Sapidによるソフトウエア解析技法 II

―ソフトウエア・メトリックスの作成―

齋　藤　邦　彦

はじめに

Sapid（www.Sapid.org/）は下流レベルのソフトウェア解析を効率的に実現するソフトウエア工学ツール・プラットフォームであり，ソフトウエアの生産性や品質の向上を目的としたプログラムを簡単に作成することができる。本論文では，ソフトウエア品質管理・評価の手法のひとつであるソフトウエア・メトリックスを求めるプログラムを開発する。

ソフトウエア・メトリックスは，ソフトウエア品質管理，品質評価の手法として確立され，開発現場でも導入されている。しかし，実際にISOやJIS等で提案されたメトリックスに基づいたソフトウエア評価を行うためには，構文解析等の手法を用いて，メトリックス評価プログラムを作成する必要がある。構文解析系も含んだプログラムを作成することは容易ではなく，開発に多くの時間が必要とされる。

本論文では，Sapidを利用して各種のメトリックスを求めるプログラムを作成する。オープンソースとして公開されている，多くのソフトウエアのメトリックスを解析し，結果を統計的に処理して，メトリックスに対する評価や比較の指標となるデータを提供する。オープンソースの解析のために，オープンソース・ソフトウエアの細粒度リポジトリ・データベースを提供するアーカイバ・サイトSappy（www.aichi-pu.ac.jp/sappy他）のデータを利用する。

第１章で関数とパラメータの名前，型を求めるプログラムを作成し，第２章第３章で関数のメトリックス情報を求めるプログラムを作成する。第４章でその他のメトリックス情報を求め，第５章でメトリックスに関するデータの活用法を提案する。

I　関数とパラメータの名前，型を求める　－オブジェクト配列の利用

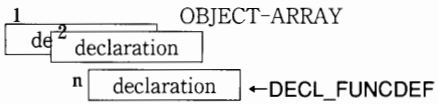
「Sapidによるソフトウエア解析技法 I」[1)]では，関数宣言，関数オプショナル宣言と言ったクラスのID番号を求める方法として，トップレベル（プログラムID）から関連をたどる方法（spdGetRelObj 関数群），オブジェクトの包含関係を利用する方法（spdGetIncludedOcc 関数群）を解説した。

ここでは，オブジェクト配列（型：SpdObjIdArray）を用いて該当するオブジェクトを求める方法を用いる。関数 spdGetObjIdArray によりオブジェクト配列を求め，繰り返し制御（for文など）を用いて順番にオブジェクトIDを求める。i 番目のオブジェクトIDは，オブジェクト配列変数の構造体メンバ変数id[i]に格納されている（例：func_array.id[i]）。

1．例題1　関数宣言オブジェクト一覧をもとめる

関数宣言オブジェクト一覧を，オブジェクト配列（クラス名：declaration, ソートDECL_FUNCDEF）として取得する。関数 spdGetObjIdArray に求めるクラス名（declaration）をパラメータとしてオブジェクト配列を求める。for文で順番にオブジェクトIDを取り出し，spdGetAttrValInt 関数でsortが関数宣言（DECL_FUNCDEF）のものを取り出す。

①オブジェクトIDをオブジェクト配列から求める　－ spdGetObjIdArray



②　関数プロトタイプ

オブジェクト配列←spdGetObjIdArray（クラス名,	名前,	条件）	
SpdObjIdArray	SpdString†	SpdString†	SpdCondObjIdFunc

③　関数プロトタイプ

```
func_array = spdGetObjIdArray("declaration", NULL, NULL);
```

†SpdString は char* でもよい。

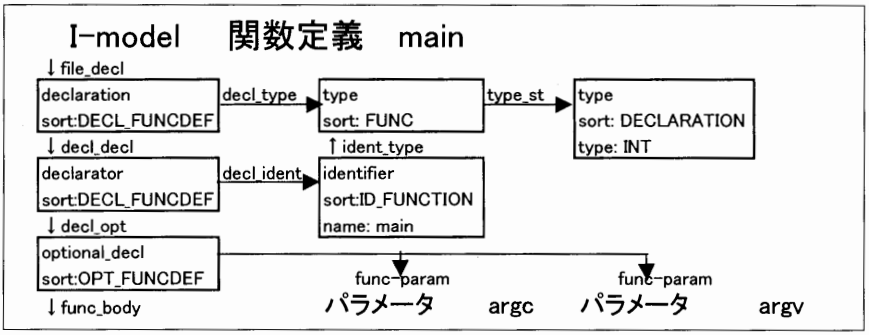
④ サンプルプログラムⅠ　ー関数名と型を求める

```
↓オブジェクト配列の宣言
 SpdObjIdArray func_array;
int l;
↓オブジェクト配列の取得
func_array=spdGetObjIdArray("declaration", NULL, NULL);
        for (l = 0; l < func_array.size; l++)
        if (spdGetAttrValInt(func_array.id[l], "sort") == DECL_FUNCDEF)
                              ↑これが関数宣言のオブジェクトIDとなる
                decr_id = spdGetARelObj(func_array.id[l], "decl_decl","any");
                                        |
```

