社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2000-45 (2001-01)

JAVAプログラムからのファンクションポイント計測に関する一考察

今川 勝博† 楠本 真二† 井上 克郎†‡

†大阪大学 大学院基礎工学研究科 情報数理系専攻
〒560-8531 大阪府豊中市待兼山町 1-3
Phone: 06-6850-6571 Fax: 06-6850-6574
‡奈良先端科学技術大学院大学 情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5
E-mail: {imagawa, kusumoto, inoue}@ics.es.osaka-u.ac.jp

あらまし
ソフトウェアの開発規模を見積もる手段としてファンクションポイント法が用いられることが多くなっている．ファンクションポイント法は，ソフトウェアの機能要件だけを抽出して定量的に計測する手法である．ファンクションポイントは一般的に要求仕様書や分析・設計書等から計測されるが，過去の開発において，それらの中間生成物が既に存在せず，最終生成物であるソースコードしか存在しない場合も多い．そこで本研究では，オブジェクト指向開発されたプログラムからファンクションポイントを計測するための手法について検討する．また，その手法に基づいてJavaソースコードからのファンクションポイント計測ツールを開発し，その評価を行なう．

キーワード ファンクションポイント，見積もり，オブジェクト指向開発，Java

キーワード

Function Point Measurement for JAVA Program

Masahiro Imagawa†, Shinji Kusumoto† and Katsuro Inoue†‡

†Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan
Phone: +81-6-6850-6571 Fax: +81-6-6850-6574
‡Graduate School of Inforamtion Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
E-mail: {imagawa, kusumoto, inoue}@ics.es.osaka-u.ac.jp

Abstract
Function point analysis (FPA) was proposed to help measure the functionality of software systems. It is widely used to estimate the system size and the effort required for the actual system development in software development. However, if an organization tries to introduce FPA, FP will have to be counted from the past software developed there, and this counting is time-consuming. For the past software, the requirements specification and design specification were sometimes missing. In such case, FP must be counted from only the source code. In this paper, we propose detailed FPA measurement rules for JAVA program and develop the function point measurement tool.

Key words Function point analysis, Estimation, Object-oriented development, Java

1 まえがき

ソフトウェアの大規模化・複雑化に伴い，高い品質を持ったソフトウェアを開発するためには，明確な開発計画の下で開発プロセスの全工程を系統づけて管理することが重要になってきている．明確な開発計画を立てるためには，ソフトウェアの規模，投入する工数，開発期間，開発に使用される技術などを予測する必要があるが，中でも重要なものは開発工数と開発期間である [11]．通常，開発工数や開発規模を予測するのに，まずソフトウェアの規模を見積もり，これに基づいて開発工数と開発期間を予測する手法がとられている．

ソフトウェアの機能的な規模を見積もる手段の一つとしてファンクションポイント法 [2] がよく用いられている．ファンクションポイント法は，ソフトウェアの機能要件だけを抽出して定量的に計測する手法で，ソフトウェア開発の初期段階における成果物である要求仕様書や設計仕様書等から計測される．従って，その値は開発環境や開発言語などの技術要件に左右されない一定の値になる．

一般に，ファンクションポイント法による見積もりを開発現場に導入する際には，過去の開発において計測されたファンクションポイント値とその開発に要した開発工数や開発期間等の実績データを蓄積し，ファンクションポイント値とそれらの間に何らかの関係式を見出す必要がある．また，蓄積されたデータ数が不十分な場合，関係式の正確性が低下し，開発規模の見積もりが不正確なものとなる．従って，過去の開発における成果物からファンクションポイントを計測しなければならない．しかし，ファンクションポイントを手作業で計測するためには教育や導入のためのコストが必要となる．更に，要求仕様書や設計仕様書等の中間成果物が存在せず，最終成果物であるソースコードしか存在しないという場合も多い．そこで，本研究ではオブジェクト指向開発方法論に基づいて開発されたシステムを対象として，ソースコードからファンクションポイント値を計測するための手法について検討する．具体的には，ファンクションポイント法の主流技法の一つであるIFPUG法に基づいて，ソースコードからファンクションポイント値の計測を行うための計測ルールを提案する．更に，提案したルールをファンクションポイント計測ツールとして実装し，その有効性の評価を行う．

