実装者に依存しないプログラム規模の測定に向けて

Toward Programmer-Independent Program Size Measurement

門田暁人[*1]　内田眞司[*2]　松本健一[*3]

あらまし　多数の企業で用いられているソフトウェア規模の尺度であるソースコード行数（SLOC；Source Lines of Code）は，同一機能を実装した場合であっても，実装者によって値が大きくばらつくという問題がある．本論文では，この問題の軽減を目的として，同一の機能仕様に基づいて実装された 9 個のプログラムを分析した結果に基づいて，トークン数，クローン量，関数の仮引数総数を組み合わせることで, 実装者に依存しにくい規模の尺度 *Size* を導出した．この 9 個のプログラムには，SLOC に 3.16 倍の差があったが，*Size* の導出値の差は 1.22 倍となった．さらに，導出した尺度 *Size* を別の仕様の 6 個のプログラムに適用した結果，SLOC の 4.66 倍の差に対して，*Size* は 1.60 倍の差となった．

1　はじめに

ソフトウェア開発の黎明期から今日に至るまで，ソフトウェア規模の尺度として，ソースコード行数（SLOC；Source Lines of Code．コメント行や空行は含まない）が用いられてきた．さらには，SLOC をベースにして，不具合検出密度（件／SLOC），テストケース密度（件／SLOC），テスト工数密度（SLOC／人時），開発生産性（SLOC／人時）など開発プロセスに関する様々な尺度が定義され，プロジェクト管理に用いられている．

ところが，誰もが気付いているように，SLOC には大きな問題がある．同一機能を実装した場合であっても，実装者の能力やプログラミング技術によって，値が大きくばらつくことである．筆者らの経験では，同一機能の実装であっても，実装者によって 2 倍以上の SLOC の差はごく普通に生じる．にも関わらず，多くの企業では，他に適当な尺度がないために SLOC に頼らざるを得ないのが現状であり，例えば，不具合検出密度（件／SLOC）に基準値を設けて，品質保証活動を行ったり，SLOC ベースで規模見積もりを行っている．その挙句に，基準値を達成しているのに品質が悪いと嘆いたり，規模見積もりが外れて赤字が発生したと嘆いている．SLOC という不完全な尺度を用いていることに，そもそもの問題があると考えられる．

SLOC に代わる尺度としては，ソフトウェア機能量の尺度として，ファンクション

*1 Akito Monden, 奈良先端科学技術大学院大学 情報科学研究科
*2 Shinji Uchida, 奈良工業高等専門学校 情報工学科
*3 Ken-ichi Matsumoto, 奈良先端科学技術大学院大学 情報科学研究科

ポイントが用いられるようになっており，一定の成功を収めている．ただし，計測にコストを要するために，採用している企業は必ずしも多くない．また，採用している企業においても，やはり計測コストのために，その適用は開発初期の規模見積もりに限られる．一方，仕様追加や変更が起こりがちな今日の開発では，実装中もしくは実装後のソフトウェア規模を把握し，プロジェクト管理や改善を行うことが求められる．そのためは，SLOC のように，低コストで計測できる規模尺度が必要である．

本論文では，実装されたソフトウェア（ソースコード）の規模を測るという原点に立ち戻って，SLOC が実装者に依存するという問題の軽減を目指す．

2 基本アイデア

2.1 規模尺度に求められる条件

プログラムを P，P の仕様を S，実装者を H とする．S に基づいて H が実装したプログラムを $P(S, H)$と表す．また P の規模を $Size(P)$とする．本論文の目的は，

[条件 1] 任意の S および H_i, H_j $(i \neq j)$について，$Size(P(S, H_i)) \fallingdotseq Size(P(S, H_j))$

を満たす規模尺度 $Size$ を求めることである．つまり，同一の仕様が与えられた場合，異なる実装者が作成した 2 つのプログラムは，ほぼ同規模となることが求められる．従来の規模尺度である SLOC は，明らかに条件 1 を満たさない．

