ソフトウェア工学 50－6
（1986. 9. 24）

# プログラム作成能力の評価尺度とデータ収集ツール

松本 健一　大西 諭　井上 克郎　工藤 英男　杉山 裕二　鳥居 宏次

大阪大学　基礎工学部

ソフトウェアの生産性や品質の評価において，ソフトウェアの生産に直接携わるプログラマの能力は，考慮すべき重要な要素の一つである．しかし，プログラマの能力の定量的評価は，非常に困難であると思われており，これまであまり試みられていなかった．
本稿では，「プログラム作成能力」をプログラマの能力であると考え、それを定量的に評価する尺度（評価式）を提案する．また，この評価式を用いて，コンパイラの製作を行った学生のプログラム作成能力を求めた結果について述べる．

A Measure of Programmer's Capability

Kenichi MATSUMOTO, Satoshi ONISHI, Katsuro INOUE, Hideo KUDO,
Yuji SUGIYAMA, and Koji TORII

Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

In the study of software metrics, measuring the 'capability' of each programmer is one of the essential problems. However, it has been believed that measuring the capability is very hard and impractical; thus, very simple but insufficient measure such as experience years or intuitive evaluation given by the manager has been used. In this paper, we propose a model of the programmer capability which is based on the sum of the life times of errors involved in the program text of a project made by the programmer. Using this model, we have computed the capabilities of students who performed compiler construction projects.

## 1．まえがき

ソフトウェアに対する需要が増大し，またソフトウェアの欠陥や故障が社会に大きな影響を与える今日，ソフトウェアの生産性と品質の向上は，ソフトウェア工学の大きな目標の一つである．ソフトウェアの生産性や品質を向上させるためには，それらを定量的に評価することが必要である．ソフトウェア・メトリックスは，ソフトウェア生産活動とソフトウェア自身の一般的な性質を定量化（モデル化）し，定量化した値をもとにソフトウェア生産活動の制御を行い，ソフトウェアの生産性や品質を効率的に向上させることを目的としている[1]．

ソフトウェア・メトリックスによるソフトウェアの生産性や品質の評価において，ソフトウェアの生産に直接携わるプログラマの能力は，考慮すべき重要な要素の一つである．特にソフトウェアの開発に要するコストや工数，作成されたソフトウェアの信頼性といったものは，プログラマの能力に大きく左右されると考えられる．しかしプログラマの能力それ自体の定量的評価が非常に難しく，これまであまり試みられていないため，これを考慮に入れないモデルが多い．また考慮に入れているものでも，「プログラマとしての経験年数」や「管理者による主観的な評価」に基づいた単純な定量化を行っている程度である[2][3]．プログラマの能力をより一般的に評価することができれば，それを組み入れた，より現実的なモデルを作ることが可能になる．

本稿では，「プログラム作成能力」をプログラマの能力であると考え，このプログラム作成能力を定量的に評価する尺度（評価式）を導出すると共に，実際のプログラム作成過程から得られたデータにそれを適用した結果について述べる．

## 2．プログラム作成能力とその評価尺度

設計・コーディングを誤りなく行え，プログラムを効率的に作成できることを，プログラマの能力と考える．本稿ではこれを「プログラム作成能力」と呼ぶ．なおここでは，一人のプログラマに一つの問題が与えられ，設計・コーディングからテスト・デバッグまで，プログラマが一人で行うようなプログラム作成過程についてのみ考える．

このプログラム作成能力は，プログラムの作成過程で作り込まれたエラーを調べることによって定量化できると考えられる．なぜなら，エラーは設計・コーディングの誤りがプログラム上で具体化したものと考えられるからである．しかし，作り込まれたエラーの数を数えるだけでは充分とは言えない．エラーごとに，プログラム自身とその作成過程に与える影響が異なるからである．したがって，エラーの影響度に応じたある種の重みを付け，各エラーごとの重みの総和をとることにする．いわば，エラーの質と量によってプログラム作成能力を定量化する．

エラーの影響度に応じた重みとしては，「エラーが作り込まれてから取り除かれるまでに経過した時間（エラー寿命と呼ぶ）」を用いることとする．なぜなら，一般に寿命の長いエラーというのは，作り込まれてからの時間が大きいため発見しにくく，それを取り除く際に影響を及ぼす範囲も広くなるからである．逆に影響度の高いエラーというのは，発見，修復が困難で，エラー寿命は長くなると考えられる．一方，プログラム作成能力からいえば，作り込んだエラーはなるべく早く取り除くべきである．