2．関数の名前と型を求める

関数宣言のオブジェクトIDから，順番に関数，パラメータの名前と型を求める。関数の型のクラス type は関数の宣言 declaration （属性：関数）から関連 decl_type（declaration-type）をたどって求める。またクラス identifier （名前）から関連 ident_type（identifier-type）で求めることもできる（図Ⅰ参照)。

パラメータはクラス optionaldecl （オプショナル宣言）から，関連 func_parm（optionaldecl-declaration）をたどり，パラメータの宣言 declaration （属性：変数）を順番に求める。そこから名前（クラス identifier，関連：decl_ident）と型（クラス： type，関連：decl_type）を求める。



図Ⅰ　I-model定義　関数main.c

サンプルプログラムⅠ（続き）　－関数名と型を求める

```
        ident_id  =   spdGetARelObj(decr_id, "decl_ident","any");
        func_name =   spdGetName(ident_id);
        type_id =     spdGetARelObj(ident_id, "ident_type","any");

        printf("FUNC_NAME= %-30s  FUNC_TYPE = ", func_name);
        getTypeName(type_id);
```

3．型の求め方

型の宣言には，int（整数型），char（文字型）といった属性のほかに，const，volatileといった修飾子，longなどのサイズ指定子，signed，unsignedといったsign指定子などがある。また，ポインター型，配列，構造体，共用体，enum型といった複雑な型，ユーザ定義の型もある。Sapidはすべての型の宣言に関する情報を保持している。本プログラムでは整数，文字型とポインター型，配列型のみを扱う。

クラス型 type の属性sortが TYPE_DECLARATION のとき，関数 getTypeName Basic により，型の名前を標準出力に出力する。関数 getTypeName は，クラス型typeの属性 sort を調べ，TYPE_POINTER（ポインター型）の場合はその情報を出力する。関連 type_st で続く type オブジェクトを求め，関数 getTypeName を再帰的に呼び出す。TYPE_ARRAY （配列型）のときも同様である。TYPE_DECLARATIONのときは関数 GetTypeNameBasic を呼び出す。

関数 GetTypeNameBasic：型を求める関数（基本型）

```
char * getTypeNameBasic(SpdObjId type_id){
    if (spdGetAttrValInt(type_id, "sort")= TYPE_DECLARATION)
        switch(spdGetAttrValInt(type_id, "type")){
                    case TYPE_INT:
                      return "INT";break;
                    case TYPE_VOID:
                      return "VOID";break;
                    case TYPE_CHAR:
                      return "CHAR";break;
                    case TYPE_DOUBLE:
                      return "DOUBLE";break;
                    case TYPE_FLOAT:
                      return "FLOAT";break;
                    case TYPE_DOTS:
                      return "...";break;
                    case TYPE_UNKNOWN:
                      return "UNKNOWN";break;
     default:
                    return -1;
  }
}
```

関数 GetTypeNameBasic：型を求める関数（配列，ポインタの場合）

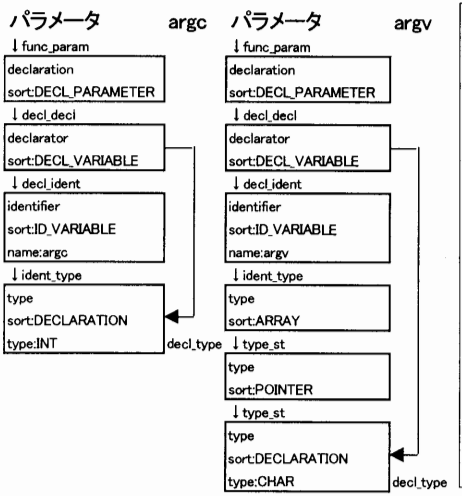
```
void getTypeName(SpdObjId type_id){
SpdObjId     type_id1;
 if ((type_id1=spdGetARelObj(type_id, "type_st","any"))!=SAPID_NON_ID)
   switch(spdGetAttrValInt(type_id, "sort")){
                  case TYPE_POINTER:
                printf(" POINTER OF ");
                getTypeName(type_id1);break;
                  case TYPE_ARRAY:
                printf(" ARRAY OF ");
                getTypeName(type_id1);break;
                  case TYPE_DECLARATION:
                printf(" %s ",getTypeNameBasic(type_id1));
                  break;
               default:
                  break;
  }
}
```

サンプルプログラムI（パラメータ情報取得まで）

```
int main(int argc, char *argv[])
{
    char          *target_program;
    SpdObjId      prog_id, decr_id, opt_id, param_id, blk_id, ident_id;
    SpdObjIdArray func_array;
    SpdOcc        opt_occ;
    SpdCursor     *opt_csr;
    int l;

    target_program = smuGetAProgramName();
    if ((prog_id = spdTarget("I-model", target_program)) == SAPID_NON_ID) {
        spdErrorExit("Program ¥"%s¥" not found.", target_program);
    }

    func_array=spdGetObjIdArray("declaration", NULL, NULL);
         for (l = 0; l < func_array.size; l++)
         if (spdGetAttrValInt(func_array.id[l], "sort") == DECL_FUNCDEF)
                  decr_id = spdGetARelObj(func_array.id[l], "decl_decl","any");
                  ident_id  =    spdGetARelObj(decr_id, "decl_ident","any");
                  func_name =   spdGetName(ident_id);
                  type_id =      spdGetARelObj(ident_id, "ident_type","any");
                              /***名前と型を出力する***/
                  printf("FUNC_NAME= %-30s   FUNC_TYPE = ", func_name);
                  getTypeName(type_id);

                  opt_id = spdGetARelObj(decr_id , "decl_opt", "any");
                  if(opt_id != SAPID_NON_ID)
                           {param_id = spdGetARelObj(opt_id ,"func_parm","any");
                            blk_id    = spdGetARelObj(opt_id ,"func_body","any");
```