ただし，常に定数を返す関数も条件 1 を満たしてしまう．そこで，尺度 $Size$ は，

[条件 2] 異なる仕様を持つ任意の P，Q について，$Size(P\|Q) \fallingdotseq \mathrm{Size}(P) + \mathrm{Size}(Q)$
ただし， $P\|Q$ は P の後ろに Q をつなげたプログラムを表す．

も満たすものとする．条件 2 は，プログラム P に機能追加して $P\|Q$ というプログラムができた場合に，その $Size$ の値は P と Q の各 Size の和になることを意味する．さらに，実用面を考慮し，次の条件を設ける．

[条件 3] 任意の S，H_i，H_j について， $Size(P) \fallingdotseq \mathrm{E}(SLOC(P))$
ただし，$SLOC(P)$は P の SLOC を表し，$\mathrm{E}(x)$は x の期待値を表す．

条件 3 は，尺度 Size が，SLOC の期待値（可能な実装の平均値）とほぼ同じ値を取るとするものである．条件 3 を満たすことで，SLOC に基づく従来のメトリクスの基準値を，そのまま Size の基準値としても用いることができる．例えば，生産性の基準値を，社内で「10 SLOC／人時」としていた場合，「10 Size／人時」に代替できる．

なお，本論文における仕様 S は機能設計書レベルの仕様であり，プログラムの入出力と動作を規定する．アルゴリズムやデータ構造については規定しないが，プログラムの入出力に影響を与える基本的なアルゴリズム（例えば，探索は，幅優先なのか深さ優先なのか）は S で規定する．また，MVC 等のアーキテクチャは S で規定するが，モジュール構造やクラス構造については規定しない．

2.2 規模尺度に関する仮説

実装者に依存しない規模尺度を考えるにあたって，まず，SLOC のばらつきに影響する要因を整理し，それら要因の計測方法を検討する．SLOC がばらつく要因としては，次のものが考えられる．

要因A) プログラムの表層的特徴[10]

コーディングスタイルの違いによって，改行の数が異なり，SLOC に影響を与える．改行を多用するコーディングスタイルでは SLOC が増大する．

要因B) モジュールの粒度

モジュールの定義や呼び出しには一定の SLOC を要するため，モジュールの粒度が細かいほど SLOC が増大する傾向がある．一般に，仕様で与えられた機能をどこまで細分化してモジュール（関数，クラス，メソッドなど）として実装するかについては，実装者に任せられることが少なくない．

要因C) 冗長な実装

コーディング技術が未熟なためにコピー&ペーストを多用する場合や，類似の機能が多数のモジュールにまたがって存在する場合には，冗長なコードが多くなり，SLOC が必要以上に増大する．

要因D) アルゴリズムやコーディング Tips

実装者が採用するアルゴリズムやコーディング Tips（プログラミング作法[9]やプログラムを短く書く技法[11]）の違いによって，SLOC に差が表れる．適切なアルゴリズムや Tips を使った場合には, プログラムをコンパクトに記述でき, SLOC が小さくなる．

要因E) ライブラリ関数

実装者がライブラリ関数・クラスに熟知していないために，本来実装が不要な機能を実装し，SLOC が増大する場合がある．

以上の要因による影響を何らかのメトリクスにより計測できれば，実装者によるばらつきを補正することが可能となり，実装者に依存しない規模の尺度を得ることができると期待される．各要因を計測する方法を，仮説 A～E として下記に述べる．

仮説A) プログラム行数ではなくトークン数を計測することで，コーディングスタイルの違いを吸収できる．

仮説B) プログラム中で定義されている関数，クラス，メソッド，および，それらに渡される引数の数などを計測することでモジュールの粒度の違いを評価できる．

仮説C) コードクローン計測[8]によって，冗長なコードの量を評価できる．

仮説D) トークンの種類数によって，高度なアルゴリズムや Tips を採用しているか否かを評価できる．高度なアルゴリズムや Tips を採用しない場合，Halsted のソフトウェアサイエンス尺度[6]における Vocabulary（語彙）が少なくなり，トークンの種類数が増大しない可能性がある．

