論　文

エラー寿命に基づくプログラマ性能の実験的評価
——大学環境におけるプログラム開発——

非会員　松本　健一[†]　　正　員　井上　克郎[†]
正　員　菊野　　亨[†]　　正　員　鳥居　宏次[†]

An Experimental Evaluation of Programmer Performance Based on Error Life Span —— For Program Development in Academic Environment ——

Kenichi MATSUMOTO [†], *Nonmember*, Katsuro INOUE [†], Tohru KIKUNO [†] *and* Koji TORII [†], *Members*

あらまし　プログラマ性能は，基本的には，プログラマをソフトウェアを生産する一つの機械（装置）とみなした上で，誤りのない設計，コーディング，およびデバッグをいかに効率良く行いうるかを測る尺度である．これまでに，プログラマ性能がソフトウェアの生産性や信頼性に大きな影響を与えることが報告されている．しかし，現状では，プログラマ性能の厳密な意味については必ずしも確立されていない．本論文では，ソフトウェアの開発過程で発生したエラーに留意して，プログラマ性能の定量的評価を試みる．ここでは，性能の高いプログラマほどエラーを作り込まず，仮に，エラーを作り込んだとしても短い期間でそれを取り除くことができると仮定する．この仮定のもとに，エラーがソフトウェアに作り込まれてから除去されるまでの時間に注目したエラー寿命を新しく導入し，エラー寿命に基づくプログラマ性能の定式化を提案した．更に，学生実験から収集したデータを用いて評価実験を行った．実験の結果，エラー寿命に基づく評価尺度とプログラマ性能の関係をいくつかの視点から具体的に示すことができた．

1. まえがき

最近，ソフトウェアに対する需要が増大し，それと共に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ことが少なくない．それに伴い，ソフトウェアの生産性と信頼性の向上はソフトウェア工学における主要な目標として位置づけられている[8]．従って，ソフトウェアの開発過程におけるプログラマの生産活動を支援することはソフトウェアの生産性や信頼性の向上にとって基礎的かつ重要な課題である．

ソフトウェアの開発過程からプログラマの生産活動に関するデータを収集し，その分析結果をプログラマにフィードバックする試みがある[1),(10)]．すなわち，プログラマが分析結果に基づいて自らの生産活動を管理することにより，ソフトウェアの生産性や信頼性の向上を目指そうとするものである．その場合，フィードバックすべきデータはプログラマの生産活動を十分に反映するように選択される必要がある．そうしたデータの有力な候補の一つとしてプログラマ性能が知られている[2),(5),(11)]．

プログラマ性能は，基本的には，プログラマをソフトウェアを生産する一つの機械（装置）とみなした上で，誤りのない設計，コーディング，およびデバッグをいかに効率良く行いうるかを測る尺度である．しかし，現状では，プログラマ性能の厳密な意味については必ずしも確立されていない．Sackman ら[11]はプログラミング性能（programming performance）という用語を，Boehm[2]はプログラマ能力（programmer capability）という用語を用いている．

† 大阪大学基礎工学部情報工学科，豊中市
Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan

これまでに，プログラマ性能がソフトウェアの生産性や信頼性に大きな影響を与えることが報告されている．例えば，Sackmanらが行った実験[11]ではデバッグの作業時間にプログラマ性能の違いによって最大で28倍もの差がある．一方，Boehmのソフトウェアの開発コスト予測モデルCOCOMO[2]においても，予測される開発コストにプログラマ能力によって最大2.03倍の差がある．しかし，プログラマ性能の定量的評価は一般に困難と考えられている．そのため，通常，「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」に基づいてプログラマ性能を求めている．

本論文では，ソフトウェアの開発過程で発生したエラーに関する次の二つの要因

E 1 ：作り込んだエラーの総数

E 2 ：エラーが作り込まれてから取り除かれるまでの時間

に留意して，プログラマ性能の定量的評価を試みる．評価における基本的な考え方として「性能の高いプログラマほどエラーを作り込まず(E1)，仮に，エラーを作り込んだとしても短い期間でそれを取り除くことができる(E2)」と仮定する．このときプログラマの性能は，各エラーが作り込まれてから取り除かれるまでの時間(これをエラー寿命と呼ぶ)の総和で表される．

