Vol. 35 No. 4　　情　報　処　理　　Apr. 1994

特別論説

情報処理最前線

ソフトウェア規模見積り技術の最近の流れ
——行数による評価から機能量による評価へ†——

西　山　茂††

1. はじめに

ソフトウェア開発をその初期段階で種々の知識を基に予測し，必要な手段を講じておくことは，高い品質をもったソフトウェアの効率的な開発に欠かせない事項である．ソフトウェア開発に関して予測の対象となるものは種々ある．たとえば，大きさ（量あるいは規模），投入する工数，開発期間，開発に使用される技術，品質（バグ数など）などである．これらはいずれも重要な予測対象であるが，中でも重要なものは開発投資額（工数）と開発期間である．予測は開発のさまざまなフェーズで実施されるが，本稿では主として開発の上流工程で実施される予測を扱う．

開発する「ソフトウェアの量☆」と，開発投資額（工数）と開発期間はなんらかの関数関係にあることは容易に推定できる．また，ソフトウェアの量は，ソフトウェアの種々の特性を評価するときの基準を作るのにも役立つ．そこで通常，ソフトウェアの量をまず推定／予測し（すなわちソフトウェアの規模を見積もり），これに基づいて開発投資額（工数）と開発期間を予測する手法がとられる．

ソフトウェアの量を表す尺度にも種々のものがあるが，以下の二つが代表的である．

a．プログラム行数（[S]LOC：[Source] Lines Of Code の略）

b．ソフトウェア機能量

プログラム行数（以下 LOC あるいは SLOC という場合もある）は，多くの人が漠然としたイメージをもっていて，ソフトウェアの量の尺度としてあまり抵抗なく受けいられてきた．これは，プログラム行数がソフトウェアの属性の中では数少ない直接「目に見える量」であり，人間の見たり触ったりすることにより物事を理解するという特質によく合っているためであろう．プログラム行数がある一面ではソフトウェアの量を表していることも事実である．しかし，最近になって，プログラム行数は予測のために使用するソフトウェアの量として適切でないという認識が高まってきている．

ソフトウェア機能量は，ソフトウェアがもっている，あるいは，もつであろう機能をある一定の手段により定量化したものである．ソフトウェア機能量の代表的なものに本稿で紹介するファンクションポイントがある．ほかにもDeMarcoのBang尺度[1]や画面数によってのソフトウェア規模を規定する方法などもこの分類に属すると考えてよい．ソフトウェア機能量は，プログラム行数に対する疑問と裏腹になって，ソフトウェアの予測や開発に関する重要な量であるとの認識が広まりつつある．しかし，ソフトウェア機能量は，プログラム行数とは対照的に，直接には目に見えない量である．これが，ソフトウェア機能量がソフトウェアにとって重要な尺度であるにもかかわらず，今に至ってもそれほど普及していない原因の一つであろうと思われる．

上記のようにソフトウェアの規模見積り法はソフトウェア機能量を使う方向に向きはじめている．本稿では，ソフトウェア規模見積りの代表的でかつ安定した手法であるファンクションポイント法について，手法の概要，利点，ソフトウェア開発プロセスとの係わりあい，利用状況などについて述べる．

---

† Current Practices of Software Sizing—Toward Functional Measures from Source Lines of Code by Shigeru NISHIYAMA (NTT Software Laboratories).

†† NTT ソフトウェア研究所

☆本稿では「大きさ」というような意味で用いる．以下では誤解が生じないと思われるところでは単に「量」ということもある．また，規模ということもある．

2. ファンクションポイント法の概要

ファンクションポイント法は，米国IBMのA. J. Albrechtによって，1979年にその初期版[2]が，1983年にほぼ現在の形に整理された版が公表された[3]．その後，これをベースに種々のファンクションポイント法が提案されてきているが，本稿では，Albrecht (1983年)版をさらに改良したIFPUG法☆[4]をベースに解説を行う．(「Albrecht」は日本ではドイツ語ふう（？）に「アルブレヒト」と呼ばれることが多いようであるが，米国では「オールブレクト (all-brekt)」と呼ばれている．)

2.1 ファンクションポイント法の視点

ソフトウェア開発は使う側（ユーザ）の要求を開発する側のもつ実現方法へ変換する変換過程であり，常に，ユーザ視点と開発者視点の二つの視点がある．ユーザ視点とは，ソフトウェアで何ができるか（機能）という視点であり，開発者視点とは，ソフトウェアをどうつくるかという視点である．ファンクションポイント法では，ソフトウェアをユーザ視点だけでみることを要求する．これが，ファンクションポイントが機能量とよばれるゆえんであり，ファンクションポイント法を理解し，実際に適用するうえで重要な概念である．これに対してLOCは開発側の視点にたつ量である．

