プログラム・ジェネレータの生産性実験†

佐藤 匡正††

本論文では，プログラム・ジェネレータ方式の代表的な言語である RPG の生産性特性について把握する．プログラム・ジェネレータ方式は記述を必要な部分のみに絞り込めるために，一般には生産性が高いと言われている．この方式では，適用性の広い共通的なプログラムの骨組を用意しておき，与えられた要求条件に合うようにパラメータ化した機能で肉付けをすることによってプログラムが生成される．プログラムが簡単にできるかどうかはこの骨組が要求条件に合うかどうかにかかっている．用意されている骨組が与えられた要求条件に合えば著しい効果が得られるが，そうでなければこの効果は得にくい．ジェネレータ方式のこのような生産性特性は汎用言語にはないクリティカルな特性である．したがって，プログラム・ジェネレータの適用に当たってはこの特性を理解しておく必要がある．この特性を把握するために，RPG, COBOL, および RPG 改を用いてプログラムの作成実験を行った．ここで，RPG 改とは RPG の手続きを高水準化したものである．実験の結果，①プログラム作成の時間を減少させるには，作成量を減らすのが効果的であるという通説を裏付けるデータが得られた．② RPG は作成量が少なくてすむ課題には効果的だが，作成量が多くなるような課題では COBOL 並の生産性であり，効果的とはいえない．③言語による差より個人差の幅の方が大きい，ことを明らかにした．

1. 序　論

プログラムを能率よく作成するための手法の一つとしてプログラム・ジェネレータ方式がある．この方式では記述を必要な部分に絞り込めるので COBOL などの汎用言語に比べてプログラム作成能率が良く生産性が高いとされている．この特徴からワークステーションやオフィスコンピュータにサポートされ広く使われている．

このプログラム・ジェネレータ方式は古くから実用化されており，代表的なものに RPG (Report Program Generator) がある．RPG は 1960 年代から今日に至るまで 20 年以上も使用され続けている長い歴史をもつ言語である．当初はリポート・プログラム・ジェネレータという名前が物語るように，経営事務等で使用される多様な報告書（リポート）を容易に作成することに限定して考案されたものであった．したがって，機能も入出力ファイルは一つに限定され，また演算機能も四則演算など報告書作成に必要なものに限定されていた．その後，複数の入出力ファイルが扱える突き合わせ（マッチング）処理機能の実現や演算機能の充実などが図られ事務処理全般に適用されるようになった．これを基本としてさらに適用域拡大をねらった改版が加えられ，ファイルの入出力タイミングの指定や通信処理記述機能を追加した RPGII が生まれた．最近では，データベースやワークステーションとのやりとりが扱える RPGIII[6] に進んでいる．

このような発展の経緯をもつが，RPG の基本的な概念は変わっていない．プログラムサイクルと呼ばれる「プログラムの骨組」が用意されている．この骨組にはファイル装置などとの基本的な入出力契機が規定されている．この骨組に肉が付けられ一本のプログラムになる．肉に相当するのは利用者の指示によって選択されるシステム機能や，RPG 利用者自身が手続きとして作成する処理である．利用者の指示は手続きを除けば表に記入する（フィルインブランク）程度の簡単なものですむ．

RPG 生産性の良さはこのプログラムの骨組にある．この骨組によって，プログラムとしての記述量が減らせ，機能のパラメータ化が可能となっている．この RPG の手法は洋服の 'イージオーダ' 方式になぞらえることができる．利用者の要望を予想して類型的な半完成品をあらかじめ用意しておく．この中から利用者の要望に近いものを選びだす．裾の長さなどの細部の要望には個別に加工調整を施すものである．このように利用されればプログラムは簡単に生産性高く作成できる．

しかしながら，RPG の生産性はすべての領域で良好であるというわけではない．不向きな領域がある．イージオーダ方式では並はずれのサイズなど，利用者の要望が想定している範囲を越えると適用しにくくなる．RPG の場合にも同様な問題点がある．細部まで要求どおりの仕様にしようとすると RPG のもつ「お

† A Productivity Measurement Experiment on Program Generators by TADAMASA SATOH (NTT Communications and Information Processing Laboratories).

