ソフトウェア　工学　12－3
（1979. 10. 26）

プログラムの文体の評価・改良

君島　浩
（富士通株式会社）

1. はじめに

プログラムを定量的・形式的に評価することは，極めて大切である。しかしながら直観を使い，試行錯誤によって，プログラムを上手に作ってみるということも大切である。そのような取組みは，定量的・形式的な取組みが正しい方向へ進むように案内し，かつそれらが成熟するまでの間の代行者となる。ここではシステムプログラムの文体の改良の経験を述べるとともに，品質及び生産性の尺度に関連した若干の考察をする。

2. プログラムの文体

プログラムのわかりやすさは，もともとの概念のわかりやすさと表現のわかりやすさの二つに分けられる。表現のわかりやすさは，更に全体的な構成の工夫によるものと，各部分の文体のわかりやすさの二つに分けられる。プログラムの文体の重要性とその訓練方法は，「プログラム書法」[1]で述べられている。筆者も実際のシステムプログラムの文体の評価・改良を試みて，文体改良の重要性を認識し，実例を教材にした文体改良教育を行っている[2]。

文体を単純な方法で改良することはできない。以下にいくつかの改良パターンを示す。言語はシステム記述言語であるが，名標の長さが8文字以下という制約を除けば，PL/Iと同じと思って読んでいただきたい。

```
T1=NQTSIZE;                                          (1)
IF  NLRECL>=MAXLRECL
    THEN  T1=0;
...
IF  T1=0
    THEN  NQBCOUNT=0;
    ELSE  NQBLENG=T1;          (T1は一時的変数)
```

このコードは一見すると改良の余地がないように見えるが，2番目のIF文の中の二つのT1を置換して，次のように改良することができる。

```
...                                                  (1')
IF  NQTSIZE=0
 |  NLRECL>=MAXLRECL
    THEN  NQBCOUNT=0;
    ELSE  NQBLENG=NQTSIZE;
```

この二つのコード(1)と(1')の品質の違いを定量的に表すことは可能だろうか．(1)から(1')に自動変換することは可能だろうか．自動変換は記号上の式の評価

や式の単純化のルーチンを使えば，一般化可能であろう．次の例はどうか．

```
T1=(BUFSIZE-1)/4096;                                    (2)
IF  T1<EXPLIMIT
   THEN T1=EXPLIMIT;
T2=BUFADD/4096;
LRECL=T1+T2;                    (T1とT2は一時的変数)
```

```
LRECL=BUFADD/4096                                       (2')
      +MAX((BUFSIZE-1)/4096,
           EXPLIMIT          );
```

これはMAX関数の存在を知っていても，それが使えるケースだということに気が付きにくいので，なかなか難しく，新入社員教育でこの種の問題を与えても正答率はあまり高くない．

```
DCL  1  NAMETABC,                                       (3)
        2  NAMETAB(10)  CHAR(4);
DCL  1  NAMESAVC,
        2  NAMESAV(10)  CHAR(4);
IF  NAMETAB(1)='/END'
   THEN  I=4;
   ELSE  DO;
      DO  I=1
          UNTIL  (NAMETAB(I)='/END');
         I=I+1;
      END;
      I=I*4;
   END;
SUBSTR(NAMESAVC,1,I)=
   SUBSTR(NAMETABC,1,I);
```

```
DO  I=1                                                 (3')
    UNTIL  (NAMETAB(I-1)='/END');
   NAMESAV(I)=NAMETAB(I);
   I=I+1;
END;
```

この例では(3')にとどまらず，(3")まで改良できる．

```
NAMESAVC=NAMETABC;                                      (3")
```

プログラムの仕様を知らなければ，この改良はできない．(3")では配列の終端

マーク「／ＥＮＤ」のはいっている要素よりあとの要素まで複写しているので，それでも構わないかどうかは，この部分のコードを読むだけではわからないからである．

3. 定量的な分析

以上の三つの改良例は作り物ではなく，実際のシステムプログラムの見直しで添削した実例である．これらの定量的な性質を少し眺めて見よう．

表１	(1)	(1')	(2)	(2')	(3)	(3')	(3'')
実行文の行数	6	4	5	3	11	5	1
実行文の記号数（単語，区切りの数）	34	22	34	19	69	36	4
フローグラフの弧の数	5	4	4	1	6	3	1

「実行文の行数」は，生産性の尺度の一つ「行数／人日」や信頼性の尺度の一つ「バグ数／行数」などによく使われる．三つの改良例を見ると，直観的に悪いと判断した原案(1)，(2)，(3)の方が，行数が多いがゆえに，生産性も信頼性もよさそうなデータが出ることに注意しなければならない．しかも直観的な良し悪しも，定量的な良し悪しも，同じ問題を解く複数の代案を比較することをしなければ，はっきりしなかったのである．