以下，**2.**では，エラー寿命とそれに基づくプログラマ性能の評価尺度について述べる．次に**3.**では，学生実験から収集した実際のプログラム開発過程のデータを用いた評価尺度の適用実験について述べる．**4.**では，プログラマ性能の評価値と学業成績との比較を行う．更に，二つのプログラム開発過程から独立に求めた，同一プログラマに対する評価値同士の比較を行う．最後に**5.**では，チームによるプログラム開発への評価尺度の適用の可能性，今後の課題などについて簡単にまとめる．

2. エラー寿命とプログラマ性能

2.1 エラーとフォールト

IEEE標準の定義[15]によると，与えられた問題を解くために必要な情報を理解する過程や，手法やツールを使用するための思考の過程でプログラマがおかす誤りをエラーと呼ぶ．一方，エラーがソフトウェア中に具体化したものがフォールトである(図1参照)．但し，エラーとフォールトは必ずしも1対1に対応せず，通常，一つのエラーが原因で複数のフォールトが発生する．

プログラマ性能を評価するために，エラーの数を数えることが考えられる．しかし，上の説明でもわかるように，エラーは抽象的なものとして定義されているため，実際に測定することは非常に困難である．これに対し，フォールトは具体的に測定可能である．そこで本論文では，IEEE標準でいうフォールトを測定し，それから推定した誤りをエラーと呼ぶ．但し，明らかに一つの誤りが原因と考えられるフォールトがいくつかある場合には，それらのフォールトをまとめて一つのエラーとする．

2.2 エラー寿命 T_e

ここでは設計，コーディング，デバッグの一連のソフトウェアの開発過程を考える．この各過程においてエラー e_i $(i=1, 2, \cdots)$ がソフトウェア中に具体化してから，取り除かれるまでの時間をエラー寿命と定義する．図2にエラー寿命の例を示す．同図で×印はエラーがソフトウェア中に具体化した時刻を，○印はそのエラーが取り除かれた時刻を示す．×印，および，○印の時刻の測定方法については**3.2**で述べる．

エラー寿命と同様の概念はこれまでにも提案されて

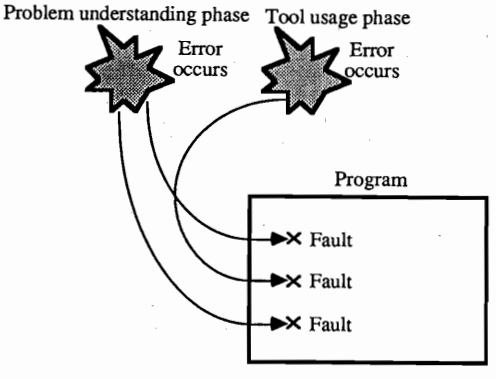


図1 エラーとフォールトの関係
Fig. 1 Error and fault.

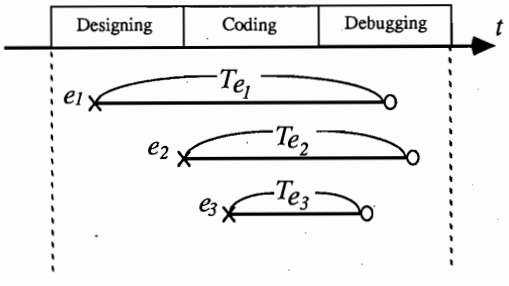


図2 エラー寿命 T_e
Fig. 2 Error life span T_e.

いる．Mills[7]はソフトウェアの品質評価の尺度としてエラー日数(error days)を利用している．ここでエラー日数とは，エラーがソフトウェア中に作り込まれてから取り除かれるまでの日数の総和である．次に，WeissとBasiliは文献[13]で，プログラムテキストの変更情報に基づいたソフトウェア開発過程の評価を行っている．そこで，ソフトウェアシステム中にエラーが存在していた時間の長さが有益な情報になりうることを指摘している．

しかし，いずれの場合にも，ソフトウェアの開発過程から収集したデータに基づいた議論は行われていない．これに対し，本論文では，学生実験からデータを実際に収集し，エラー寿命を算出する．更に，エラー寿命とプログラマ性能との関係についても議論する．

2.3 プログラマ性能の評価尺度 s

ソフトウェア中に作り込まれたエラーがソフトウェアの生産性や信頼性に悪影響を及ぼすことが知られている[12]．従って，プログラマの性能を評価する単純な方法はエラー数を数えることである．すなわち，エラー数が多いほどプログラマの性能が低いとみる．