4．パラメータ宣言の求め方

パラメータ部分の宣言は optionaldecl （オプショナル宣言）から，関連 func_parm （ optionaldecl-declaration ）をたどり，パラメータの宣言 declaration （属性：変数）を spdGetRelObjInit 関数群によって順番に求める。そこから名前（クラス：identifier，関連：decl_ident ）と型（クラス：type，関連：decl_type ）を求める。



int main(int argc, char * argv[])

［解説］

オプショナル宣言（optionaldecl）から

↓func_param：optionaldecl-declaration

↓decl_decl：declaration-declarator

↓decl_ident：declarator-identifier

↓ident_type：identifier-type

と型（type）をたどる。ポインター型，配列型は関連 type_st で順番にたどる。ポインター，配列は属性sortで指示される。

図Ⅱ　関数mainのパラメータのI-model

サンプルプログラムⅠ（パラメータ情報取得まで　続き）

```
param_id = spdGetARelObj(opt_id ,"func_parm","any");
param_csr = spdGetRelObjInit(param_id, "decl_decl", "any");
  while ((para_id = spdGetRelObj(param_csr)) != SAPID_NON_ID) {
          paraident_id = spdGetARelObj(para_id, "decl_ident","any");
          param_name =  spdGetName(paraident_id);
          paratype_id = spdGetARelObj(paraident_id, "ident_type","any");
          printf("PARAM_NAME= %-30s PARAM_TYPE=", param_name);
          getTypeName(paratype_id);
          printf("¥n");
        }
```

Ⅱ　関数呼び出しを求める　－基本メトリックス

ソースコードの手続き的メトリックスを，プログラム・メトリックスと関数メトリックスに分ける。プログラム・メトリックスは関数メトリックスのデータを集計して求める。関数メトリックスの基本項目は，関数の引数の数，行数，コメント行数，コールする関数の情報などの項目からなる。関数が所属するプログラム，ファイル，関数とパラメータの名前，型は，「Sapidによるソフトウエア解析技法Ⅰ」[1)]と第１章の例題Ⅰから求めることができる。本章では関数呼び出しの回数，位置といった情報を求めるプログラムを作成する。

1．例題2　関数呼び出しの情報を求める

ある関数の中の関数呼び出し情報を求める手順を考える。関数呼び出しは，関数中のオブジェクト identifier （名前）の出現として求める。関数定義部の名前にアクセスしないために，I-modelの中で関数定義名（identifier）より下位にある関数のオプショナル宣言を基点として，identifier （名前）の出現（occurrence）を spdGetIncludedOcc 関数群で求める。順番にsortが関数であるものを選び，配列に格納する。同じ関数が複数回出現したときは，重複して処理を行なわない。

アルゴリズム

0　関数名を格納する配列func_idを用意する（準備）

1．クラス identifier（名前）のオブジェクト出現（occurrence）の取得

2．属性：sort（種類）が関数（DECL_FUNCDEF）であるものを選ぶ

3．関名を格納する配列を調べ，新規の名前かどうかを判断する

4．新規の関数であった場合は配列に格納し，情報を出力する

関数 getFuncCall　（関数呼び出しの情報を求める）

```
#define MAX_FUNC_SIZE 100
void getFuncCall(SpdObjId funcDecl_id)
{
   SpdOcc       func_occ;
   SpdCursor   *func_csr;
   SpdObjId     func_id[MAX_FUNC_SIZE],func_id1;
   int            i=0,j=0;

Visibleな関数のCallを求める手順
     Func_csr = spdGetIncludedOccInit(funcDecl_id, "identifier",funcDecl_id);
 名前の出現を求める
     while ((func_occ = spdGetIncludedOcc(func_csr)).relationId != SAPID_NON_ID)
        if (spdGetAttrValInt(func_occ.objectId, "sort") == ID_FUNCTION){
      関数の名前の出現だけ抜き出す
            while(j<=i)
             if(func_id[j++]==func_occ.objectId)
                 goto EXIT2;
      重複する関数名の出現があった場合は取り除く
            func_id[i]=func_occ.objectId;
            printf(" Calling Function Name %d:%s ¥n",i,spdGetName(func_id[i]));
            i++;
            EXIT2:
            j=0;
           }
    }
}
```