ファンクションポイントが何を表しているかを具体的に理解するために，次に，まずファンクションポイントの求め方をみる．

2.2 ファンクションポイントの求め方

ファンクションポイントは次の6つの手順により求める．

step 1：測定対象（ソフトウェアシステム）を明確にする（アプリケーション境界）
step 2：ファンクションを抽出する
step 3：ファンクションの複雑さを評価する
step 4：複雑さの評価値を合計する
step 5：システム特性を評価する
step 6：ファンクションポイントを求める

上記のステップは論理的なステップであり，普通，step 2とstep 3はほぼ同時に行われる．以下各手順を詳しくみていく．

（1） 測定対象（ソフトウェアシステム）を明確にする

測った値の意味を明確にするためには，測る対象がどれなのかをはっきりさせる必要がある．ファンクションポイント法では，「ユーザ視点にたって」測る対象をはっきりさせることを要求する．この測定対象を取り囲む仮想的な境界を「アプリケーション境界」と呼んでいる．

なお，ファンクションポイント法では，ソフトウェアあるいはソフトウェアシステムをアプリケーションと呼ぶことが多い．本稿でも特に断わらない限りこれらを同じ意味として使う．

（2） ファンクションを抽出する

アプリケーション境界を明確にした後，測定対象のソフトウェアの要求仕様書や機能仕様書から，「ユーザ視点にたって」**図-1**に示す5つのファンクション（機能）を抽出する．なお，「ファンクション」も「機能」も同じ意味であるが，Function Point Analysisの一般的な邦訳がファンクションポイント法であるため，ここではファンクションと呼ぶことにする．

5つのファンクションの抽出はアプリケーション内で扱われるデータの性質に着目して行われる．これがファンクションポイント法の特徴である．

a．外部入力：アプリケーション境界を越えてシステムにデータを入力する機能をいう．ユーザが直接入力する場合だけでなく他アプリケーションが作成したデータをMTや通信回線を経由して入力する場合も外部入力に含まれる．

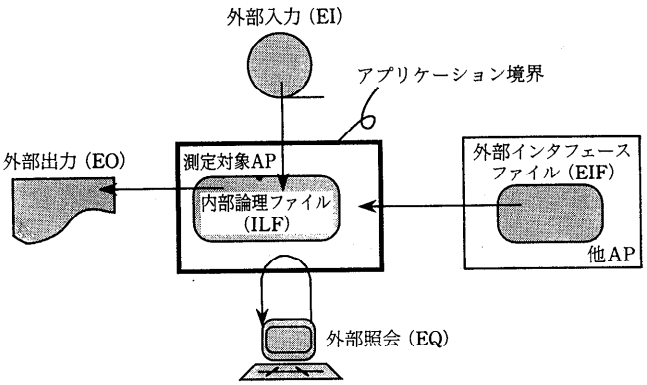


図-1 ファンクションポイント法の5つのファンクション

☆ IFPUG: International Function Point Users Group

b．外部出力：アプリケーション境界を越えてシステムからデータを出力する機能をいう．プリンタに出力される帳票，画面出力，MT などの2次記憶に出力されるデータがこれに相当する．

c．外部照会：利用者の入力をトリガとしてシステム内のデータを出力するような入出力が組となった機能をいう．

d．内部論理ファイル：アプリケーション境界内で維持される論理的な関連をもったデータの集合のうち，ユーザ視点からみて必要なものを扱う機能をいう．ここで維持とは，内部論理ファイルのデータの生成，使用，追加，更新，削除などを指す．

e．外部インタフェースファイル：アプリケーション境界外のソフトウェアが維持する論理的な関連をもったデータの集合のうちユーザからみて必要なものを参照する機能をいう．

（3） ファンクションの複雑さを評価する

抽出された個々のファンクションの「複雑さ」を決め，それに応じた点数を与える．複雑さは全てのファンクションともその度合いが「高い」か「中くらい」か「低い」かの3種類である．複雑さの度合いは，ファンクションが扱うデータ要素の数と以下の事項の多寡の組合せによって判断される．

a．入出力機能で参照するファイル数（外部入力，外部出力，外部照会の場合）

b．ファイルが含むレコードの種類数（内部論理ファイル，外部インタフェースファイルの場合）

点数はファンクションと複雑さの度合いの組合せであらかじめ決められている．**表-1** と**表-2** に，外部入力と内部論理ファイルに対する複雑さ評価および点数付けのための表を示す．