しかし，ソフトウェアの生産性や信頼性に与える影響がどのエラーも同じであるとは考えにくいので，エラー数を数えるだけでは評価としては不十分である．エラーがもつ影響度に応じた重み付けが必要であることが指摘されている[3]．

ここではその重みとしてエラー寿命を用いることを提案する．その主な理由は次のとおりである．エラー寿命が長ければ，そのエラーに関連するプログラムコード(の細部)について忘れてしまう可能性が大きい．更に，エラーが影響を及ぼすコードの量も多くなる．そのため，エラー除去に必要な労力は大きくなり，作業内容もそれだけ複雑になってしまう．

この考えに基づき，プログラマ性能の評価尺度 s を次のように定義する．まず，エラー寿命 T_e は与えられた問題の難しさ p にも依存すると考える．すなわち，問題が難しいほど，プログラムの開発作業は複雑になり，エラーは発生しやすく，エラー寿命も長くなると仮定する．更に，性能の高いプログラマほど s の値が大きくなるように逆数を用いて次式(1)のように定める．

$$s=(\text{エラー寿命 } T_e \text{ の総和}/f(p))^{-1} \qquad (1)$$

但し，f：正規化関数

p：与えられた問題の難しさ

とする．

この s の定義では，開発すべきプログラムの要求仕様に基づき，プログラマが一人で，設計，コーディングからテスト，デバッグまでを行うことを前提としている(チーム構成でソフトウェア開発を行う場合については**5.**で述べる)．更に，プログラムはあらかじめ定められた期限までに完成されるものとし，その期間内に要求仕様は変更されないものとする．

3. 評価尺度の適用実験

3.1 データ収集源

ここでは本学部情報工学科の学生に対して行われた三つの学生実験(実験Ⅰ，実験Ⅱ，実験Ⅲ)からデータを収集した．実験Ⅰは昭和60年度第2学期に学部3年生を対象として実施した．実験Ⅱは昭和61年度第2学期に学部3年生を対象として，実験Ⅲは昭和60年度第2学期に学部2年生を対象として行った．従って，実験Ⅱと実験Ⅲは同一の学生に対して行われている．どの実験も比較的小規模なもので，開発期間も比較的短かった．なお，各学生は事前にコンパイラに関する講義[14]を受けているが，コンパイラの作成経験はない．

実験Ⅰではプログラミング言語CまたはPascalを用いて，C，PascalまたはPL/Iのサブセット(ソース言語と呼ぶ)のコンパイラを作成する．ソース言語の仕様の内で必要最低限度の部分だけがあらかじめ与えられ，各学生の判断に基づいてその拡張が許されている．更に，構文エラー発生時の処理内容の決定は学生に任されている．従って厳密にいえば，与えられた問題の難しさは学生により多少異なる．作成されたプログラムのサイズは約1,000～2,000行であった．

実験Ⅱでは，プログラミング言語Cを用いてPascalのサブセット(ソース言語)のコンパイラを作成する．ソース言語の仕様は，実験Ⅰに比べ，かなり簡略化されている．その他の特徴は実験Ⅰと同じである．作成されたプログラムのサイズは約1,000行であった．

実験Ⅲでは，いわゆる酒屋問題[4](一種の在庫管理問題)に対するプログラムを作成する．作成すべきプログラムの要求仕様は，実験Ⅰおよび実験Ⅱと異なり，厳密に定められている．従って，与えられた問題の難しさに対する各学生の間の差は小さい．プログラムサイズは約300行であった．

各学生実験ともに約40名の学生が参加した．今回用いるデータは，あらかじめ定めた期限までに実験を終了した学生に関するものに限った．その結果として，実験Ⅰからは9人(#1～#9で引用する)のデータが，実験Ⅱおよび実験Ⅲからは6人(#11～#16で引用する)

のデータが収集できた(後述の表1，表2参照)．なお，実験IIと実験IIIの中の#11～#16はそれぞれ同一の学生を表す．

3.2 評価値の算出

今回の実験では，プログラムテキストが変更される度に，プログラマにその変更理由を端末から入力させた．プログラマが実験中に作成したプログラムテキストの変更情報(変更時刻，変更箇所，変更理由)を，人手を介して分析してエラー寿命 T_e を求めた[6]．従って，プログラムテキスト中に具体化しなかったエラー(例えば，コーディング以前に発見されて取り除かれたエラーなど)はエラー e に含めない．また，シンタックスエラーもエラー e に含めないことにした．

