プログラム構造の複雑さ尺度の評価と導出法の提案†

花田收悦†† 高橋宗雄††
永瀬淳夫††† 黒田幸明†††

ソフトウェア・メトリックスに関する研究の一つとして，プログラム構造の複雑さに着目した幾つかの構造尺度が提案されているが，プログラム構造の一面を表すものが多く構造の複雑さを網羅していない．また，実用大規模システムへの適用性について十分な評価は行われていない．本論文では，従来の代表的な構造尺度について大規模システムの開発過程で得られた信頼性データを用いて，構造尺度と信頼性（単位規模当りのバグ数）との関係を分析し，その結果，実用システムの構造尺度としては必ずしも適切ではないことを示す．次に，データ構造，データ参照，処理，制御構造およびプログラム・インタフェースの五つの部分構造に基づいて，構造の複雑さを網羅的に考慮した構造尺度の導出法を提案し，これを用いて大規模システムプログラムの構造尺度を求め信頼性との関係を分析する．分析の結果，構造の複雑さが信頼性に与える影響度合は，大規模システムプログラムの場合 15% 程度であり，残り 85% は構造の複雑さ以外の要因であることを明らかにする．

1. まえがき

近年，ソフトウェア開発や保守における品質予測や品質管理を的確に行うことをねらいとして，プログラムの信頼性を表す尺度を求める研究が行われている．これらの尺度は，大別して以下のように分類される．

（1） プログラムバグの発生傾向をみることによって，信頼性を表すもの．

（2） プログラムの計算の複雑さによって，信頼性を表すもの．

（3） プログラム構造の複雑さによって，信頼性を表すもの．

これらのうち，(1)は，プログラムをデバッグしてみない限り，信頼性の予測は不可能であるうえ，信頼性を左右する要因の抽出を目的にしているわけではないこと，また(2)は，コーディングされたプログラムから導出可能であるが，問題を解くアルゴリズムそのもののむずかしさを表現するものであり信頼性向上の要因を見つけ出すことはむずかしいことから，いずれの尺度も品質管理用として十分とはいえない．さらに(2)は，導出手順が複雑であるという欠点をもつ．これに対して(3)は，コーディングされたプログラムから機械的に導出できるうえ，複雑な構造を抽出して良形な構造に変更する手段を講ずることが可能であるなど，品質管理用としても優れている．本論文で扱う尺度もこの構造に関するものであり，以下，構造尺度またはたんに尺度とよぶことにする．

これまでに，プログラム構造の複雑さに関して，データ参照・処理や制御構造など，プログラム構造の一部の特性に着目した構造尺度がいくつか提案されている．たとえば，Halstead[1]は，データ参照とオペレータの数などに基づいて，プログラム作成に要する労力(effort)の予測値を経験的に求め，これを構造尺度として提案した．一方，制御構造に着目した尺度として，グラフ理論を応用した McCabe[2] の Cyclomatic 数が代表的である．また，Cyclomatic 数に条件式のオペランド数を加える拡張を行った Myers[3] の尺度や，さらにオペレータ数を考慮した Hansen[4] の尺度なども提案されている．

しかしながら，これらの構造尺度には，次のような問題点がある．

（1） 経験的に，あるいは理論的に導出した構造尺度は，データ参照・処理や制御構造などプログラム構造の一面を表してはいるが，構造の複雑さをどの程度網羅しているのかが明確ではない．

（2） 実用の大規模システムに適用して評価を行っているものが少ない．

（3） これまでプログラムの品質予測に用いられてきた規模[5]と比較した有意差の評価や，規模の影響の除去が行われていない．

本論文では，これまでに提案されている代表的構造尺度が大規模システムプログラムの信頼性とどのよう

† An Evaluation and Derivation Method of Complexity Measures on Program Structure by SHUETSU HANATA, MUNEO TAKAHASHI (Software Engineering Section, Yokosuka Electrical Communication Laboratory, NTT), ATSUO NAGASE and KOHMEI KURODA (Processing Programs Section, Yokosuka Electrical Communication Laboratory, NTT).
†† 日本電信電話公社横須賀電気通信研究所ソフトウェア技術研究室
††† 日本電信電話公社横須賀電気通信研究所処理プログラム研究室

