて作成した学習モデルにおいて判定が「否」から「良」に変化することが確認できた．

5.2.3　考　　察

「否」と判定されたソースコードに対して，表 5 に示した予約語を用いて判定が「良」に変化したということは，表 8 より，その修正は適切であったと言える．つまり，上級者・初級者間で差異が存在するソースコード特徴量を用いて修正の指針を示すことは有用となり得ると言える．

6. ま と め

本研究では，ソースコード品質を評価する手法として，ソースコード特徴量を用いた，機械学習による良否の判定を行う手法と，ソースコードの修正の指針を提示する手法を提案した．

評価実験として，否と判定されたソースコードを，提示されたソースコード特徴量を用いて修正し，改めて入力ソースコードとしたところ，良と判定される場合があった．したがって，ソースコード特徴量はソースコードの適切な編集への指針となることが判明し，ソースコードの良否の判定を利用することは，プログラミングの学習を支援することにつながると言える．

今後の研究課題として，判定の精度を向上させるために，新たにメトリクスや，コードクローンの数やタイプを特徴ベクトルとして追加することなども考えられる．また，上級者が提出したソースコードのうち，初級者に近いと判定され得るソースコードによる判定ミスをなくすため，テストデータにおいて異なる判定がされたソースコードの目的変数を動的に変更して学習を行うことが考えられる．さらに，現状においては，修正の提示は，対象となる予約語が利用されているかいないかという基準で行っている．全体の構造を加味し，利用されているものに対しても頻度に言及する指針を提示することができれば，より適切な編集を行うことができるのではないかと考えられる．

謝辞　本研究は JSPS 科研費 18H04094 の助成を受けた．

文　　献

[1] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer, "A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each," *Proc. of ICSE 2012*, pp. 3-13, 2012.

[2] Quinn Hanam, Fernando S. de M. Brito, and Ali Mesbah, "Discovering Bug Patterns in JavaScript," *ACM SIGCSE Bulletin*, pp. 144-156, 2016.

[3] Hao Zhong and Zhendong Su, "An Empirical Study on Real Bug Fixes," *Proc. of ICSE 2015*, pp. 913-923, 2015.

[4] 堤祥吾，プログラミングコンテスト初級者・上級者間におけるソースコード特徴量の比較，大阪大学大学院情報科学研究科修士論文， 2018.

[5] 堤祥吾，吉田則裕，崔恩瀞，井上克郎，"プログラミングコンテスト参加者を対象とした編集作業の特徴調査，" 情報処理学会論文誌， Vol.197， No.6， pp.1-8， 2017.

[6] Arpad E Elo, "The Rating of Chess Players, Past and Present," Arco Pub., 1978.

[7] Mikhail Mirzayanov, "Codeforces Rating System," Codeforces, `http://code forces.com/blog/entry/102`, 2010.

[8] Thomas J McCabe. "A complexity measure," *IEEE Trans. Softw. Eng.*, Vol. 2, No. 4, pp. 308-320, 1976.

[9] Vladimir I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady 10*, pp. 707-710, 1966.

[10] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus, "Fine-grained and accurate source code differencing," *Proc. of ASE 2014*, pp. 313-324, 2014.