サンプルプログラムⅡ　（関数メトリックスを求める）

```
int main(int argc, char *argv[])
{
    char          *target_program;
    SpdObjId      prog_id, decr_id, opt_id, param_id, blk_id, ident_id;
    SpdObjIdArray func_array;
    SpdOcc        opt_occ;
    SpdCursor     *opt_csr;
    int l;

    target_program = smuGetAProgramName();
    if ((prog_id = spdTarget("I-model", target_program)) == SAPID_NON_ID) {
        spdErrorExit("Program ¥"%s¥" not found.", target_program);
    }

    func_array=spdGetObjIdArray("declaration", NULL, NULL);
         for (l = 0; l < func_array.size; l++)
          if (spdGetAttrValInt(func_array.id[l], "sort") == DECL_FUNCDEF)
                    decr_id = spdGetARelObj(func_array.id[l], "decl_decl","any");
                    opt_id = spdGetARelObj(decr_id , "decl_opt", "any");
                                                    |
    関数呼び出し情報を出力
                    getFuncCall(func_array.id[l]);
    関数被呼び出し情報を出力
                    getFuncCalled(func_array.id[l]);
    変数メトリックス情報を出力
                    getVarNum(prog_id);

                                                    |
```

関数 getFuncCall （改良版）

```
void getFuncCall(SpdObjId funcDecl_id){
   SpdOcc       func_occ;
   SpdCursor    *func_csr;
   int          j;

構造体の定義
   struct FUNC{
         int func_size;
         SpdObjId *func_id;
     };
   struct FUNC func;
構造体の初期化
   func.func_size = 0;
   func.func_id = NULL;

     func_csr = spdGetIncludedOccInit(funcDecl_id, "identifier",funcDecl_id);
     while ((func_occ = spdGetIncludedOcc(func_csr)).relationId != SAPID_NON_ID)
          if (spdGetAttrValInt(func_occ.objectId, "sort") == ID_FUNCTION){
       関数の名前の出現だけ抜き出す。
              for(j=0; j <func.func_size ;j++)
                 if(func.func_id[j]==func_occ.objectId)
                       goto EXIT2;
       構造体を配列のように使う方法（注参照）
              func.func_id=spdRealloc(func.func_id, sizeof(SpdObjId) * (func.func_size + 1));
              func.func_id[func.func_size] =   func_occ.objectId;
         結果の出力
              printf(" Calling Function Name %s ¥n",
                       spdGetName(func.func_id[func.func_size]));
              func.func_size ++;
              EXIT2:
             }
         }
}
```

注 func.func_id=(SpdObjId)malloc(sizeof(SpdObjId) * (func.func_size + 1));　でも可

関数 getFuncCall は，関数の呼び出し数を最大100として配列を宣言しており，対象とするソフトウエアの関数呼び出し数が100を超えると処理できなくなる。そのため，構造体を用いて，サイズに対応するようプログラムを改良する。

2．関数呼び出しの複雑さ

呼び出される関数も他の関数を呼ぶ。その関数呼び出し構造をツリーとして表現し，関数呼び出しの複雑さの情報を求める。関数の複雑さは，関数呼び出しの総数，呼び出しの最大ネスト数といった情報である。

各関数ごとに関数呼び出しの情報を配列（クラス optionaldecl ）に格納し，再帰的に getFuncCall を適用する。ただし同じ名前の関数が出現した場合の無限ループの可能性もあり，実用的なソフトウエアとするためには，これを考慮する必要がある。

関数 getFuncCall　（関数の複雑さを求める）

```
func_id[i]=func_occ.objectId;
printf(" Calling Function Name %d:%s ¥n",i,spdGetName(func_id[i]));

                              ******  追加部分  ******
オプショナル宣言オブジェクト一覧から、呼び出した関数と同じものを選ぶ
func_array=spdGetObjIdArray("optionaldecl", NULL, NULL);
        for (k = 0; k < func_array.size; k++)
        if (spdGetAttrValInt(func_array.id[k], "sort") == DECL_FUNCDEF){
      オプショナル宣言→宣言子→名前とたどる
          decr_id = spdGetARelObj(func_array.id[k], "decl_opt","any");
          ident_id = spdGetARelObj(decr_id, "decl_ident","any");
       関数と同じ名前（ID）かどうかを判断
          if(func.func_id[func.func_size] ==ident_id)
              再帰的呼び出しの場合を排除
                if(func_array.id[k] !=funcDecl_id){
              再帰的に関数呼び出し情報を出力
                    printf(" The %s is call the function¥n¥t" ,spdGetName(ident_id));
                    getFuncCall(func_array.id[k]);
                    printf("%s   func list end ¥n" ,spdGetName(ident_id));
                }
        }
```

3．関数の呼び出し位置

関数の呼び出し位置は，出現した関数名 identifier から，関連 ident_ref（名前－式：identifier－expression ）を求め， spdGetOffset 関数で，オフセットを求める。関数自身のオフセットを加えて，ファイルの先頭からの絶対位置を求める。最

終的にオフセットを行，桁数に変換して実際の位置を求める必要がある。これらの情報はSapidのユーティリティSPIEにより取得できる。

関数 getFuncCall　（関数の複雑さを求める　続き）

```
SpdRel  ident_rel; 追加部分(宣言)
                                                    |
 func_id[I]=func_occ.objectId;
                              /******  追加部分（手順）  ******/
 ident_rel = spdGetARel(func_id[I], "ident_ref","any");
 offset = spdGetOffset(ident_rel.relationId);
                  ↓関数オフセット値（グローバル変数）
 offset = offset + func_offset;
 printf("Calling Function Name %d:%s offset = ¥n",i,spdGetName(func_id[i]),offset);
                                                    |
```