エラー寿命の総和は，次の式のようにプログラム作成過程の各時刻にプログラム中に存在したエラー数を時間で積分した値になる．

$$(\text{エラー寿命の総和}) = \int N_t \, dt$$

$N_t$：プログラム作成過程の時刻 $t$ に
プログラム中に存在したエラー数

この式は，エラー平均寿命の概念の導入により，次のように変換される．

$$\begin{aligned}(\text{エラー寿命の総和}) &= \frac{\int N_t \, dt}{(\text{総エラー数})} \times (\text{総エラー数}) \\ &= (\text{エラー平均寿命}) \times (\text{総エラー数})\end{aligned}$$

このように，エラー寿命の総和は，エラー平均寿命と総エラー数の積になる．エラー平均寿命と総エラー数はプログラム作成能力とともに，与えられた問題の難しさに依存すると思われる．与えられた問題の難しさの影響を評価尺度から取り除き正規化するため，与えられた問題の難しさ $p$ から標準的なエラー平均寿命を与える関数 $f(p)$ および，$p$ から標準的な総エラー数を与える関数 $g(p)$ により割る．さらに，プログラム作成能力が高いほど評価尺度の値が良くなるように逆数をとり，それをプログラム作成能力の評価尺度 $C_{fg}$ とする．時間軸 $t$ は，工数を反映した時間とする．

$$\frac{1}{C_{fg}} = \frac{(\text{エラー平均寿命})}{f(p)} \times \frac{(\text{総エラー数})}{g(p)}$$

$$= \frac{\int N_t \, dt}{f(p) \cdot g(p)}$$

p　：与えられた問題の難しさ
f(p)：標準的なエラー平均寿命を与える関数
g(p)：標準的な総エラー数を与える関数

一つの問題が与えられた際，エラーの総数が小さくなれば，$C_{fg}$は大きくなる（エラー平均寿命が大きくならないと仮定した場合）．また，エラーが取り除かれるまでの時間が短くなると，大きくなる（エラー数が増えないと仮定した場合）．

## 3．実験

### 3．1　データ収集源

データは，大阪大学基礎工学部情報工学科３年生に対して行われた学生実験より収集した．学生実験では，同じ目的で作られたプログラムが多数入手できること，プログラマである学生の能力にある程度バラつきがあるなど，データ収集上の利点がある．学生実験のあらましは以下の通りである．

- 作成するプログラムは，C，PascalまたはPL/Iのコンパイラのサブセット（行数約2000行）である．コンパイルされるプログラムの言語仕様については最低限必要な部分だけは全学生共通に決められているが，各学生の判断により仕様を拡張することが許されている．このため問題の難しさは学生により多少異なる．
- プログラミング言語はＣまたはPascal．すべての学生はどちらも使用経験がある．
- すべての学生は事前に簡単なコンパイラの講義を受けているが，コンパイラを実際に作成するのは初めてである．

### 3．2　収集データ

今回の実験では，$C_{fg}$の評価式中の各パラメータに次の値を用いることにし，これらをデータとして収集した．

t ＝［累積端末使用時間］

- 工数とは完全に一致していないが，他に工数を容易にかつ客観的に得る方法がないと思われた．またこれは，容易に自動収集可能である．

$N_t$＝［時刻ｔにプログラム中に存在した設計・コーディングの誤りによる論理エラー数］

- プログラムが与えられた仕様を満たし，正しく動作するために，何らかの修正が加えられたプログラムの部分の数．
- シンタックスエラーは含まない．

f(p),g(p)＝
［実験終了時のプログラムサイズ(語い数)］

- 問題の難しさｐはコンパイラを作るべき言語の仕様の複雑さと考えて良い．
- 学生ごとの仕様の違いがあまり大きくなく，基本的なデータ構造やアルゴリズムにも差がないことから，仕様の難しさは，実験終了時のプログラムサイズで表わせよう．
- f(p)，g(p)はｐの単調増加関数と考えられる．今回のようにｐの範囲が狭い場合には，$f(p)=p$，$g(p)=p$と近似できよう．
- プログラムサイズとしては語い数を用いる．語い数とプログラムテキストの行数には表１に示すように高い相関があるので，語い数の代わりに行数を用いてもよいであろう．

表１．　プログラムテキストの行数と語い数の相関係数

| | b | c |
|---|---|---|
| a | 0.97 | 0.91 |
| b | — | 0.95 |

ａ：行数(空白，コメント行含む)
ｂ：行数(空白，コメント行除く)
ｃ；語い数(コメント分を除く)

