プログラム開発におけるチーム性能のモデルに基づく実験的評価†

——プログラマ性能モデルの拡張——

松本健一†† 楠本真二††
菊野亨†† 鳥居宏次††

ソフトウェアの大規模化に伴い，ソフトウェア開発の作業はプログラマ1人ではなく，複数のプログラマから構成されるチームによって行われる．したがって，ソフトウェア開発における生産性について考えるとき，個々のプログラマの性能だけでなく，チームの性能を定量的に評価することが重要である．これまでに，プログラマの性能を測る尺度として，エラー寿命に注目したプログラマ性能 *SI* が既に提案されている．本論文では，プログラマ性能 *SI* を拡張して，チームの性能を定義し，その実験的評価を試みる．ここでは，次の3種類のチーム性能モデルを定義する．(1)チーム性能 *ST*1…チームを構成する各プログラマの性能 *SI* の和をチーム性能とする．(2)チーム性能 *ST*2…チームを構成する各プログラマの性能 *SI* の平均値をチーム性能とする．(3)チーム性能 *ST*3…チーム全体を1人の仮想的なプログラマとみなしたときの，仮想プログラマの性能 *SI* をチーム性能とする．あるコンピュータメーカの新人研修における実際のプログラム開発過程に，チーム性能モデルを適用した結果，チーム性能 *ST*3 が最も適していることが分かった．次に，チームを構成する各プログラマに対し，性能 *SI* に応じた作業量を割り当てることによってチーム性能 *ST*3 を最大にできることを示す．

1. はじめに

ソフトウェアが大規模化し，複雑になるのに伴い，その開発作業はプログラマ1人ではなく，複数のプログラマから構成されるチームによって行われるようになってきた．したがって，例えばソフトウェア開発における生産性について考えるとき，個々のプログラマの性能だけでなく，チームとしての性能を定量的に評価することが重要である．しかし，チームの性能の定量的評価についての研究は，筆者らの知る限り，ほとんどなされていないのが現状である．

チームによるソフトウェア開発の成功事例の報告としては Baker のチーフ・プログラマ・チーム[1]がある．ここでは，チーム内における作業分担を徹底し，各人の能力を最大限に発揮することにより，生産性の向上を目指すことを目的としている．

一方，モデルに基づいたチーム構成の議論が Scott ら[15]によって報告されている．Scott らは文献 15) で，チームの構成員である各プログラマを一台のプロセッサに対応付け，プログラム開発チームをマルチプロセッサ上のコンピュータネットワークとしてモデル化している．シミュレーション実験によって，ネットワークの形状（対話が行われるプログラマの対を列挙したもの），ネットワーク上のプロセッサ数（プログラマの数），および，性能の高いプロセッサの配置（リーダの選定方法）がチーム全体の生産性に与える影響について解析している．

チームによるソフトウェア開発では，チームを構成する各プログラマの性能だけでなく，チームの構成方法やチーム内での作業分担など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指摘されている[11]．しかし，実際のソフトウェア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．特に，個々のプログラマの性能とチームの性能との関係や，各プログラマに割り当てられる作業量とチームの性能との関係については，ほとんど分かっていない．

一方，プログラマ個人の性能を測る尺度については，プログラマの生産活動を十分に反映したものが多く提案されている．プログラミング性能[14]，プログラマ生産性[4]，プログラマ能力[3]などがその代表的なものである．筆者らも，同様の概念として，エラー寿命に注目したプログラマ性能 *SI*[7)-9)] を提案し，その有効性の確認を1985年10月～1986年2月に実施した実際のソフトウェア開発過程から収集したデータに基づいて行ってきている[7),8)]．さらに，プログラマ性能の

† An Experimental Evaluation of Team Performance in Program Development Based on Model—Extension of Programmer Performance Model—by KEN-ICHI MATSUMOTO, SHINJI KUSUMOTO, TOHRU KIKUNO and KOJI TORII (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University).

†† 大阪大学基礎工学部情報工学科

評価を自動化したプロトタイプシステムを UNIX 上で実現しており，このシステムを利用すれば実際のソフトウェア開発過程に対する実験的評価が比較的容易に行える[9]．

本論文では，各プログラマの性能を定量的に評価する尺度であるエラー寿命に基づくプログラマ性能 SI[7)-9)] を拡張して，チームの性能を定義することを試みる．ここでは次の3種類のチーム性能モデルを導入し，これらのモデルの妥当性検証のための適用実験を行う．

（1） チームを構成する各プログラマの性能 SI の和をチーム性能 ($ST1$) とする．

（2） チームを構成する各プログラマの性能 SI の平均値をチーム性能 ($ST2$) とする．

（3） チーム全体を1人の仮想的なプログラマとみなしたときの，仮想プログラマの性能 SI をチーム性能 ($ST3$) とする．

あるコンピュータメーカの新人研修におけるプログラム開発から実際に収集したデータに基づいて行った実験的評価の結果として，チーム性能 $ST3$ が最も適していることを示す．さらに，チーム性能 $ST3$ の定義の下にチーム性能を最大にする，チーム内での各プログラマへの作業量の割当方法について議論する．具体的には，チームを構成する各プログラマに対し，性能 SI の値に比例した作業量を割り当てることによってチーム性能 $ST3$ を最大にできる可能性があることを示す．

