ソフトウェア工学 71−12
（1990．2．9）

モジュール間依存度を組み合わせたプログラムの複雑度について

梁 海述† 辻野 嘉宏†† 都倉 信樹††

†江原大学校 韓国
††大阪大学 基礎工学部

プログラムの複雑度は，ソフトエアを客観的でかつ定量的に評価する有用な尺度である．しかし，従来の尺度では，ある1つの要素だけ（例えば，プログラムのサイズ，制御流れ，データフロー情報など）で複雑度を測定している．本原稿では，従来の尺度の問題点を解決するために,モジュール間の依存度を組み合わせた新しいプログラムの複雑度尺度 MCM (Module Complexity Metrics) を提案する．提案尺度MCMはモジュール複雑度に影響を与えるいくつかの要因を統合的に測定できる長所を持つ．

The Complexity Measure of Program Based on the Inter-Module Dependency

Haesool Yang† Yoshihiro Tsujino†† and Nobuki Tokura††

† Kangweon National University, Chuncheon, KOREA
†† Faculty of Engineering Science, Osaka University, JAPAN

The program complexity is one of useful measures for objective and quantative estimation of software. However, the known methods to measure the program complexity is insufficient in that each method measures only one factor (e.g. program size, control flow, data flow or information). In this paper, we propose a new measure MCM(Module Complexity Metrics). The advantage of MCM is that it can measure comprehensively several factors influencing module complexity.

1．まえがき

最近，ソフトウェアシステムが大きくなるにつれて，ソフトウェアの開発・維持費用の軽減や品質保証に関する研究が盛んに行われてきた．その中でも，ソフトウェア複雑度などの品質を定量的に測定しようとする研究に対する関心が増えつつある．これらの定量的な測定をソフトウェアライフサイクルの各段階で用いれば，ソフトウェア品質向上や費用軽減に大きく役に立つことが示されている[1,2,3]．特に，文献(3)では，ソフトウェア複雑度が増加すればするほど，品質は落ちることも示されている．

詳細なソフトウェア複雑度測定は，以下の点でソフトウェア開発や維持保守に役に立つ．

・もし，システム設計段階で作成された仕様から複雑度の測定が可能なら，システム設計者はシステムが完成される前に，品質を落とす要素を見つけて変更できる．

・プロジェクト管理者はソフトウェア複雑度尺度を用いて，生産されるソフトウェア製品をより詳細に評価，管理できる．また，スケジューリングや費用割当などの意思決定においても役にたつ．

・システム作成者はソフトウェア複雑度をツールとして用いることで，品質のよいコード生成ができる．

・システムテスタは，ソフトウェア複雑度尺度を用いて，どの部分にテストがより必要かを決めるのに役に立つ．

このように，ソフトウェア複雑度測定はソフトウェア品質を向上させるのに役に立ち，これまで，ソフトウェア複雑度を測定する多くの尺度が提案されている．代表的な複雑度測定要素としては，プログラムの大きさ，プログラムの制御フロー，プログラムのデータ構造及び，データフローがあるが，これらの測定方法に共通の問題点はプログラムの複雑度に影響を与える1つの要素だけを尺度としてソフトウェアの複雑度を測定している点である．従って，これらはソフトウェアの複雑度を正確に表現できず[5,6]，より正確にソフトウェア複雑度を表することができる尺度が必要である．本稿では，モジュール化プログラムの複雑度に影響を与えるいくつかの要素を考慮した尺度 Module Complexity Metrics (MCM) を提案する．

提案尺度 MCMは従来の尺度では測定できないいろいろな観点の要素を測定し，組み合わせることによって，ソフトウェア品質の評価において，信頼性のある測定結果を得ることができる．また，本稿では，提案した尺度 MCMと従来の代表的な尺度との測定値を比較し，その相関関係を統計処理を用いて分析・評価することによって，提案する尺度 MCMは従来の尺度が持つ性質を保ちながら，プログラムの複雑度をより詳細に測定できることを示す．

2．従来のプログラム複雑度の測定法

2.1 複雑度測定の意味と尺度

複雑度は，次のようないくつかの意味を表す．

(1)プログラムを理解できる程度

(2)プログラムの誤りを修正，保守できる程度

(3)プログラムを他人に説明できる程度

(4)プログラムを変更できる程度

(5)プログラムを作成するときの努力の程度

プログラムの複雑度測定にはいくつかの方法が用いられているが，文献(4)ではプログラムの複雑度の値で複雑度の程度を分類する方法が示されている．また，文献(4)の方法は測定の簡単さと実用面で，複雑度測定に適合していることも示されている．文献(4)の方法によって，プログラムA，B，Cすべてが”若干難しい”であるということだけでなく，BがAより”少し難しい”ことも表すことができる．つまり，複雑度に関するプログラム間の順序関係を表すことができる．