グローバル変数を求める手順

```
extern func_offset =0;  追加部分(宣言)
int main(int argc, char *argv[]){
 SpdRel  ident_rel_func; 追加部分(宣言)
                                                    |
    func_array=spdGetObjIdArray("declaration", NULL, NULL);
         for (l = 0; l < func_array.size; l++)
         if (spdGetAttrValInt(func_array.id[l], "sort") == DECL_FUNCDEF)
                                  *****  追加部分  *****/
             ident_rel_func = spdGetARel(func_array.id[l], "file_decl","any");
             func_offset = spdGetOffset(ident_rel_func.relationId);
                                                    |
```

Ⅲ　関数の被呼び出し

第２章では，ある関数における関数呼び出し（call）に関する様々な情報を求めた。第３章では，関数が他の関数で呼ばれる場合（called）の情報を求める。プログラム内における全ての関数の中での出現を調べる必要があり，関数（クラス optionaldecl ）のオブジェクト配列の中から当該関数の名前（クラス：identifier 属性：name）の occurrence を持つものを選ぶ。同じ関数が複数回出現する場合は重複を取り除く。

アルゴリズム

１．クラス optionaldecl の配列オブジェクトの取得

２．optionaldecl から　identifier の名前が関数名と等しく属性 sort が関数であるものを選ぶ

3．当該関数の identifier を occurrence として持つものを選ぶ

4．新規の呼び出し元関数であった場合はエントリーする

関数 getFuncCalled　関数の被呼び出し状況を求める

```
void print_Calling_FuncName(SpdObjId decl_id){
    SpdObjId     decr_id, ident_id;
    decr_id    = spdGetARelObj(decl_id, "decl_decl","any");
    ident_id   = spdGetARelObj(decr_id, "decl_ident","any");
    printf("%-15s Called this Function ¥n",spdGetName(ident_id));
}

void getFuncCalled(SpdObjId called_func_id){

    SpdOcc       func_occ;
    SpdCursor   *func_csr;
    SpdObjIdArray func_array;
    SpdObjId     decr_id, ident_id, func_id[MAX_FUNC_SIZE];
    int          l;

    decr_id    = spdGetARelObj(decl_id, "decl_decl","any");
    ident_id   = spdGetARelObj(decr_id, "decl_ident","any");
    func_array=spdGetObjIdArray("declaration", NULL, NULL);

    for (l = 0; l < func_array.size; l++)
        if(spdGetAttrValInt(func_array.id[l], "sort") == DECL_FUNCDEF){
          func_csr=spdGetIncludedOccInit(func_array.id[l],"identifier",func_array.id[l]);
          while((func_occ = spdGetIncludedOcc(func_csr)).relationId != SAPID_NON_ID)
            if(func_occ.objectId == ident_id){
              if(func_array.id[l]!=called_func_id){
                print_Calling_FuncName(func_array.id[l]);
                getFuncCalled(func_array.id[l]);
              }
            }
        }
}
```

Ⅳ　代表的なソースコードメトリックス

その他のソースコードのメトリックスとして，内部変数と変数の総数の比，ループのネスト数，Cyclomatic数といった情報を求める。変数に関する情報は，関数のときと同じく，クラス identifier を用いて取得する。プログラムの構造に関する情報はクラス block に関する情報を利用する。

1．内部変数に関するメトリックス

変数の情報は変数名，型，有効範囲，出現位置，含まれる式などからなる。変数名，型は1章の関数のプログラムを改良して求める。内部，外部，静的といった変数の種類はSapidの基本情報（SDB）には含まれない。変数の宣言を

参照して，宣言に関する情報として調べる。変数の有効範囲や変数を含む式を取得するために，expression，block をといったクラスを参照する。

関数 getVarNum：変数のメトリックス情報を求める

```
#define MAX_VAR_SIZE 100
void getVarNum(prog_id){
   SpdObjIdArray id_array;
   SpdObjId      file_id, decl_id, decr_id, gvar_id[MAX_VAR_SIZE];
   SpdCursor    *decl_csr;
   int count=0, count2=0, i, j, gvar_num=0;
 グローバル変数のリスト（配列）を求める
   file_id = spdGetARelObj(prog_id,"prog_file","any" );
   decl_csr = spdGetRelObjInit(file_id,"file_decl","any");

   while((decl_id = spdGetRelObj(decl_csr))!= SAPID_NON_ID)
          {decr_id = spdGetARelObj(decl_id, "decl_decl","any");
                 if(spdGetAttrValInt(decl_id, "sort") == DECL_VARIABLE)
                 gvar_id[gvar_num++];
                }
 変数の出現を求め、ローカル、グローバルを判断する
   id_array = spdGetObjIdArray("identifier", NULL, NULL);
   for (i = 0; i < id_array.size; i++)
        if  (spdGetAttrValInt(id_array.id[i], "sort") == ID_VARIABLE){
                 count++;
                 for(j = 0; j< gvar_num; j++)
                       if (id_array.id[i]==gvar_id[j]){
                          count2++;goto EXIT;
                        }
                 EXIT:
                 }
    printf("Number of  Local Variables: %d  Local/Total: %-5.2f%% ¥n"
           ,count,(float)(count-count2)*100/count);
}
```