以下に，本論文の構成について簡単にまとめる．先ず，2章では，文献 7)〜9) を参照して，エラー寿命とそれに基づくプログラマ性能モデル SI について紹介する．次に3章では，プログラマ性能モデル SI を拡張して，チーム性能のモデル ST を新しく定義する．ここでは3つの基本的な立場に対応して，3つのモデル $ST1$, $ST2$, $ST3$ を定義する．4章では，1988年4月〜8月に，あるコンピュータメーカの新人研修における実際のプログラム開発過程を対象として実施されたチーム性能モデルの適用実験について述べる．次に，5章では，チームを構成する各プログラマに対して，プログラマ性能モデル SI による評価を収集した実験データ（個人の開発作業工数）を用いて行う．さらに，チーム性能モデル $ST1$, $ST2$, $ST3$ の間の比較評価を収集した実験データ（チームとしての開発作業工数）に基づいて行う．引き続き6章では，チーム性能モデル $ST3$ の下で，チーム性能を最大にする作業量の決定方法について議論する．最後に7章では，今後の課題などについて簡単にまとめる．

2. プログラマ性能のモデル

2.1 エラー寿命 T_e

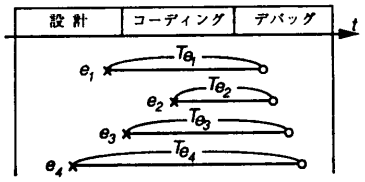
Weiss と Basili は文献 16) で，プログラムテキストの変更情報に基づいたソフトウェア開発過程の評価を行っている．その中で，ソフトウェアシステム中にエラーが存在していた時間の長さが有益な情報になることを指摘している．ここでは，この時間の候補として，エラーの寿命を次のように定義する[7),8)]．

エラー e がフォールトとして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 e に対するエラー寿命 T_e と定める*．**図1**にエラー寿命 T_e の例を示す．同図において×印はエラーがフォールトとしてソフトウェアに具体的に現れた時刻を，○印はそのフォールトが取り除かれた時刻を示す．

なお，エラー寿命に似た概念はこれまでにも提案されている．例えば，Mills[10] は最終的に開発されたシステムの品質を測るためにエラー日数 (error days) という概念を提案している．ここでエラー日数とは，各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を合計したものである．

2.2 プログラマ性能

プログラマ性能とは，基本的には，プログラマをソフトウェアを生産する1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表す尺度である．プログラマ性能と同様の概念としては，プログラミング性能[14]，プログラマ生産性[4]，プログラマ能力[3] など



× エラー e_i の混入
○ エラー e_i の除去
Te_i エラー e_i の寿命

図1 エラー寿命
Fig. 1 Error life span.

* IEEE 標準[17]の定義によると，ソフトウェアの開発作業において人間がおかす誤りをエラー (error)，エラーがソフトウェア中に具体化したものをフォールト (fault) と呼ぶ．本論文でもこの IEEE 標準に従うものとする．(なお，文献 10), 16) においてはここでいうフォールトをエラーと呼んでいる．)

がある．

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[3),14)]．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が求められている．

プログラマ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．しかし，ソフトウェアの生産性や信頼性に与える影響の度合がどのエラーも同じであるとは考えにくい．そこで，プログラマ性能を評価するためには，単にエラー数を数えるのではなく，それぞれのエラーが持つ影響度に応じた重み付けをした上で総和をとる必要のあることが指摘されている[6)]．

その重みとしてエラー寿命を用いることを，筆者らは既に文献 7)，8) で提案してきた．以降，エラー寿命に基づいてプログラマ性能を定義する枠組みをプログラマ性能モデル SI と表す．このモデル SI の基本的な考え方は，「性能の高いプログラマほどエラーをおかさず，仮にエラーをおかしたとしてもその結果として具体化したフォールトを短い期間で取り除くことができる」と言える．ただし，プログラマが1人で設計，コーディングからテスト，デバッグまでを行うものとする．（チームでソフトウェア開発を行う場合には3章で述べるチーム性能モデルを用いる．）