また，プログラムの複雑度を測定する尺度としてはいくつかのものがあるが，その中で主要な尺度は次の3種に分類できる[5,6,7,8,9]．

1)プログラムの大きさに基づく尺度

2)プログラムの制御構造に基づく尺度

3)プログラムのデータ構造及びデータフローに基づく尺度

上記の方法はそれぞれプログラムの複雑度をある1つの観点から測定している．

以下，上記の尺度の代表的な測定方法を説明する．

2.2 プログラムの大きさに基づく測定法

(1)ライン数 (Lines of Code)

プログラムのライン数で複雑度を測定する方法である．ライン数が大きいプログラムが小さいプログラムより複雑であるとはいえない．従って，ライン数のみでプログラムの複雑度を評価するには問題点がある．

(2) Halsteadの Software Science

Software Science (SS) はプログラムの尺度の中で最も利用されている測定方法である[7]．この方法は，プログラムの中のオペレータとオペランドの数に基づいている．オペレータとオペランドを用いてプログラムのLengthとかプログラムの Levelなどを定義した．この尺度はプログラム作成時間と努力を予測するのに有効であることが示されている[12]．

Software Scienceは，次の5つからなる．

・Vocabulary ：$n = n1 + n2$

・Length ：$N = N1 + N2$

・Volume ：$V = N \log n$

・Level ：$L = V^*/V$

・Program Effort ：$E = V/L$

但し，n1：プログラム中のオペレータの種類の数

n2：プログラム中のオペランドの種類の数

N1：プログラム中のオペレータの出現数

N2：プログラム中のオペランドの出現数

V*：最も理想的なプログラム言語でプログラムを記述したときのプログラムの Volume

Vocabulary nの値が大きいほどプログラマがプログラムを理解するのに多くの Wordを理解すべきであること

を意味する． nを制限することによって，プログラムの複雑度を減少させることができる．

Lengthは，プログラムの大きさを測定する１つの方法である． Nの値が大きいほどプログラムの理解が容易でなく，維持するのにも多くの努力が要る． Nの値の測定はモジュールでのライン数測定の代わりにも用いられる．抽象化の Level Lは，L＝2･n2／(n1･N2)を推定値として用いられる．

Volume Vはプログラムを書き表すために必要な量つまり，ビット量である．

Program Effort Eは Volumeを Levelで割った値で，(N1+N2) log (n1+n2) ／ (2*n2/(n1*N2)) である．

2.3 プログラムの制御構造に基づく測定法

(1)McCabeの Cyclomatic数

コントロールフローグラフに基づく複雑度測定方法で，Cyclomatic C(G)は次のようにして計算される[8]．

C(G)＝e-n+2p

G ：プログラムのコントロールフローグラフ

e ：Gの辺数

n ：Gの頂点数

p ：Gの連結成分数

McCabeはあるプログラムの複雑度 C(G)の値が１０より小さいとき，適切なモジュール性を持ち，１０より大きいとき，分割するのが望ましいと提案した．

そして，C(G)の値が大きくなるほど，プログラムの開発時間，費用とエラーが発生する確率が大きいことが示された．しかし，この尺度はプログラムの制御構造にのみ依存するのでプログラム複雑度に影響を与える他の要因は反映できない問題点がある．

(2)Gilbの論理的複雑度尺度(Logical Complexity Metrics)

プログラム内にどれぐらいの Decision Making Logicがあるかを測定する方法である．

(3)Woodwardの Knot数

制御流れによって現れる Knot数の和でプログラムの複雑度を表す方法である[9]．しかし，構造化プログラムの場合，制御流れの交差点が現れないので，常に同一の複雑度 0を持ち，複雑度を区別できない問題点を持つ．

その他に，制御構造に基づく測定方法として， Scope尺度，Chenの尺度などがある．

2.4 プログラムのデータ構造とデータフローに基づく測定法

(1)Chapineの Q測定

プログラム内のデータがどのように使われるかによってデータを次のように分類する．

a) Pデータ：セグメントを生成するのに必要な入力データ

b) Mデータ：セグメント内で生成され，変化するデータ

c) Cデータ：セグメント内での制御に使われるデータ

d) Tデータ：セグメントで変化されずに使われるデータ

Q尺度は，上記のデータがそれぞれ異なる比重(Weight) を持って，プログラムの複雑度に影響を与えることに基づいている[10]．

(2)情報流れ (Information Flow)

