日本ファジィ学会誌 Vol.10, No.5, pp.796-803 (1998)

解説

ソフトウェアメトリクス†

松本 健一*

1. はじめに

初心者にも馴染みやすいプログラミング言語や開発環境の登場により，ソフトウェア作りを趣味とする人が増えつつある．しかし，社会で広く利用されているソフトウェアの大部分は，依然として，専門家の手により工業製品として開発されている．工業製品である以上，その特性や開発／利用過程を定量的に評価する活動は，品質や生産性を改善する上で不可欠である．本解説では，ソフトウェアとその開発／利用過程の定量的評価尺度(ソフトウェアメトリクス：Software Metrics)とはいかなるものであるのかを，いくつかの具体例を交えて紹介する．

2. 定量的評価の必要性

ソフトウェア工学 (Software Engineering) という用語が1968年に初めてNATO主催のワークショップと書籍で使われて30年が経つ．当初は社会的ニーズを満足させるためにソフトウェアを如何に利用すべきかが盛んに議論された．その後，ソフトウェアの出荷量が増大するにつれて，開発期間やコストの予測や制御に関心が集まった．そして，ソフトウェアに対する社会の依存度が増大した1990年代に入ると，ソフトウェアの品質が特に重要視されるようになった[4]．

品質の時代と呼ばれる今日はまた，ソフトウェアの開発／利用形態が多様化した時代でもある[4]．様々な人々がソフトウェアを利用し，利用目的や利用環境は驚異的に広がりつつある．ソフトウェア開発では道具としてソフトウェアが多用されるため，利用形態の多様化はそのまま開発形態の多様化でもある．多様化を象徴するキーワードとしては，オブジェクト指向開発，コンポーネントウェア，エンドユーザコンピューティング，World Wide Web (WWW)等がある．

ソフトウェアの開発／利用形態の多様化は，新たな理論や技術の開発を促進すると同時に，理論や技術の妥当性や有効性を改めて検証する必要性を高くしている．なぜならば，理論や技術の多くは，その適用対象をある程度限定し，適用上の条件も持っている．ソフトウェアの開発／利用形態が変化すれば，新たな理論や技術はもちろんのこと，既存の理論や技術であっても，従来通りに用いることが適当とは必ずしも言えなくなる．例えば，コーディング作業の進捗は，従来，作成されたプログラムコードの規模(サイズ)である程度把握することが出来た．しかし，オブジェクト指向技術を用いた開発では，ソフトウェア部品の流用や再利用により，作業終了時と大差のない規模のプログラムコードが作業開始直後から存在する場合がある．プログラムコードの規模に基づく進捗管理をオブジェクト指向開発でも盲目的に行うことは適当でない．

ソフトウェアの新しい開発／利用形態における理論や技術の妥当性や有効性の議論は，抽象的，あるいは，定性的な情報だけでなく，具体的，かつ，定量的な情報にも基づいて行うべきである．例えば，比較的小規模でコントロールが容易な環境下での観察実験(Case Study)や様々なコンテキストでの反復実験(Replicated Experiment)は，

† Software Metrics

* Ken-ichi MATSUMOTO
奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

新しい開発／利用形態への認識を深め，理論や技術との関係を明らかにする有効な方法である[16]．実験を重視したこうした研究形態はEmpirical Software Engineeringと呼ばれている[16][22][23]．そして，Empirical Software Engineeringで最も重要な役割を果たすのがソフトウェアメトリクスである．

3. ソフトウェアメトリクス

メトリクス(Metrics)という用語は，日本語では「定量的評価尺度」，あるいは，単に「尺度」と訳される場合が多い．従って，ソフトウェアメトリクスとは，「ソフトウェアとその開発／利用過程を対象とした(定量的評価)尺度」ということになる．なお，以降の説明では簡単のため，開発／利用過程のうちの開発過程のみを取り上げるが，利用過程についても同様の議論が可能である．