2.3 プログラマ性能モデル SI

上述の考えに基づき，プログラマ性能の評価尺度を次のように定義する．

まず，プログラマの性能は，ある1つのソフトウェア開発過程においてそのプログラマがおかしたすべてのエラーの寿命 T_i の総和で表されるものとする．ただし，エラー数，および，エラー寿命はプログラマに与えられた問題（仕様）の難しさ p にも依存すると考える．したがって，エラー寿命の総和を $f(p)$ で正規化する．ここで f は正規化関数である．さらに，性能の高いプログラマほどプログラマ性能の値が大きくなるようにする．このとき，プログラマ性能の評価尺度は次式(1)のように定めることができる[7),8)]．

$$SI=\left(\frac{\sum T_i}{f(p)}\right)^{-1} \qquad (1)$$

以降，この式(1)をプログラマ性能の評価尺度 SI と呼ぶ．

2.4 評価尺度 SI の検証

プログラマ性能の評価尺度 SI の妥当性の検証を，大学における学生実験（1985年10月～1986年2月に実施した）から収集したデータに基づいて試みている[7),8)]．これらの評価においては，正規化関数 $f(p)$ として，最終プログラムサイズ L の2乗 (L^2) を用いる．（その理由については文献 7)，8) に詳しく述べているので，それらを参照されたい．）

したがって，式(1)は次式(2)のように書き換えられる．

$$SI=\left(\frac{\sum T_i}{L^2}\right)^{-1} \qquad (2)$$

式(2)で定義される評価尺度 SI の妥当性に関して，次の(1)～(3)に示す結果を既に得ている[7),8)]．

(1) エラー数の総和よりもエラー寿命の総和の方が，プログラム開発に要した工数（すなわち，生産性）との間に高い相関がみられた．（相関係数の値は前者が0.45で，後者が0.82であった．）

(2) Moher と Schneider の実験結果[12)]より，学生プログラマの性能を決定する主な要因は，大学におけるコンピュータサイエンスの講義の成績である．この成績の平均点と評価尺度 SI による評価値との間に高い相関（相関係数の値は0.75であった）がみられた．

(3) 異なる2つのプログラム開発過程から測定した同一プログラマの SI の値はほぼ同一であった．

以上の結果より，式(2)で定義される評価尺度 SI がプログラマ性能の評価尺度として妥当であると考えられる．

3. チーム性能モデルの定義

ここでは2.2節，2.3節で述べたプログラマ性能の評価尺度 SI を拡張して，n ($n\geq 2$) 人のプログラマで構成されるチームの性能を測る尺度を新しく定義する．なお，定義に当たっては，プログラマ性能とチーム性能の間の関係に注目して次の(1)～(3)に示す3つの異なるモデルを導入する．これらの比較検討を行うため，本論文では実際のプログラム開発過程にそれぞれのモデルの適用実験を行う（詳細は4章で述べる）．

便宜的に，プログラマ j ($1\leq j\leq n$) に対する SI, L, $\sum T_i$ を添字 j を付けてそれぞれ SI_j, L_j, E_j と表す．したがって，式(2)よりプログラマ j の性能の評価尺度は次式(3)となる．

$$SI_j=\left(\frac{E_j}{L_j^2}\right)^{-1} \quad (3)$$

(1) チーム性能モデル $ST1$…チームを構成する各プログラマのプログラマ性能 SI の和がチーム性能であると定める.

容易に分かるように,このモデル $ST1$ の定義では,チーム作業による作業効率の向上(例えば,知識や技術の共有による効率の向上)や,逆に作業効率の低下(例えば,プログラマ間の対話のためのオーバヘッドによる効率の低下)は考慮されない.このモデルではチーム性能の評価尺度を次式(4)として定義する.

$$ST1=\sum_{j=1}^{n} SI_j \quad (4)$$

(2) チーム性能モデル $ST2$…チームを構成する各プログラマのプログラマ性能 SI の平均値がチーム性能であると定める.

定義より明らかなように,このモデル $ST2$ ではチーム作業による作業効率の向上と低下を考慮している.しかし,作業効率への影響を個別に取り上げるのではなく,全体としてプログラマ性能が平均化されるという立場をとっている.このモデルではチーム性能の評価尺度を次式(5)で定義する.

$$ST2=\frac{1}{n}\sum_{j=1}^{n} SI_j \quad (5)$$

(3) チーム性能モデル $ST3$…チーム全体(n 人のプログラマ)を1人の仮想的なプログラマとみなす.そのときの仮想プログラマに対するプログラマ性能 SI がチーム性能であると定める.

このモデル $ST3$ では n 人のプログラマによるチーム作業が全体的には,1人のプログラマの作業とみなせるという立場をとる.このモデルではチーム性能の評価尺度を,式(2)と同様の考え方に立って,次式(6)として定義する.

$$ST3=\left(\frac{\sum_{j=1}^{n} E_j}{\left(\sum_{j=1}^{n} L_j\right)^2}\right)^{-1} \quad (6)$$

4. モデルの適用実験

4.1 データ収集源

チーム性能の評価尺度の実験的評価を行うため,あるコンピュータメーカにおける新人研修(1988年4月～8月に実施した)からデータを収集した.本研修の主な特徴は次のとおりである.

(1) 8つのチーム(#1～#8で表す)がプログラム開発を行う.各チームは同じ事務処理用のファイル処理システムを COBOL を用いて作成する.

(2) ファイル処理システムは 18 個のモジュールから構成されている.作成すべきモジュールの各プログラマへの割当は各チームに任されている.