以降，2では，ファンクションポイント法，及びIFPUG法について説明する．3では，Java言語 [9] を用いて開発されたシステムのソースコードからファンクションポイント値を計測するための手法，及び今回の提案手法に基づいて開発したファンクションポイント計測ツールについて説明する．4では，実際にJavaソースコードから計測したファンクションポイント値についての評価を行なう．最後に5で，まとめと今後の課題について述べる．

2 ファンクションポイント

2.1 ファンクションポイント法

ファンクションポイント法は，ユーザの要求から機能要求仕様の大きさを定量的に測定する手法で，A.J.Albrechtによって1979年に提案された．求められる計測値は，機能量または機能規模と呼ばれる．また，機能量の単位としては，伝統的にファンクションポイントという呼称が使われている．機能量の計測では，計測対象ソフトウェアの機能のうち，画面や帳票，ファイルなどを通じた情報の入出力に着目し，それらを種類別に数え上げ，それぞれの複雑さによって重み付けを行ない加算した値を機能量とする．このようにして得られる機能量の規模尺度としての長所は，(1) 規則に従って計測される値であるため，誰が計測しても同じ値が得られる，(2) 開発環境や開発言語などの技術要件に左右されず，機能仕様にだけ依存する，という点が挙げられる．

現在，ファンクションポイント法は目的等に応じて様々な改良や変更が行なわれ，数十種類の計測方法がある [5][6]．本研究では，数多くのファンクションポイント法の中から，ファンクションポイント標準化の中心的組織であるIFPUGが定めており，日本においても主流技法として用いられているIFPUG法を用いてファンクションポイントの計測を行なう．

2.2 IFPUG法 [1]

IFPUG法は，Albrecht版のファンクションポイント法に対して複雑さの評価の客観化やルールの精密化・適正化などの変更を行なったバージョンである．IFPUG法は，図1に示す手順で計測する．

Step1 算出種類の選択

算出種類を，(1) アプリケーションファンクションポイント: アプリケーションソフトウェアの大きさを表すファンクションポイント，(2) 新規開発

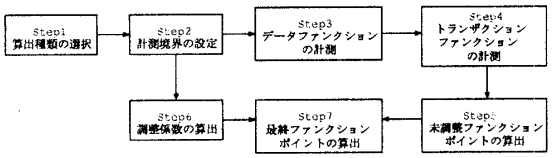

図 1: ファンクションポイント計測手順

プロジェクトファンクションポイント: 新規にアプリケーションを開発するプロジェクトの規模を知るために使用するファンクションポイント, (3) 機能改良プロジェクトファンクションポイント:すでに存在するアプリケーションを改良するプロジェクトの規模を知るために使用するファンクションポイント, の 3 種類から選択する.

Step2 計測境界の設定

計測境界とは, ファンクションポイントを測定したい対象アプリケーション自体を境界の内部, 他のアプリケーションおよびユーザを境界の外部になるように設定する境界のことである. 境界の大きさはサブシステム程度を目安にし, 一つの境界はユーザからみて機能面で閉じていなければならない.

Step3 データファンクションの計測

データファンクションとは, アプリケーション中にあり, ユーザが認識できる論理的な意味でのデータのまとまりのことである. データファンクションにはファンクションタイプと複雑さの要素があり, それらを用いてデータファンクションのファンクションポイントが決定される. データファンクションには以下の二種類のファンクションタイプがある.

- 内部論理ファイル: 計測対象のアプリケーションによってデータが更新されるデータファンクション.
- 外部インターフェースファイル: 計測対象のアプリケーションによってデータが参照されるデータファンクション (データは更新されない).

次に, それぞれのデータファンクションをデータ項目数, レコード種類数という二つの要素によって低・中・高の三段階の複雑さに分類する.

- データ項目数 (Data Element Type,DET): データファンクションを構成する, ユーザが認識できる論理的なデータ項目の数.
- レコード種類数 (Record Element Type,RET): データファンクションの中に存在する, 異なる意味合いを持つデータのまとまりの個数.

更に, データファンクションの複雑さを決定するために表 1 を使用する.

表 1: データファンクションの複雑さ

RET \ DET	1-19	20-50	51-
1	低	低	中
2-5	低	中	高
6-	中	高	高

Step4 トランザクションファンクションの計測

トランザクションファンクションとは, アプリケーションに対するデータの出入りを伴う処理のことである. トランザクションファンクションにもファンクションタイプと複雑さの要素があり, それらを用いてトランザクションファンクションのファンクションポイントが決定される. トランザクションファンクションには以下の三種類のファンクションタイプがある.