な関係にあるかを，規模による構造尺度の正規化値を用いて分散分析により評価する．この結果，実用システムの構造尺度としてこれらの尺度が必ずしも十分でないことを示す．次に，網羅的に構造の複雑さを求めるために，プログラム構造を，データ構造，データ参照，処理，制御構造およびプログラム・インタフェースの五つの部分構造に分類したのち，これらの部分構造の複雑さを網羅する構造尺度の導出法を提案する．さらに，この導出法を用いて実用の大規模システムプログラム開発過程で得られたデータに基づいて，構造尺度を求める．この構造尺度によれば，プログラム構造の複雑さは，プログラムの信頼性（プログラムバグ）に15%程度影響を与えていることを，また残りの85%は構造の複雑さでは説明できない要因によっていることを示す．

2. 従来構造尺度と信頼性との関係分析

筆者らは，これまでにプログラム構造の複雑さ要因を，データ構造，データ参照，処理，制御構造およびプログラム・インタフェースの五つの部分構造ごとに列挙し，プログラムの信頼性についてプログラム・バグ数（以下バグ数と略す）を基準として，各要因との関係の分析手法を提案し，以下の項目を明らかにした[6]．

（1） バグ数と相関のありそうな要因であっても，プログラム規模の影響を除去すると相関のないものが多い．

（2） バグ数に影響を与えている複雑さの要因は，部分構造全体に分布している．

この結果は，構造尺度が次の条件を備えるべきことを示している．

（1） 尺度は，規模との関係を分析したのち，その影響を除去するなどして，バグ数との相関を調べる必要がある．

（2） 尺度は，部分構造全体を網羅するものでなければならない．

本章では，従来の尺度がこのような条件を満たしているか否かを，大規模システム開発で得られたデータを用いて分析を加える．

2.1 分析の準備

本評価に用いるサンプルおよび測定方式は，複雑さ要因の分析[6]に使用したものと同等であり，次に示すとおりである．

（1） サンプル・プログラム

PL/I風のシステム記述言語SYSL[7]で作成された全体規模が300キロステップ（ks）のシステムプログラムであり，OSのジョブ管理，障害復旧プログラムを中心に258個のモジュールである．

（2） 測定方式

コンパイラを改造したツール（MEASURE*）で計測した要因の使用数をデータベースに格納し，統計処理プログラム（COSTAT**[8]）により，各モジュール（外部手続き）ごとの尺度とバグ数との関係を求める（**図1**）．

（3） 測定項目

これまでに提案されている尺度のなかで，代表的なcyclomatic数およびeffortを対象とする．

（a） cyclomatic数

McCabeの尺度は，プログラムコントロール・フローグラフGのcyclomatic数$V(G)$であり，Gの節点，有向辺の個数をそれぞれn, eとしたとき，以下で定義される[2]．

$$V(G)=e-n+2$$

（b） effort

effort（以下Eと略す）は，プログラム内の一意なオペレータ数および一意なオペランド数をそれぞれn_1, n_2，またオペレータ総数およびオペランド総数をそれぞれN_1, N_2とし，サブプログラム数をpとしたとき，以下で定義される．

$$E=n_1N_1(N_1+N_2)\log_2(n_1+n_2)/((p+1)n_2)$$

2.2 分析手法

プログラムの規模として実行文数を基準に選び，尺

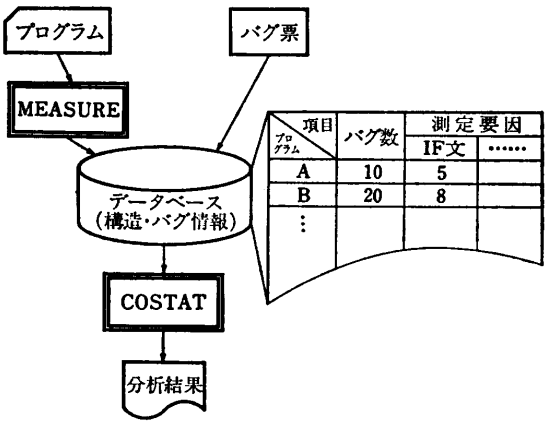

図1 複雑さとバグ数の測定方式

Fig. 1 Method to derive the relation between complexity factors and errors.