モジュール間での情報流れに基づく測定法で，次のように定義される[3]．

$INFO = Len1*(fan\text{-}in*fan\text{-}out)^2$

Len1 ：プログラムの行数

fan-in ：そのモジュールを呼び出すモジュール数

fan-out：そのモジュールが呼び出すモジュール数

この他にも，文献 (15)で提案された尺度 D(G)などがある．

いままで述べた従来の複雑度尺度はある１つの側面だけを反映するので，プログラムの複雑度を正確に表せない問題点がある[14,15]．従って，本稿では，いくつかの側面を反映できるモジュール複雑度尺度 (MCM) を提案する．

３．モジュール複雑度尺度ＭＣＭの提案

ここでは，プログラム複雑度に影響するいくつかの要素と，プログラムの構造化を考慮したプログラムのモジュール複雑度尺度 MCM (Module Complexity Metrics) を提案する．

MCMは，モジュール内部複雑度 IMC (Inter-Module Complexity) とインタフェース複雑度 IC (Interface Complexity)及び，ネスティングレベル (NL) からなる．

これらに関する詳細は，3.1と 3.2で述べる．

3.1 モジュール内部複雑度 (IMC) の測定方法

本稿では， IMCとして文献(15)で提案したプログラムから変換された制御データグラフ上で代入－参照チェインを考慮し，データフロー情報を複雑度に反映させた複雑度尺度Ｄを用いる．

1)制御データグラフ

本稿で考える制御データグラフ G＝ (V, E) は，構造化プログラム（簡単のため，代入文と，If文， While文のみからなると考える）の制御フローから得られる有向グラフである．以下，制御の分岐がない順次的な代入文の列をブロックという．各頂点 u∈ Vは，モジュールのブロックまたは，条件判断に対応している．また，各有向辺(u,v)∈Eは，頂点 uから頂点 vへの制御フローを表す．構造化プログラムから有向グラフ Gを得るために，構造化プログラムに適用される規則を図１に示す．例えば，図１ (１) は，If文に適用される規則で，G(st1)とG(st2)は，If文の then節と else節に対応する部分グラフを表している．

2)代入－参照チェイン

モジュール内の変数 xに対して， xに代入された位置 pと pで代入された xの値が参照される位置 qの組を x

の代入－参照チェインという．このとき，位置 pでの x への代入は，位置 qで参照される xに対する１つの解釈を与える． xが参照された位置 qで qとチェインを持つ代入位置の数が多ければ， xに対する解釈の数が増加するので，モジュール内の変数 xの役割を理解することが困難になる．つまり，モジュール内での各変数に対する代入－参照チェイン数は，プログラムの複雑度と密接な関係を持っている．

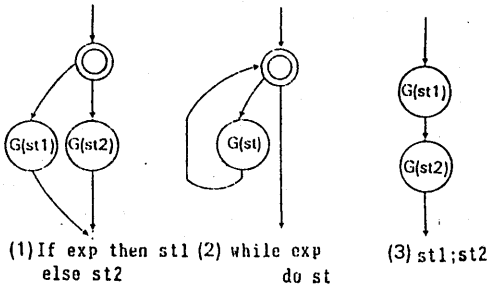

図１．制御データグラフの構成

3)モジュール内部複雑度尺度Ｄ

モジュール内のすべての変数に対する代入－参照チェインの全体数を用いて，複雑度尺度を定義する方法がある．しかし，この方法は計算が難しく，計算するのに時間がかかる．また，計算できたとしてもその値が広範囲に渡ってばらつくので，いくつかのモジュールの複雑度を比較・評価するための実用的な複雑度尺度としては不適当である．

一般的に， n個の制御構造を持つモジュールは，複雑度の値を nをパラメータとする範囲内に持つことが望ましい．つまり，複雑度が一定の範囲内に現れると，いくつかのモジュール間の相対的な複雑度の評価ができる．しかし，モジュール内のすべての変数に対して，代入－参照チェインを求めることによって，複雑度が広範囲の値をとると，複雑度に対する相対的な評価の基準がないので，複雑度の比較ができない．

このような問題点があるので，本稿では１つの変数を想定し，その変数に対する代入－参照チェイン数を用いてモジュールの複雑度尺度を定める．

4)尺度Ｄのための仮定

一般的なモジュール内での各々の制御構造とブロックにおいて，１つ以上の代入と参照が行われるが，計算と議論を簡単にするために，ここでは，次の仮定をおく．

（仮定１）制御データグラフ G内の各ブロックでは高々１回の代入があり，条件部にはない．

（仮定２）参照は，制御構造の条件部にのみ高々１回あり，ブロック内にはない．

以下，代入のないブロックは議論に影響しないので，略する．１つの制御構造について代入と参照が高々１つである．

5)尺度Ｄの定義と計算方法

Gの出口に参照があるとするとき，その参照とチェインを構成する代入を出力代入と呼ぶ．このとき，制御データグラフ Gの複雑度尺度Ｄは次のように定義される．ある１つの変数 xについての Gの出口における出力代入数を最大にするような代入の配置を持つ Gにおいて， x の代入－参照チェインの最大数をＤとする．構造化プログラムＰの複雑度Ｄは，Ｐから変換された制御データグラフ Gの複雑度Ｄで定義される．