- 外部入力: 計測境界外からのデータ入力によりデータファンクションが更新される処理.
- 外部出力: 計測境界外へのデータ出力を含む処理のうち, データファンクションのデータを加工して出力する処理.
- 外部照会: 計測境界とのデータ入出力により, データファンクションのデータを参照する処理.

次に, それぞれのデータファンクションをデータ項目数, 関連ファイル数という二つの要素によって低・中・高の三段階の複雑さに分類する.

- データ項目数 (Data Element Type, DET): 計測境界を出入りするデータ項目の個数.
- 関連ファイル数 (File Type Referenced, FTR): 対象となるトランザクションファンクションの処理中にデータが更新または参照されるデータファンクションの個数.

トランザクションファンクションの複雑さを決定するための表については文献 [1] を参照されたい.

Step5 未調整ファンクションポイントの算出

Step3, Step4 の結果をもとに, ファンクションタイプ・複雑さ別に表 2 を用いて重み付けを行ない, それらの値を合計した値が未調整ファンクションポイントとなる.

Step6 調整係数の算出

未調整ファンクションポイントは「データのまとまり」と「データの出入り」のみに着目した値であり, 性能, 信頼性, ユーザーインタフェースなどについては考慮されていない. そこでシステム特性をファンクションポイントに反映させるため

表 2: 未調整ファンクションポイント算出表

ファンクションタイプ \ 複雑さ	低	中	高
内部論理ファイル	7	10	15
外部インターフェースファイル	5	7	10
外部入力	3	4	6
外部出力	4	5	7
外部照会	3	4	6

に，表 3 に示すシステム特性の 14 項目を 6 段階で評価し，その結果から調整係数を算出する．調整係数はファンクションポイント算出の際に，未調整ファンクションポイントを補正する役割がある．

表 3: システム特性の 14 項目

1	データ通信機能	8	オンライン更新
2	分散データ処理	9	複雑な処理
3	性能条件	10	再利用性
4	高負荷構成	11	インストレーションの容易さ
5	トランザクション率	12	運用の容易さ
6	オンラインデータ入力	13	複雑サイト
7	エンドユーザーの効率	14	変更の容易さ

$$\text{調整係数} = 0.01 \times \text{影響度の合計} + 0.65$$

Step7 最終ファンクションポイントの計測

未調整ファンクションポイントと調整係数を用いて最終ファンクションポイントを算出する．Step1 の算出種類によって算出方法が異なる．

- アプリケーション FP = 未調整 FP × 調整係数
- 新規開発 FP = (未調整 FP + 移行分未調整 FP) × 調整係数
- 機能改良 FP = (変更部分の新未調整 FP + 追加部分の未調整 FP + 移行分未調整 FP) × 新調整係数 + 削除部分の未調整 FP × 旧調整係数

2.3 実用面における課題

定量的にソフトウェアの機能量を計測することができるファンクションポイント法を用いることで，ソフトウェア開発プロジェクトに際して，開発工数や開発期間を見積もることが可能となる．

しかし，詳細な部分のファンクションポイントの計測には測定者の判断が必要になる．結果として，同一プロダクトに対してのファンクションポイントの計測であっても，計測する人間によって誤差が生じてしまうという問題点が指摘されている [4]．例えば，同じ組織内の人間が同じプロダクトに対して測定した場合は 12%，違う組織の人間が測定した場合は 30%以上の誤差が出るという報告もされている [3]．

また，ファンクションポイントを実際に応用して見積りを行うためには，過去の開発において計測されたファンクションポイント値とその開発に要した工数や期間に関するデータを蓄積し，その関係式を導き出さなければならない．そのためには関係式が得られるだけの十分な過去の開発に関するデータが必要となり，その計測のためのコストが現場への導入の妨げとなっている．更に，一般にファンクションポイントは要求仕様書や設計仕様書から計測されるが，過去の開発における成果物として，最終成果物であるソースコードしか存在しないことも多く，そのような場合にはファンクションポイント値の計測が難しい．