2 . Cyclomatic数

プログラムの複雑さを計測する指標として，ブロックのネストの数，ブロックの数などを用いる。Cyclomatic complexity数 はThomas McCabe により1976年に提案された，プログラムの健全性（soundness），信頼性（confidence）を計測する指標である。Cyclomatic complexity数は，プログラム・モジュールの複雑さを，有向グラフとして表現したときの線形独立なパスの合計であらわされる。

① Cyclomatic の複雑数

$CC = E - N + p$

E ＝ グラフのエッジ数

N ＝ グラフのノード数

p ＝ 要素数

② Cyclomatic の複雑さの基準（Thomas McCabeによる）

1-10	リスクのない平坦なプログラム
11-20	少し複雑，危険度小
21-50	複雑，危険度大
50以上	非常に危険

3．最大ネストと各ブロック

条件，繰り返しといった制御文や，中括弧{，}で囲まれたブロックに関する複雑さの情報を求める。Sapidでは，ブロックの種類を，内部にブロックを含むhierachicalブロックとbasicブロックに分ける。basic ブロックは内部にブロックを含まないブロックの最小単位である。

関数 get_depth とサンプルプログラム

```
                                        |
        トップレベルのblockのIDを求める
        blk_id    = spdGetARelObj(opt_id ,"func_body","any");
        関数のネストの深さを求める
        max_depth= get_depth(blk_id);
                                        |

int get_depth(SpdObjId block,int depth)
 {
    SpdObjId        blk_id,block_id;
    SpdCursor       *blk_csr;
    extern int       max;
    if  (spdGetAttrValInt(block, "sort") == BLOCK_BASIC)
                return depth;

    blk_csr = spdGetRelObjInit( block, "blk_blk", "any");
    while ((blk_id = spdGetRelObj( blk_csr)) != SAPID_NON_ID){

          if (blk_id >= block){ return NULL; }
          if(spdGetAttrValInt(blk_id, "sort") == BLOCK_IF||
                spdGetAttrValInt(blk_id, "sort") == BLOCK_FOR||
                spdGetAttrValInt(blk_id, "sort") == BLOCK_DO||
                spdGetAttrValInt(blk_id, "sort") == BLOCK_SWITCH||
                spdGetAttrValInt(blk_id, "sort") == BLOCK_WHILE||
                spdGetAttrValInt(blk_id, "sort") == BLOCK_HIERACHICAL
            )
                depth++;
          depth = get_depth(blk_id,depth);
          if (max < depth)
              max = depth;
     }
     return max;
}
```

分岐を含むhierachical ブロックは，分岐部分のbranch ブロックと分岐以外のブロックcompound ブロックに分けられる。branch ブロックは分岐の種類によってif文，while文などに分けられる。関数 get_depth は，あるブロックのネストの深さを返す。再帰的に呼び出され，呼び出されたブロックIDと，そのブロックのトップレベルからのネストの深さをパラメータ（depth）にもつ。

関数（クラス optionaldecl ）から，関連 func-param をたどりトップレベルのブロックのIDを求めて，get_depth 関数にパラメータとして渡す。各分岐ブロックを調べるときの情報を，適当な構造体を定義し，格納することで，if や for といったブロックごとの情報を得る。

V　ソースコード・メトリックスの取得と利用

メトリックス情報をCSVやXMLといったデータ形式で出力し，情報を活用する方法を考察する。CSVデータ形式はEXCELで処理可能であり，関数ごとの各種のメトリックスを集計して，プログラム・メトリックスを取得し，最大，最小，平均，標準偏差といた統計的なデータ処理を行い，メトリックス関連の資料を作成する。

多くのオープンソース・ソフトウエアに対して同様の処理を行い，データを集計・解析することも可能である。オープンソースをSapidアプリケーションで利用するためには，細粒度ソフトウエアリポジトリ・データベースSDBが必要であるが，本論文では，オープンソースのSDBをアーカイブし，インターネット上で提供しているインターネットサイトsappyを利用する。

1．CSVデータの処理

4章までに求めたメトリックスに，ソースコードの行数（SLOC）とコメント行数（COM）といった基本メトリックスを加え集計し，最大，最小，平均，標準偏差を求める。図Ⅲは文末の参考資料Ⅲのプログラム Aladdin[3)] に関するメトリックスデータの解析結果である。

①その他の指標

MVG：Cyclomatic数

Depth：関数callツリーの最大パス

Fanin：関数callの数，（ｖ）関数callの複雑さ

Fanout：関数calledの数，（ｖ）関数calledの複雑さ

番号	LOC	MVG	COM	LOC/COM	MVG/COM	Depth	FaninV	Fanin	FanoutV	Fanout
				途中	省略					
420	10	0	9	1.1	0	0	1	1	15	15
421	30	2	29	1	0.069	2	5	5	1	1
422	10	0	9	1.1	0	0	0	0	15	15
合計	28605	2937	6853	3290.2	492.591	845	5060	10312	1874	4335
平均	67.8	6.96	16.2	7.7967	1.16728	2	12	24.44	4.44	10.7
最大	900	168	700	90	219	18	335	1170	85	243
最小	3	0	0	0	0	0	0	0	0	0
偏差	112	14	43	10.565	10.6954	1.8	24.2	82.87	10.2	28.9