### 3．3　データ収集ツール

$C_{fg}$を求めるために作成したツールの主なものについて説明する．

・端末使用時刻の記録ツール
　端末が使用されるごとに，その開始時刻と終了時刻を各学生について記録する．

・プログラムサイズの計測ツール
　プログラムテキストのファイルを入力するとその行数と語い数が出力される．

・プログラムテキストの履歴収集ツール
　エディタによってプログラムテキストの更新が行われるごとに自動的に起動され，更新の前後のプログラムテキストを比較，変更場所を求めてそれを学生別，ファイル別に記録するツール．この時，更新時刻と更新目的も記録される．更新目的は，更新作業終了時にCRTに表示したメニュー（表2参照）から学生自身に選択してもらう．

表2．更新目的を選択するメニュー

| |
|---|
| 1:基本設計の変更 |
| 2:ロジック・エラーの訂正 |
| 3:シンタックス・エラーの訂正 |
| 4:プログラムの入力 |
| 5:プログラムの改良 |
| 6:テスト用ファイルの作成・変更 |
| 7:エラーチェック用ステートメントの挿入・削除 |
| o:その他 |

・テキスト履歴の差分リスト作成ツール
　テキストの履歴データをもとに，図1のような形式のリストを出力する．リストは更新順に，更新時刻・更新目的と共に，更新前後のプログラムテキストを左右に対応づけられて出力される．

・プログラム作成状況のグラフ作成ツール
　プログラム作成過程におけるプログラムサイズの変化，エディタの使用回数，テキスト更新目的などのデータをまとめ，図2に示すグラフを作成するツール．$C_{fg}$を求めるためのツールではないが，作成したグラフは，$C_{fg}$による評価の対象とする学生の選択や評価結果の分析の参考資料とした．

### 3．4　$\int N_t dt$の算出方法

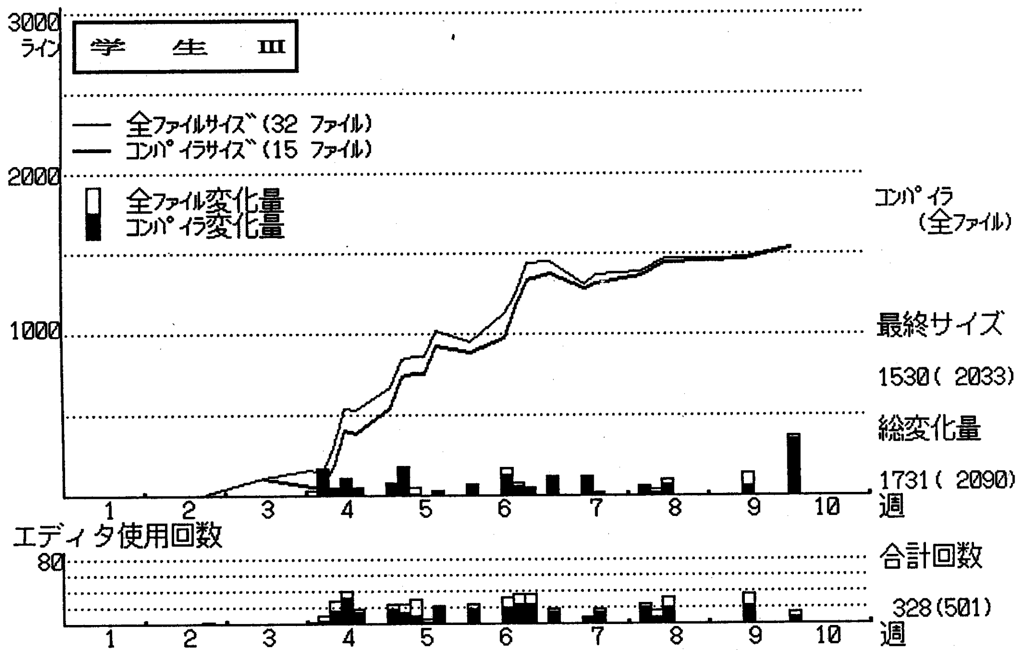
$\int N_t dt$を求めるには，プログラムテキストの履歴をもとに，まずエラーの特定を行う．そしてそれが作り込まれた時刻と取り除かれた時刻から，そのエラーの寿命を求める．すべてのエラーの寿命の総和が$\int N_t dt$である．エラーの特定と，それが作り込まれた時刻と取り除かれた時刻の決定は，図1に示したテキスト履歴の差分リストをもとに，手作業で行った．作り込まれた時刻と取り除かれた時刻は，端末使用時刻の記録をもとに累積端末使用時間に変換する．なお累積端末使用時間 t の刻み幅は15分とした．

```
****************** Wed Dec  4 22:38:35 GMT+9:00 1985 ******************
更新時刻
-----------------------------------------------------------   -----------------------------------------------------------
 101      EXPR *result;                                        101      EXPR *result;
 102                                                           102
 103    printf("***********exp_¥n");                           103    printf("***********exp_¥n");
 104                                                           104
 105      result = exp_or ();                                  105      result = exp_or ();
 106c*    if (is_left_val(result->n_mode)) {                   106c*    if (is_left_val(result) {
 107          insymbol ();                                     107          insymbol ();
 108          if (op_exp(sy)) {                                108          if (op_exp(sy)) {
 109              result = cons_node (sy, 0, result, exp_());  109              result = cons_node (sy, 0, result, exp_());
 110              return result;                               110              return result;
 111          } else                                           111          } else
--- end line is 316                                           --- end line is 316
```