複雑度を計算するための前段階として，出力代入数を最大にするという条件のもとで，代入－参照チェイン数を最大にする代入と参照を次のように配置する必要がある．

［アルゴリズムＣＡ］

① While構造の場合には，条件で参照され，ループ内の最初で代入されるようにする．

②If構造の場合には，条件で参照され，$H(d) \geqq H(e)$なら， eの側の分岐の最初で代入されるようにし，$H(d) < H(e)$なら， dの側の分岐の最初で代入されるようにする．但し，H(g)は制御データグラフ Gの高さと呼び，” Gの入口から出口までの閉路のない経路上での条件部の最大数”と定義する． □

このとき，次の定理が成り立つ．

［定理２］[15] ＣＡの時間計算量はＯ（n）である． □

［定理３］[15] アルゴリズムＣＡは，任意の制御データグラフGに対して，提案尺度 D(G)を実現する代入，参照の配置を決定する．そして，$D(G) = Duc(G) + m \cdot H(G)$である．但し，Duc(G)はＣＡが決定する G内の代入による代入－参照チェインの数， mは Gの入力代入の数（$m \geqq 1$）である． □

［定理４］[15] n個の制御構造を持つ構造化プログラムの提案尺度 D(G)について， $n \leqq D(G) \leqq n(n+3)/2$が成り立つ． □

3.2 インタフェース複雑度（IC）の測定

モジュール間には，いろんな情報が流れるので，それを反映させるために，ICを用いて，モジュール間の複雑度を測定する．図２にプログラムのモジュール間のインタフェースを示している．図２の (a)のＫモジュールは (b)でのＫモジュールに比べてインタフェースが複雑でない．しかし，モジュールに関する複雑度を Halstead の Software Science尺度とMcCabeの C(G)では反映できないので，図２の (a)と (b)でのＫモジュールの区別ができない．

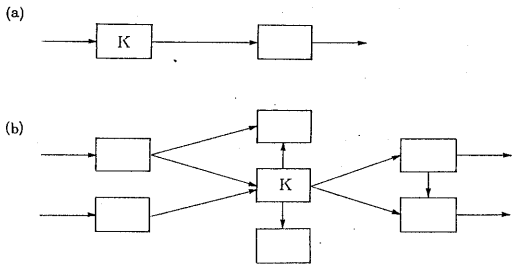

図２．モジュール間のインタフェース複雑度の比較

1)ICの測定方法

あるプログラムAが図3のようなモジュールからなる場合，モジュール間の各枝はインタフェース（または，情報流れ）を表す．ここで，情報とは，モジュール間の制御，データの受渡しを意味し，これらのインタフェースが多いモジュールはモジュール内の複雑度を増加させる要因になる．つまり，図3のjモジュールは枝1，2，3のインタフェース，kモジュールは2，3，4のインタフェースからそれぞれ影響を受けるので，i，mモジュールと比べ，複雑度増加要因が多いことが分かる．これらのモジュール間の複雑度に影響を与える要因をインタフェース複雑度（IC）であると考える．

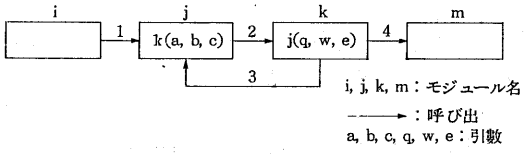

図3．モジュール間の呼び出し関係

次に，モジュール間のインタフェースに影響を与えるインタフェース要因（Interface Factor）を定義する．

［定義1］モジュール Mの FCN(Function Call Number)：M中のモジュール呼び出しにおける引数の総和．すなわち，

$$FCN(M) = \sum_{モジュールN} (Mに含まれる\ Nへの呼び出しの静的な数 * Nの引数の数) \quad \square$$

例えば，図3の j モジュール内で，k (a,b,c) でkモジュールのみを1回呼び出したとき，jモジュール内での FCNは3になる．また，jモジュールでkモジュールとmモジュールを呼び出したとき，kモジュール呼び出しときの FCNと，mモジュール呼び出しときの FCNの値の和が jモジュールの FCNになる．