図Ⅲ　EXCELによるデータ処理結果

2．XMLデータの処理

XMLはデータ交換の標準フォーマットであり，適当なスタイルシートを用いて，ブラウザから参照することもできる。サンプルプログラムⅢはCSVデータをXMLデータに変換するプログラムである。

サンプルプログラムⅢ　XMLデータ変換

```
#include <stdio.h>
#define  MaxTextSize 10000

main(){
        FILE *fp,*fp1;
        int  line=0,  …  以下、各項目の宣言
        char fname[100];
        char tmp[MaxTextSize],tmp1[2];
        int c,n=0;

入力と出力のファイルを開く
        fp=fopen("output.dat","r");
        fp1=fopen("output.xml","w");

XMLのバージョンやsスタイルシートを指定する
       strcpy(tmp,"<?xml version=¥"1.0¥" encoding=¥"Shift_JIS¥"?>
       <?xml-stylesheet type=¥"text/xsl¥" href=¥"Output1.xsl¥"?>
        <metrics-report>¥n");
        fprintf(fp1,"%s",tmp);
DTDのタグの生成を開始＊
       while((c=fgetc(fp))!=EOF){
        strcpy(tmp,"¥t<function>¥n");
        fprintf(fp1,"%s",tmp);
        strcpy(tmp,"¥t¥t<name>");
        sprintf(tmp1,"%c",(char) c);
        strcat(tmp,tmp1);
        while ((c=fgetc(fp))!=(int)',')
            {sprintf(tmp1,"%c",(char)c);
                strcat(tmp,tmp1);
            }
        fprintf(fp1,"%s </name>¥n",tmp);

        以下、各項目ごとに、タグを生成する。
```

*DTDの定義は省略

出力結果　XMLデータ形式

```
<?xml version="1.0" encoding="Shift_JIS"?>
<?xml-stylesheet type="text/xsl" href="Output1.xsl"?>
<!DOCTYPE doc SYSTEM "file://c:¥metrics.dtd">
        <metrics-report>
        <function>
                <name>1:Extract_Eigenvector </name>
                <line>28 </line>
                <MVG>6 </MVG>
                <comment>0 </comment>
                <line_com>0 </line_com>
                <MVG_com>0 </MVG_com>
                <depth>2 </depth>
                <FANIN>5 </FANIN>
                <FANIN_visible>5 </FANIN_visible>
                <FANOUT>1 </FANOUT>
                <FANOUT_visible>1 </FANOUT_visible>
        </function>
                                        | 続く
```

おわりに

Sapidのソフトウエア解析技法の例題として，ソースコード・メトリックス取得プログラムを詳細に解説した。オープンソースのソフトウエアに関するソースコード・メトリックス情報を利用して，ソフトウエアの品質解析の指標を開発する手法を提案し，オープンソースの新たな利用方法を提起した。

今後の課題として，ドキュメントに関するメトリックスを求める手法を考える。Sapidのドキュメント管理バージョンDapidを利用してコメントやドキュメント，仕様書などを資源として利用することが可能である。

［参考資料Ⅰ］　SAPID関数の名前について

① **SpdGetARelObj，spdGetRelObjInit**

SAPID関数	取得関数	［一つ取得］	参照オブジェクト	［カーソル初期化］
Spd	Get	[A]	RelObj	Init

② **spdGetIncludedOccInit**

SAPID関数	取得関数	［一つ取得］	出現を対象	［カーソル初期化］
Spd	Get	[A]	IncludedOcc	Init

③ **SpdGetObjIdArray**：

SAPID関数	取得関数	オブジェクトID配列
Spd	Get	ObjIdArray

［参考資料Ⅱ］　HTML形式に出力１　対象プログラムmain[1)]

プログラムmainは「Sapidによるソフトウエア解析技法Ⅰ」[1)]で用いた，30行ほどの簡単なサンプルである。Ⅴ章までで求めた各種メトリックスを，HTMLの表形式で出力する。

メトリックス１：(REBOOTESPRITプロジェクトより参照)[2)]