* A Module Analyzer for Program Stucture

** Conversational Statistical Analysis Program

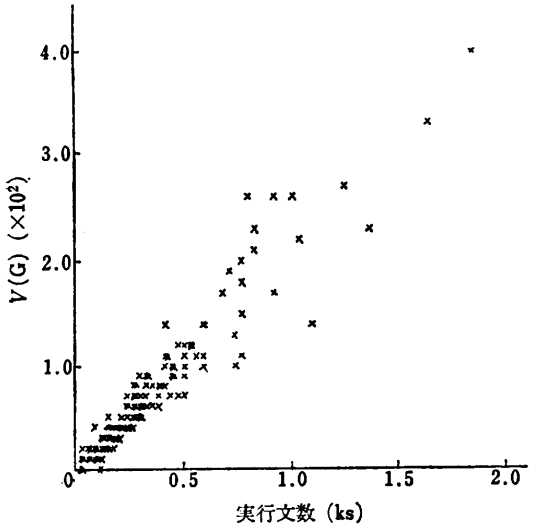

図 2　実行文数と $V(G)$ の関係
Fig. 2　Steps versus $V(G)$.

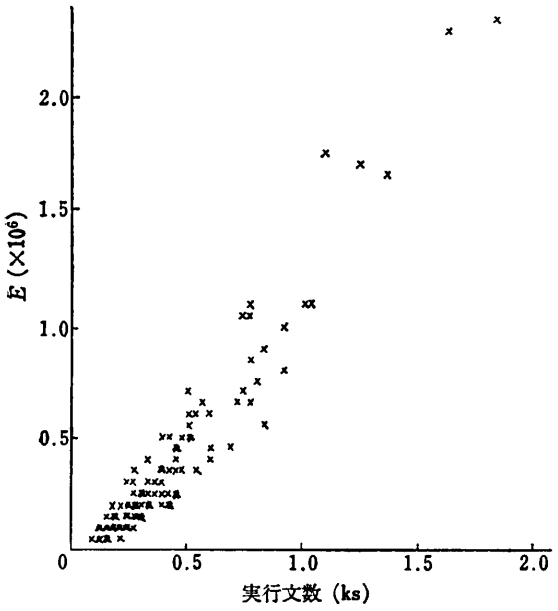

図 3　実行文数と E の関係
Fig. 3　Steps versus E.

度との関係を調べた．その結果，両者の間にはきわめて強い相関が認められた（図 2, 3 ）．この結果は従来から指摘されているもの[9]と一致する．

従来の報告では，各尺度とバグ数との関係を相関係数の大小で評価しているものが多いが，以下に示す理由により，尺度とバグ数の関係分析には不適当であると判断した．

（1）　相互に強い相関をもつ尺度の影響を除去していない．とくに，バグ数との間に強い相関のあることが認められている実行文数の影響を除去する必要がある．

（2）　相関係数は 2 変数間にリニアな関係があるか否かを判定するものであり，両者の関係がリニアでない場合には相関がないと判定されることが多い．

本論文では，上記の問題点を解決する以下の分析手法を導入する．

（1）　尺度およびバグ数をそれぞれ実行文数で正規化する．

（2）　正規化の結果，バグ数の微少な変化に対してバグ率（バグ数／実行文数）が大きく変動する実行文数の小さいモジュールは，統計処理上不適当なので除外する．この結果，分析対象モジュールは 100 ステップ以上の 127 個とする．

（3）　尺度使用率（尺度／実行文数）の低い順にモジュールを $n(>1)$ グループに分け，各グループに水準値 $l=1, \cdots, n$ を割り当て，水準値を説明変数，バグ率を基準変数とする一元配置の分散分析により尺度使用率とバグ率の関係を分析する．ここで，各グループ内のモジュール数(繰返し数)は同一になるようにする（図 4 ）．