エラー寿命 T_e を測る時間の単位としては学生が端末を操作していた時間(端末使用時間)の累積を用いる．厳密にいえば，端末使用時間はプログラム開発に費やした実際の時間を表さない．しかし，端末使用時間が自動収集可能なデータであること，および，机上でよく考えた上で端末を操作するプログラマほどエラー寿命の評価が有利になること，などを理由に用いた．

エラー寿命 T_e の総和は次式(2)のように書き換えられる．但し，avgはエラー寿命 T_e の平均値，N はエラーの総数である．

$$\sum T_e = \mathrm{avg} \cdot N \qquad (2)$$

ここで，avgと，N は共に問題の難しさ p に依存すると考えられる．従って，avgの正規化関数 $f_1(p)$ と N の正規化関数 $f_2(p)$ を導入して，式(2)より式(3)を求める．

$$\frac{\sum T_e}{f(p)} = \frac{\mathrm{avg}}{f_1(p)} \cdot \frac{N}{f_2(p)} \qquad (3)$$

但し

$$f(p) = f_1(p) \cdot f_2(p) \qquad (4)$$

とする．

各実験において各学生が用いたアルゴリズムやデータ構造はほぼ同じである．また，問題の難しさ p も各学生の間で大きな差はない．そこで，p を実験終了時のプログラムサイズ(行数) L で近似的に表すことにする．更に，avg，N は共に p に同程度依存すると仮定する．以上より，ここでは，単純に，

$$f_1(p) = f_2(p) = p \fallingdotseq L \qquad (5)$$

$$f(p) = f_1(p) \cdot f_2(p) \fallingdotseq L^2 \qquad (6)$$

と置くことにする．

3.3 実験データ

各学生実験から収集したデータ，および，それらより算出したプログラマ性能に対する評価値 s を表にまとめる．実験Iに関しては表1に示す．実験IIと実験IIIについては表2に示す．

表1より実験Iでは，学生#2のエラー数101個が最も多く，学生#7のエラー数26個が最も少ない．しかし，エラー寿命の総和は，学生#1が最大で値93750を，学生#6が最小で値11550をとっている．以上より，エラーを多く作り込む学生ほどエラー寿命の総和も大きいとは限らないことがわかる．すなわち，エラー数とエラー寿命との間には必ずしも相関がない．

表2より，6人中1人の学生#13を除けば，評価値 s は二つの実験IIと実験IIIにおいてほぼ同じになった．なお，学生#13は実験IIのコンパイラの作成において，ソース言語の文法を誤解していたことがわかった．そのため，実験IIにおいて多くのエラーを発生し評価値を低くしていた．

評価値 s の平均値は実験IIが31.3，実験IIIが54.6(学生#13を除くと実験IIが34.1，実験IIIが46.8)であ

表1 実験データ(実験I)

学生	ソース言語	プログラミング言語	プログラムサイズ(行)	総端末使用時間(分)	エラー数	エラー寿命の総和(分)	エラー寿命の平均(分)	評価値 s
#1	PL/I	C	2098	7955	77	93750	1218	46.9
#2	PL/I	C	1685	6202	101	29715	294	95.5
#3	PL/I	C	1530	5906	61	28620	469	81.8
#4	PL/I	C	1789	8021	78	67020	859	47.8
#5	C	C	1094	4754	55	32145	584	37.2
#6	C	C	1661	3463	35	11550	330	238.9
#7	PASCAL	PASCAL	2111	5838	26	24045	923	185.3
#8	C	PASCAL	1084	8651	49	66765	1363	17.6
#9	PASCAL	PASCAL	2420	5139	33	49170	1490	119.1

表2 実験データ（実験II，実験III）

学生	実験II：コンパイラ作成			実験III：酒屋問題		
	プログラムサイズ(行)	エラー寿命の総和(分)	評価値 s	プログラムサイズ(行)	エラー寿命の総和(分)	評価値 s
#11	1251	65205	24.0	322	3300	31.4
#12	963	24690	37.6	326	2430	43.7
#13	1366	107745	17.3	326	1140	93.2
#14	1260	35790	44.4	340	1800	64.2
#15	998	25830	38.6	298	2055	43.2
#16	1149	50940	25.9	296	1695	51.7