そこで，以降ではソースコードからファンクションポイント値の計測を行なうための手法について検討する．

3 提案するファンクションポイント計測手法

ここでは，オブジェクト指向プログラミングで作成されたシステムからファンクションポイントを計測するために提案する手法について説明する．本提案手法では，IFPUG 法に基づいて計測を行なうため，中心となるのはシステムからのデータファンクションとトランザクションファンクションの二種類のファンクションを抽出する作業である．以下では，まずファンクションポイントの基本的な計測方針について述べ，次にファンクションの抽出方法と複雑さの決定方法，及び，今回試作したファンクションポイント計測ツールについて説明する．

3.1 基本方針

ファンクションの基本的な抽出方針として，計測対象となるシステムのすべての機能に対応するような入力データを，例えばユースケース [8] に対応させて作成する．作成された入力データに基づいて実際にシステムを動作させて，ファンクションポイント計測に必要な情報を得る方法をとる．

ファンクションポイント計測に必要な情報を，ソースコードの静的な解析結果のみからではなくシステムの実行過程から抽出する理由としては，主に次の 2 点がある．

- 実際に動作する機能に対するファンクションポイントを計測するため．つまり，ソースコード中に含まれるユーザに見えないコード等 (例えば，開発保守用のコード) は計測対象としない．
- 計測に必要な情報量を少なくするため．

データファンクションとトランザクションファンクションそれぞれの基本的な抽出方針を以下に示す.

- データファンクション: システムを構成するクラスのいずれかをデータファンクションとする. IFPUG 法におけるデータファンクションは, システム内に存在する論理的なデータのまとまりのことを指すが, オブジェクト指向におけるクラスも同様の意味を持つと考える.
- トランザクションファンクション: データファンクションであるクラスと他のクラスとのメッセージの送受信をトランザクションファンクションとする. IFPUG 法におけるトランザクションファンクションは, データファンクションを更新・参照するようなシステムへの入出力を伴う処理のことであるが, データファンクションをクラスと考えたときに, クラスが保持する情報を更新・参照するメッセージの送受信がトランザクションファンクションであると考える.

以降では, ファンクションの識別方法と複雑さの決定方法の詳細について説明する.

3.2 データファンクションの計測

データファンクションの識別

計測対象システムを構成するクラスの中から, ユーザがデータファンクションをクラス単位で指定する.

次に, データファンクションを内部論理ファイルまたは外部インターフェースファイルのいずれかに分類する. 分類方法は, データファンクションに指定されたクラスについて, システムの実行中に, 引数が渡されるメソド呼び出しが一度でもある場合は, 内部論理ファイルとする. つまり, メソド呼び出しの際に渡される引数によって, クラスの保持するデータが更新されると考える. そのようなメソド呼び出しが一度もない場合には, クラスの保持するデータは更新されることがないと考え, 外部インターフェースファイルとする.

データファンクションの複雑さ

IFPUG 法では, データファンクションの複雑さを決定する要素として, DET と RET の二つがある. DET は, データファンクションを構成するデータ項目の個数のことを意味し, RET は一つのデータファンクション中に存在する異なる意味合いを持つデータのまとまりの個数を意味する.

本手法ではクラスをデータファンクションに対応させているため, DET はクラスで定義されているクラス変数の中で, int, char, boolean 等の型として宣言されている変数の個数とし, RET はクラス型として宣言されているクラス変数の個数とする. つまり, クラス型として宣言されているクラス変数は, ある意味合いを持ったデータのまとまりであると考える.

DET, RET の値によって, データファンクションの複雑さを低・中・高の三段階に分類し, その複雑さによって重み付を行ないファンクションポイントを算出する. その際に用いる分類表と算出表は IFPUG 法で用いられているものと同じものを使用する.

3.3 トランザクションファンクションの計測

トランザクションファンクションの識別

データファンクションに指定されたクラスと他のクラスとのメッセージの送受信をトランザクションファンクションとする.

次に, 抽出されたトランザクションファンクションを外部入力, 外部出力, 外部照会のいずれかに分類する. 例えば, データファンクションに指定されたクラスで宣言されているメソドが, データファンクションではない他のクラスで呼び出された場合, メソドに引数が渡されていれば, データファンクションとなるクラスの保持するデータを更新するメソド呼び出しであると考え, このメソド呼び出しは外部入力であるとする. トランザクションファンクションの抽出及び分類のルールを図 2 に示す.