［定義2］モジュール Mの AVN (Accept Variable Number)：モジュール Mが呼び出されたときの引数の総数．

$$AVN(M) = (プログラム中でMを呼び出す静的な回数 * Mの引数の数) \quad \square$$

例えば，図3のjモジュールにおいて，k (a,b,c) でkモジュールを1回呼び出したとき，kモジュールはjモジュールから1回呼び出され，引数の数は3なので，kモジュールでの AVNは3である．

［定義3］モジュール Mの EVN(External Variable Number)：モジュール Mで参照する各大局変数の出現回数の総和である．

$$EVN(M) = \sum_{x} (大局変数\ xが\ Mで参照される静的数) \quad \square$$

例えば，図3のプログラム Aにおいて，x，y，zが大局変数として宣言されたとき，jモジュールで xを2回，yを1回参照し，zを参照しないなら，jモジュールでの EVNは3になる．

ICにおいて，EVNは大局変数がプログラム内に多く存在し，各モジュール間の参照が多ければ複雑度が増加することを表す．

［定義4］RVN (Return Variable Number)：あるモジュールが他のモジュールを呼び出す場合，その呼び出しと関連のある引数は呼び出されたモジュールで使用した後，その情報を呼び出したモジュールに戻す．RVN(M)はモジュール Mが呼び出した各モジュール Nの引数の使用回数の総和である．

$$RVN(M) = \sum_{Nへの呼び出し} (Mが呼び出したモジュール\ Nでの仮引数の静的参照回数) \quad \square$$

但し，RVNは，呼び出しモジュールと呼び出されたモジュールの両方に計算される．

例えば，図3のjモジュールにおいて，k(a,b,c) でkモジュールを1回呼び出し，kモジュールで対応する仮引数がそれぞれ2回，1回，3回使われたとき，jとkモジュールでの RVNは6になる．ここで，注意すべき点は，RVNは呼び出しモジュールと呼び出されたモジュールの複雑度を同時に増加させることである．

また，2つのモジュール間のインタフェースは定義1～4で定義されたインタフェース要因の数だけでなく，参照関係のあるモジュールの NL (Nesting Level，以下に定義する）にも関係があることより，インタフェース要因の数に NL＋1の値（NLP：Nesting Level Plus）を掛けて測定した．また，NL＝0の場合，ICの値が 0になることを防ぐために，NL＋1にしている．

次に，NLを定義する．

［定義5］(a) モジュールMに制御構造が存在しないとき（つまり，順次的な命令だけからなる），Mの文の Nesting Levelは 0である．

(b) Mに制御構造が存在するときは，各文の Nesting Levelはその文を含む制御構造の数である．□

NLはプログラム複雑度を増加させる要因で，NLPは次のように計算される．

・NLPの計算例

```
if( )
  if( )      ····· ① NLP＝NL＋1＝3
    if( )
      a=b;   ····· ② NLP＝NL＋1＝4
```

上の例で，①ように1つのif構造内で他のifがネストされている場合，NLPは3で，②のように NLが3である構造の中に1つの文が存在するとき，NLPは4になる．

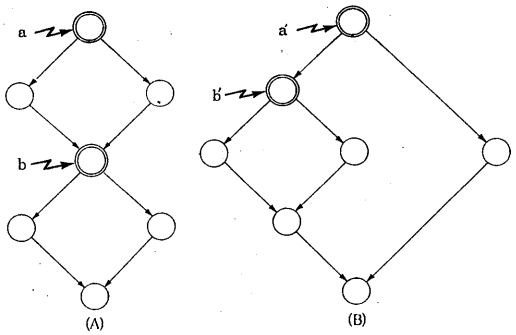

図4．モジュールでの Nesting Levelの比較

また，Nesting Levelはモジュール内の制御構造をプログラムの複雑度の要素として考慮したもので，従来のMcCabeの Cyclomatic数では測定できないモジュールの制御流れに関する複雑度を測定できる．例えば，図４の(A)と (B)モジュールにおいて，Cyclomatic数は共に３であるが，実際に(A)，(B)モジュールは相異なることが分かる．つまり，(A)，(B)モジュールにおいて，いろんな Nesting Levelの値を求めることができるが，(A)モジュールでの a，b部分と，(B)モジュールでの a',b'部分の Nesting Levelの値を比較する場合，(B)モジュールが (A)モジュールより大きいことが分かる．図４の(A)モジュールにおいて，a，b部分の Nesting Levelはそれぞれ１であるが，(B)モジュールにおいて，a'部分は Nesting Levelが１，b'部分は２であることがわかる．つまり，モジュール内の制御構造がどう構成されているかによって，文の Nesting Levelが相異なることが分かる．これは，McCabeの Cyclomatic数では反映できないモジュール構造上のネストされた制御構造を複雑度に反映させていることを表す．