更新箇所　更新前　更新後　更新目的のメニュー番号

```
### Main menu ###
2
```

図1．テキストの差分リスト出力例



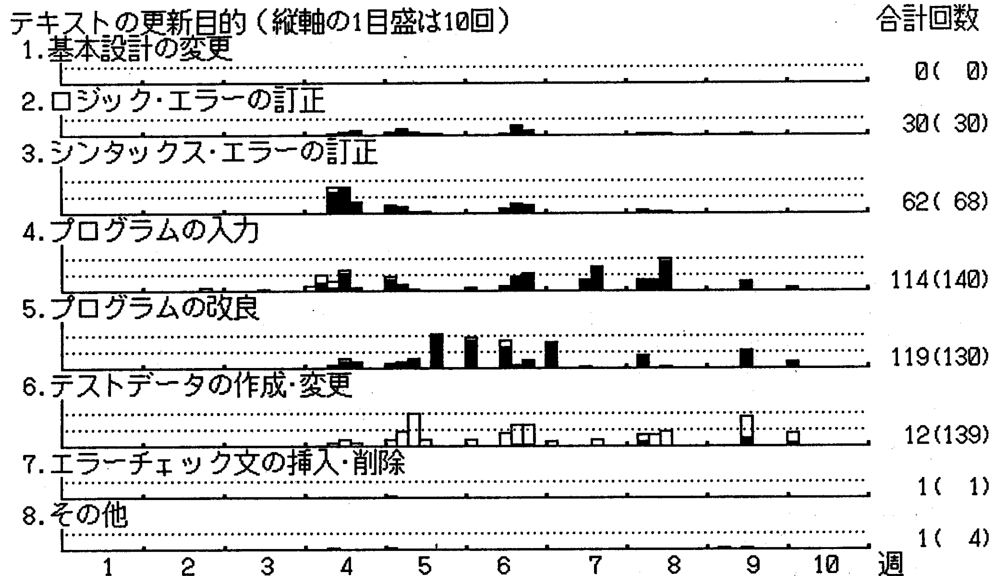




図2. プログラム作成状況グラフ出力例

## 3．5　実験結果

収集したデータをもとに，９人の学生について$C_{fg}$を求めた．

９人の学生に関する主な収集データと$C_{fg}$を表３に示す．また，図３に，$\int N_t dt$の値が最大の学生と最小の学生の$N_t$のグラフを示す．

$C_{fg}$の最小値は726，最大値は11413であった．$C_{fg}$の大小は，各学生のプログラム作成能力に対するわれわれの直観的な評価と一致している．$C_{fg}$が特に大きい学生と小さい学生についてその理由を分析した結果を述べる．

●$C_{fg}$が大きい学生

- 学生Ⅶ　（$C_{fg}$＝11413）
- 学生Ⅸ　（$C_{fg}$＝　6304）

彼らは共にPascalコンパイラをPascalで記述している．彼らがプログラム作成上参考にした本として報告しているものにはPascalコンパイラをPascalで記述したプログラムが掲載されている．彼らは，主要なアルゴリズムとデータ構造をこのプログラムからそのまま用いている．このため総エラー数も少なく，順調に作業が進んだ．