（4） 複雑さの評価値を合計する

抽出された全てのファンクションに対する複雑さの評価値を加え合わせる．これにより得られた合計値を未調整ファンクションポイントと呼ぶことがある．

（5） システム特性を評価する

上記(3)の複雑さの評価だけでは，アプリケーションの複雑さが必ずしも十分に評価されない場合がある．そこで，**表-3** に示す14種類の評価項目の評価値により(4)で得られた複雑さの評価値の合計値を調整する．この 14 種類の評価項目をシステム特性と呼ぶ．システム特性の各項目はそれぞれ 0〜5 点の 6 段階で評価され，14 項目全ての評価値の合計値を求め，システム特性の評価値とする．

（6） ファンクションポイントを求める

以上の手順によって得られた値を，次の式に代入して，アプリケーションのファンクションポイントを求める．

$$F_p = S_c \times (0.65 + 0.01 \times S_g)$$

ここで，

F_p：ファンクションポイント　S_c：複雑さの評価値の合計　S_g：システム特性の評価値

表-1 複雑さの評価（EI：外部入力）

参照ファイル数＼データ要素数	1〜4	5〜15	16〜
0, 1	低	低	中
2	低	中	高
3〜	中	高	高

低＝3　中＝4　高＝6

表-2 複雑さの評価（ILF：内部論理ファイル）

レコード種類数＼データ要素数	1〜19	20〜50	51〜
1	低	低	中
2〜5	低	中	高
6〜	中	高	高

低＝7　中＝10　高＝15

表-3 システム特性

No.	分類	特性項目
1	アルゴリズムの難しさ	
2	オンライン処理	通信機能を介したデータ収集の複雑さ
3		オンラインデータ入力の割合
4		内部データベースがオンライン更新される割合
5	分散処理	分散処理の高度さ
6		複数組織使用に対する設計製造への影響
7	処理性能	トランザクション量が設計に与える影響
8		レスポンスタイム，スループットへの要求の厳しさ
9		高負荷による設計に対する要求の厳しさ
10	使い勝手，操作性	エンドユーザの使い勝手のよさに対する要求の度合い
11		エンドユーザが機能変更可能にする要求の度合い
12		システムの運用の容易さに対する要求の度合い
13		システム移行およびインストールの容易さに対する要求の度合い
14	再利用可能とする設計の要求度合い	

通常この式は，ファンクションの複雑さの評価により求められた値を ±35% の範囲で補正すると解釈されている．実際，標準的なシステムでは，上記の式の括弧内は1に近い値になるといわれている．

ここで示した式は，新規開発のアプリケーションのファンクションポイントを求める式である．このほかに，上記の式をベースに機能拡張の場合や機能拡張後のファンクションポイントを求める式がある．

以上でみたようにファンクションポイントの測定は手順化されており，このため，属人性が排除され，再現性が高く，測定値として信頼性の高いものである．

図-2 に簡単化したファンクションポイントの計数例を示す．

2.3 ファンクションポイントの意味

Albrecht が二つの論文で述べていることを要約した以下の記述がファンクションポイントの意味をよく表している．

「ファンクションポイントはプログラムの複雑さに関連しており，『アプリケーションの全ての機能を含む外部表現』と比例関係にある量である．」ここで，外部表現とは前節で述べた「ユーザ視点からの要求」に対応する．さらに，Albrecht はファンクションポイントが工数とのよい相関があり，ソフトウェアの見積りや評価に使えることを示した．

言い換えれば，ファンクションポイントは，「ソフトウェアをユーザ視点でみたときに，そのソフトウェアの全てを表している量であり，かつ，ソフトウェア開発の工数とよい相関がある実用的な量である」ということである．

ファンクションポイントの複雑さの評価値は，Albrecht が「議論と試行により決定」[2),3)] したものである．システム特性についても同様である．これらの値や項目は，欧米での約 10 年にわたる使用実績や次節で述べる「精度」のよさなどからみて，適用分野（後述）を特定すれば妥当なものであると考えてよい．さらに，社間比較などのためには全ての利用者が，同じ項目，値を使っているということが重要であるという点に注意する必要がある．

2.4 ファンクションポイントの精度

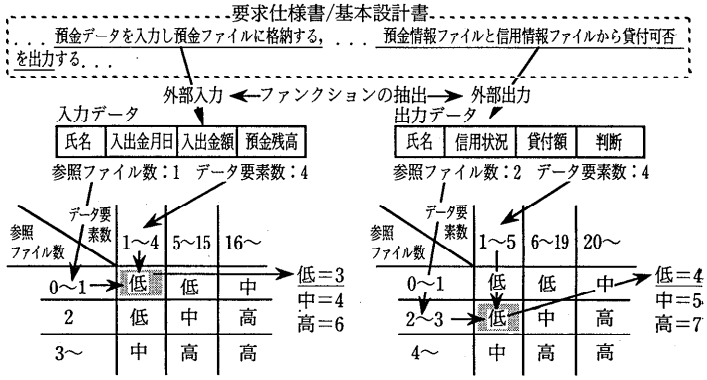
ファンクションポイント法の精度についてみてみる．ここで精度とは，再現性の精度，工数見積りの精度，および，プログラム行数見積りの精度，を意味する．