本分析では繰返し数を大きくするため，グループ数

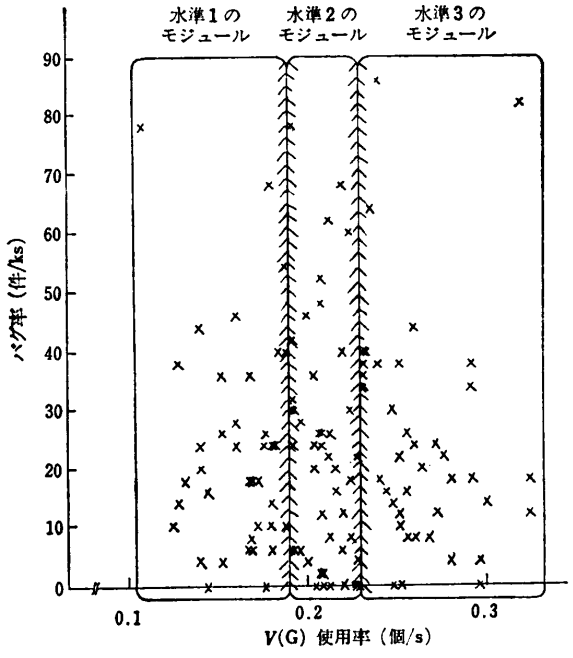

図 4　水準化の方法
Fig. 4　Classification method.

表 1 $V(G)$ と E の分散分析表
Table 1 Anova lists of $V(G)$ and E.

要因	平方和	自由度	不偏分散	F値	検定結果	純平方和	寄与率
$V(G)$	0.0005	2	0.00023	0.62		0.0000	0.0
誤差	0.0458	124	0.00037			0.0463	100.0
合計	0.0463	126				0.0463	100.0
E	0.0006	2	0.00032	0.87		0.0000	0.0
誤差	0.0456	124	0.00037			0.0463	100.0
合計	0.0462	126				0.0463	100.0

nを3とする.

2.3 分析結果

$V(G)$ およびEに関する分散分析表を**表1**に示す. この結果は，これらの尺度使用率がバグ率に影響を与えているとはいえない（換言すれば，構造の複雑さ尺度として実行文数よりも優れているとはいえない）ことを示しているが，その理由としては以下のものが考えられる.

（1） 従来の分析で有意となったものは，実行文数の影響を除去していない.

（2） $V(G)$ は制御構造の複雑さを，またEはデータ参照と処理の複雑さを包含しているが，プログラム構造の複雑さ全体を網羅しているとはいえない.

3. プログラム構造尺度の導出と分析

本章では網羅的な構造尺度の導出法を述べ，その導出法を2章に示したサンプルに適用して新尺度を求めるとともに，新尺度がバグ数にどの程度影響を与えているか推定する.

3.1 構造尺度の導出法

Zolnowski ら[10]は，複雑と思われる構造要因を直観的に列挙し，その全要因の使用率を2レベルの値をもつ質的データに変換したのち，その一次結合の値 (Complexity Score) を構造尺度としている. しかし，この導出法には次のような問題がある.

（1） バグに影響を与えていない要因も尺度に含まれている.

（2） プログラマの直観に基づいて評価したプログラムの複雑さ順位と，この尺度を比較することによって尺度の妥当性を判定しており，尺度の検証法として十分とはいえない.

本論文では，これらの問題点を解決するため，前述の五つの部分構造に含まれる複雑さ要因に基づいて，次のような手順で構造尺度 (C_i) を求める. なお，各部分構造の複雑さ要因としては，複雑さ要因の分析[6]に使用したものと同一である（**表2**）.

表 2 プログラム構造の複雑さ要因
Table 2 Complexity factors of program structure.

部分構造	複雑さの要因	理由
データ構造	・レジスタ変数	ハードウェアの意識
	・構造体	データの多さ
	・定数	意味のわかりにくさ
	・ビット列変数	意味のわかりにくさ
	・ポインタ変数	アドレスの意識
	・BASED 変数	変数割付けの意識
データ参照	・変数参照	変数参照の多さ
	・レジスタ割付け／解放文	ハードウェアの意識
	・グローバル変数参照	変数参照の非局所性
処理	・アドレス処理（ADDR 組込み関数）	アドレスの意識
	・メモリビット操作 (UNSPEC 組込み関数)	ハードウェアの意識
	・SUBSTR 組込み関数	列操作の多さ
	・型変換	処理の複雑さ
	・代入文	処理の多さ
	・ポインタ演算	アドレスの意識
	・DO ループ制御変数の変更	処理の複雑さ
制御構造	・IF 文	パスの多さ
	・GOTO 文	制御の流れの複雑さ
	・内部モジュール呼出し	変数参照の非局所性
プログラムインタフェース	・複数入口，出口	制御の流れの複雑さ
	・外部モジュールの呼出し	インタラクションの多さ
	・PROCEDURE 文のオプション（レジスタ退避・回復指定等の入口・出口処理指定）	コンベンションの複雑さ
	・入出力パラメータ	コンベンションの複雑さ

step 1: プログラムの部分構造 $S_i(i=1\sim5)$ に含まれる複雑さ要因（表2）とバグ数の関係を分散分析により求める.