- 学生Ⅵ　（$C_{fg}$＝　7024）

作成するコンパイラの仕様を要求された必要最小限に抑えている．他の$C_{fg}$の小さい学生の中にはプログラムを作り始めた頃欲張って仕様の拡張をしようとしているが，大半は実現できず，無駄な努力をしている．この学生は，プログラム作成の形態も，設計・コーディングとタイプイン・テスト・デバッグを明確に分けている．机上での充分な設計・コーディングを行っているので，端末での作業は非常に効率よいものになっている．その証拠に端末使用時間が非常に小さい．

●$C_{fg}$が小さい学生

- 学生Ⅷ　（$C_{fg}$＝　726）

コンパイラに対する理解度が低い．このことは学生自身が提出したレポートで述べている．理解度が低いため，設計段階での誤りが多く，コンパイラの仕様の変更や文法規則の変更が目立つ．このような変更はエラーとしてかなり深刻なものである．

プログラムを作成し始めてからそれがコンパイル可能になるまで，かなり時間がかかっている（約12日間）．このためエラーの寿命が長くなっている．この原因の一つは，プログラミング言語がPascalのために分離コンパイルができなかったことにある．しかしそれ以上に，手続きや関数単位での段階的なコンパイルとテストを行わず，まずプログラム全体をタイプインしようとするプログラム作成形態が原因である．

表３を見ると，エラー平均寿命がプログラマによってかなり違うことがわかる．エラー平均寿命を語い数で割った値でも同じである．エラー平均寿命がプログラマによって異なるということは，評価尺度にエラー寿命を組み込んだ意味があったということである．

$1/C_{fg}$とその主なパラメータの間の相関係数を調べた結果を表４に示す．

もし$1/C_{fg}$と常に強い相関のあるパラメータが存在するならば，それを用いて評価式の簡略化が可能かもしれない．しかし，今回の結果では，相関のあるものはなかった．

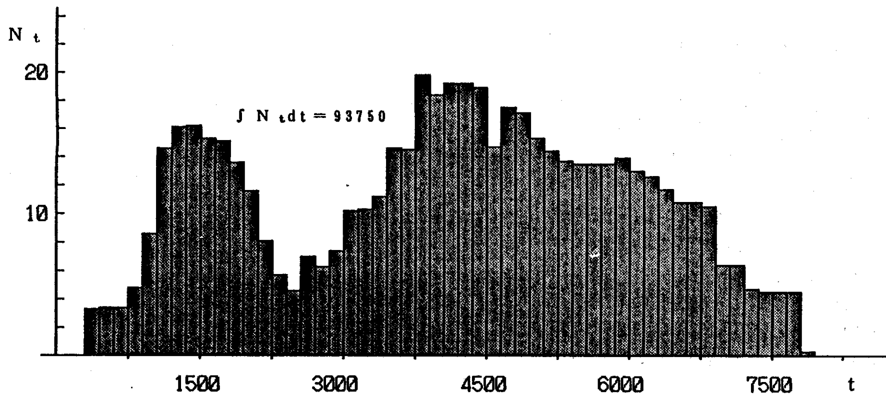
表４．$1/C_{fg}$とパラメータ間の相関係数

| | b | c | d | e | f |
|---|---|---|---|---|---|
| a | -0.16 | -0.40 | 0.41 | 0.03 | -0.55 |
| b | − | 0.45 | 0.52 | 0.82 | 0.66 |
| c | − | − | -0.35 | 0.35 | 0.11 |
| d | − | − | − | 0.68 | 0.49 |
| e | − | − | − | − | 0.54 |

a：語い数
b：総端末使用時間
c：総エラー数
d：エラー平均寿命
e：$\int N_t dt$
f：$1/C_{fg}$

表3.　主な収集データと$C_{fg}$

| 学　生 | I | II | III | IV | V | VI | VII | VIII | IX |
|---|---|---|---|---|---|---|---|---|---|
| プログラミング言語 | C | C | C | C | C | C | Pascal | Pascal | Pascal |
| ターゲット言語 | PL/I | PL/I | PL/I | PL/I | C | C | Pascal | C | Pascal |
| 実験終了時のプログラムサイズ（語い数） | 12388 | 9849 | 9308 | 10638 | 8701 | 9007 | 16566 | 6960 | 17606 |
| 総端末使用時間　（分） | 7955 | 6202 | 5906 | 8021 | 4754 | 3463 | 5838 | 8651 | 5139 |
| 総エラー数 | 77 | 101 | 61 | 78 | 55 | 35 | 26 | 49 | 33 |
| エラー平均寿命　（分） | 1218 | 294 | 469 | 859 | 584 | 330 | 923 | 1363 | 1490 |
| エラー平均寿命／語い数 | 0.098 | 0.030 | 0.050 | 0.081 | 0.067 | 0.037 | 0.056 | 0.196 | 0.85 |
| $\int N_t dt$ | 93750 | 29715 | 28620 | 67020 | 32145 | 11550 | 24045 | 66765 | 49170 |
| $C_{fg}$ | 1637 | 3264 | 3027 | 1689 | 2355 | 7024 | 11413 | 725 | 6304 |