（1） 再現性の精度

再現性とは，何度やっても同じ結果になるか，あるいは，だれがやっても同じ結果になるかということを示す度合いである．後者は信頼性と呼ばれることもある．

Kemerer は，信頼性を評価するため，27 のシステムを 54 人の計測者に独立に計測させる実験を行った[5)]．この結果，各計測者のファンクションポイントは強い相関をもち（相関計数：0.8），計測値の誤差が小さい（誤差の相対値のメディアンが約 0.12）ことを示し，ファンクションポイント法は信頼性が高いと主張している．

さらに，Kemerer らは標準的な解釈と異なる解釈をされる機能とそのファンクションポイント計測値への影響について研究を行っている[6)]．この結果，1)ファンクションポイント法の規則の解釈は統一的であること，2)大多数の項目の解釈の相



（a） ファンクション抽出と複雑さの評価

上のような手順を繰り返し，その総和をとる．(本文 step 1〜step 4)
(…+3+4+…)＝200
とする．

（b） システム特性評価

14 種類のシステム特性の評価の合計をとる．(本文 step 5) **30** とする．

（c） ファンクションポイント算出

ファンクションポイントは (本文 step 6)
200×(0.65+0.01×30)＝**190**

図-2 ファンクションポイント算出手順の例

違はファンクションポイント計測値への影響が小さいこと，3)解釈の相違のある項目についても規則を整備することにより解釈を一意にできること，からファンクションポイント法の信頼性は高いとしている．

筆者らも別の角度から，計数法の問題点を検討したが，マニュアル記述の充実とエキスパートの支援により計数は精度高く行えるという結果を得ている[7]．

（2） 工数見積りの精度

Albrecht は 24 種類のアプリケーションのデータから求めた次のような工数とファンクションポイントの関係式（回帰式）を示した．

$E_{mh}=54\times F_p-13390$ （相関係数：0.935）

ここで，

E_{mh}：工数（人時） F_p：ファンクションポイント

Kemerer は，15 のシステムについて上記の Albrecht の式による工数，COCOMO (Constructive COst MOdel の略)[8] による工数，および SLIM[9] による工数と，実工数を比較した[10]．その結果，算出された工数と実工数の比は，ファンクションポイントの場合は 1.2，LOC を使う COCOMO と SLIM の場合はそれぞれ 5.6 と 9.4 であった．この結果は，ファンクションポイントが工数を予測するためのよい量であることを示していると考えてよい．ただし，ファンクションポイント法に相当有利な結果となっているのは，Kemerer と Albrecht のデータの特性が類似していたためであろうと推定されている．

（3） プログラム行数見積りの精度

後述する LOC の問題点などから，筆者はプログラム行数とファンクションポイントとの関係はファンクションポイントの精度としては重要でないと考えている．しかし，プログラム行数からファンクションポイントへの移行期には便利なツールとなる．たとえば，Albrecht は次のような回帰式を求めている．

$S_s=118.7\times F_p-6490$ （相関係数：0.854）

ここで，

S_s：COBOL の LOC F_p：ファンクションポイント

2.5 ファンクションポイント法の適用対象ソフトウェア

ファンクションポイント法は入出力やファイルを機能としてとらえるため，このような処理が支配的なビジネスソフトウェア（バンキング，事務処理計算など）がその適用領域であるといわれている．CPU の処理が支配的な科学技術計算や OS などのソフトウェアに適用するため，ソフトウェアの特徴（ファンクションポイント法のファンクション）をとらえてソフトウェアの規模を定量化するというファンクションポイント法の考え方を拡張する試みは行われているが，報告例は多くない．

3. ファンクションポイントとプログラム行数 (LOC)

開発投資額（工数）と開発期間の予測は，一般にソフトウェアの量（あるいは規模）に基づいて行われることは前述した．したがって，工数と期間の予測の精度はソフトウェアの量の精度（見積り精度）に依存する．また，これらの予測は開発のできるだけ早い時期に精度よく行えることが重要である．

ソフトウェアの量を表すものとして従来多く用いられてきたプログラム行数 (LOC) は，開発の初期段階では不明であるため，通常，経験者の経験と勘（と度胸．この三つを併せて KKD と呼ぶことがある）によって推定される（見積もられる）．LOC およびそれを用いる方法は，以下のような問題点をもっており，ソフトウェア開発を予測するためのソフトウェアの量および手法としては必ずしも妥当なものではない．

a．属人的であり（経験者の「経験と勘」），再現性にとぼしいこと（定まった手法がない）

b．LOC は開発の結果として得られるものであり，開発初期に予測することは困難である（精度がよくない）

c．開発技術，開発環境に依存し，開発するソフトウェアの仕様に固有な量を表していない

これに対し，ファンクションポイントおよびファンクションポイント法は以下のような利点があり，ソフトウェア開発を予測するためのソフトウェアの量および手法として優れている．さらに，ソフトウェアのさまざまな属性を評価するための基準量としても用いることができる．