仮説E) プログラム中で呼び出されているライブラリ関数の種類数を計測することで，ライブラリ関数の活用度合いを評価できる．

以上の仮説に基づいて，本論文では，トークン数，トークン種類数，関数の数，コ

ードクローン含有量などのメトリクス計測を行うことで，実装者に依存しないソースコード規模の測定値を得ることを目指す．

3　規模尺度 *Size* の導出

3.1 概要

本章では，ある特定の仕様に基づいて 9 人の実装者の作成した C 言語プログラムのメトリクスを計測し，仮説 A～E の検証を行うとともに，距離の尺度の導出を試みる．プログラムの仕様，実装者，およびメトリクス計測ツールは，次のとおりである．

- 仕様
  3×3 盤面の 8 パズルを幅優先探索によって解くプログラム(8 パズルプログラム)である．標準入力から初期の盤面（数値の並び）を読み取り，解を求め，初期状態から最終状態まで各盤面を標準出力に表示する．幅優先探索では，木の深い部分を探索するほどノード数が指数的に増加するため，探索を効率化する必要がある．そこで，あるノードを調べるときに，そのノードに対応する盤の状況がすでに調べたノードのそれと同じである場合には探索を打ち切る．また，ある深さのノードの中で調べるべきノードの数が 10,000 を超えた場合には警告メッセージを出力して停止する．
- 実装者
  奈良先端科学技術大学院大学 情報科学研究科の 9 名の学生である．
- メトリクス計測ツール
  SLOC の計測には，Resource Standard Metrics[12]を使用した．コードクローンの計測には CCFinderX[1]を用い, 10 トークン以上重複するコード列をクローンとして検出した．トークン数およびトークン種類数の計測には，参考文献[4]のツールを用いた．

3.2 仮説の検証

実装された 9 個の 8 パズルプログラムの特徴を表 1 に，ソースコードメトリクスを計測した結果を表 2 に示す．表 1 の列「SLOC」は，空行やコメント文だけの行を除いたソースコード行数を表す．列「主なコーディング Tips」は，プログラム中で使用されている主なコーディング Tips，アルゴリズム，データ構造を表す．列「改行」は，改行を多用するコーディングスタイルの有無を表す．列「未実装の機能」は，実装されていない機能仕様の有無を表す．なお，未実装の機能があった場合，その分量は（筆者の主観的には）仕様全体の 1 割程度であった．列「類似」は，類似するプログラムを示す．以降では，2.2 節で述べた仮説 A)～E)の検証を行う．

仮説A) プログラム(c)と(e)は非常に類似した（大部分がコピーであると思われる）実装であったが，(e)は仕様の一部を未実装であった．従って，コーディングスタイルが同じであれば，(c)と比べて(e)の SLOC は 1 割程度小さくなるはずであるが，実際には SLOC はほぼ同じ値を取っていた．その原因は，(e)には改行を多用するコーディングスタイルが採用されていたことにある．表 2 のトークン数を見ると，

表 1.　実装された 8 パズルプログラムの特徴

P	SLOC	主なコーディング Tips	改行	未実装の機能	類似
(a)	165	キュー，隣接リスト，ハッシュ			
(b)	203	キュー，ハッシュ			
(c)	119	キュー，隣接リスト，ハッシュ			(e)
(d)	329	3 次元配列	○		
(e)	116	キュー，隣接リスト，ハッシュ	○	○	(c)
(f)	159	キュー，隣接リスト，ハッシュ	○	○	
(g)	135	キュー，隣接リスト，ハッシュ			
(h)	366	キュー，ハッシュ			
(i)	168	キュー，隣接リスト，ハッシュ			