ソフトウェアメトリクスは，ソフトウェアの品質と生産性の改善を目的として用いられる．より具体的な利用目的としては次のようなものが挙げられる[7][8]．

- ソフトウェアやその開発過程の現状を知る．
- 開発作業の進捗状況を把握し，必要な指示を与える．
- ソフトウェアの開発コスト，出荷可能日，機能，品質等，開発管理の対象となる特性を予測する．
- ユーザの要求や目標を定量的に表す．
- ソフトウェア開発における利害得失を分析する．

これら利用目的からも分るように，ソフトウェアメトリクスは，ユーザ，プロジェクト管理者，開発者，そして，経営者のために用いられる[2]．

ソフトウェアメトリクスを分類するために，まず，評価対象であるソフトウェアとその開発過程をモデル化する．一般に，ソフトウェア開発とは，ソフトウェアに対するユーザの要求を実行可能なプログラムコードに「変換」する作業である．但し，ユーザ要求と実行可能なプログラムコードでは，その抽象度や表現形態等に大きな差がある．そこで，要求仕様書，設計書，ソースプログラム等のいくつかの中間プロダクトを設定し，それらの間での比較的小さな「変換作業」の連鎖によって開発を進めることになる．なお，実際のソフトウェア開発では，実行可能なプログラムコードに付随して操作マニュアル等が作成され，ユーザに提供される．ここでは，開発者からユーザに提供されるプロダクトを総称して最終プロダクトと呼ぶ．

個々の変換作業は，元となるプロダクト(入力プロダクト)，変換により得られるプロダクト(出力プロダクト)，及び，作業実行に必要な資源によって定義される(図1参照)．ここで，入力プロダクトは，ユーザからの要求，あるいは，他の変換作業の出力プロダクト(中間プロダクト)であり，出力プロダクトは最終プロダクト，あるいは，他の変換作業への入力プロダクト(中間プロダクト)である．また，資源としては，時間，人間(開発者)，ハードウェア，ソフトウェア等が挙げられる．例えば，コーディング作業における入力プロダクトは「詳細設計書」等，出力プロダクトは「プログラムコード」等であり，作業担当者，作業に費やされた時間，コーディング環境等が資源となる．なお，ソフトウェア開発では，コードレビューや単体テスト等，いわゆる中間プロダクトを生成しない作業が存在する．そうした作業は，入力プロダクト(の一部)が出力プロダクト(の一部)でもあり，作業によってプロダクトの特性のみが変化すると考える．例えば，コードレビューでは，プログラムコードは入力プロダクト，かつ，出力プロ

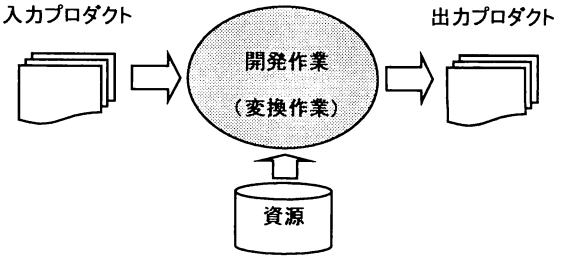

図1 ソフトウェア開発作業モデル

ダクトである．作業によって変化する特性はプログラムコード中の残存バグ数である．

以上の議論に基づき，ここでは，プロダクトメトリクス(Product Metrics)，資源メトリクス(Resource Metrics)，及び，プロセスメトリクス(Process Metrics)の3つにソフトウェアメトリクスを分類する．

(1) プロダクトメトリクス

プロダクトメトリクスは，最終プロダクト，及び，中間プロダクトの特性を評価するための尺度である．変換作業単位で見れば，入出力プロダクトの特性を評価する尺度と言える．評価される特性としては，規模(サイズ)，複雑さ，残存バグ数，信頼性，理解容易性等が挙げられる．特に，最終プロダクトの特性は，そのままソフトウェアの品質を表すものとなる．

これまでに提案されたメトリクスの多くは，このプロダクトメトリクスである．しかも，そのほとんどが，プログラムコードや擬似コードを対象としたメトリクス(コードメトリクス)である．これは，プログラムコードが，ほとんどのソフトウェア開発において作成されるプロダクトであり，ファイルとして蓄積されている場合には，ツール等による機械的な評価(尺度適用)が比較的容易なためである．