（1） 量として信頼できる

計測過程が手順化されており，計測過程を記録に残すことも，それから算出過程を再現することも容易である．この性質は，ファンクションポイントの信頼性を高めることに役立つ．

また，ファンクションポイントは，要求仕様書や機能仕様書から定められた手順に従ってファンクションを抽出し，評価することにより「測る」量である．「測る」は「予測」と違い，精度は必要なだけ上げることができる（ただし，ファンクションポイント法は数学的に厳密な体系ではないから，限界はある）．さらによいことに，ファンクションは開発の初期段階で比較的容易に抽出可能である．特に，ファンクションポイント法で重要な意味をもつ入出力は，この段階でも一つ一つ明確に識別できていることが多い．このため，開発の初期段階でかなりの精度でソフトウェアの量が確定できる．

（2） 開発の初期段階に用いる量として適切である

ファンクションポイントはユーザ視点にたった量である．ソフトウェアのユーザにとっては機能が重要であり，どんなふうに作られるか（これがプログラム行数に対応しているともいえる）は二義的であることが多い．開発の初期段階の見積りは，ユーザとの折衝に使用されることが多い．ユーザにとっては理解しやすく，また，開発者にとってはユーザとの共通的な認識をもてるユーザ視点にたった量に基づく見積りは重要である．

（3） 開発技術，開発環境に依存せず，開発するソフトウェアの仕様に固有な量を表している

ファンクションポイントの開発技術，開発環境，言語（これらを開発手段と呼ぶことにする）への依存度がきわめて低いことは，ファンクションポイントの計測手順から明らかである．このことは開発手段がソフトウェア開発に与える影響をファンクションポイントにより詳細に分析できるということを示している．このため，開発工数，期間を予測する場合には，開発手段の影響を予測に正しく反映でき，より精度の高い予測を行えることを意味する．

表-4 に以上の議論をまとめた，LOC とファンクションポイントの比較を示す．

4. ソフトウェア開発プロセスとファンクションポイント

ファンクションポイントは，ソフトウェアの基本的属性である「量」を表している．このため，ファンクションポイントはソフトウェア開発の種種の局面で使用可能である．ただし，ファンクションポイントはソフトウェアのツクリをみないため，ツクリを管理する進捗管理には適していないと考える．したがって，ファンクションポイント（法）のソフトウェア開発での適用方法は，1)見積りと，2)評価が代表的であろう．**図-3** にソフトウェア開発プロセスとファンクションポイント利用法の関係を示す．ファンクションポイントを用いれば，ソフトウェア開発の種々の様相を分析することが可能である．そのためには，ファンクションポイントとともに，開発に関するさまざまなデータを蓄積しておくことが重要になる．なお，ここでは説明の容易さから開発モデルとしてウォータフォールモデルを想定しているが，ファンクションポイント法自体はモデルとは独立であり，プロトタイピングなどのモデルにも適用可能である．

（1） 見　積　り

要求分析工程から基本設計工程で作成されたドキュメントからファンクションポイントを計測する．計測したファンクションポイントと過去の開発データから当該ソフトウェア開発のコストや期間の見積りを行う．

精度がそれほど問題にならなければ（たとえば，よくいわれる，「とりあえずオーダが分かればよい」などの類），開発の計画段階からも使用

表-4 LOC とファンクションポイントの比較

開発工数／期間の見積りに必要な属性（特に上流工程での見積り）		LOC			ファンクションポイント	
ソフトウェアの量としての信頼性	算出法	低い	経験と勘	属人的	高い	決められた手順
	再現性		とぼしい			優れている
	獲得法		予測			計測
開発初期段階での適用性	精度高く獲得できること	できない			できる	
	ユーザとの交渉に使えること	使いにくい（ソフトウェアのツクリに関係した量であり，ユーザには理解しにくい）			使いやすい（ソフトウェアの機能と直結した量でありユーザには理解しやすい）	
ソフトウェアの仕様に固有であること		固有でない（開発技術，環境，言語に強く依存）			固有である（開発技術，環境，言語に依存しない）	

可能である．このときは，ファンクションの複雑さ，入出力，内部論理ファイルの種類などは過去のデータから仮定する．これらの見積りの基礎としたデータも後で実績データなどから検証が可能である．