(c)が 883, (e)が 782 となっており, (e)における未実装の機能量を反映できている．このことは，仮説 A)を支持している．

仮説B)　ここでは「キューとハッシュを用いているが隣接リストを用いていない」という共通した性質を持つプログラム (b)と(h)に着目する．SLOC は(b)が 203，(h)が 366 と約 1.5 倍の差がある．トークン数でみるとこの差はさらに広がり(b)は 1406 で(h)は 3342 と 2 倍以上の差となる．(b)と(h)の大きな違いは，(b)の関数の個数が 8 であるのに対し，(h)が 22 という点である．また，仮引数の総数は(b)は 7 であるのに対し(h)は 25 である. このことは, 関数や仮引数の総数が多いと SLOC が大きくなるという仮説 B)を支持している．

仮説C)　クローン含有率の最も大きいプログラムは(d)であり，実にコードの 73%がクローンである．それに伴い，SLOC も 329 と大きくなっている．このことは，仮説 C)を支持しているといえる. プログラム(d)よりも SLOC が大きいものとして(h)があり，クローン含有率は 38.4%にとどまるが，(h)については仮説 B)で述べたよ

表 2.　実装された 8 パズルプログラムのメトリクス値

P	SLOC	関数の数	仮引数の総数	トークン数	トークン種類数	クローン含有率
(a)	165	9	7	991	114	0.058
(b)	203	8	7	1406	154	0.250
(c)	119	4	2	883	100	0.174
(d)	329	1	0	3342	107	0.730
(e)	116	5	2	782	101	0.050
(f)	159	4	1	1090	109	0.287
(g)	135	4	2	965	119	0.135
(h)	366	22	25	2522	209	0.384
(i)	168	8	8	1274	138	0.200

うに，関数や仮引数の総数の多さが SLOC の増大に影響したと考えられる．

仮説D) トークンの種類数が多いプログラムは(b)と(h)であったが，いずれも隣接リストを用いていない．この結果より，「適切なアルゴリズムや Tips を採用しない場合，語彙が乏しくなりトークン種類数が減る」という仮説 D)は棄却された．ただし，(b)，(h)に(d)を加えた 3 つのプログラムに着目すると，いずれも隣接リストやハッシュといったプログラムをコンパクトに書くための技術が使われていないために，冗長なコードが多く，クローン含有率が高くなっている．このことから，トークン種類数ではなくクローン含有率を計測することで，アルゴリズムや Tips の違いによる SLOC の増減を評価できる可能性がある．

仮説E) 9 個のプログラムはいずれも標準ライブラリ関数を適切に用いていたために，仮説 E)の真偽を判断することはできなかった．

以上のことから，仮説 A)～C)については，仮説が検証されたとまでは言えないものの，仮説を支持する結果が得られており，仮説に明らかに反する事実も見られなかった．また，仮説 D)は棄却されたが，トークン種類数の代わりにクローン含有率を用いることが有望であることが分かった．仮説 E)については，本論文で用いたプログラムには該当がなく，以降では考慮の対象外とする．

3.3 規模尺度 *Size* の定義と導出

3.2 節の結果から，実装者に依存しない規模の尺度を求めるにあたって，トークン数，クローン含有率，関数数，仮引数の総数といったメトリクスを組み合わせて用いることが有望であることが確認できた．メトリクスの組み合わせ方には様々な方法が考え

$$\hat{Size} = k_1 N_1 + k_2 N_2 + \cdots + k_n N_n + C \qquad \cdots\cdots (1)$$