2)NICの測定方法

NIC (Nested Interface Complexity) は，Nesting Level Plus (NLP) 上にインタフェース複雑度 (IC) の測定要因を考慮した要因である．つまり，各々のインタフェース要因は，その要因が生じる位置の NL (Nesting Level) が増加するほど，複雑度が増加することである．１つのモジュールでの NICは次のように計算される．

NIC＝FCN(NLP)＋AVN(NLP)＋EVN(NLP)＋RVN(NLP)

但し，FCN (NLP)は，FCNの要因が生ずる文の NLPを引数の数に乗じて計算した FCNの値であり，AVN(NLP)，EVN(NLP)，RVN(NLP)の値も同様である．例えば，図３のｊモジュールで NICを測定する場合，相対モジュールは枝１，２，３でつながっているｉ，ｋモジュールを意味する．つまり，ｊモジュールにおいて NICを測定する場合，ｉモジュールとｋモジュールとの NICを出せばよいが，ｊモジュールおいて，ｋモジュールとの NICの測定例を説明する．まず，ｋモジュールとの IC要因 (FCN，RVN，AVN) を求め，各々のICが発生するところに NLPを乗ずるが，FCN(NLP)において，NLPはKを呼び出したｊモジュールの呼び出し文での NLPで，AVN(NLP)での NLPはｊモジュールでの相対モジュールであるｋモジュールでのｊモジュールを呼び出す文の NLP値になる．

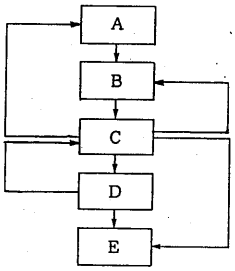

図５．モジュール間のインタフェース複雑度の例

また，ｊモジュールでｋモジュールを呼び出し，その値を帰すなら，RVN(NLP)は ｊモジュールの相対モジュールであるｋモジュールでの仮引数が参照される文の NLPを用いて計算する．また，NICの値から，次のような事実を得ることができる．例えば，図５のように，モジュールＡ～Ｅからなるプログラム Ｐを考える．Ｐにおいて，Ｃの NICの値は他のモジュールに比べて大きい値を持つので，ＣはＰの中でエラーが発生する可能性が最も大きい．また，Ｃを変更したいとき，その影響が広い範囲に及びやすいので，変更が難しいモジュールであることが分かる．

3.3 例題分析

ここでは，Ｃ言語で作成した図６のサンプルプログラムで測定した IMCと NICを示す．

```
 1 # define MAXLINE 1000 /* maximum input line size */
 2
 3 main ( ) /* find longest line */
 4 {
 5  int len ;
 6  int max ;
 7  char line [MAXLINE] ;
 8  char save [MAXLINE] ;
 9
10  max=0 ;
11  while ((len=getline (line, MAXLINE)) >0)
12     if (len> max) {
13         max=len ;
14         copy (line, save) ;
15     }
16     if (max> 0)
17      printf ("%s\n", save) ;
18 }
19
20 getline (s, lim) /* get line into s, return length */
21 char s [] ;
22 int lim ;
23 {
24   int c, i ;
25
26   for (i=1 ; i<lim-1 &&(c=getchar( ) ) ! = EOF
27                      && c ! ='\n' ; ++ i) s[i]=c ;
28   if (c==‘\n’)  {
29       s[i]=c ;
30        ++i ;
31    }
32   s[i]= ‘\0’ ;
33  return(i) ;
34 }
35
36 copy (s1, s2) /* copy s1 to s2 ; assume s2 big emough */
37 char s1[], s2[] ;
38 {
39   int i ;
40   i=0 ;
41   while ((s2[i]=s1[i])  ! =‘\0’)
42        ++i ;
43 }
```

図６．サンプルプログラム

1)IMCの測定

図６のプログラムを制御データグラフに変換して，尺

度Dによる代入－参照チェイン数でモジュール内部複雑度を測定すると，図７の通りである．

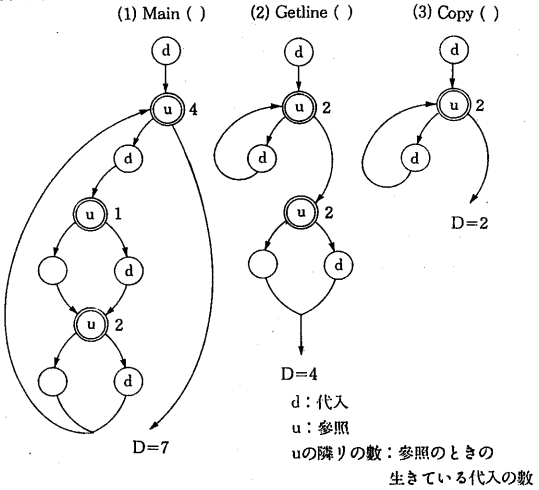

図７．図６のサンプルプログラムの計算例

2)NICの測定

(1)FCN(NLP)