step 2: S_i 内の，分析の結果有意となった j 番目の要因を e_{ij} とする.

step 3: 各サンプルモジュール $k(1\sim127)$ の e_{ij} の使用率 f_{ijk} を低い順に並べて $n(>1)$ レベルに水準分けし，水準値 $l=1, \cdots, n$ をおのおの割り当てる.

step 4: f_{ijk} を l で置き換え，これを $\overline{f_{ijk}}$ とする.

step 5: S_i ごとの尺度 C_i を次のように求める.

$$C_i=\sum_j \overline{f_{ijk}}$$

step 6: プログラム構造全体の尺度 C_t を C_i の総和により求める. すなわち,

$$C_t=\sum_i C_i$$

この方法において，step 2 でバグ数に影響のない要因は除外される. step 6 で構造全体を網羅する尺

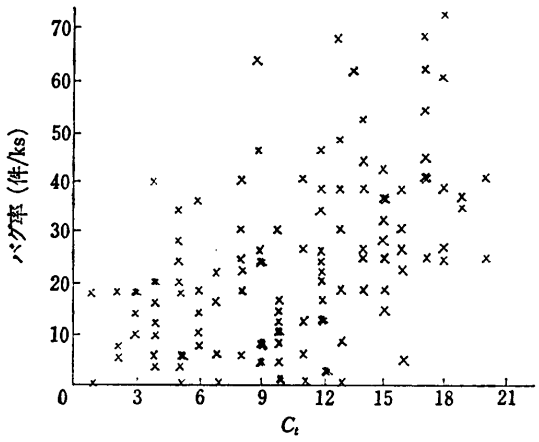

図 5 構造尺度 C_t とバグ率の関係

Fig. 5 Complexity measure C_t versus errors/step.

表 3 C_t の分散分析表

Table 3 Anova list of C_t.

要因	平方和	自由度	不偏分散	F値	検定結果	純平方和	寄与率
C_t	0.0077	2	0.00387	12.6	**	0.0071	15.5
誤差	0.0382	124	0.00031			0.0388	84.5
合計	0.0459	126				0.0459	100.0

**：99% 有意

度が構築される．なお，本分析においては，繰返し数が大きくなるように，$n=3$ とした．

3.2 尺度の評価

（1） C_t とバグ率との関係

モジュールごとの C_t とバグ率の散布図を**図 5** に示す．この結果から次のことが推測される．

（i） バグ率の変動が大きく，C_t のみでバグ率を予測することは困難である．

（ii） ある C_t 値の最高のバグ率は，C_t が大きいほど，高くなる傾向にある．このことは，構造を単純にすれば，バグ率を一定値以下に抑止しうることを示している．

（2） 分散分析

2.2 節に示した分析手法を適用して，C_t が構造尺度として有意か否かを，統計的に検定する．分析は，C_t 値を $n(>1)$ 水準化し，その値 $\bar{C}_t$ を説明変数，バグ率を基準変数とする分散分析法により，C_t が構造尺度として有意か否か検定する．本分析では $n=3$ とした．

分析結果を**表 3** に，推定値のグラフを**図 6** におのおの示す．この結果から次のことがいえる．

（i） C_t は構造尺度として有意である．

（ii） C_t によって説明できるバグ率の変動量の割合は 15% 程度であり，残り 85% は構造以外の要因の影響による．

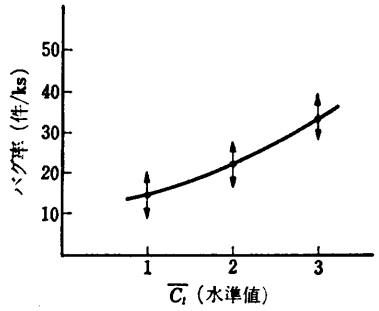

図 6 構造尺度 C_t の推定値のグラフ

Fig. 6 Estimate graph of complexity measure C_t.