表3 成　績

学　生	成　績
#1	69.6
#2	71.5
#3	74.0
#4	65.8
#5	72.9
#6	82.8
#7	71.6
#8	69.6
#9	73.4

る．

4. 考　　察

4.1 エラー寿命と端末使用時間の比較

ここでは，エラー寿命による重み付け（**2.3**参照）がプログラマ性能の評価に有用であるか否かについて検討する．そのため，プログラマ性能に直接的に対応していると考えられる，プログラム作成に要した時間（総端末使用時間）との相関の高さを示す．

まず，表1の実験データに基づいて，エラー寿命の総和と総端末使用時間の比較を行った．その結果を図3に示す．両者の間の相関係数の値は0.82であった．

一方，エラー数と総端末使用時間の比較を行ったところ，両者の間の相関係数の値は0.45であった．

以上の二つの結果より，単純にエラー数を評価するよりも，エラー寿命による重み付けを行って評価した方が，プログラム作成の効率を良く表現していると考えられる．

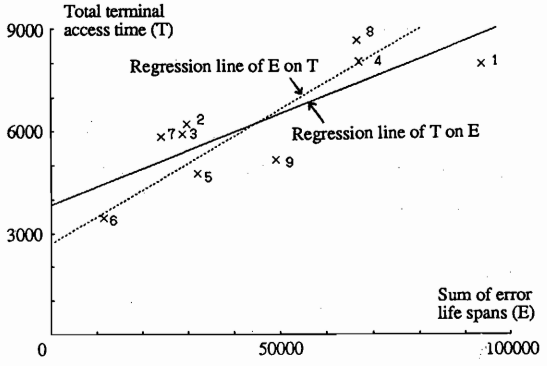


図3　エラー寿命 T_e の総和と総端末使用時間の比較
Fig. 3　Sums of error life spans vs. total terminal access times.

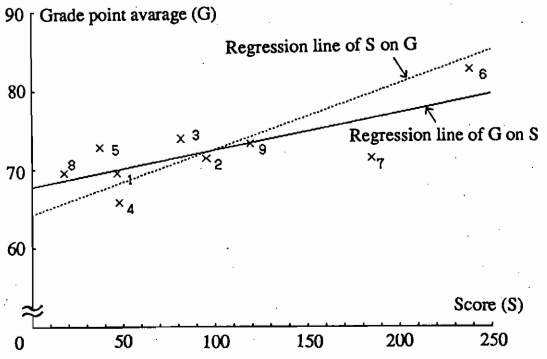


図4　評価値と成績の比較
Fig. 4　Scores vs. grade point averages.

4.2 評価値と学業成績との比較

Moher と Schneider の実験結果[9]より，プログラマの「経験（experience）」と「素質（aptitude）」がプログラマ性能を決定すると考えられる．特に，プログラマが学生の場合，「経験」は大学で修得したコンピュータサイエンスおよびプログラミングに関する講義の総数で近似的に測れると考えている[9]．一方，「素質」はそれらの講義の成績の平均点で測れると考えている．

今回の実験の場合，各学生の経験（すなわち修得したコンピュータサイエンスおよびプログラミングに関する講義の総数）はほぼ同じである．従って，もしプログラマ性能に差があるなら，それは各学生の素質の差によって説明される．

そこで，各学生のコンピュータサイエンスおよびプログラミングに関する講義の成績の平均点を調べた（表3参照）．評価値 s（表1）と成績の平均点の比較を行った結果を図4に示す．両者の間の相関係数は0.75であった．これより，提案した尺度 s はプログラマ性能をかなり良く表現していると考えられる．

4.3 同一人物の評価値の比較

同一人物のプログラマ性能は開発するプログラムや

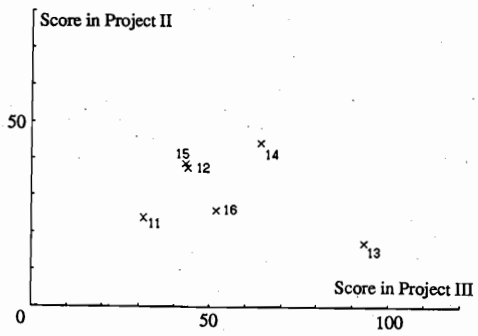


図5　二つのプログラム開発過程での評価値の比較
Fig. 5　Scores in two projects.