最も基本的なコードメトリクスは，規模を表すプログラム行数(LOC：line of code)である[3]．LOCは多くの開発者にとって馴染みやすく，算出も容易である．プログラムの他の特性，例えば，信頼性などをLOCで予測する方法も提案されている[9]．

但し，LOCの定義は提案者や利用者によってそれぞれ微妙に異なり，その利用には注意が必要である[11]．例えば，多くのプログラムコードは，データ定義行，実行行，注釈行(コメント行)，空行から構成されているが，このうち注釈行をLOCとして数えるかどうかでその値は何倍も異なってくる．しかも，注釈行を数えるべきかどうかの判断は，注釈行がどのように作られたのか，LOCを何に利用するのか，といったコンテキストに大きく依存する．プログラムを理解する上で重要な情報が注釈行としてプログラムに埋め込まれている場合には，注釈行もLOCとして数えるべきであろう．一方，コードジェネレータ等で機械的に生成された形式的な注釈行であれば，LOCとして数えるのは適当でないかもしれない．

ソフトウェアの複雑さとは，ソフトウェアの作成や理解の難しさを表す特性である．複雑さの尺度(複雑さメトリクス)は数多く提案されているが，その大半はプログラムコードを対象としている[7][20]．しかも，プログラムコードの意味的構造の複雑さを扱うメトリクスはほとんど知られておらず，プログラムに含まれる情報量や条件分岐数といった形式的構造の複雑さを評価するものが多い．

例えば，MaCabeの提案するサイクロマチック数(Cyclomatic number)は，プログラムの制御の流れを有向グラフで表現し，そのグラフの持つ性質に基づいて複雑さを評価する尺度である[14]．サイクロマチック数を求めるために，あるプログラムを有向グラフに変換した結果を図2に示す．図2のグラフGの節点は逐次実行される連続したプログラム部分であり，枝はプログラム中での制御の分岐を表す．但し，グラフを強結合とするために，プログラムの最後を表す節点から先頭を表す節点への枝が便宜的に追加されている．グラフ理論によれば，有向グラフGが強結合であれば，1次独立な閉路の数$V(G)$は，枝の数eと節点の数nを用いて

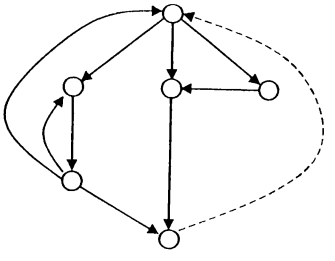

図2 プログラムのグラフ表現

$$V(G)=e+n+1$$

となる．この$V(G)$をサイクロマチック数と呼ぶ．サイクロマチック数は，プログラムの全ての経路を表現するために最小限必要な閉路数(基本パス数)であり，プログラムをテストするために最低限必要なテストデータ数と考えることが出来る．また，サイクロマチック数はモジュール分割の基準としても用いられており，各モジュールのサイクロマチック数を10以下とするのが望ましいと経験的に言われている．なお，サイクロマチック数は，プログラム中に現れる制御の分岐数に1を加えた数に等しいことが知られており，算出のために図2のようなグラフを描く必要は必ずしもない．

(2) 資源メトリクス

資源メトリクスは，変換作業における資源の消費量，及び，消費される資源の特性を評価するための尺度である．主な評価対象は，時間(作業時間)と人間(開発者)である．

作業時間はソフトウェアの生産性を議論する上で基本となるデータである．それだけに，開発の形態や実状に即した尺度を用いることが求められる．例えば，開発期間が数ヶ月から数年におよぶプロジェクトで，プロジェクト専属の開発者が作業を行うような場合，日常的に利用している，いわゆる「カレンダー時間」を単位として作業時間を評価することが出来る．一方，開発期間が比較的短く，各々の開発者がいくつものプロジェクトに同時に参画しているような場合，計算機の利用状況等を考慮した独自の時間単位を定義する必要がある．