3.3 構造尺度の利用法

本論文に示した尺度の導出法は，プログラム構造とバグ数に関するデータが蓄積されているシステムであれば適用可能である．これにより，種々の問題領域を代表する尺度を抽出することができる．

抽出された尺度に基づいて，ソフトウェア開発過程のコーディング終了時点で，モジュールごとに複雑さを測定することにより，次のような行動をとることが可能となり，品質管理に貢献できる．

（1） 構造尺度に従い，統計的にみてバグの発生しやすい構造をもつモジュールを指摘することができる．

（2） この場合，構造の変更が可能ならば，より簡明なものに修正する．具体的には，部分構造内の要因のうち，プログラムの信頼性に影響を及ぼすものに対して，使用制限や変更を指示することができる．

（3） 複雑さが本質的に回避できない場合であれば，そのモジュールについてとくに注意深くテストやレビュを行う個所を指摘することができる．

4. あとがき

本論文では，これまでに提案されているプログラム構造の複雑さ尺度の測定方式とプログラム信頼性との分析方式について述べ，大規模システムプログラム（規模約 300 ks）の開発過程で得られたデータを用いて，バグ数との関係を分析した．本方式の特徴は，以下のとおりである．

（1） 構造尺度と相関の強いプログラム規模の影響を除去することにより，構造尺度のバグ数への真の影響度合を求めることができる．

（2） 構造尺度とバグ数との関係がリニアでない場合でも，相互の関係を求めることができる．

また，本分析法により，これまでに提案された構造尺度はプログラム規模の影響を除去すると，バグ率への影響は少なく，プログラム構造を網羅する尺度が必要なことを明らかにした．

さらに，このような要求を満たす尺度導出法を提案し，大規模システムプログラムでは，構造尺度のバグ率への影響度合は 15% 程度であることを明らかにした．本導出法は，構造の複雑さ要因とバグに関するデータが完備されているシステムであれば適用可能である．これにより，おのおのの問題領域固有の構造尺度を得ることができるうえ，尺度による品質管理も可能であり，実用的な導出法である．

今後は，各種の問題領域を網羅する構造尺度の確立が課題である．

謝辞 おわりに，ソフトウェア・メトリックスについてご教示いただいた当研究所データ通信研究部戸田巌部長，本研究を開始するに際し，適切な問題提起とご助言をいただいた伊吹公夫特別研究室長，筑後道夫元データ処理研究部統括調査役および熱心にご討論いただいた関係各位に深く感謝します．

参 考 文 献

1) Halstead, M. H.: *Elements of Software Science*, Elsevier North-Holland, New York (1977).
2) McCabe, T. J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol. SE-2, No. 4, pp. 308-320 (1976).
3) Myers, G. J.: An Extension to the Cyclomatic Measure of Program Complexity SIGPLAN Notices, Vol. 12, No. 10, pp. 61-64 (1977).
4) Hansen, W. J.: Measure of Program Complexity by the Pair (Cyclomatic Number, Operator Count), SIGPLAN Notices, Vol. 13, No. 3, pp. 29-33 (1978).
5) Akiyama, F.: An Example of Software System Debugging, Proc. IFIP Congress, pp. 353-359 (1971).
6) 花田收悦，高橋宗雄，永瀬淳夫，黒田幸明：プログラム構造と信頼性に関する分析，情報処理学会論文誌，Vol. 23, No. 4, pp. 9-15 (1982).
7) 寺島信義：DIPS-1 における高能率システム製造用言語の実用化，情報処理，Vol. 16, No. 6, pp. 492-498 (1975).
8) 中庭源内，中野良平，下村隆夫：会話形統計解析システム，通研実報，Vol. 27, No. 4, pp. 661-674 (1978).
9) Sunohara, T., Takano, A., Uehara, K. and Ohkawa, T.: Program Complexity Measure for Software Development Management, Proc. 5th ICSE, pp. 100-106 (1981).
10) Zolnowski, J. and Simmons, D.: Measuring Program Complexity, Proc. IEEE COMPCON 77, pp. 336-340 (1977).

（昭和57年4月12日受付）
（昭和57年5月19日採録）