(3) 各チームは3人～5人のプログラマで構成される.チームの構成は研修中の成績等に基づき,チーム間で能力差が生じないように教官によって決定された.

(4) 各チームに2台の作業用端末が割り当てられる.

4.2 計測の自動化

2.4 節で引用した学生実験[7)-9)]では,プログラマが実験中に作成したすべての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,人手を介して分析を行いエラー寿命を求めていた.この分析には多くの時間と労力がかかるため,今回の実験のように多くのプログラマのデータの分析を必要とする場合には,このままの適用は事実上不可能である.

そこで,テキストの変更情報を機械的に処理し,エラー,および,各エラーの寿命を近似的に求める方法について検討する.(この近似的方法の詳細については文献 9) で述べている.)

Dunsmore と Gannon の実験結果[5)],および,2.4節の学生実験でのエラー寿命の分析結果[7)]より,ここでは次の2つを仮定する.

(1) 各編集作業において行われるプログラムテキストの“変更”はプログラム中のフォールトを取り除くためのものである.

(2) 各編集作業で“変更”された行の集合を,それが“作成”された編集作業ごとに分類する.こうして構成される各部分集合は1つのエラーに対応している.

これらの仮定により,仮定(2)中の各部分集合ごとにエラー寿命が定義される.具体的には,各部分集合に属する行ごとに,それらの行を“作成”するための編集作業の時刻から,それらを“変更”するための編集作業の時刻までの経過時間がエラー寿命として定義される[9)].

図2を用いて具体的に説明する.同図で t_i ($0\leq i\leq$

3) は i 回目の編集作業が終了した時刻を表す．P_i は時刻 t_i において作成されたプログラムテキスト，l_j $(1\leq j\leq 5)$ は時刻 t_3 において変更された行を表す．この図の時刻 t_3 に注目する．仮定(2)より，変更された行を2つの部分集合 $\{l_1, l_2, l_3\}$ と $\{l_4, l_5\}$ に分ける．この編集作業では，2つの部分集合に対応して2つのエラーが取り除かれたものとする．さらに，集合 $\{l_1, l_2, l_3\}$ に対応するエラーの寿命を t_3-t_1 とし，集合 $\{l_4, l_5\}$ に対応するエラーの寿命を t_3-t_2 とする．

この定義によるエラー寿命（の近似値）は計算機を用いて自動計測が可能であり，そのためのツールをUNIX 上で既に開発している[9]．

この考え方に従った方法で，学生実験における各プログラマのエラー寿命の総和の推定を行った[9]．エラー寿命の総和の人手による実測値と，提案する上述の方法により近似的に求めたその推定値との相関を**図3**に示す．図3における相関係数の値は0.90であった．したがって，上述の自動計測の方法は，エラー寿命の総和をかなりよく推定できることが分かった*．今回の新人研修への適用実験ではこの自動計測の方法を採用する．

なお，2.4 節の学生実験ではエラー寿命を測る時間として端末使用時間を用いた．それは開発作業全体に対する端末上での作業の割合が高かったからである．一方，今回の実験では，4.1 節の特徴(4)より端末上での作業の割合はそれほど高くないことが予想される．そこで，端末上での作業も含め，設計，コーディング，単体テスト，結合テスト等に要した時間を各プログラマに申告させた．そして**図4**に示すように，

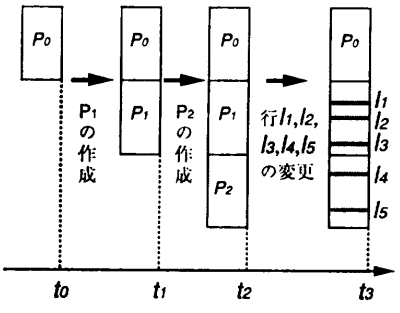


図2 エラー寿命の推定
Fig. 2 Estimation of error life span.

* 現時点では，評価尺度 SI はプログラマ性能を相対的に比較，評価するものであり，SI の値の絶対値に意味はない．したがって，エラー寿命の総和も，その絶対値を推定する必要はなく，実測値と相関の高い値を推定値とすれば十分である．この意味での推定精度の高さを表しているのが，図3における相関係数の値である．

ツールにより自動計測した端末使用時間とプログラマからの申告時間を併合して新しく作業時間を定義した．この作業時間に基づいてエラー寿命を求めることにした．

4.3 収集データ

収集データの評価に当たっては，18個のモジュールの中から次の2つの条件を満足するモジュールにだけ注目した．その結果，9個のモジュールが選択され，評価に用いられた．

(1) モジュールのサイズに関する条件……8チームの平均で，サイズが100行以上のモジュール．（サイズが余りにも小さいモジュールは評価の対象から除いた．）

(2) モジュールの機能に関する条件……8チームの平均で，データ定義部分のサイズがモジュール全体のサイズの50%以下のモジュール．（データ定義部だけから構成されているモジュールは評価の対象から除いた．）