開発者の評価では，作業に必要な知識，経験，適性といった，開発者の内的な特性を評価対象とする必要がある．しかし，人間の内面を直接的に評価することは困難であり，従来は「プログラマとしての経験年数」等で大まかに分類する程度であった．最近では，作業実行に必要な知識量と開発者の持つ知識量の関係をオージブモデル(累積正規モデル)で表し，作業習熟も表現可能な開発者評価モデルも提案されている[10]．

(3) プロセスメトリクス

プロセスメトリクスは，個々の変換作業，及び，開発作業全体の特性を評価するための尺度である．評価される特性としては，進捗状況，誤り発生率，作業効率，生産性等が挙げられる．

個々の変換作業のプロセスメトリクスは，その作業の入出力プロダクトや資源の特性間の関係(モデル)として定義される場合が多い．これは，作業内容自体の定量的評価が難しいためで，作業で作り出されたプロダクトの善し悪しや消費された資源の量に基づいて間接的に評価していると言える．

例えば，ソフトウェア信頼度成長モデル(SRGM：Software Reliability Growth Model)は，テスト時刻(テスト開始からの経過時間)と累積発見バグ数の関係としてテスト作業(テストプロセス)を表現するモデルである．一般的なSRGMでは次の(A 1)(A 2)を仮定している．

(A 1) テスト作業で発見されたソフトウェア中のバグは全て修正され，除去される．

(A 2) テスト作業(のバグ修正，除去の過程)において新たなバグが作り込まれることはない．従って，テストの進行に伴い，ソフトウェア中のバグは必ず少なくなる．

更に，テスト作業において発見された総バグ数を確率量としてモデル化し，その確率過程に対して非同次ポアソン過程(NHPP：Non-Homogeneous Poisson Process)を仮定する．すると，テスト開始からテスト時刻tまでに発見される総バグ数$N(t)$は次式で表される[19]．

$$Pr\{N(t)=n\}=\frac{\{H(t)\}^n}{n!}\times\exp[-H(t)] \qquad (n=0,1,2,\cdots)$$

$$H(t)=\int_0^t h(t)\,dx \qquad t\geq 0$$

ここで$H(t)$はNHPPにおける平均値関数(mean value function)で，$N(t)$の期待値を表す．また，$h(t)$は時刻tにおける単位時間当りの発見

バグ数の期待値を表す．更に，テスト開始時にソフトウェア中に存在していたバグ数(初期バグ数)N_0の期待値は$H(\infty)$であり，ソフトウェア中の時刻tにおける残存バグ数の期待値は$H(\infty)-H(t)$となる(図3参照)．

SRGMを用いれば，テスト時刻，初期バグ数(の推定値)，及び，残存バグ数(の推定値)に基づいて，テスト作業の進捗や効率を評価することが出来る．但し，テスト作業で発見されたバグ毎にその発見時刻の情報(データ)が，評価には必要となる．また，平均値関数に特徴を持たせた数多くのSRGMが提案されており，テスト作業の実態に応じてモデルを選択する必要がある．

開発作業全体を対象としたプロセスメトリクスの例としては，変換作業の実行系列に基づくものがある[12]．例えば，ソフトウェア開発ではプロダクトに作り込まれたバグの種類や数に応じて，設計作業やコーディング作業をやり直すことがある．こうした作業のやり直しを変換作業の実行系列のパターンとして定義し，その発生回数によって開発作業全体の作業効率を評価することができる[12]．

4. ソフトウェアメトリクスの特性

3.で紹介した例以外にも様々なソフトウェアメトリクスがこれまでに数多く提案されている．利用に際しては，評価の対象や目的に応じたメトリクスを選択するのはもちろんであるが，メトリクスの持つ次のような特性についても検討の必要がある．

(1) 適用可能時期

ソフトウェア開発は数多くの作業で構成されており，開発期間も長く数年に及ぶ開発プロジェクトも珍しくない．ソフトウェアの品質が期待通りでないからといって，プロジェクト終了間際に作業のやり直しを行っていたのでは期限までに開発を終了することは出来ない．開発者の安易な増員もプロジェクトをかえって混乱させ，開発に要する