†† 日本電信電話(株)情報通信処理研究所

仕着せ」機能では不足なことがある．この不足がごくわずかでも小回りが利きにくいので不足分は自分で書かねばならない．この場合は，不足分だけでなく関連する処理の記述も必要となる．このために記述量が増え RPG の効果は失われる．例えば，入力文字列の妥当性チェックは RPG では不足なことがある．この不足分は自分で書かねばならない．書くのはこのチェックだけでなくこれに付随する誤りの後処理も含まれる．一般に，誤りの後処理の種類は多様である．エラー表示の契機や修復処理は複雑な要求条件になることも少なくない．この場合には，要求条件を満たすように半固定のプログラム骨組を命令文や標識の組合せによって調整する．この調整は実現不可能な場合もあり一般に複雑になる．

このように RPG は，要求される条件によっては生産性が期待されたほどでもないということもある．「骨組」を持たない手続き型言語にはない特性である．

実際のシステムでは，いろいろな業務プログラムが要求される．要求は RPG 向きのものも多いが，そうでないものも混在する．RPG を適用するにあたってはこの RPG 向きでない要求条件のプログラム開発が全体の成否の鍵となる．RPG 向きのプログラムでの生産性特性はもとより，不向きなものでの特性が把握できれば開発管理がしやすくなる．このためには，実務的なレベルでの定量的なデータが欲しい．ここでは，このデータを，プログラム作成実験を通じて得ることにする．

このような実験は，職業プログラマを被験者とした Sackman[4]，学生を使った広瀬ら[5]の業績がある．しかし，これらの結果はシステム向きのデータであったり，実務性に欠けていたりする．実務に使えるような，業務プログラムでの生産性データが要望される．筆者らはこの観点から生産性実験を行い，汎用言語 COBOL と比較して必ずしも効果的ではないことを指摘した[1]．この原因は RPG のプログラム作成量が減らないことによる．その後，この改善策の実現を試みた[2]．そこで，この改善策の効果も含めて RPG の総合的な生産性について定量的に論ずることにする．

本論文では，第2章で言語の差異を浮き彫りにできる実験方法を計画し，第3章でこの実験によって得られたデータを定量的に分析し RPG の生産性特性を明らかにする．

2. 生産性実験

2.1 実験の目的

RPG の生産性特性を現状と潜在能力の二面から把握する．「現状」は汎用言語との比較によって，「潜在能力」はプログラム作成量の削減度合によって，生産性実験を通じて把握する．つまり，

(目的 1) 汎用言語 COBOL に対する RPG の生産性特性を得る．

(目的 2) 手続き記述を高水準化した RPG の方言の効果を得る．

ここでいう RPG の方言とは，多項目演算や IF 文，繰り返しの DO 文などによって手続き記述を高水準化したものである[2),3)]．以下，これを RPG 改と呼ぶ．

2.2 「生産性」の内容

「生産性」という用語の一般的な定義はまだ確定していない．「作成能率」に近い意味と解釈できる．そこで，生産性特性の要因としてはプログラム作成作業に関連する事項に着目する．ここでは，プログラム完成までの作成時間，プログラム作成量，修正量を取り上げる．

2.3 実験の難しさ

複数のプログラム言語の生産性の高低を論ずるための手っとり早い方法は，実際に作成してみて作成にかかった時間を計ってみればよい．この場合，作成時間が実際に即した代表値でなければ意味はない．普遍的な解釈を可能とするデータが欲しい．作成時間を決める要因が言語だけであれば一点のみのデータであっても代表値と看做せる．しかし，この時間を決定する要因には言語以外にもある．どんな人が，どんな課題を書いたか，どのように時間を計ったかなど，作成者の能力，課題の難易，計測精度に関する要因がある．これらの要因は計測しにくいだけでなく複合化しているために個々の要因を独立なものとしては扱いにくい．

しかも，実際に即したデータを得るには実際を模したプログラム作成実験でなければならない．ところが，このような実験では，実験とはいえ，一件のデータを得るのに実務的なプログラムを作成するのと変わらないほど，時間と経費が嵩む．必要とする時間と経費の制約から得られるデータは限定される．経済性，時間を勘案して，かつ言語以外の要因の影響を少なくできる実験計画が必要となる．

2.4 実験方法

基本的には，プログラム作成者ごとに，与えられた

課題を作成するのに要した時間，プログラム作成量，修正量などの量を得る．用いる言語はCOBOL, RPG, RPG 改の三種類である．プログラム作成者は職業プログラマ（14名）である．課題は実務から選んだ7題である．図1に概要を示す．

（1） 能力差への配慮

プログラム作成者は経験や才能などから生ずる能力差がある．これを緩和するためにグループ分けする．ただし，現状では能力差を知る術がないのでプログラマとしての経験年数を目安とする．グループとしての水準が同程度か否かは実験後のデータで確認する．