表1に新人研修から収集，分析した各チームのデータをまとめる．構成員 $m1$，$m2$ 等は各チームを構成するプログラマを表す．プログラムサイズ L_j（行）は各プログラマ m_j が作成したモジュールサイズの合計

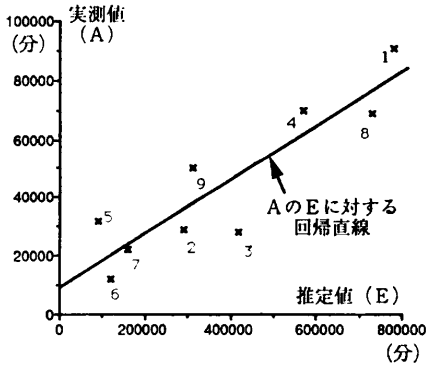


図3 実測値と推定値の比較
Fig. 3 Actual values vs. estimated values of $\sum T_e$.

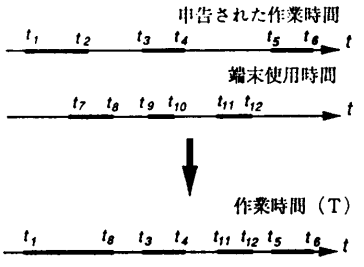


図4 作業時間の定義
Fig. 4 Definition of programming efforts.

表 1 実験データ
Table 1 Data of experiment.

チーム	構成員	プログラムサイズ L_j	エラー寿命の総和 E_j (分)	総作業時間 T_j (分)	プログラマ性能評価値 SI_j
#1	$m1$	289	1490	2009	56
	$m2$	263	10608	5488	7
	$m3$	385	7192	2652	21
	$m4$	137	6109	3633	3
	$m5$	95	6679	3523	1
#2	$m1$	365	7899	3999	17
	$m2$	278	8510	2706	9
	$m3$	249	6877	2730	9
	$m4$	155	1855	3766	12
	$m5$	107	101	2646	113
#3	$m1$	221	13329	3730	4
	$m2$	600	37620	3409	10
	$m3$	362	27689	4809	5
#4	$m1$	333	22972	4354	5
	$m2$	230	4896	3039	11
	$m3$	364	3970	4220	33
	$m4$	319	21612	3214	5
#5	$m1$	393	12035	3681	13
	$m2$	270	1569	4061	46
	$m3$	342	11907	4173	10
	$m4$	240	15147	3886	4
#6	$m1$	569	22470	3429	14
	$m2$	375	11789	3243	12
	$m3$	155	1818	2874	13
#7	$m1$	387	17634	3768	8
	$m2$	328	14194	3407	8
	$m3$	264	4747	2704	15
	$m4$	126	5092	3627	3
	$m5$	172	208	2467	142
#8	$m1$	583	14497	4426	23
	$m2$	203	2268	3211	18
	$m3$	233	14775	4699	4
	$m4$	169	25621	5366	1

である．次に，エラー寿命の総和 E_j (分) は 4.2 節で説明した仮定の下でツールを用いて自動計測した値である．また，総作業時間 T_j (分) は図 4 の方法で求めた作業時間（端末使用時間と申告された時間を併合した時間）の合計を表す．最後に，評価値 SI_j は各プログラマ m_j のプログラマ性能を表しており，式(3)を用いて求めた．

5. モデルの比較

5.1 エラー寿命と作業時間に関する考察

まず，チームによるソフトウェア開発を行っている各プログラマのプログラマ性能の分析を行う．表 1 から算出される，各プログラマのエラー寿命の総和と総作業時間の関係を**図 5** に示す．この場合の相関係数の値は 0.46 である．2.4 節で述べた学生実験（すなわち，1 人のプログラマが単独で開発を行うソフトウェア開発）の場合のエラー寿命の総和と総作業時間の相関係数の値は 0.82 であった[7]．したがって，チームを構成したことによってこの相関係数の値がかなり低くなったことが分かる．（もちろん，データ収集の対象者が同一でないので，断定的な結論は導けない．）

次に，この相関係数の値の低下の原因について考察する．チームによる開発という作業形態そのものが各プログラマに大きな影響を与えているためと考えられる．

エラー寿命への影響について考える．例えば，チームによるコードレビュが行われると，1 人のプログラマが単独で開発を行う場合に比べ，幾つかのエラーの寿命は短くなる．一方，異なるプログラマが開発しているモジュール間のインタフェース部に相当するモジュール内のフォールトは，プログラマ間の対話がうまく行われないと発見が困難になり，そのフォールトの原因となったエラーの寿命は長くなる．

次に，作業時間への影響について考える．他のプログラマの協力の下でモジュールの開発が行われるならば，その作業に要する時間は短くなる．一方，他のプログラマの作業への協力のために必要となる時間や，他のプログラマとの対話に要する時間などは，1 人のプログラマが単独で開発を行う場合には本来必要のないものである．したがって，これらの時間によってチーム開発におけるプログラマの作業時間が長くなる．