ファンクションポイントの精度は，要求仕様分析あるいは機能設計の精度であることに注意する必要がある．計測の対象となるものがきちんとできていなければ，いくら精密に計測しても無意味である．ファンクションポイント法を導入するにあたって，「仕様は変更されるもの」といった従来ソフトウェア開発の「常識」を洗い直してみる必要がある．「Garbage in, Garbage out」という言葉があるが，ファンクションポイント法を適用するときにも，肝に銘じておくべき言葉である．

（2） 評　　価

試験から受け入れにかけてファンクションポイントを使用して，開発したソフトウェアの評価を行うことができる．その際，必要があればファンクションポイントを再計測する．再計測は，開発したシステムの最終的なファンクションポイントを決定するためである．ファンクションポイントを使った指標の例には次のようなものがある．

a．仕様変更率＝最終ファンクションポイント／初期ファンクションポイント

b．生産性1＝プログラム行数／ファンクションポイント

c．生産性2＝投入コスト／ファンクションポイント

d．生産性3＝ドキュメント枚数／ファンクションポイント

e．品　　質＝摘出バグ数／ファンクションポイント

ファンクションポイントは，少なくとも従来の量の指標であったプログラム行数を置き換え，より精度の高い指標を与える．また，上記の仕様変更率のような新しい指標を与えることもできる．ファンクションポイントのこのような性質が，ソフトウェア開発の定量化を前進させると考える．

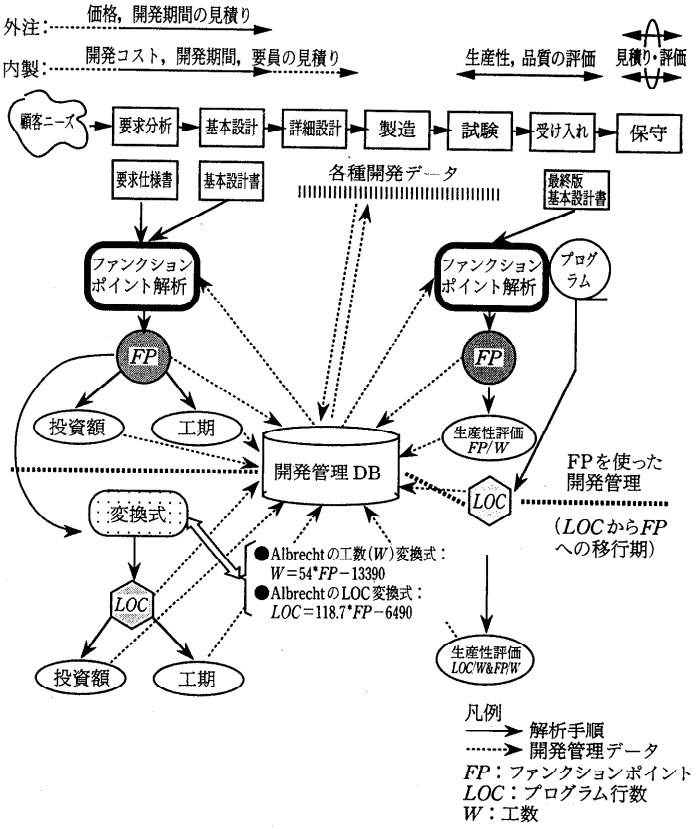


図-3 ソフトウェア開発プロセスとファンクションポイントの利用法

5. ファンクションポイント法の利用状況

ファンクションポイント法の国内，海外の利用状況，および，標準化の状況について簡単に述べる．

（1） 日本国内の利用状況

日本でのファンクションポイント法利用はあまり活発ではない．しかしながら，大手の何社かはすでにファンクションポイント法を実際の開発に使用していると聞いている．筆者の属するNTTにおいても試験的に使用を開始した．また，多くのソフトウェアハウスが関心を寄せており，自社内でいろいろ研究を行っているとも聞いている．

ファンクションポイント法が普及しない原因はさまざまであろうが，その導入法にも問題がある．斉藤らは，ファンクションポイント導入を阻害する要因を考察し，トップダウン的な導入，他

手法との併用，手法の選択，必要なドキュメントを提案している[11].

（2） 外国の状況

恐らく，世界中で最もファンクションポイント法が普及している国は米国であり，これを支えているのが IFPUG である．IFPUG は，米国に本拠をおくファンクションポイント法のユーザ団体であり，現時点での de facto なファンクションポイント標準化団体でもある．その会員数は個人，団体合わせて 500 を超え，伸び率はおおむね年率 15% であると言われている[12]．会員は金融，公共事業，ソフトウェアハウス，コンピュータメーカなど多彩であるが，筆者の印象では，通信関係の会社の加入が多いようである．また，オーストラリア，カナダ，南アフリカ，イギリス，フランス，オランダ，ドイツに支部をもつ．

米国で実際にどの程度ファンクションポイント法が使用されているかはなかなかつかめないが，昨年秋の IFPUG のユーザ会議で報告された以下の AT&T のデータが一つの参考になる[13].