開発環境が異なっていてもある程度一定のはずである．従って，同一人物に対する評価値が，測定の度に大きく異なったのではあまり信頼できない．

表2は異なる二つのプログラム開発過程（実験IIと実験III）から収集した同一人物に対する評価値 s を示している．プログラマ性能の評価値 s の比較結果を図5に示す．

3.3 で述べた理由により，学生# 13を考察の対象からはずすことにする．このとき，図5中の×印は傾き45度の直線上にほぼ位置している．従って，比較的小規模なプログラム開発過程においては，今回提案した評価値 s は安定しているといえる．

5. むすび

5.1 まとめ

本論文では，エラーがソフトウェアに作り込まれてから除去されるまでの時間に注目したエラー寿命を新しく導入し，エラー寿命に基づくプログラマ性能の定式化を提案した．

更に，提案した尺度の有効性についての具体的な検証を行うため，学生実験から収集したデータを用いて考察した（大学という教育，研究の場で実験データの収集を行ったため，考察に用いた実験データはデータの総数が小さく，提案した尺度の妥当性はあくまでも限定的にしか示せていない）．まず，エラー寿命の総和と総端末使用時間を比較したところ，それらの間の相関はかなり高かった．すなわち，各学生のエラー寿命に基づく評価値と筆者らの直観的な評価とが一致した．更に，各学生のコンピュータサイエンスおよびプログラミングに関する講義の成績と比較したところ，それらの間の相関はかなり高かった．最後に，同一プログラマに対する評価値を二つの異なるプログラム開発過程で比較したところほぼ同じであった．

5.2 評価尺度の適用可能範囲について

今回の実験では，ツールを用いて自動的に実験データを収集することを考えた．具体的には，**3.2** でも述べたように，プログラムテキストの変更情報に基づいてエラー寿命を測定し，評価値 s を算出した．従って，評価の対象となっているのは，主として，コーディング以降の活動におけるプログラマ性能である．

もし，例えば設計段階で作成されるプロダクト（設計書）の変更情報が収集可能であれば，「設計に関する」プログラマ性能も評価値 s で議論できる．

更に，**2.** ではプログラマが1人で，設計，コーディング，デバッグの各段階を行うプログラム開発過程を仮定していた．しかしエラー寿命の考えは，これらの各段階に独立に適用できる．また，プログラマが1人でなく，数人のチームでプログラム開発を行う場合でもエラー寿命を測ることができる．その場合のエラー寿命はチームの性能を表すことになる．

5.3 正規化の指針について

提案した評価値の算出方法では，**3.2** の式(5)，(6)に示すように，実験終了時のプログラムサイズで正規化を行っている．これには，(1)与えられた問題の難しさに大きな差がない，(2)プログラム中で用いるアルゴリズムやデータ構造がほぼ同じである，ということが仮定されている．

従って，例えば簡単なファイル処理を実行する事務処理プログラムと複雑な処理を実行するシステムプログラム（OS，コンパイラ，等）の場合のように，比較すべきプログラムの間でアルゴリズムやデータ構造の複雑さに大きな差が存在するときにはその点を十分に考慮した正規化が必要になる．

また，プログラム中に多くのエラーが残ったままでは精度の高い評価値は期待できない．そこで，テストの手法，残存エラー数，等も考慮すべきである．

今回の実験では，エラー寿命を測る時間の単位として端末使用時間を採用した．しかし，プログラマの作業形態などを考慮して，プログラマが単位時間当りに費やす労力が一定と見なせるものであれば他のものでも良い．他の候補としては，例えばカレンダー時間，CPU時間，コマンドの使用回数などがある．