番号	値	メトリックス	要素	基準
M１	Developing	ドキュメントのページ数／ソースコードの行数	理解性	ドキュメンテーション
M２		ドキュメントのアクセス可能性のチェックリスト	理解性	ドキュメンテーション
M３	37	コメント行／ソースコードの行数	理解性	自己記述性
M４		モジュールの自己記述性のチェックリスト	理解性	自己記述性
M５	9.25	ソースコードの行数／モジュール数	可搬性・柔軟性	モジュラリティ
M６	Developing	マシン依存コード行数／実行部コードの行数	可搬性	環境独立性
M７	Developing	システム依存コード行数／実行部コード行数	可搬性	環境独立性
M８	18	Fan-in, Fan-outの複雑さ（cyclomatic number）	理解性・信頼性	モジュラリティー・要素複雑性
M９	5	複雑さ（cyclomatic number）	理解性・信頼性	コード複雑性
M10		モジュールの自己記述性のチェックリスト	柔軟性	汎用性
M11		不良耐久性のチェックリスト	信頼性	不良耐久性
M12		テスト総数／検出エラー数		
M13		分離された要求数／全要求数		
M14		データ無矛盾性チェックリスト		
M15		手続き無矛盾性チェックリスト		
M16	0.135	複雑度／ソースコードの行数	理解性・信頼性	要素複雑性
M17	2	最大ネスト数	理解性・信頼性	要素複雑性
M18	5	内部変数の数／変数の総数		
M19	0.139	変数の総数／実行部コードの行数		

［参考資料Ⅲ］　HTML形式出力(2) 対象プログラムAladdin[3)]

Aladdinは，linuxで利用できる行列処理ソフトウエアである。Maryland大学Systems Research（USA）で開発されたオープンソースであり，関数の数が422個の中規模のプログラムである。プログラム・メトリックスと関数メトリックスについてのデータをHTMLの表形式で出力する。メトリックスのタグ記号については，第5章を参照のこと。各関数名のリンク（参照）先は，SPIEで生成した情報のソースコード部分（当該関数の先頭行）である。

メトリックス2：Procedual Metrics[3)]

Metric	Tag	Overall	Per Function
Line of Source Code	SLOC	6149	14.6 M5
Cyclomatic number	MVG	2.94e+03 M9	6.96
1 Number of if statement	N-if	1473	3.49
2 Number of for statement	N-for	1240	2.94
3 Number of while statement	N-while	23	0.0545
4 Number of switch statement	N-switch	200	0.474
5 Number of do statement	N-do	1	0.00237
6 Number of pain statement	N-plain	0	0
Max Block Depth	MBD	18 M17	N/A
Fanin	Fin	5444 M8	12.9
Fanout	Fout	3921 M8	9.29
	Max Depth	13	N/A
Lines of comment	COM	807	1.91
SLOC/COM	L_C	7.61958 M3	N/A
MVG/COM	M_C	3.63941 M16	N/A

Functions

Fuction Name	LOC	MVG	COM	L_C	M_C	DEPTH	FUN_IN	FUN_OUT	F_IO	F_IO(V)
1:Extract_Eigenvector matrix.i:6121	28	6	0	0	0	2	10(V=5)	1(V=1)	7.11	3.61
2:Extract_Eigenvalue matrix.i:6092	26	4	0	0	0	2	10(V=5)	1(V=1)	7.11	3.61
3:Solve_Eigen matrix.i:5994	96	11	4	24	2.75	2	460(V=30)	1(V=1)	325	21.2
4:MatrixSolveEigen matrix.i:5845	147	11	15	9.8	0.73	3	403(V=29)	2(V=1)	285	20.5
5:MatrixPut matrix.i:5737	82	7	6	13.7	1.17	3	16(V=8)	1(V=1)	11.3	5.7
途中省略 7～417										
416:dktq06 elmt_shell_4n_q.i:6812	34	1	31	1.1	0.032	1	0(V=0)	18(V=2)	12.7	1.41
417:dktb06 elmt_shell_4n_q.i:6710	96	3	92	1.04	0.032	1	56(V=28)	16(V=1)	41.2	19.8
418:elmt_shell_4nodes_q elmt_shell_4n_q.i:5915	789	87	744	1.06	0.117	5	212(V=106)	15(V=15)	150	75.7
419:Lamina_Sys elmt_lamina_sys.i:5773	308	9	293	1.05	0.030	2	50(V=25)	0(V=0)	35.4	17.7
420:sld02 elmt_fiber.i:5776	10	0	9	1.11	0	0	2(V=1)	15(V=15)	10.7	10.6
421:SetUpFiberRespondBuff elmt_fiber.i:5744	30	2	29	1.03	0.069	2	10(V=5)	1(V=1)	7.11	3.61
422:elmt_fiber elmt_fiber.i:5724	10	0	9	1.11	0	0	0(V=0)	15(V=15)	10.6	10.6

[参照文献・ホームページ]

1) 斉藤邦彦（2000）「Sapidによるソフトウエア解析技法Ⅰ」（彦根論叢第325号）p126

2) 日科技連編（1994）21世紀へのソフトウエア品質管理技術　（日科技連出版社）p306

3) http://www.isr.umd.edu/~austin/aladdin.html

[参考文献]

(1)山本 晋一郎，阿草 清滋(1995)「細粒度リポジトリに基づいたツール・プラットフォームとその応用」 情報処理学会ソフトウェア工学研究会，Vol.102，No.7，pp.37－42

(2)大橋 洋貴，山本 晋一郎，阿草 清滋(1999)「ソフトウェア空間をトラバースする柔軟な検索」日本ソフトウェア科学会第16回大会論文集，pp.149－152

(3)齋藤　邦彦(1999)「ソフトウエア工学ツールプラットフォームSapid」彦根論叢第310号 pp.183-199

(4)カーニハン，リッチー(1983)「プログラムング言語C」共立出版株式会社