上述のように，1 人のプログラマによる開発に比べ，チームによる開発の場合にはその作業形態上の特

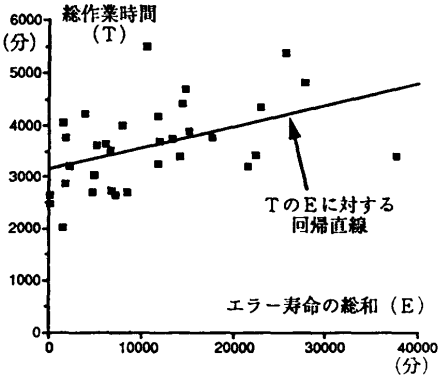


図 5 エラー寿命の総和と総作業時間
Fig 5 Sums of error life spans vs. total programming efforts.

徴がエラー寿命と作業時間の関係をより複雑なものにしていると考えられる．

5.2 3つの評価尺度の比較

ここでは，3章で定義したチーム性能に対する3つの評価尺度 $ST1$, $ST2$, $ST3$ の比較を，表1に示した収集データに基づいて行う．

表2に8つのチーム（#1〜#8）に対する評価尺度 $ST1$, $ST2$, $ST3$ の値とチームのデバッグ時間（分）を示す．ここで，チームのデバッグ時間とは，各構成員が独自に行った単体テストの終了後に，チームの構成員全員が集まって実行する結合テストにおけるデバッグ時間である．このデバッグ時間は，プログラム開発におけるチームの生産性を評価する尺度の1つの有力な候補であると考えられる．

表2より，どのチームに関しても3つの評価尺度の値には次式(7)のような関係が見られる．

$$ST2<ST3<ST1 \qquad (7)$$

次に，**表3**に3つの評価尺度 $ST1$, $ST2$, $ST3$ の値とチームのデバッグ時間の間の相関係数の値をまとめる．表3より $ST1$ と $ST2$ の値の間には非常に高い相関がある（相関係数の値は0.99となっている）ことが分かる．ただし，この相関の高さは，もし各チームごとのプログラマ数に大きな差がなければ，$ST1$ と $ST2$ の定義式から導かれる当然の結果である．

さらに，表3の中でチームのデバッグ時間と各評価尺度の値との相関係数に注目すると，3つの尺度の中では $ST3$ の場合が最も高い．このときの相関係数の値（絶対値）は0.83であり，2.4節で述べた学生実験における SI に関する相関係数の値と比べても，十分に大きい値となっている．したがって，3つの評価尺度の中では $ST3$ が，チーム性能を評価するのに最も適していると考えられる．

表2 チーム性能とチームのデバッグ時間
Table 2 Team performance and team debugging efforts.

チーム	$ST1$	$ST2$	$ST3$	チームのデバッグ時間（分）
#1	88	18	43	490
#2	160	32	53	460
#3	19	6	18	2200
#4	54	14	29	2170
#5	73	18	38	980
#6	39	13	33	650
#7	176	35	39	570
#8	46	12	25	1430

表3 チーム性能とチームのデバッグ時間との相関係数
Table 3 Coefficients of correlation among team performance and team debugging efforts.

	$ST1$	$ST2$	$ST3$
$ST2$	0.99	—	—
$ST3$	0.80	0.79	—
チームのデバッグ時間	−0.69	−0.66	−0.83

6. モデル $ST3$ に基づく作業量の割当

6.1 プログラマの作業量とチーム性能

現実のソフトウェア開発では，限られた性能を持った，しかも限られた人数のプログラマによってチームを構成しなければならない場合が多い．したがって，各プログラマの性能を最大限に発揮させ，チーム全体の性能を高くできるようなチーム内での作業量の決定方法について検討することは非常に重要である．

ここでは，チーム性能の評価尺度 $ST3$ の値を最大にすることを目指したチーム内での各プログラマの作業量の決定方法について議論する．

まず，$ST3$ の値がとりうる範囲について考える．$ST3$ の定義式(6)より次の関係が導かれる．

$$ST3=\left[\frac{\sum_{j=1}^{n} E_j}{\left(\sum_{j=1}^{n} L_j\right)^2}\right]^{-1} \leq \sum_{j=1}^{n}\left(\frac{E_j}{L_j^2}\right)^{-1} \qquad (8)$$

ただし，等号が成立するのは，

$$\frac{L_1}{SI_1}=\frac{L_2}{SI_2}=\cdots=\frac{L_n}{SI_n} \qquad (9)$$

のときである．この式(9)より次のことが分かる．チーム性能の値が最大となるのは，そのチームを構成する各プログラマ j が自分のプログラマ性能 SI_j に比例したサイズ L_j のプログラムを開発するときである．すなわち，式(9)を満足するように各プログラマの作業量が決定されるとき $ST3$ の値は最大となり，式(3)，(4)，(8)より

$$ST3=\sum_{j=1}^{n} SI_j=ST1 \qquad (10)$$

が導かれる．

表1に示した実験データに対する各プログラマの作業量と各チームの性能の関係を**表4**に示す．表4中の K_j は L_j/SI_j を表す．ここで，SI_j はプログラマ j のプログラマ性能，L_j はプログラマ j の実際の作業量（開発したモジュールサイズの合計）である．同表で“—”は対応するプログラマが存在しないことを示す．K_{opt} は $ST3$ が最大となる K_j の最適値を表し，次

表 4 プログラマの作業量とチーム性能の関係
Table 4 Relations between programmer activities and team performance.