トランザクションファンクションの複雑さ

IFPUG 法では, トランザクションファンクションの複雑さを決定する要素として, DET と FTR の二つがある. DET は計測境界を出入りするデータ項目の個数のことを意味し, FTR は対象となるトランザクションファンクションの処理中にデータが更新または参照されるデータファンクションの個数を意味する. 本手法ではトランザクションファンクションは, クラス間のメッセージ送受信に対応するため DET はメソド呼び出しの引数の個数とする. FTR については, 今回提案するトランザクションファンクション抽出ルール (図 2 参照) では, 一つのトランザクションファンクション, つまりメソド呼び出しが複数のデータファンクションを更新または参照することがない. したがって FTR はすべてのトランザクションファンクション

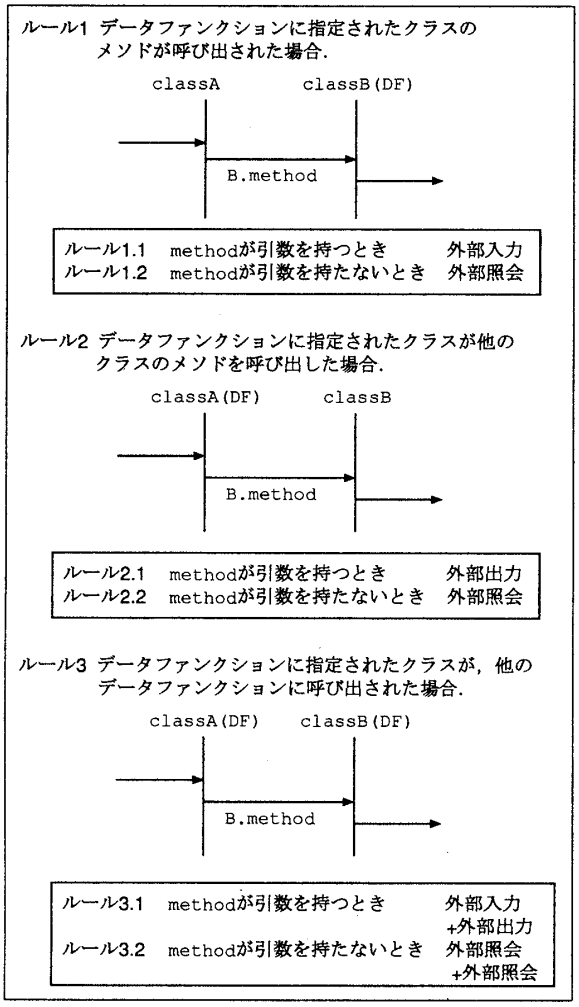

図 2: トランザクションファンクション抽出ルール

について 1 となる.

また，その他のルールとして，次の二点がある.

- 同一 DF 間でのメソド呼び出しはトランザクションファンクションとしない.
- 実行中に現れた異なるトランザクションファンクションについて，メソドを呼び出したクラス，メソドが呼び出されたクラス，呼び出されたメソド，引数の個数，引数の型が同じであれば，同一トランザクションファンクションとし，一回だけ計測する.

DET，FTR の値によって，トランザクションファンクションの複雑さを低・中・高の三段階に分類し，その複雑さによって重み付を行ないファンクションポイントを算出する．その際に用いる分類表と算出表は IFPUG 法で用いられているものと同じものを使用する.