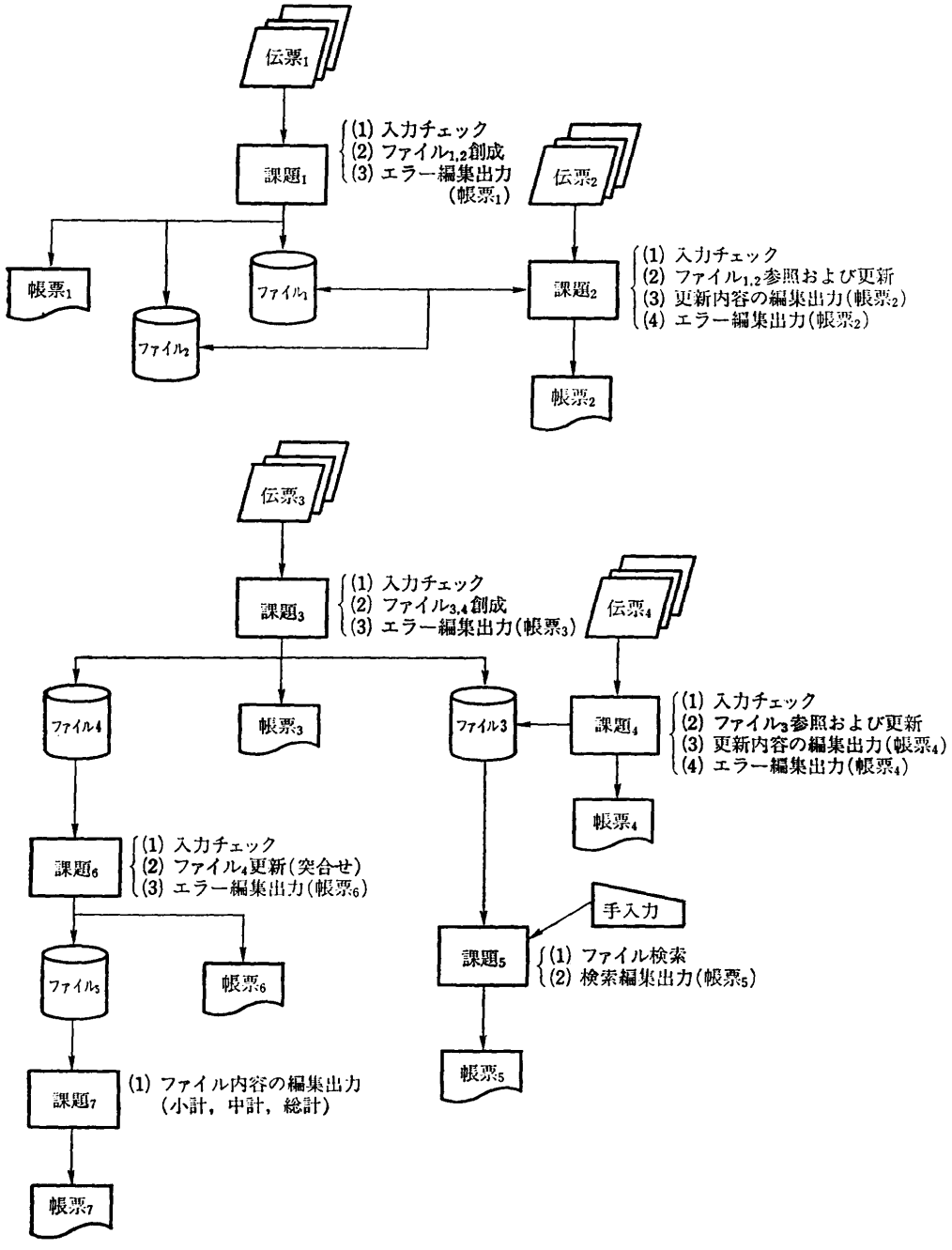

図 1 課題の構造
Fig. 1 The composition of subjects.