図６のサンプルプログラムで，FCNは Mainモジュールだけで発生するので，NIC要素である FCN(NLP)は図８のように４と６になる．

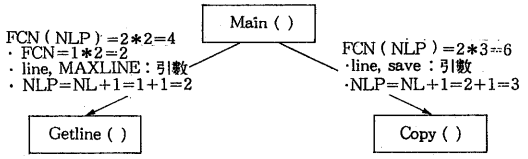

図８．図６の FCN(NLP)の計算

(2)AVN(NLP)

Getlineモジュールの観点（Getline←Main）で，AVN(NLP)は次のように計算される．

・AVN＝(呼び出された回数＊引数の数)の総和＝1*2＝2

・NLP：相対モジュールの NLPで，Getlineの相対モジュールは Mainなので，ここでの NLPは Mainモジュールでの Getlineで，FCN(NLP)と同様に２である．つまり，AVN(NLP)は４で，この値は Mainモジュールでの GetlineでのFCN(NLP)＝4と同じ値を持つ．また，Copyモジュールの観点（Copy←Main）でのAVN(NLP)は図９と同様に６になる．つまり，モジュールＡがモジュールＢを呼び出す状態ではＡの FCN(NLP)＝ＢのAVN(NLP)である．

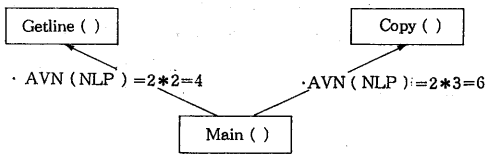

図９．図６の AVN(NLP)の計算

(3)RVN(NLP)

Getlineモジュールの観点（Getline←Main）で，Mainモジュールで Getlineモジュールを呼び出し，その引数を使用するので，Getlineモジュールの複雑度に影響を与え，そのときの Getlineモジュールの NLPを意味するので，次のように計算される．

・RVN＝呼び出されたモジュールで引数を使用した回数＝3＋1＝4(Sは 27，29，32ラインでそれぞれ１回ずつ使用され，limは 26ラインで１回使用)

・NLP＝引数 Sの NLPは，2(line 27，29),1 (line 32)で，limも2(line 26)なので，全体 RVN(NLP)は 7になる．

また，Copyモジュール観点（Copy←Main）でも同様にすると，それぞれの変数ごとに NLPをかけて加えると，4になる(S1:使用１回＊NLP(2)＝2，S2:使用１回＊NLP(2)＝2)．

Mainモジュールの観点で，RVN(NLP)は呼び出したモジュールがそれぞれ Getlineと Copyなので，両方から受けた値を出せばよい．つまり，まず，Mainモジュールで，Getlineモジュールを呼び出し，その結果をうけるので，RVN(NLP)は７になり，Copyでは４になる．従って，Mainモジュールでの RVN(NLP)は 11になる．図６のサンプルプログラムでは，大局変数がないので，EVNの計算例は省略する．

上のような方法で，図６のサンプルプログラムに対して，モジュール間のインタフェースと複雑度を測定した例が図１０である．図１０において，Mainモジュールから Getlineモジュールへの FCN(NLP)は Mainモジュールのみに関連し，GetlineモジュールのNIC計算には含まれない．その理由は，Getlineモジュールにおいて，同じ引数(line，MAXLINE) を使い，重複された概念は導入されないからである．従って，Mainモジュールの NIC計算はそれぞれの FCN(NLP) の値を加えた値(10) と，RVN(NLP)の値を加えた値(11)を加えればよい．他のモジュールからは，Mainモジュールは呼び出されないので，Mainモジュールでの AVN(NLP)は 0である．表１はサンプルプログラムに対する従来の尺度と提案尺度 MCMの測定結果である．図１０と表１を見ると，Mainモジュールの複雑度 MCM(D(G)+NIC)の測定値は 28で，Getlineモジュールは15，Copyモジュールは１２で，複雑度に影響

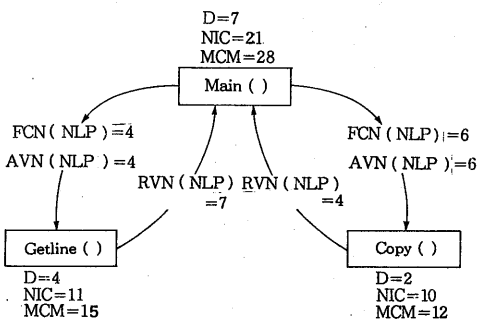

図１０．図６のサンプルプログラムの計算例

を与えるいろんな要素を測定し，組み合わせることによって，より正確にモジュール複雑度を測定できることを示した．また，３つのモジュールの複雑度を加えることによって，プログラムの全体複雑度を求めることができる．