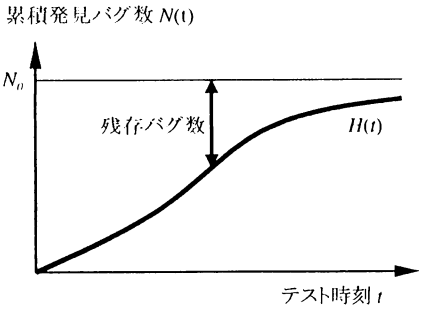

図3 ソフトウェア信頼度成長モデル

期間を更に延ばす結果となる[17]．

評価の対象や目的が同じであれば，開発作業の出来るだけ初期に適用可能となるメトリクスを用いるべきである．例えば，プログラム行数(LOC)はプロダクトの規模を表すメトリクスであるが，当然のことながらプログラムコードが作成されるまで適用することは出来ない．これに対して，同じくプロダクト規模のメトリクスであるファンクションポイントは，要求分析の段階から適用可能であり，評価結果を計画立案に活用することは十分可能である[1]．

(2) 予測性

早い時期から適用可能なソフトウェアメトリクスの中には，将来のある時点でのソフトウェアや開発過程の特性値を予測するものがある．予測能力を持つソフトウェアメトリクスの利用価値は高いが，適用時期と予測精度の関係を知った上で利用する必要がある．

例えば，SRGMを用いれば，理屈の上では，テスト作業開始直後から初期バグ数を予測することは可能である．しかし，大半のSRGMでは，累積発見バグ数が最低でも20～30個，テスト時間にして7割程度の作業が終了しないと予測値は安定せず誤差も大きい[13]．複数のSRGMを実測データに適用し，最も適合度の高いものを選んで利用するといったことも行われているが，適合度の優劣の判定も，初期バグ数の予測値が安定するのを確かめてから行うべきである．

(3) コスト

ソフトウェアメトリクスによる評価には，データ収集等の作業が必要であり，ソフトウェア開発プロジェクトのコストを最大10%上昇させると言われている[15]．開発プロジェクトの予算規模や評価目的から考えて，余りにも評価コストの高いソフトウェアメトリクスは，その利用を慎重に検討する必要がある．

(4) 数学的性質

ソフトウェアメトリクスの多くは数学的モデルであり，モデルとして満たすべき条件(必要条件)が明らかにされている場合がある．例えば，Weyukerはプログラムコードの複雑さメトリクスを対象として，メトリクスが持つべき数学的性質(プロパティ)として次の9つを挙げている[18]．なお，ここで，$|P|$はプログラムPの複雑さを表す．また，$P \equiv Q$は2つのプログラムPとQの機能が同じであることを表す．更に，$P;Q$はプログラムPの後ろにQを繋げたプログラムを表す．

性質1：$(\exists P)(\exists Q)(|P| \neq |Q|)$．

性質2：cを非負数とする．複雑さがcとなるプログラムはたかだか有限個しか存在しない．

性質3：$(\exists P)(\exists Q)(P \neq Q \& |P| = |Q|)$．

性質4：$(\exists P)(\exists Q)(P \equiv Q \& |P| \neq |Q|)$．

性質5：

$(\forall P)(\forall Q)(|P| \leqq |P;Q| \& |Q| \leqq |P;Q|)$．

性質6：

6a：$(\exists P)(\exists Q)(\exists R)$

$(|P| = |Q| \& |P;R| \neq |Q;R|)$．

6b：$(\exists P)(\exists Q)(\exists R)$

$(|P| = |Q| \& |R;P| \neq |R;Q|)$．

性質7：Pを構成する文の順序のみを変えたものをQとすると$|P| \neq |Q|$であるようなプログラムP，Qが存在する．

性質8：Pの識別子名を付け替えたものをQとすると$|P| = |Q|$である．

性質9：$(\exists P)(\exists Q)(|P| + |Q| < |P;Q|)$．

但し，Weyukerの提案には批判的な意見もある[6]．例えば，ZuseはWeyukerの性質は統計学上のメトリクスの性質と一致していないと指摘している[21]．また，Cherniaviskyらは，これらの性質はメトリクスとしての必要条件を与えているにすぎないので，有用性についての評価に用いるには注意が必要であると述べている[5]．