（2） 経験度への配慮

言語についての経験度を未知レベルに合わせる．つまり，RPG を全く知らないプログラム作成者を選ぶ．ただし，他の言語についての経験は問わない．

修得時間を短くするため，実験開始にあたって入門用講習会を開く．また，実験中に遭遇するトラブルの解決にあたって相談できる要員を配置する．なお，データの独立性を保つために作成者同士の相談は禁止する．

（3） かけもち作業の禁止

実験に専念できるように，実験期間中は他の仕事とのかけもちは禁止する．

（4） 言語の交代プログラミング

RPG と COBOL は互いに異質の言語仕様をもつ．同一の作成者が同一の問題を二つの言語を用いて作成しても言語の慣れによる生産性データへの影響は無視できる．RPG と COBOL については言語を交代して二回プログラミングする．RPG と RPG 改は類似しているので言語の影響は無視できない．交代プログラミングはしない．

2.5 計測方法

実験での計測量は主として記述量と作成時間である．記述量には，プログラム作成量と修正量がある．この単位を言語に共通な尺度である「行数」とする．行数はプログラムの記述量を表す尺度として最適とはいえない．この対案として，文字数や文数などの尺度も考えられるが，これらは区切りの単位として COBOL と RPG に共通する概念とはいえないし，行数よりも精度が優れているという保証はない．行数は量の区切りとしてはどのような言語にも共通なので，一般的に広く普及している．この普及度から精密な尺度とはいえないが行数を用いる．

プログラム作成量は，この行数を完了時点を基準として計測する．コメントやコンパイラ等の処理系への制御情報，デバグ用の記述などを除いて言語としての実質的な行数を計る．機械化可能で計測誤差は少ない．修正量はプログラムの修正に要した行数である．

これに対して，作成時間の計測では個人の手作業である．作業項目の分類の判断や精度に対する認識に個人差がある．この個人差による計測誤差を少なくするために分類の判断基準を設ける．この基準を盛り込んだ帳票を制定し，これにプログラム作成者各自が計った時間などのデータを記入する．

制定した基準の概略を次に述べる．

（1） 時間の定義

●修得時間……言語説明書を読んだり相談員との相談に要する時間である．

コーディングやデバグ中も対象である．修得のために作る練習用のプログラムの作成時間も含まれる．

●論理構築時間……課題の理解および処理ロジックの作成に要する時間である．

●コーディング時間……ソースプログラムの記述に要する時間である．デバグのためのコーディングは含まれないが，解法の行き詰まりなどでの全面的な書き直しは含まれる．

●デバグ時間……動作確認用データの作成，誤り箇所の分析，この修正，およびこれらに関する計算機操作などにかかる時間である．

●作成時間……言語を修得した後に完成までに要する時間である．論理構築時間，コーディング時間，デバグ時間の合計である．

（2） プログラム作成の完了時期

プログラム作成の完了時期は各自の判断に委ねると誤差が大きくなる．このために，共通な基準を設ける．プログラム作成者からの申告に基づいてあらかじめ用意してある試験データを投入し，動作が妥当であれば完成と看なす「受け入れ試験」方式とする．

（3） 記録表の制定と計測体制

計測は被験者各自が行う．記録の煩わしさを緩和し，計測精度を確保するために記録表を制定する．様式は作業形態に応じてデバグとその他に分け，二種類用意する．誤記や記入もれのないように，相談員を兼ねた集計担当者を設け毎日，記録のチェックおよび集計を行う．

また，作成者同士の相談や，かけもち作業の混入などの禁止事項の監視の役目も果たす．

3. 分　　析

上の生産性実験から 66 件のデータが得られた．これを**表 1** に示す．このデータに基づく考察を述べる．

（1） 生産性の分析

[観察 1] RPG と COBOL の生産性に差はない．

☆ 作成時間には言語間の差はみられない．

次が裏付けである．No. 1〜16 のデータについて，

……RPG と COBOL の作成時間(**図 2**) の比較観察．

……平均値に関する t 検定 5% 有意．

☆ 作成時間に差がないのはプログラムとしての経

表 1 実験データ
Table 1 Data.