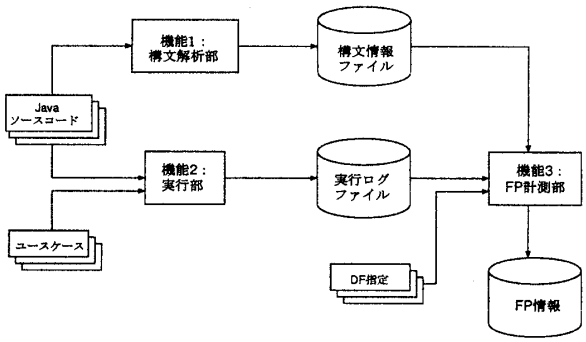

図 3: 計測システムの構成

3.4 計測ツール

今回提案した手法に基づいて，Java ソースコードからファンクションポイント計測を行なうためのツールを開発した．ツールは Windows2000 上で Java を用いて実装している．図 3 に計測システムの構成を示す．開発したシステムは構文解析部，実行部，FP 計測部の三つの機能を持つ．構文解析部は，計測対象となるシステムのソースコードを入力として，FP 算出に用いるための構文情報ファイルを出力する．構文情報ファイルの詳細は後述する．実行部は，対象となるシステムのソースコードとユースケースを入力として，実際に計測対象システムを動作させ，その実行の過程から FP 算出に必要な情報を抽出し，実行ログファイルとして出力する．実行ログファイルについての詳細も後述する．最後に FP 計測部は，構文情報ファイルと実行ログファイル，及びユーザによるデータファンクションの指定を入力として FP を算出し出力する.

構文情報ファイル

構文情報ファイルの形式を表 4 に示す．構文情報ファイルに記述されている情報として，まずクラス名がある．次に，そのクラスで宣言されているクラス変数の名前，その変数の型，フラグ (その変数がデータファンクションの複雑さを決定する際に DET と見なされるのか，もしくは RET と見なされるのかを判別するために用いる) からなる「CV」の行がある．次に，そのクラスで宣言されているメソド名があり，そのメソドで宣言されている変数についても，クラス変数と同様の情報が「MV」の行に記述されている．表 4 は一つのクラスについての情報で，構文情報ファイルには全クラス分がこのような形式で記述されている.

表 4: 構文情報ファイルの形式

C	クラス名		
CV	クラス変数名	型	DETflag
⋮	⋮	⋮	⋮
M	メソド名		
MV	メソド変数名	型	DETflag
⋮	⋮	⋮	⋮
M			
MV			

表 5: 実行ログファイルの例

Begin	Sakaya.Sakaya.1.String
Begin	Souko.Souko.2.String.int
Begin	Haisou.Haisou.0
End	Haisou.Haisou.0
End	Souko.Souko.2.String.int
End	Sakaya.Sakaya.1.String

実行ログファイル

実行ログファイルの例を表 5 に示す．実行ログファイルにおいて，「Begin」で始まる行はシステムの実行中にあるメソドが呼び出されたことを意味する．逆に，「End」で始まる行はあるメソドが終了したことを意味する．「Begin」「End」の後には「クラス名.メソド名.引数の個数」と続き，引数の個数に応じて引数の型が記述される．表 5 の例では，まず，Sakaya クラスの Sakaya メソドが引数一つで呼び出され，その引数は String 型である．次に Souko クラスの Souko メソドが引数二つで呼び出され，その引数は String 型と int 型である．最後に Haisou クラスの Haisou メソドが引数なしで呼び出され，呼び出されたメソドが再帰的に終了している．表 5 を図式化したものを図 4 に示す．

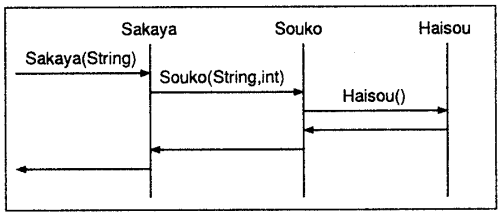

図 4: 表 5 の対応図

4 評価実験

4.1 適用

開発した FP 計測ツールを用いて本提案手法の評価実験を行なった．計測対象は，Java で記述された約 700 行からなる酒屋問題のシステム [12] である．このシステムは在庫管理や発注受付等の機能を持った事務処理系のシステムである．Java ソースコードから計測された FP 値を評価するための比較対象として酒屋問題の要求仕様書からも FP 値の計測を行なった．その結果を表 6 に示す．また，その際の計測画面を図 5 に示す．

表 6: 計測結果

ソースコードからの計測値	要求仕様書からの計測値
44	40

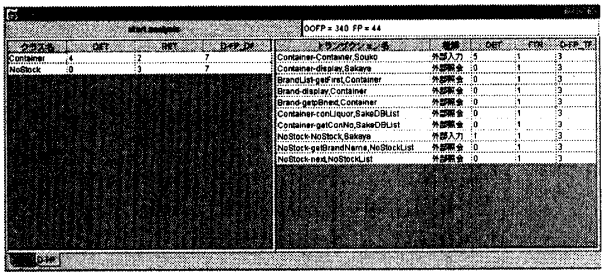

図 5: ツールの計測画面

4.2 分析

計測対象とした酒屋問題は比較的小規模なシステムであったが，ソースコードから計測したファンクションポイント値と要求仕様書から計測したファンクションポイント値には誤差が生じた．この誤差は，要求仕様書上で一つのトランザクションファンクションとして計測された処理が，実際のソースコード上ではデータファンクションに指定されたクラスと他のクラスとの複数回のメッセージ送受信によって実装されていることが原因であった．つまり，本提案手法におけるトランザクションファンクション抽出ルールでは，一つのメソド呼び出しを一つのトランザクションファンクションとして抽出するため，ある処理を実行するための複数回のメソド呼び出しから複数個のトランザクションファンクションを抽出していたのである．