チーム	K_1	K_2	K_3	K_4	K_5	K_{opt}	$\frac{\sum_{j=1}^{n}\|K_j-K_{opt}\|}{K_{opt}}$	$\frac{ST1-ST3}{ST1}$(%)
#1	5.2	37.6	18.3	45.7	95.0	13.3	11.4	51
#2	21.5	30.9	27.7	12.9	0.9	7.2	9.8	67
#3	55.3	60.0	72.4	—	—	62.3	0.3	5
#4	66.6	20.9	11.0	63.8	—	23.1	4.2	46
#5	30.2	5.9	34.2	60.0	—	17.1	4.9	48
#6	40.6	31.3	11.9	—	—	28.2	1.1	15
#7	48.4	41.0	17.6	42.0	1.2	7.3	17.3	78
#8	25.3	11.3	58.3	169.0	—	25.8	7.4	46

式(11)で定義される.

$$K_{opt}=\frac{\text{開発するプログラムサイズの合計}}{\text{チーム内のプログラマ性能の和}} \quad (11)$$

この K_j と K_{opt} の差（つまり，各プログラマの比例配分されなかった作業量の割合）を次式(12)で評価する.

$$\frac{\sum_{j=1}^{n}|K_j-K_{opt}|}{K_{opt}} \quad (12)$$

また，チーム性能の最大値（式(10)より，この最大値は $ST1$ に等しい）に比べて，実際のチーム性能の値 $ST3$ がどの程度低下しているのかを次式(13)で評価する.

$$\frac{ST1-ST3}{ST1}\times 100(\%) \quad (13)$$

表4の K_j と K_{opt} の差（式(12)）について見ると，チーム #3, #6 が式(9)にほぼ近いのに対し，チーム #1, #2, #7 では非常に偏った作業量の割当が行われていることが分かる．次に，チーム性能の低下率（式(13)）に関しては，K_j と K_{opt} の差と同様の傾向が見られる．すなわち，チーム #3, #6 が非常に小さいのに対し，チーム #1, #2, #7 が大きくなっている.

表2において $ST3$ の値が最も大きいチーム #2 と $ST3$ の値が最も小さいチーム #3 のチーム性能の低下率（式(13)）について調べてみる．チーム #3 がわずかに5％であるのに対し，チーム #2 は 67% もの低下を起こしている．したがって，各プログラマへの作業量の比例配分化に関しては，チーム性能とは逆に，チーム #3 がチーム #2 よりはるかに優れていたことが分かった.

6.2 最適な作業量の決定

プログラマの作業量とチーム性能との関係に基づき，チーム性能の評価値 $ST3$ を最大にする各プログラマの作業量の決定方法について議論する．ここでは，あるプロジェクト P におけるプログラム開発を考える．このプロジェクトには n 人のプログラマが参加するものとする．今，プロジェクト P に対し次の(1)～(3)を仮定する.

(1) 各プログラマのプログラマ性能は既知である（例えば，P と類似しており，しかも既に完了しているプロジェクト P' におけるプログラマ性能の値を利用する）.

(2) プロジェクト P ではチームによる開発を行うため，各プログラマのプログラマ性能は一般に変化する．しかし，その変化の大きさはプログラマ性能に比例し，かつ，その割合は一定であるとする.

(3) プロジェクト P で開発するプログラムの最終的なサイズ，および，各部分（モジュール等）の最終的なサイズが予測可能である.

このとき，チーム性能の評価値 $ST3$ を最大にするためには，各プログラマ j に対し式(9)で定義される作業量 L_j を割り当てればよい．もし，式(9)を完全に満たす割当が存在しない場合には，式(12)の値が可能な限り小さくなるように L_j の値を決めることが必要となる.

7. おわりに

本論文では，チームによるソフトウェア開発を対象として，チームとしての生産性を定量的に評価する試みを行った．具体的には既にその有効性を実験的に示しているプログラマ性能モデル SI を拡張して，チーム性能モデル ST を提案した．さらに，モデル ST の有効性の検証を行うため，企業の新人研修におけるプログラム開発過程から実際にデータを収集し，そのデータに基づいて分析を行った.