5. おわりに

ソフトウェアメトリクスによる評価は，ソフトウェアの品質や生産性の改善に不可欠であるが，煩雑な作業を伴うことも確かである．特に，評価に必要なデータの収集と統計的処理は，メトリクスの提案者によってその具体的方法が示されていない場合が多く，ソフトウェアメトリクスを初めて使う者にとっては敷居が高いと言える．しかし最近では，ソフトウェアメトリクスによる評価作業をパッケージ化し，ソフトウェアツールによって評価の自動化，あるいは，支援を積極的に行う例が増えている．

宮川らが開発した「ソフトウエア信頼性成長モデルツール(以下では「SRGM Tool」と呼ぶ)」は，SRGMを用いて残存バグ数や平均故障待ち時間を推定することができる[24]．SRGM Toolへの入力は，テストインスタンス，累積フォルト数，テスタ担当者の人数または能力の系列のみである．利用できるモデルは，宮川らが提案する超幾何分布モデルに加えて，代表的なSRGMであるゴンペルツモデル，遅延S字型NHPPモデル，指数型NHPPモデルの計4つである(図4参照)．SRGM Toolの利用者は，自分たちが所有するデータを用いて，宮川らが提案するモデルの良さを容易に確かめることが出来る．

SRGM Toolの最大の特徴は，Javaで記述されたアプレットとして実現されており，インターネット上で公開されている点にある[24]．UNIX，Windows，Macintosh上のWeb Browserで実行可能であり，実行速度も速い．ツールの配布や更新も簡単で，利用者に特別な知識は不要である．更に，SRGMの解説文が合わせて公開されており，幅広い利用と利用結果のフィードバックが期待される．

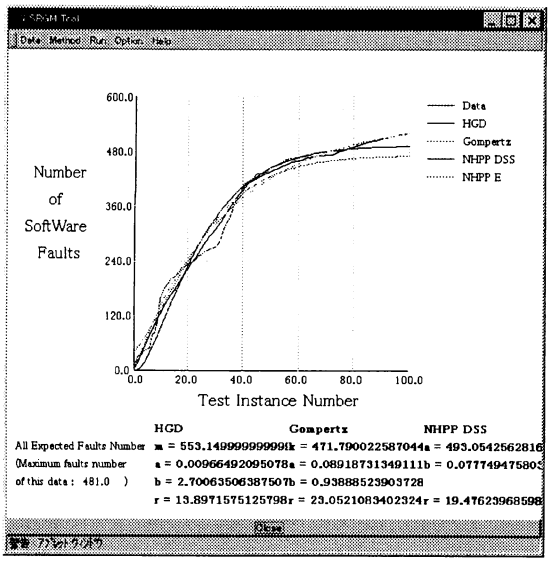

図4 SRGM Toolの出力例

ソフトウェアメトリクスによる評価作業を全て自動化することはなかなか容易でない．しかし，こうした試みは，ソフトウェアメトリクスの妥当性や有効性をより多様なコンテキストで検証可能とする．更に，研究者間やユーザ間での知識やデータの共有を促進し，ソフトウェアメトリクスによる評価をより一般的なものとする可能性を秘めている．

参 考 文 献

[1]A. J. Albrecht, "Measuring application development productivity," *Proc. of Joint SHARE/GUIDE/IBM Application Development Symposium*, pp.83-92 , 1979.

[2]V. R. Basili, "Goal Question Metric Paradigm," *in Encyclopedia of Software Engineering*, J. J. Marciniak, ed., John Weily and Sons, pp.528-532, 1994.

[3]V. R. Basili and D. H. Hutchens, "An empirical study of a syntactic complexity family," *IEEE Transactions on Software Engineering*, 9, 6, pp.652-663, 1983.