実験 No.	作成者	課題	言語[a]	経験年数[b]	修得時間〔時間〕	計測時間〔時間〕				作成量〔ソース行数〕	デバグ回数
						論理	コーディング	デバグ	合計		
1	1	1	1	1	36.5	14.3	37.7	33.7	85.7	475	
2	2	1	1	2	8.08	5.67	32.8	37.5	76.0	466	
3	3	1	1	3	19.8	5.25	21.5	9.37	36.1	349	
4	4	1	1	4	13.5	5.05	12.5	10.5	28.1	354	
5	1	2	1	1	7.95	15.2	20.0	36.5	71.7	370	
6	2	2	1	2	0.50	7.17	9.75	16.1	33.0	406	
7	3	2	1	3	2.08	2.25	12.6	6.60	21.5	332	
8	4	2	1	4	0.98	0	12.4	5.98	18.4	325	
9	1	1	2	1	27.3	14.8	19.2	48.2	82.2	482	
10	2	1	2	2	19.5	5.17	19.1	38.7	63.0	449	
11	3	1	2	3	32.9	4.75	17.1	12.3	34.2	421	
12	4	1	2	4	12.2	2.15	15.9	15.1	33.2	515	
13	1	2	2	1	5.08	0.72	18.7	39.6	59.0	280	
14	2	2	2	2	0.33	5.00	9.42	17.4	31.8	281	
15	3	2	2	3	3.17	3.18	12.2	15.6	31.0	274	
16	4	2	2	4	0.52	0.90	7.25	10.8	19.0	270	
17	5	3	3	3	6.0	4.1	6.6	2.4	13.1	279	6
18	6	3	3	3	19.4	5.8	10.9	13.8	30.5	214	5
19	7	3	3	2	22.5	1.4	16.0	27.3	44.7	274	12
20	8	3	3	1	8.2	3.6	16.3	20.8	40.7	296	6
21	9	3	3	1	11.3	6.6	10.0	17.8	34.4	343	10
22	5	4	3	3	0	0.8	2.3	3.8	6.9	105	2
23	6	4	3	3	0.6	2.3	5.7	4.0	12.0	100	6
24	7	4	3	2	0	2.3	6.2	16.3	24.8	94	8
25	8	4	3	1	0.6	3.9	4.6	22.3	30.8	112	7
26	9	4	3	1	0.7	0.2	4.8	23.6	28.6	106	7
27	5	5	3	3	0.5	0.5	2.0	1.8	3.8	46	3
28	6	5	3	3	0	1.4	4.2	1.6	7.2	50	4
29	7	5	3	2	0	4.3	8.6	5.0	17.9	48	7
30	8	5	3	1	0.8	0.3	1.5	9.3	11.1	50	2
31	9	5	3	1	0	0	2.7	2.4	5.1	46	4
32	5	6	3	3	24.7	3.5	7.8	15.5	26.8	274	6
33	6	6	3	3	0	2.4	12.2	35.1	49.7	203	6
34	7	6	3	2	3.3	1.1	7.5	35.9	44.5	239	11
35	8	6	3	1	0	0.5	3.6	56.2	60.3	325	13
36	9	6	3	1	0	0	3.9	35.1	39.0	292	7
37	5	7	3	3	0	0.3	1.7	0.7	2.7	86	2
38	6	7	3	3	0	1.0	2.3	0.7	4.0	80	3
39	7	7	3	2	0	0	4.3	11.6	15.9	90	7
40	8	7	3	1	4.1	0	1.3	12.3	14.1	80	5
41	9	7	3	1	0	0	3.0	10.3	13.3	93	10
42	10	3	2	3	7.5	1.8	11.7	23.3	36.8	599	9
43	11	3	2	2	9.2	1.24	15.9	58.3	75.4	856	20
44	12	3	2	1	2.9	3.7	13.2	9.3	26.2	561	6
45	13	3	2	1	17.3	12.7	16.7	11.3	40.7	579	8
46	14	3	2	1	14.8	4.3	11.6	40.5	56.4	554	6
47	10	4	2	3	0.4	0.6	5.2	19.7	25.5	199	7
48	11	4	2	2	1.8	0.9	4.4	50.6	55.9	155	22
49	12	4	2	1	0.3	0.9	2.8	17.0	20.7	147	5
50	13	4	2	1	0.8	3.0	9.2	19.8	32.0	180	2
51	14	4	2	1	0	0	4.0	21.8	25.8	179	9
52	10	5	2	3	0	0.5	1.3	8.4	10.2	48	7
53	11	5	2	2	0	0.7	2.4	2.7	5.8	71	3
54	12	5	2	1	0	0.8	5.5	9.7	16.0	45	5
55	13	5	2	1	0	3.0	5.2	3.1	11.3	47	3
56	14	5	2	1	0	0	7.7	17.3	25.0	53	8
57	10	6	2	3	0	0.6	9.7	71.8	82.1	391	13
58	11	6	2	2	0.9	0.3	12.4	61.7	74.4	501	18
59	12	6	2	1	0	0	6.2	47.6	53.8	380	9
60	13	6	2	1	1.0	19.2	13.0	36.2	68.4	403	11
61	14	6	2	1	0	2.8	8.3	63.6	74.7	389	19
62	10	7	2	3	0	0	5.6	11.2	16.8	103	7
63	11	7	2	2	0	0.8	2.5	6.7	10.0	105	6
64	12	7	2	1	0	0.5	2.3	5.1	7.9	91	4
65	13	7	2	1	0	1.3	5.7	14.7	21.7	112	5
66	14	7	2	1	0	0	3.2	6.9	10.1	98	4

〈図3参照〉

[a] 言語；1: COBOL, 2: RPG, 3: RPG 改

[b] 経験年数；1: 1年未満, 2: 2年未満, 3: 3年未満, 4: 3年以上

験年数によって言語への親しみやすさが変わるためと推定される.