4.3 考察

今回の評価実験では，要求仕様書上で一つのトランザクションファンクションとして計測されていたファンクションが，ソースコード上では複数のトランザクションファンクションとして計測されていたために誤差が生じた．

同様に，要求仕様書上では一つのデータファンクションであっても，ソースコード上では複数のクラスで実装されるという場合も考えられるため，さらに大規模なシステムに対して本提案手法を適

用した場合，誤差が大きくなると考えられる．その対策として次のような方法が考えられる．

- 複数個のクラスをまとめて一つのデータファンクションとするルールを追加する．この場合，単にデータファンクションの個数を少なくできるだけではなく，まとめられたクラス間でのメッセージ送受信は，同一データファンクション間でのメッセージ送受信となり，トランザクションファンクションとして計測されなくなるため，トランザクションファンクションの個数も少なくすることができる．
- 複数回のメソド呼び出しを一つのトランザクションファンクションとするルールを追加する．

5 まとめ

本研究では，ファンクションポイント法の主流技法であるIFPUG法のルールに基づいて，Javaソースコードからファンクションポイント値を計測するための計測ルールの提案を行なった．また，提案したルールに基づいてファンクションポイント計測ツールを開発し，その評価を行なった．今後の課題としては，

- 複数個のクラスを一つのデータファンクションとするルールや複数回のメソド呼び出しを一つのトランザクションファンクションとするルール等，計測値の正確性が向上するであろうと予測できるルールを用いて計測ツールの改良を行なう．
- さらに規模の大きいシステムに対して計測ツールを適用し，新たな計測ルールの追加を行なう．
- これまでに開発してきたファンクションポイント計測ツール[7][10]と統合する．

などが挙げられる．

参考文献

[1] IFPUG: "Function Point Counting Practices Manual, Release 4.0", International Function Point Users Group(1994).

[2] A.J.Albrecht:Function Point Analysis, in Jhon J.Marciniak, editor, Encyclopedia of Software Engineering, Vol.1, John Wiley & Sons, pp.518-524, (1994).

[3] B.A.Kitchenham:"The Problem with Function Points", IEEE Software, Vol.14, No.2, pp.29-31, March/April, (1997).

[4] G.C.Low and D.Ross Jeffery:"Function Points in the Estimation and Evaluation of the Software Process", IEEE Transactions on Software Engineering, Vol.16, No.1, pp.64-71, January, (1990).

[5] C.Symons:"Software Sizing and Estimating", John Wiley & Sons(1991).

[6] C.Symons:"Function Point Analysis: Difficulties and Inprovements", IEEE Transactions on Software Engineering, Vol.14, No.1, pp.2-10, January, (1988).

[7] T.Uemura, S.Kusumoto and K.Inoue: "Function Point Measurement Tool for UML Design Specification", Proceedings of the Sixth International Software Metrics Symposium, pp.62-69, (1999).

[8] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard: Object-Oriented Software Engineering - A Use Case Driven Approach -, Addison-Wesley(1992).

[9] J. Gosling, B. Joy, and G. Steele: The JAVA™ Language Specification, Addison-Wesley(1996).

[10] 柏本, 楠本, 井上, 鈴木, 湯浦, 津田: "イベントトレース図に基づく要求仕様書からのファンクションポイント計測手法", 情報処理学会論文誌, Vol. 41, no. 6, pp.1895-1904(2000).

[11] 中村 永:"科学技術計算とリアルタイム制御に向くソフト計測手法", 日経エレクトロニクス, No.658, pp.175-185(1996-03).

[12] 山崎利治:"共通問題によるプログラム設計技法解説", 情報処理学会誌, Vol.25, No.9, p.934(1984).