$\hat{Size}$ ：目的変数（*Size*）の推定値
N_j ：説明変数（トークン数，関数数，パラメータ数…）
k_j ：偏回帰係数
C ：定数項

られるが，本論文では，最も単純な方法として，式(1)のようにメトリクスの線形結合により規模尺度 *Size* を定義する．

式(1)のパラメータ（係数 k_j および定数 C）推定には，フィットデータが必要であり，また，フィットデータにおいて，2.1 節の条件 1～3 を満たすように教師信号，すなわち，目的変数 *Size* の値を与える必要がある．ここでは，フィットデータとして，実験で用いた 9 個のプログラム（P_a,...P_i と記す）に加えて，それぞれを 2 倍した（すなわち，2 個連結した）プログラム $P_a||P_a$，$P_b||P_b$，$P_c||P_c$，…を用意した．条件 2 を満たすためには，本来，異なる仕様を持つプログラムを連結したものを用意すべきであるが，P_a～P_i は同一仕様であるため，ここでは，連結したプログラム $P||P$ の前半部分と後半部分の仕様が異なりコードの重複がない（前半部分と後半部分にまたがるコードクローンが検出されない）ものと仮定してコードクローンの計測を行った．この仮定のも

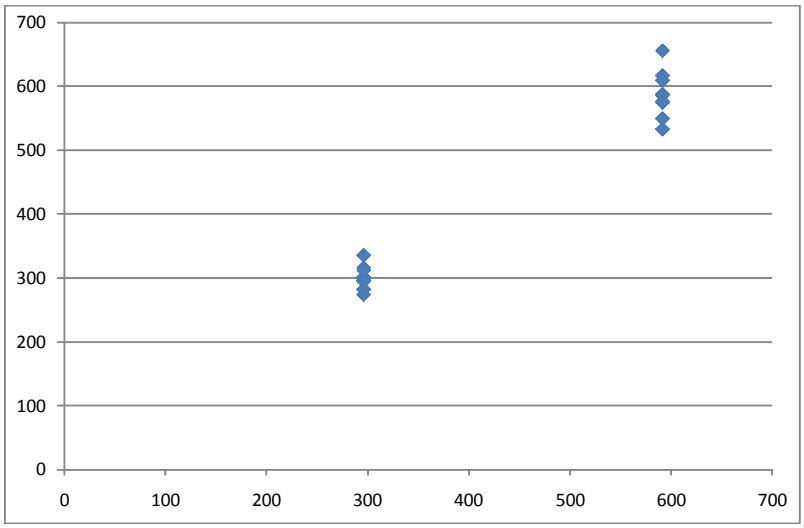

図 1.　*Size* の理想値―導出値の散布図

とでは，*Size*(*P*||*P*) ≒ 2×Size(*P*)となることが期待される．また，条件 1～3 を満たすために，各プログラムの連結数を *k*，P_a,...P_i の SLOC の平均値を E(SLOC(P_a,...,P_i))とすると，各プログラムについて *Size* = *k*×E(SLOC(P_a,...,P_i))を教師信号として与えた．

式(1)に取り入れる説明変数を決定するにあたっては，式(1)では各説明変数が目的変数に対して（乗法的ではなく）加法的に作用することを考慮し，「クローン含有率」という比率の尺度ではなく，正規化前の「クローンの量（トークン数）」を用いた．また，多重共線性を避けるために，各説明変数間の相関係数を調査した結果，関数の数と仮引数の総数の相関係数が 0.984 と極めて高かったため，関数の数を説明変数から除外した．

重回帰分析によりパラメータ推定した結果，得られた回帰式を表 3 に示す．各偏回帰係数および定数項の有意性検定を行った結果，有意水準 1%で全ての係数および定数の有意性を確認できた．この回帰式が，*Size* を表す尺度となる．

図 1 は，横軸に *Size* の理想値（教師信号），縦軸に回帰式による *Size* の導出値を取ったグラフである．図より分かるように，*Size* は（今回実験で用いた仕様に関しては）条件 1～3 をほぼ満たしているといえる．

表 4 に，SLOC と *Size* の理想値，導出値の比較結果を示す．SLOC は最小値 116，最大値 366 であり 3.16 倍の開きがあったが，*Size* の導出値は最小 274，最大 335 であり