「実行文の記号数」は，コンパイラの字句解析の単位で数えたもので，空白の多少に影響されず，一つの文の長短を反映してくれるので，行数よりも頼りになると考えて評価してみた．三つの例では，行数で比較しても記号数で比較しても，あまり変わらないようである．

「フローグラフの弧の数」は，試験の手間に関係があると考えて評価してみた．それぞれのコードは，図１のようなグラフと見なして，弧を数えた．

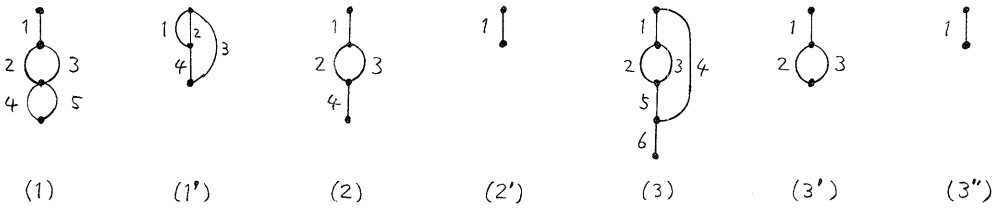

図１　各コードのグラフ

これらの例に関していえば，直観的な文体改良は，主にプログラムの複雑さを量的に減らそうとしたのではないかと推察することができる．文体の改良には，名標の吟味などのような質的改良や，段付けなどのような形式の改良なども含まれるが，最も効果的だと感じられるのは，この種の量の削減である．プログラムのわかりやすさの評価尺度は「単位当たりのわかりやすさ」であってはならない．

4. 設計の技術と管理

文体改良は一般には機械的にはできない。つまりコーディングといえども，一般の製造作業とは異なる。ましてプログラム全体の大きさや理解しやすさは，アルゴリズムやモジュール構成の工夫，つまり設計次第で大きく変わるものである。「行数／人日」や「バグ数／行数」などを眺めても，設計者の技術力に個人差があるのを放っているのなら，生産性や品質を管理できないばかりか，真実と逆の評価をしてしまう恐れさえある。この種の評価尺度はむしろ定数として考え，行数，人日，バグ数などの絶対値の方を，地道に減らすよう努力するのが，設計の管理の正統的なやり方であろう。「行数／人日」や「バグ数／行数」が，納期見積りなどの係数（定数）として以外に役に立つのは，その実績値が平均から大きくはずれたら，普通はありえないような大失策や超美技があったのではないかということを，知らせてくれることである。

電気工学のカリキュラムの中心となるのは，原理（電磁気学など），設計基礎（部品，電気回路など），応用設計（ラジオ設計など）などであって，設計管理学ではない。ソフトウェア開発においても，プログラム理論，アルゴリズム・データ構造，具体的ソフトウェア方式論（制御プログラム方式，コンパイラ方式など）などが重要である。残念ながら，この種の教育界の供給力は，まだまだ需要を満たしていないし，出版物においても設計管理学に類するものばかりが目に付く。このような状況なので，現場においてはプロジェクトの外側からの管理よりも前に，中にとびこんでの技術的な指導や見直しが重要である。

5. 設計の見直し

文体改良の実例は，新人教育の教材や実務の手引書として使われている。悪例改良なので，設計や芸術の指導で必要な，「見せる」，「話す」，「やらせる」，「しかる」，「ほめる」の条件がそろっている。しかし重要なのは文体の評価・改良の技巧だけでなく，見直しの大切さとその精神も示していることである。プログラムの見直しといえば，バグ（仕様に対する誤り）を探すことであるかのように，考えられがちである。しかしながらプログラムのような設計の産物の見直しは，誤り探しではいけない。(1)，(2)，(3)，(3′) のコードが，すべて仕様には合っていることに注目願いたい。見直しの中心は，誤りを探すことではなく，原案を理解し，よりよい代案を探すことである。文体などを改良せずに，わかりにくい原案を眺めても，バグは探しつくせないし，行数を減らせば自然にバグの絶対数が減ってくれる。なによりもプログラムの真の価値は，バグがないことではないのである（もちろんバグがあっては話にならないが）。プログラムをうまく作るには，よい手本を見て学ぶことと，文体改良のような見直しを実施することがまず大切であろう。

参考文献

1) B.W.Kernighan and P.J.Plauger, "The Elements of Programming Style," McGraw-Hill, 1974. 木村泉訳，「プログラム書法」，共立出版，1976.
2) 君島，河田，「ソフトウェア開発における品質向上の技法」，FUJITSU, Vol.29, No.6, 1978, pp.213-223.