今回の実験的評価は企業の新人研修から収集したデータを用いて行っているため，得られた結果はあくまでも限定的なものである．チーム性能の評価尺度の有効性を十分に示すには，より多くの，かつ，より多様なソフトウェア開発過程を対象とした適用実験を行うことが今後は必要である．その場合，モデルを適用するソフトウェア開発プロジェクトの特性を把握し，プロジェクトの実情にあった適用方法を検討する必要がある．例えば，プログラミング言語としてアセンブリ言語と高級言語を併用するプロジェクトにおいては，プログラムテキスト1行に記述することのできる命令，処理の内容や複雑さを言語間で比較し，その差によってLを正規化する必要がある．

また，データ収集に伴うプログラマへの負荷ができる限り小さく，かつ，信頼性の高いデータが収集できる仕組みの開発が必要となる．さらに，ソフトウェアの生産性や信頼性の向上を目指すという本研究の本来の目標からすれば，収集，分析したデータをプログラマに効果的にフィードバックする仕組み[2),13)]も重要である．現在，データの収集，管理，分析，そして，フィードバックを一貫して支援するソフトウェア開発環境について検討中である．なお，UNIX上で稼働するプロトタイプシステム[9)]の開発を既に完了しており，幾つかのソフトウェア開発過程に適用して，事例データの蓄積と分析を進めている．

参考文献

1) Baker, F.T.: Chief Programmer Team Management of Production Programming, *IBM Syst. J.*, Vol. 11, No. 1, pp. 56-73 (1972).
2) Basili, V. R. and Rombach, H. D.: The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Trans. Softw. Eng.*, Vol. SE-14, No. 6, pp. 758-773 (1988).
3) Boehm, B. W.: *Software Engineering Economics*, Prentice-Hall (1981).
4) Chen, E. T.: Program Complexity and Programmer Productivity, *IEEE Trans. Softw. Eng.*, Vol. SE-4, No. 3, pp. 187-194 (1978).
5) Dunsmore, H. E. and Gannon, J. D.: Analysis of the Effects of Programming Factors on Programming Effort, *J. Syst. Softw.*, Vol. 1, pp. 141-153 (1980).
6) 福田：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).
7) 松本，井上，菊野，鳥居：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―，電子情報通信学会論文誌 D, Vol. J71-D, No. 10, pp. 1959-1965 (1988).
8) Matsumoto, K., Inoue, K., Kudo, H., Sugiyama, Y. and Torii, K.: Error Life Span and Programmer Performance, *Proc. 11th COMPSAC*, pp. 259-265 (1987).
9) 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).
10) Mills, H.: Software Development, *IEEE Trans. Softw. Eng.*, Vol. SE-2, No. 4, pp. 265-273 (1976).
11) 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).
12) Moher, T. and Schneider, G. M.: Methods for Improving Controlled Experimentation in Software Engineering, *Proc. 5th ICSE*, pp. 224-233 (1981).
13) 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，第33回情報処理学会全国大会論文集，1G-1 (1986).
14) Sackman, H., Erikson, W. J. and Grant, E. E.: Exploratory Experimental Studies Comparing Online and Offline Programming Performance, *Comm. ACM*, Vol. 11, No. 1, pp. 3-11 (1968).
15) Scott, R. F. and Simmons, D. B.: Predicting Programming Group Productivity—A Communications Model, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 4, pp. 411-414 (1975).
16) Weiss, D. M. and Basili, V. R.: Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 2, pp. 157-168 (1985).
17) IEEE Standard Glossary of Software Engineering Terminology, IEEE, Rep. IEEE-Std-729-1983 (1983).

(平成元年8月31日受付)
(平成2年9月11日採録)

松本 健一（正会員）

昭和37年生．昭和60年大阪大学基礎工学部情報工学科卒業．平成元年同大学院博士課程中退．同年同大学基礎工学部助手．工学博士．ソフトウェア開発における計測環境，ソフトウェアの品質保証の枠組に関する研究に従事．電子情報通信学会会員．

楠本 真二（学生会員）

昭和40年生．昭和63年大阪大学基礎工学部情報工学科卒業．現在，同大学院博士課程在学中．ソフトウェアの生産性や品質の定量的評価，チームによるソフトウェア開発に関する研究に従事．電子情報通信学会会員．

菊野 亨（正会員）

昭和22年生．昭和45年大阪大学基礎工学部制御工学科卒業．昭和50年同大学院博士課程修了．同年広島大学工学部講師．同大助教授を経て，昭和62年大阪大学基礎工学部助教授．工学博士．主に，分散処理システム，フォールトトレラントシステム，組合せ最適化問題の解法に関する研究に従事．電子情報通信学会，ACM, IEEE 各会員．

鳥居 宏次（正会員）

昭和13年生．昭和37年大阪大学工学部通信工学科卒業．昭和42年同大学院博士課程修了．工学博士．同年電気試験所（現電子技術総合研究所）入所．昭和50年ソフトウェア部言語処理研究室室長．昭和59年大阪大学基礎工学部情報工学科教授．ソフトウェア工学の研究に従事．電子情報通信学会，日本ソフトウェア学会，人工知能学会，ACM, IEEE 各会員．