[4]V. R. Basili and J. D. Musa, "The future engineering of software : A management perspective," *IEEE Computer*, 24, 9, pp.90-96, 1991.

[5]J.C. Cherniavisky and C. H. Smith, "On Weyuker's axioms for software complexity measures," *IEEE Transactions on Software Engineering*, 17, 6, pp. 636-638, 1991.

[6]S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," *IEEE Transactions on Software Engineering*, 20, 6, pp.476-493, 1994.

[7]S. D. Conte, H.E. Dunsmore and V.Y. Shen, "*Software Engineering Metrics and Models*," The Benjamin/Cummings Pub., 1986.

[8]J. R. Dunham and E. Krusei, "The measurement task area", *IEEE Computer*, 16, 11, pp.47-54, 1983.

[9]L. Elshoff, "An investigation into the effects of the computing method used on software science measurements," *SIGPLAN Notices*, 13, 2, pp.30-45, 1978.

[10]N. Hanakawa, S. Morisaki, and K. Matsumoto, "A learning curve based simulation model for software development," *Proc. of 20th International Conference on Software Engineering*, pp.350-359, 1998.

[11]C. Jones, *Applied Software Measurement-Assuring Productivity and Quality*, McGraw-Hill, 1991.

[12]楠本真二，松本健一，菊野亨，鳥居宏次，"ペトリネットによるプログラム開発演習のモデル化とそのモデルによるプログラマ作業効率の定量的評価"，電子情報通信学会論文誌，J76-D-I，9，pp.484-492，1993.

[13]松本健一，菊野亨，鳥居宏次，"S字型ソフトウェア信頼度成長モデルの大学環境における実験的評価 －推定精度の比較と習熟係数の決定－"，電子情報通信学会論文誌，J73-D-I，2，pp.175-182，1990.

[14]T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, SE-2, 4, pp.308-320 , 1976.

[15]S. L. Pfleeger and J.C. Fitzgerald, Jr., "Software metrics tool kit Support for selection, collection and analysis," *Information and Software Technology*, 33, 7, pp.477-482, 1991.

[16]A. A. Porter, L. G. Votta and V. R. Basili, "Comparing detection methods for software requirements inspections : A replicated experiment," *IEEE Trans. Software Engineering*, 21, 6, pp.563-575, 1995.

[17]L. H. Putnam, "A general empirical solution to the macro software sizing and estimation problem," *IEEE Transactions on Software Engineering*, SE-4, 4, pp.345-361, 1978.

[18]E. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, 14, 9, pp.1357-1365, 1988.

[19]S. Yamada and S. Osaki, "Software reliability growth modeling : Models and applications," *IEEE Transactions on Software Engineering*, SE-11, 12, pp.1431-1437, 1985.

[20]山田 茂，高橋 宗雄，ソフトウェアマネージメントモデル入門 －ソフトウェア品質の可視化と評価法－，共

立出版，1993.
[21]H. Zuse, "*Software Complexity : Measures and Methods*," Walter de Gruyter, 1991.
[22]http : //www.cs.pdx.edu/emp-se/.
[23]http : //wwwagse.informatik.uni-kl.de/ISERN/isern.html.
[24]http : //www.dcl.c.dendai.ac.jp/hgdm/.

[問い合わせ先]
〒630-0101
奈良県生駒市高山町8916-5
奈良先端科学技術大学院大学
情報科学研究科
松本 健一
TEL ：0743-72-5311
FAX ：0743-72-5319
E-mail : matumoto@is.aist-nara.ac.jp

(1998年8月31日 受 付)

著 者 紹 介

松本 健一 (まつもと けんいち)

奈良先端科学技術大学院大学情報科学研究科

1985年 大阪大学基礎工学部情報工学科卒業. 1987年 同大大学院基礎工学研究科博士前期課程修了. 1989年 同大学院大博士後期課程中途退学. 同年 大阪大学基礎工学部助手. 1993年 奈良先端科学技術大学院大学情報科学研究科助教授，現在に至る. ソフトウェア計測，ソフトウェアプロセスなどの研究に従事. 工学博士