表１．サンプルプログラム(図６)の MCMの測定結果

尺度		Main	Getline	Copy
従来の尺度	C(G)	4	3	2
	n1	11	16	8
	n2	6	6	6
	N1	28	35	11
	N2	14	19	6
	E	2203	6100	259
提案	D(G)	7	4	2
	NIC	21	11	10
	MCM	28	15	12

3.4　評価

本稿で提案した尺度 MCMの妥当性を示すため，任意のＣプログラム５０個に対して，従来の尺度との相関関係を評価した．

まず，従来の INFO (Information Flow Matrics) と C(G)，SS尺度との相関関係を表２に示す．

表２．INFOと C(G)，SS尺度との相関関係

	INFO	C(G)	SS
INFO	1.000	0.345	0.383

表２から，INFOと C(G)，SSとの相関関係が低いことが分かる．これに関する解釈として，プログラムの複雑度を測定する尺度として，INFO，C(G)，SSの１つだけを用いることは好ましくないことである．

表３．提案尺度 MCMと従来の尺度との相関関係

	MCM	C(G)	SS	INFO	IMC	NIC
MCM	1.000	0.780	0.805	0.854	0.837	0.886

表３からは，MCMが従来の尺度との相関関係が高いことが分かる．これは，MCMがソフトウェア複雑度に影響を与える性質を総合的によく反映していることである．また，MCMの構成に，IMCより NICが影響していることが分かる．

４．むすび

プログラムの複雑度は，プログラムを客観的かつ，定量的に評価できる尺度である．

本稿では，モジュール間依存度を組み合わせた新しいモジュール複雑度尺度ＭＣＭを提案した．また，ＭＣＭが従来の複雑度尺度の性質を保ちながら，従来の尺度より詳細に複雑度を表せることを示した．

今後の課題として，プログラムモジュールの抽象化，モジュール複雑度測定方法と，Adaのような並行処理および，実時間制御が可能なプログラミング言語の複雑度要因を分析し，新しい複雑度測定方法を考案することなどがある．

参考文献

[1] J.Martin and C.McClure:" Software maintenance : The problem and it's solution" , Prentice-Hall (1983).

[2] L.J.Artur :" Measuring Programmer Productivity and Software Quality" ,John Wiley & Sons,Inc. (1975).

[3] S. Henry and D.Kafura:" Sortware Structure Metrics Based on Information Flow" , IEEE Trans. Software Eng.,Vol.SE-7,No.5, pp.510-519(1981).

[4] W.Harrison, K.Magel, R.Kluczny and A.Dekock:" Applying Software Complexity Metrics to Program Maintenance" ,IEEE Computer(Sep.1982).

[5] 梁　海述，荒木俊郎：" プログラムの制御流れ複雑度と構造度の精度改善について"，情報処理学会ソフトウェア工学研究会資料，56-5(Sep.1987).

[6] 梁　海述，辻野嘉宏，都倉信樹：" データ流れ情報によるソフトウェアの複雑度について"，信学技報，COMP88-77(1989).

[7] M. Halstead :" Element of Software Science" , Elsevier North-Halland, New York(1977).

[8] T. J. McCabe :" A Complexity Measure" , IEEE Trans. Software Eng., Vol. SE-2, No.4, pp.308-320(1976).

[9] M.R. Woodward, M.A. Hennell and D.Hedley :" A Measure of Control flow Complexity in Program Text" ,IEEE Trans. Software Eng.,Vol.SE-4, No. 1, pp.45-50(1979).

[10]N. Chapine:" A Measure of Software Complexity", proc. of NCC, pp.995-1002(1979).

[11]N.S. Coulter:" Software Science and Cognitive Psychology" ,IEEE Trans. Software Eng., Vol.SE-13, No.6, pp.166-177(1983).

[12]有沢　誠：" ソフトウェア工学"，岩波コンピュータサイエンス，岩波書店(1988).

[13]W. Hansen :" Mesurenment of Program Complexity by the Pair" , ACM SIGPLAN NOTICES, pp.146-152 (1987).

[14]M. Waite, S. Prata and D. Matine :" C Primer Plus" , Howard W. Sanms & Co., Inc. (1984).

[15]梁　海述，辻野嘉宏，都倉信樹：" データフロー情報を考慮したソフトウェアの複雑度について"，信学論(DI)，J72-D-I, 11, pp.789-796(1989).