次が根拠である.

No. 1～16 のデータを経験年数2年で二分すると,作成時間は次のように二分される.

2年以下なら, RPG<COBOL, 2年より上なら RPG>COBOL である.

この理由は次のように推定される. 経験は主としてアセンブラ等手続き型言語によるものである. したがって, 経験の長い作成者は手続型の COBOL にはすぐ馴染むが RPG では手続きの概念が基本ではないので扱いにくい. 経験の浅い作成者ではどちらでもそれほど大きな影響はない.

☆ デバグ性は大差ない.

デバグのやりやすさの一つの指標として修正量の推移を観察する. 修正回数が長くかつ量が多ければデバグはやりにくい. この推移が類似していれば差はないと考えられる.

修正量の推移は類似している.

……**図3**の観察からいえる.

[観察 2] RPG 改は RPG より作成時間は短くてすむ.

☆ 平均作成時間はいずれの課題でも RPG 改の方が少ない.

……RPG と RPG 改の作成時間 (図 2) の比較観察からいえる.

☆ 短いのは主としてデバグ時間が少ないためである.

……RPG と RPG 改のデバグ時間 (図 2) の比較観察からいえる. RPG 改のほうが RPG よりもデバグがやりやすいことを示している. 手続き記述の高水準化の効果と推定される.

☆ 作成時間の差は言語差を表している.

● グループ全体としての能力差はない.

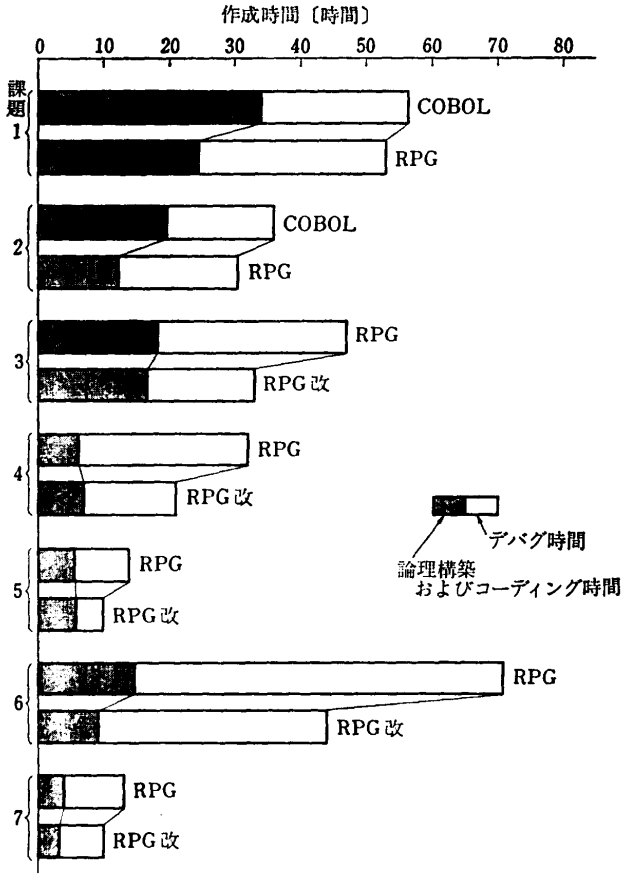

図 2 作成時間 (平均値) の比較
(COBOL-RPG-RPG 改)
Fig. 2 Comparison of average production time.