5.4 今後の課題

大学の教育の現場で実験データの収集を行ったため，考察に用いた実験データの総数は小さくなっている．提案した評価尺度の妥当性を十分に示すためにも，多

人数のプログラマ（例えば，企業におけるプログラマ）を対象にした適用実験を行うことは重要な今後の課題の一つである．

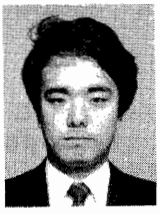
エラー寿命の算出は，人手を介して行った．そのため，多くの時間と労力が必要であった．プログラム中のエラーを特定する技術など，今のところ自動化が困難な問題がエラー寿命の算出には含まれている．現在，自動収集が容易なデータにのみ基づいてエラー寿命を近似的に求める方法について検討中である[5]．これが実現すれば，エラー寿命に基づく情報をプログラマに実時間でフィードバックすることも可能になる．

謝辞 本研究を進めるにあたり，有益なご助言を頂いた大阪大学基礎工学部の杉山裕二助教授に深謝する．また，実験IIIのデータ収集に関してご協力を頂いた大阪大学情報処理教育センターの工藤英男氏に感謝する．

文　献

(1) V. R. Basili and H. D. Rombach : "Tailoring the software process and project goals and environments", Proc. of 9th ICSE, pp. 345-357 (1987).
(2) B. W. Boehm : "Software Engineering Economics", Prentice-Hall (1981).
(3) 福田昌弘："バグ重大度の定量化の試み"，情報処理学会研究報告，SW-32-2 (1983).
(4) 二村，雨宮，山崎，淵："新しいプログラミング・パラダイムによる共通問題の設計"，情報処理，**26**, 5, pp. 458-468 (昭 60-05).
(5) K. Matsumoto, K. Inoue, H. Kudo, Y. Sugiyama and K. Torii : "Error life span and programmer performance", Proc. of 11th COMPSAC, pp. 259-265 (1987).
(6) 松本，大西，井上，工藤，杉山，鳥居："プログラム作成能力の評価尺度とデータ収集ツール"，情報処理学会研究報告，SW-50-6 (1986).
(7) H. Mills : "Software development", IEEE Trans. Software Eng., **SE-2**, 4, pp. 265-273 (1976).
(8) 宮本　勲："ソフトウェア・エンジニアリング：現状と展望"，TBS 出版会 (昭 57).
(9) T. Moher and G. M. Schneider : "Methods for improving controlled experimentation in software engineering", Proc. 5th ICSE pp. 224-233 (1981).
(10) 大西，松本，杉山，鳥居："ソフトウェア開発におけるメトリックス環境"，情報処理学会全国大会，1G-1 (1986).
(11) H. Sackman W. J. Erikson and E. E. Grant : "Exploratory experimental studies comparing online and offline programming performance", Commun. ACM, **11**, 1, pp. 3-11 (1968).
(12) D. Weiss : "Evaluating software development by error analysis : The data from the architecture research facility", J. Syst. & Software, 1, pp. 57-70 (1979).
(13) D. M. Weiss and V. R. Basili : "Evaluating software development by analysis of changes : Some data from the software engineering laboratory", IEEE Trans. Software. Eng., **SE-11**, 2, pp. 157-168 (1985).
(14) N. Wirth : "Algorithm＋Data Structure＝Programs", Prentice-Hall (1976).
(15) "IEEE Standard Glossary of Software Engineering Terminology", IEEE, 342 E. 47th St., New York, Rep. IEEE-Std-729-1983 (1983).

（昭和 63 年 4 月 11 日受付，6 月 7 日再受付）



松本　健一

昭 60 阪大・基礎工・情報卒．現在，同大大学院後期課程在学中．ソフトウェアの生産性や信頼性に関する研究に従事．情報処理学会会員．

井上　克郎

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59～61 ハワイ大助教授．関数型言語の処理系等の研究に従事．工博．情報処理学会，ACM，IEEE 各会員．

菊野　亨

昭 45 阪大・基礎工・制御卒．昭 50 同大大学院博士課程了．同年広島大・工・講師．同大助教授を経て，昭 62 阪大・基礎工・情報助教授．工博．主に分散処理システム，VLSI 向きアルゴリズム，組合せ最適化問題の解法に関する研究に従事．情報処理学会，ACM，IEEE 各会員．

鳥居　宏次

昭 37 阪大・工・通信卒．昭 42 同大大学院博士課程了．工博．同年電気試験所（現電子技術総合研究所）入所．昭 50 ソフトウェア部言語処理研究室室長．昭 59 阪大・基礎工・情報教授．ソフトウェア工学の研究に従事．情報処理学会，日本ソフトウェア科学会，人工知能学会，ACM，IEEE 各会員．