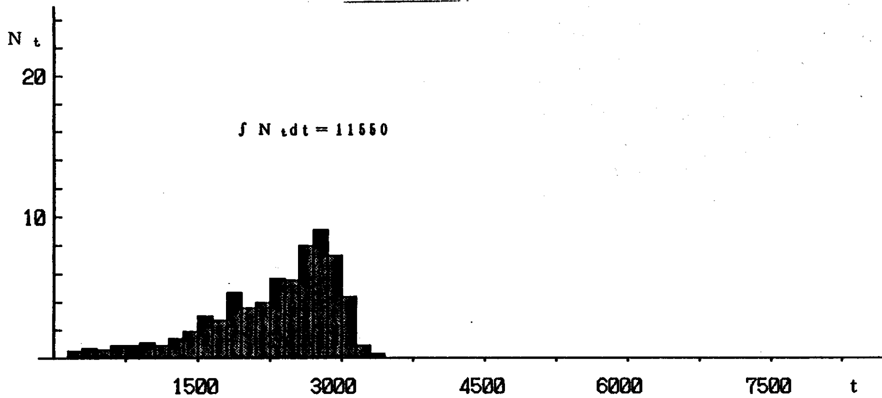




図3.　$N_t - t$　グラフ

## 5．あとがき

本稿では，これまであまり行われていない，プログラマの能力を定量的に評価する尺度の導出，および，実際のプログラム作成過程から得られたデータにそれを適用した結果について述べた．

評価尺度の導出においては，プログラマがプログラム中に作り込むエラーに注目した．特に，「エラー寿命」というものによって，エラーの質的な違いを客観的に表わし，評価尺度に盛り込んだ．実験の結果，この「エラー寿命」は，プログラマによってかなり差のあることがわかった．

評価尺度を，実際のプログラム作成過程に適用する際に問題となるのは，関数f(p)，g(p)の扱いである．今回の実験では，問題の難しさｐのばらつきが小さいことから，f(p)=p，g(p)=pとした．f(p)，g(p)を決定するには，難しさｐがかなり違う問題を，多くの同じプログラマに与え，その結果を分析する必要がある．

f(p)，g(p)の決定の他にも，評価尺度に対するより詳しい分析を行うために今後さらに適用例を増やしていくつもりである．いろいろな角度からの検討結果を，評価式に加味することにより，プログラマの能力を客観的に評価する，より一般的な評価尺度としていきたい．

評価尺度の検討とは別に，今のところ人手を介して行っている$\int N_t dt$の算出作業の省力化が必要である．しかし$\int N_t dt$の算出にはプログラム中のエラーを特定する技術など問題があるので，完全な自動化は難しいと思われる．そこで，この評価尺度との相関が高く，かつ自動収集が容易なデータを用いた簡略式を開発することを考えている．

いずれにしても，評価尺度に関する基礎的なデータを揃えるとともに，評価尺度を容易に適用できる方法やツールを整備していくことが大事である．そして，ソフトウェア・メトリックスの既存のモデルにこの評価尺度を加味し，そのモデルがより一般性を持つことが示されるならば，このプログラム作成能力の評価尺度の正当性や有効性に対する評価が行えるはずである．

## 参考文献

[1] 日本情報処理開発協会：“システムの高信頼性技術に関する調査研究（電子応用システム）成果報告書”，（1985）．

[2] 宮本：“ソフトウェア・エンジニアリング：現状と展望”，ＴＢＳ出版会（1982）．

[3] Barry W. Boehm：“SOFTWARE ENGINEERING ECONOMICS”，Prentice-Hall,Inc., Englewood Cliffs,N.J.，(1981)．

[4] 福田：“バグ重大度の定量化の試み”，ソフトウェア工学 32-2 （1983.9.28）．

[5] 岡野他：“バグ検出率とバグ生存率による品質評価方法の一例”，情報処理学会第29回全国大会，5R-5（1984）．

[6] 松本他：“ソフトウェア信頼性モデルの比較とプログラム作成能力の評価基準について”，昭和61年度電子通信学会総合全国大会，1709（1986）．