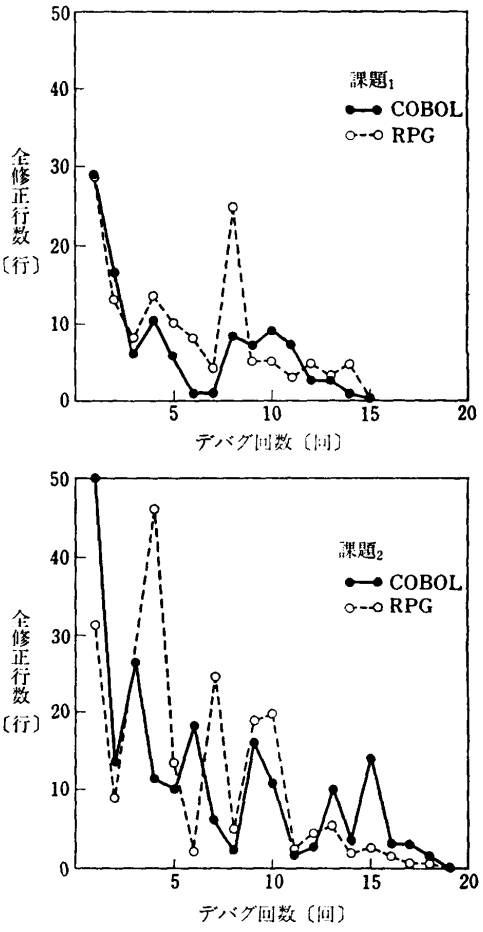

図 3 修正量の推移
Fig. 3 The change of modified lines.

表 2 作成時間と作成量の最大最小比
Table 2 The Max/Min value of production time and amount of description.

課題 / 言語	1		2		3		4		5		6		7	
	COBOL	RPG	COBOL	RPG	RPG 改	RPG	RPG 改	RPG	RPG 改	RPG	RPG 改	RPG	RPG 改	RPG
作成時間比 [Max/Min]	3.05	2.48	3.90	3.11	3.41	2.88	4.46	1.25	4.71	4.31	2.41	1.52	5.89	2.75
作成量比 [Max/Min]	1.36	1.22	1.25	1.04	1.60	1.55	1.19	1.35	1.09	1.58	1.60	1.32	1.16	1.23

……作成時間のばらつきはほぼ等しい.

RPG 改 : $\mu=137$, $\sigma=36.9$; RPG : $\mu=190$, $\sigma=42.6$

図 4 に散布図を示す.

[観察 3] 作成時間, 作成量には正の相関がある.

……単回帰によれば, $t=0.0928m+9.25$ ($r=0.724$) である. t : 作成時間 [時間], m : 作成量 [行数].

なお, 言語別の概括的な傾向は, 「RPG, RPG 改ではばらつきが大きく, COBOL では小さい. また, COBOL では作成量増加に対する作成時間の増加は大きい」といえる.

(2) 個人能力の差

[観察 4] 個人差の幅は大きい.

☆ プログラマとしての経験年数による差が大きい.

RPG と COBOL の実験では言語を交代したので全作成時間は個人的な能力を代表している. **図 5** に示す. これによれば**経験 4 年では 1 年未満の 1/3** である.

☆ 課題ごとの差は最大 6 倍である.

課題ごとの作成時間の最大値と最小値の比をとる. **表 2** に整理する. これから,

……作成時間については 1.25～6 である.

……RPG や RPG 改では個人差が多い. 言語ごとの詳細は,

COBOL : 3.05～3.90

RPG : 1.25～4.31

RPG 改 : 2.41～5.89

☆ プログラム作成量の個人差は最大で 6 割り増しである (1.04～1.60).

☆ 修得時間は個人差が最大 10 倍ある.

RPG<RPG 改<COBOL の順である (**表 3**). 平均値では 20 時間が一つの目安である.

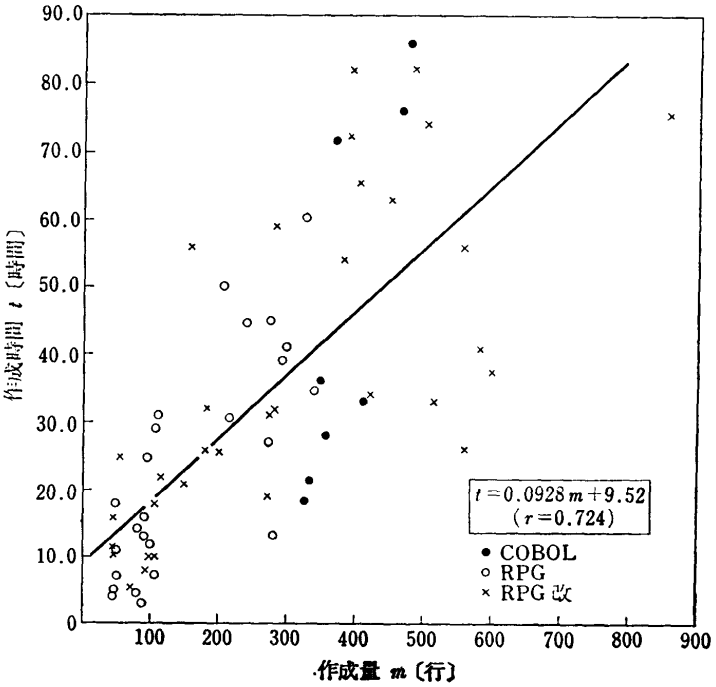

図 4 作成時間と記述量の散布図
Fig 4 The scattering chart between production time and line-of-code.