- 10年以上使用して，計測したファンクションポイント数は 800,000 になる（**2.4**(3) の回帰式を使うと約 95 M-LOC になる）
- 600 のプロジェクトに適用した（AT&T の全プロジェクトの 20%）
- 計測には 60 人年を投入した

ヨーロッパもファンクションポイント法の使用に比較的熱心なようである．早稲田大学の東らが実施したソフトウェア管理とメトリクスのアンケートの結果によれば，回答したヨーロッパ企業の 25% 以上が生産性の指標としてファンクションポイント/単位時間を使うとしている．これは，LOC/単位時間に次いでよく使われている指標である[14].

（3） 標　準　化

ファンクションポイント法を ISO の新しい検討項目 (NWI) とするという提案が昨年 (1993年) の 6月の国際会議（東京で開催）で行われ，同年 12 月の投票で正式な検討項目として認められた[15]．今後検討が活発化していくものと思われる．この提案は IFPUG 派であるオランダ，オーストラリアが主体となって行われており，標準は IFPUG ベースとなることが予想される．

6. ファンクションポイント法のバリエーション

ファンクションポイント法には種々のバリエーションがあるが，それらは適用領域を拡張するかしないかで大別できる．

（1） 適用領域を拡張しないもの

この分類に属する de facto standard な手法が IFPUG 法である．IFPUG 法は，複雑さの評価の客観化やルールの精密化／適性化などの変更は行っているが，基本的には Albrecht の手法をそのまま引き継いでいる．

IFPUG 法以外の手法としては日本国内では JISA☆[16], IPA☆☆[17], COSDES☆☆☆[18] による提案がある．

国外では，C. Jones らによる SPR 法[19], C. R. Symons による Mark II 法[20]などがある．

（2） 適用領域を拡張するもの

これに属する手法の数は多くないが，科学技術計算やリアルタイムソフトウェアへの適用を目的として C. Joncs が開発したフィーチャポイント法[19], プロセス制御用ソフトウェアへの適用を目的として Mukhopadhyay らによって開発された手法などがある[21].

7. 今後の課題

ファンクションポイント法を有効な尺度（メトリクス）とするためには，ソフトウェア開発に関する各種のデータを蓄積するとともに，その分析のノウハウも蓄積する必要がある．これらは，ファンクションポイント法を使っていく開発現場の課題である．

ファンクションポイント法の技術的な研究課題としては次のようなものがある．

（1） ファンクションポイント法に適した仕様記述法

現在でもよく使われている自然言語と図表類による仕様書記述では，ファンクションや関連ファイルが明確に判定できるような記述法を考案する必要がある．

☆ 情報サービス産業協会 (Japan Information Service Industry Association)
☆☆ 情報処理振興事業協会 (Information-Technology Promotion Agency, Japan)
☆☆☆ ソフトウェア開発見積システム技術委員会 (Committee on Software Development Estimation System)

今後普及が進むとも思われる構造化分析設計法に対して IFPUG は近く計数規約をこれに適合するように改定する予定にしている (CPM 4.0). しかし，これは完成を意味するわけではなく，今後適用を積み上げ，開発現場で発生する具体的な問題点を分析し，技法にフィードバックをかけていく必要がある.

オブジェクト指向分析設計法との関連は検討が始まったところである.

(2) 適用分野の拡張

データが少なく CPU を多く使うようなソフトウェア（いわゆる CPU リッチなソフト．たとえば，科学技術計算，OS など）にも機能量に相当する概念を適用し，その量を正しく表すことが必要である．このアプローチはすでに始まっているが，評価は確定しておらず，この分野の研究はこれからと言わねばならない.

(3) 自 動 計 測

ファンクションポイントをより正確に，より手軽に計測するためには，自動計測ツールが必要である．CASE ツールと組み合わせてファンクションポイント計測の自動化を試みているツールもあるが，本来の自動化とはかなりの隔たりがある．自動計測率の向上のための研究がさらに必要である.

8. おわりに

ファンクションポイント法について，手法の概略とファンクションの意味，従来技法との比較，適用法，動向などについて述べてきた．本稿により，ファンクションポイント法の全体が幾分明らかにできたと考える．ファンクションポイント法は 100% の完成度をもった手法ではない．しかし，従来のプログラム行数をベースにした技法に比べてかなり前進した技法であると考える.

ファンクションポイント法は，よいマニュアルとスキルの積み上げで，精度高くかつ速く計数できる．ファンクションポイントの計測工数は，大きなプロジェクトであればほとんど無視できるオーダである．試験的に使用して感触をつかむことが大事であろう.