表 3.　得られた回帰式

説明変数	偏回帰係数	標準化偏回帰係数
トークン数	0.393	3.98
仮引数の総数	-12.2	-0.978
クローンの量	-0.422	-3.36
（定数項）	15.5	

表 4.　8 パズルプログラムの SLOC と *Size* の比較

P	SLOC	Size の理想値	*Size* の導出値
(a)	165	296	296
(b)	203	296	335
(c)	119	296	274
(d)	329	296	301
(e)	116	296	282
(f)	159	296	300
(g)	135	296	316
(h)	366	296	294
(i)	168	296	312

1.22 倍の開きに収まった.

4　規模尺度の評価

3.3 節で得られた *Size* の導出式の一般性の評価を目的として, 別の仕様に基づいて 6 名の実装者の作成した C 言語プログラムの規模を計測する．プログラムの仕様は，与えられたテキスト入力をハフマン符号により符号化するものである．6 名の実装者の内訳は, 奈良先端科学技術大学院大学 情報科学研究科の 1 名の大学教員と 5 名の大学院生（いずれもソフトウェア開発経験のある社会人学生）である．

6 個のプログラムのメトリクス，及び，*Size* の導出値を表 5 に示す．表より分かるように，SLOC は最小 107，最大 499 であり 4.66 倍の開きがあったが，Size の導出値は最小 165，最大 264 であり 1.60 倍の開きに収まった．このことから，3.3 節で得られた *Size* の導出式は，異なる仕様のプログラムに対しても実装者による規模のばらつきを抑える効果があり，ある程度の一般性を持っていることが期待される．

5　関連研究

産業界では，できるだけ属人性を除いたソースコード規模を得るために，ソースコード成形ツール（formatter, beautifier, pretty printer などと呼ばれる）を用いてから SLOC を計測することが行われている．本論文では，SLOC ではなくトークン数を計測する

表 5.　ハフマン符号プログラムのメトリクス値と *Size* の導出値

P	SLOC	仮引数の総数	トークン数	クローンの量	Size の導出値
(s)	334	7	2135	1433	165
(t)	153	5	839	204	198
(u)	499	0	2411	1656	264
(v)	107	0	625	132	205
(w)	130	3	730	93	226
(x)	113	1	786	289	190

ことで同様の効果を得ており，また，コードクローンやモジュールの粒度といった他の要因も考慮している点が異なる．

プログラムから属人性を除去することを目的として，ソースコードのバリエーションを除去する方法[7]が提案されており，実装のばらつきの軽減に役立つと考えられる．ただし，方法[7]は，局所的な計算の順序の違いを除去できるが，より大域的なアルゴリズムの違いは除去できず，コードクローンやモジュールの粒度の違いも除去できない．そのため，規模の尺度を得るという本論文の目的には，効果が小さいと考えられる．

Halstead[6]は，語彙の量を考慮したソースコード規模の尺度 Volume を提案している．Volume は，全トークン N のうち n 個がユニークであるようなプログラムを記述するのに必要なビット数を表し，(トークン数 N)×$\log_2$(トークン種類数 n)で与えられる．この尺度を表 1 の 9 個のプログラムに適用した結果，最小値 5207，最大値 22530 となり，その差は 4.33 倍であった．従って，Volume は実装者に依存しない規模の尺度とはなり得ないと言える．

ソースコードからファンクションポイントを計測する試みも行われている[1]．この試みが成功すれば，本論文の目的とする規模尺度が得られると期待される．ただし，現状では，トランザクションファンクションにおける外部出力と外部照会の区別の難しいことや，自動計測が難しいといった課題が存在するため，実用化に向けての今後の発展が望まれる．

6 まとめ

本論文では，SLOC が実装者に依存するという問題の軽減を目的として，まず，規模の尺度に求められる 3 つの条件を定義した．次に，SLOC が実装者に依存する要因を整理し，各要因を定量的に計測するための 5 つの仮説 A)～E)を設けた．