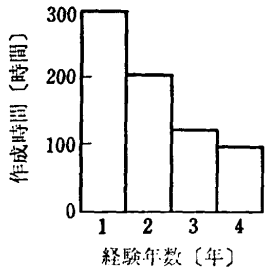

図 5 経験年数と作成時間
Fig. 5 Experience and production time.

4. 結　論

RPG, COBOL, および RPG 改を用いてプログラムの作成実験を行い, 次の観察を得た.

表 3 言語の修得時間最大最小比
Table 3 The Max/Min value of language learning time.

番号＼言語	COBOL	RPG	RPG 改
1	44.5	32.4	31.2
2	8.13	19.8	20.0
3	21.9	36.1	25.8
4	14.5	12.7	13.7
5		7.9	12.0
6		11.9	
7		3.2	
8		19.1	
9		14.8	
平均値	22.3	17.5	20.5
最大/最小	5.5	10.1	2.6

① 作成時間とプログラム作成量の間には正の相関がある.

② 作成量が同程度の課題では RPG も COBOL も生産性は変わらない.

③ RPG の手続き記述を高水準化することによって作成量の削減を図った RPG 改は作成時間が減少した.

④ 作成時間に関しての個人差の幅は最大で 6 倍である. プログラマとしての経験年数が利く. RPG は個人差が少ないという通説に従うデータは得られなかった.

これらの観察から次の結論が導かれる.

結論 1 プログラム作成の時間を減少させるには一般に, その作成量を減らすのが効果的である.

結論 2 RPG は汎用言語 COBOL に比較して作成量が少なくてすむ課題では効果的である. しかし, 作成量の多くなるような課題では効果的とはいえない. ただし, このような課題であっても COBOL 並の生産性である.

結論 3 言語による差よりも経験年数や能力などによる個人差の幅の方が大きい. どのような言語を使用するにしてもこの個人差を制御できるような管理技術の確立が当面の課題となる.

謝辞 本実験の設定および実施に協力いただいた NTT 情報処理研究所主任研究員の馬場康彦氏に心からお礼申し上げる. 併せて, 本実験の意義を理解し被験者として本実験に参加していただいたプログラマの方々に感謝する.

査読者から「観察」内の項目について重要な示唆をいただいた. 感謝する.

参考文献

1) 佐藤匡正, 馬場康彦 : RPG の生産性, 電子通信学会, 電子計算機研究会資料, EC 74-54 (1974).
2) 宮崎末広, 佐藤匡正, 馬場康彦 : TSS 向き事務処理言語 RPG, 電子通信学会総合全国大会予稿集, p. 6-196 (1976).
3) DIPS マニュアル BIG 説明書, D 12 F-10-M 003(01) 日本電信電話公社横須賀電気通信研究所 (1975).
4) Sackman, H. : *Computers, System Science, and Evolving Society—The Challenge of Man-Machine Digital System*, John Wiley & Sons, Inc. (1967). (竹中監訳 : マンマシンディジタルシステム)
5) 広瀬通孝, 石井威望 : ソフトウェアメトリックスにおけるミクロ分析の手法, 情報処理学会ソフトウェア工学研究会資料, 37-4 (1984. 7).
6) IBM システム/38 RPGⅢ 解説およびプログラマ手引き N : SC 21-7725-7.

(昭和 61 年 7 月 3 日受付)
(昭和 62 年 3 月 25 日採録)

佐藤 匡正 (正会員)

昭和 42 年横浜国立大学工学部電気工学科卒業. 同年日本電信電話公社に入社. 電気通信研究所, データ通信本部などにおいて銀行業務処理システム, 言語処理プログラムの開発に従事. 現在, 日本電信電話(株)から日本情報通信(株)に出向. プログラム設計方法とプログラム構造との関係に興味をもっている. 情報処理学会論文賞受賞 (昭和 56 年度). 電子情報通信学会会員.