残念ながらファンクションポイント法に関する公開された日本語のよい文献は少ない．英文ではあるが，具体的な事例に基づいてファンクションポイント法を解説した，B. Dreger の Function Point Analysis[22] はよい参考になるであろう.

拙い論説であったが，本稿がきっかけとなって，ファンクションポイント法の理解に役立てば幸甚である.

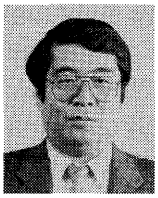
謝辞 最後になったが，本稿を書くにあたって有益なご助言をいただいたソフトウェア研究所古山主幹研究員，斉藤主任研究員に感謝する.

参 考 文 献

1) DeMarco, T.: Controlling Software Project: Management, Measurement & Estimation, Yordon Press (1982).
2) Albrecht, A. J.: Measuring Application Development Productivity, Proc. Joint SHARE/GUIDE/IBM Appl. Develop. Symposium, pp. 83-92 (1979).
3) Albrecht, A. and Gaffney, J. Jr.: Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Tr. on SE, Vol. 9, No. 6, pp. 639-648 (1983).
4) IFPUG: IFPUG Function Point Counting Practice Manual, Release 3.4 (1992).
5) Kemerer, C. F.: Reliability of Function Points Measurement, CACM, Vol. 36, No. 2, pp. 85-97 (1993).
6) Kemerer, C. F.: Improving the Reliability of Function Point Measurement: An Empirical Study, IEEE Tr. on Software Engineering, Vol. 18, No. 11, pp. 1011-1024 (1992).
7) 西山，斎藤，古山：ファンクションポイント計数上の問題点に関する分析，1994 年電子情報通信学会春季大会，D—95.
8) Boehm, B.: Software Engineering Economics, Prentice-Hall (1981).
9) Putnam, L. H.: A General Empirical Solution to the Macro Software Sizing, and Estimation Problem, IEEE Tr. on Software Engineering, Vol. 4, pp. 345-381 (1978).
10) Kemerer, C. F.: An Empirical Validation of Software Cost Estimation Models, CACM, Vol. 30, No. 5, pp. 416-429 (1987).
11) 斉藤，西山，古山：FP 法導入に関する一提案，情報処理学会第 48 回全国大会，4K—10.
12) IFPUG: New Member Orientation, IFPUG 1993 Fall Conference Proceeding.
13) Lubashevsky, A.: Living on the Edge at AT&T, IFPUG 1993 Fall Conference Proceeding.
14) Azuma, M. and Mole, D.: Software Management Practice and Metrics: EC and Japan—Some Result of Questionaire Survey, Proc. 2nd International Conference on Achieving Quality Software, pp. 57-71 (1993).
15) ISO/IEC JTC 1/N 2629 Attachment N 1067 A: Function Point Analysis (1993).

16) 情報サービス産業協会 (JISA): 情報処理工学に関する調査研究 ソフトウェアコストモデルの定量的評価, 昭和59年度, 昭和60年度, 昭和61年度.
17) 情報処理振興事業協会 (IPA): ソフトウェアの規模見積り手法の調査研究最終報告, 平成2年3月 (元技—109).
18) ソフトウェア開発見積りシステム技術委員会 (COSDES): COSDES 版 FPA ミニセミナー資料 (平成5年10月).
19) Jones, C. 著, 鶴保征城, 富野 壽監訳: ソフトウェア開発の定量化手法, 共立出版 (1993).
20) Symons, C.: Software Sizing and Estimating, John Wiley & Sons (1991).
21) Mukhopadhyay, T.: Software Effort Models for Early Estimation of Process Control Applications, IEEE Tr. on Software Engineering, Vol. 18, No. 10, pp. 915–924 (1992).
22) Dreger, B.: Function Point Analysis, Prentice-Hall (1991).

(平成6年2月16日受付)



西山 茂 (正会員)

1950年生. 1973年電気通信大学電気通信学部電波通信学科卒業, 1975年同大学院電波通信専攻科修士課程修了, 同年日本電信電話公社 (現, NTT) 武蔵野電気通信研究所に入所, 現在 NTT ソフトウェア研究所に勤務. 入所後伝送関係の研究に従事した後, 1980年よりソフトウェアの研究にテーマを移し, 日本語プログラミング, APL, 言語変換, Ada, ソフトウェア開発環境の研究実用化を行う. 現在, ソフトウェア見積り技術の研究, ソフトウェア品質保証技術の研究に従事. 訳書「ソフトウェア開発の定量化手法」(共訳, 共立出版), ISO/IEC JTC1/SC7/WC6 委員, 電子情報通信学会会員.