同一の機能仕様に基づいて作成された 9 個のプログラムを用いて各仮説の検証を行った結果，トークン数，クローン量，関数の数，関数の仮引数の総数を組み合わせて用いることが，規模尺度の導出に有望であることが分かった．そこで，これらのメトリクスの線形結合として規模尺度 Size を定義し，回帰式のパラメータ推定を行った．9 個のプログラムには，SLOC に 3.16 倍の差があったが，*Size* の導出値の差は 1.22 倍となった．

さらに，導出した規模尺度 *Size* の一般性を評価するために，別の仕様に基づいて作成された 6 個のプログラムを計測した結果，SLOC は 4.66 倍の差があったが，Size は 1.60 倍の差に収まった．このことから，尺度 *Size* は，異なる仕様のプログラムに対しても実装者による規模のばらつきを抑える効果があり，ある程度の一般性を持っていることが期待される．

得られた規模尺度 *Size* は，トークン数，クローン量，関数の仮引数の総数という，自動計測可能なメトリクスから導出できる．また，2.2 節の条件 3 により，SLOC の代替として用いることが容易であり，今後，産業界での利用が期待される．

今後の課題として，より多くの仕様，及び，実装者の作成したプログラム群を用い

て，より一般性のある尺度 Size の導出式を求めるとともに，その評価を行うことが重要となる．また，ソフトウェアメトリクスのセオリー[2][5][13]に基づいて，条件 1～3 以外に尺度 *Size* が満たすべき性質について，検討していくことが重要となる．

謝辞

本研究の一部は，文部科学省「次世代ＩＴ基盤構築のための研究開発」の委託に基づいて行われた．また，本研究の一部は，文部科学省科学研究費補助金（基盤研究(C)：課題番号 22500028）に基づいて行われた．

参考文献

[1] 赤池輝彦，楠本真二，英繁雄，芝元俊久，″ソースコードからのファンクションポイント計測とその適用″，信学技報，No.SS2007-54，pp.97-102，2007.
[2] Briand, L.C., Morasca, S., and Basili, V.R., ″Property-Based Software Engineering Measurement,″ IEEE Trans. Software Engineering, Vol.22, No.1, pp.68-86, Jan. 1996.
[3] CCFinderX, http://www.ccfinder.net/
[4] C/C++言語のトークンを抽出，http://www.vector.co.jp/soft/winnt/prog/se482039.html
[5] Fenton, N.E., ″Software Measurement: A Necessary Scientific Basis,″ IEEE Trans. Software Engineering, Vol.20, No.3, pp.199-206, 1994.
[6] Halstead, M.H., ″Elements of Software Science (Operating and programming systems series)″, Elsevier Science Inc., New York, 1977.
[7] 服部徳秀，石井直宏，″ソースコードのバリエーション除去システム，電子情報通信学会論文誌,″ Vol.J80-D-I, No.1, pp.50-59, 1997.
[8] Kamiya, T., Kusumoto, S., and Inoue, K., ″CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code,″ IEEE Trans. Software Engineering, vol. 28, no. 7, pp. 654-670, 2002.
[9] Kernighan, B., Pike, R., ″プログラミング作法,″ アスキー，2000.
[10] 永井洋一，シムウォンボ，三輪誠，近山隆，″機械学習を用いたプログラムの表層的特徴による分類,″ 第 9 回プログラミングおよびシステムに関するワークショップ，2006.
[11] Ozy, ″Short Coding～達人達の技法,″ 毎日コミュニケーションズ，2007.
[12] Resource Standard Metrics, http://msquaredtechnologies.com/m2rsm/
[13] Weyuker, E. ″Evaluating software complexity measures,″ IEEE Transactions on Software Engineering, Vol.14, No.9, pp.1357-1365, 1988.