特別研究報告

題目

実行履歴から抽出された
トランザクションファンクション候補の分類手法の提案と実装

指導教員
楠本 真二 教授

報告者
枝川 拓人

平成 19 年 2 月 20 日

大阪大学 基礎工学部 情報科学科

平成 18 年度 特別研究報告

実行履歴から抽出された
トランザクションファンクション候補の分類手法の提案と実装

枝川 拓人

内容梗概

ファンクションポイント (FP) 法は，ソフトウェアの持つ機能数を基にソフトウェアの規模を測定する手法である．測定された FP 値は，工数見積等に利用される．通常，FP 値は設計仕様書から計測されるが，最終生成物であるソースコードから FP 値を計測する手法が求められている．私の所属する研究グループでは，十分な入力データを与えてプログラムを実行し，その実行履歴情報に基づいて FP を自動計測する手法について研究を行っている．

FP 計測では，ユーザが識別できる入出力機能であるトランザクションファンクション (TF) を抽出する必要がある．通常，実行履歴情報には同一の TF が複数回現れてしまう．FP 計測時には，同一の TF を重複して数えないために，実行履歴情報から同一の TF を識別する必要がある．

本研究では，実行履歴を解析して得られた TF 候補の集合を，同一の TF 集合に分類する手法を提案する．提案手法は，複数のメトリクスを使用して TF 候補間の類似度を測定し，各類似度を総合的に判断して TF 候補を分類する．提案手法を，ある企業で使用されている Web アプリケーションの実行履歴より得られた複数の TF 候補に対して適用した．その結果，高い精度で同一の TF の分類を実現することができた．

主な用語

見積り
ファンクションポイント
メトリクス
類似判定
分類

目 次

1　まえがき

効率の良い開発には，正確な見積りが必要不可欠である．ソフトウェアの分野においてもそれは例外ではなく，その要求から様々な見積り手法が考案された．ファンクションポイント法（FP 法）もその一つである．FP 法ではソフトウェアの機能数を基にソフトウェアの規模を計測する，計測値は FP と呼ばれ，工数見積りの基礎になる．FP から工数を正確に取得するには，開発に要した工数と計測された FP を蓄積して，工数と FP の関係式を作らなければならない．そのため，新たに FP を導入して工数見積りにしたい場合，過去の FP を計測する必要がある．だが，FP は通常仕様書から計測するため，仕様書が存在しない場合や，仕様書が存在しても最終成果物に実装された機能に対応していない場合，過去に開発された FP の計測が困難な状況に陥ることがある．これらの問題に対応するため，私の所属する研究グループでは，最終成果物 (ソースコード等) からの FP 自動計測というテーマで研究を行っている．

FP はデータのまとまりであるデータファンクション (DF) と，入出力機能であるトランザクションファンクション (TF) で構成される．研究グループはこの TF を計測する手法として，計測対象ソフトウェアから実行履歴を取得し，TF が実行された形跡を TF 候補として，実行履歴上から抽出するという手法を提案している．しかしこの手法では，１つの TF が複数回実行された場合，TF 候補も複数抽出されてしまうという問題が生じる．この問題を解決するため，同一 TF から発生した TF 候補をまとめる「TF 候補の分類」という作業が必要になる．この「TF 候補の分類」が本研究のテーマである．

本研究における提案手法は，複数のメトリクスを使用して TF 候補間の類似度を測定し，各類似度を総合的に判断して TF 候補を分類する，というものである．評価実験として，ある企業で使用されている Web アプリケーションの実行履歴から抽出された二つの TF 候補群と，授業の一環で学生によって作成された Web アプリケーションの実行履歴から抽出された TF 候補群に対して提案手法を適用した．結果，全ての適用実験において精度の高い分類結果が得られた．

以降，２節ではソフトウェア見積りの概要について説明する．続いて３節では，ファンクションポイント法について説明する．４節では，研究グループが提案するファンクションポイント計測手法を紹介する．５節では，トランザクションファンクション候補の類似度の計測手法について述べ，６節でその評価実験と考察を行う．７節では，類似度を用いた具体的な分類手法を提案し，８節でその評価実験と考察を行う．最後に９節でまとめと今後の課題について述べる．

2　ソフトウェア見積り

2.1　見積り

ソフトウェアの開発において，見積りが占める重要度は大きい．効率のよい，混乱のない，迅速な開発を行うためには適当な開発プランをたてることが肝要であり，その為には安定して正確な情報の取得，すなわち見積りが必要不可欠になる．このことはソフトウェアの分野に限った話ではない．

見積りは昔から行われてはいたが，経験豊富な職人の根拠のない独断によって為されるものがほとんどであった．このような，言わば勘のような手法で安定して正確な見積りを提供できる職人は少なく，勘である以上その手法の普及は望めない．また，根拠がないため顧客を納得させることも難しい．

結果,「安定して正確な」「誰にでも出来て」「誰にでも理解できる」見積りを目指し，様々な手法が考案されることとなった．以下，その一部を紹介する．

2.2　見積り手法

ソフトウェア開発における見積りの対象としては「工数・工期・コスト・品質」など様々なものが考えられる．その中で「工数」を計測する手法として代表的なものに COCOMO 法 [1] というものがある．この COCOMO 法は，開発チームの能力や開発環境などを補正値として「ソフトウェアの規模」に掛け合わせることで工数を導き出す，というのが大まかな手法であるが，この手法を適用するには「ソフトウェアの規模」を知る必要がある．

ソフトウェアの規模を計測する基準として，LOC(Line Of Code) つまりコードの行数というものがある．LOC は計測が単純であり，ソフトウェアの規模にある程度の相関があることは自明であるため，良く用いられてきた．しかし，LOC で規模を計測するにあたり，2 つの大きな問題がある．1 つ目は，設計段階で計測するには推測に頼るしかないこと，2 つ目は，無駄なコードが製品の価値を上げるという矛盾が生じること，である．特に 2 つ目の問題は重大で，見積りがソフトウェアの価値を下げるなどは，あってはならないことである．

こうして LOC に対する不満が高まる中，これに変わる規模計測基準として注目されている手法がファンクションポイント法である．

3　ファンクションポイント法

3.1　概要

ファンクションポイント法（以下 FP 法）は，1979 年に Allan J. Albrecht によって提案されたソフトウェア規模計測手法である [2]．FP 法は，実装に依存しない計測手法を目標として開発された．計測手法は，ユーザ視点でソフトウェアの機能を抽出し，その処理内容の複雑さなどから重み付けをして，合計することでファンクションポイント (以下 FP) を計測する，というものである．

3.2　IFPUG 法

FP 法で明確に定義されていない「機能の定義の仕方」「重み付けの仕方」などに対して，様々な標準化がなされた．その中で最も広く使用されている手法が IFPUG 法である．本研究でも IFPUG 法を対象としているため，以下 IFPUG 法に限定して説明していく．

3.2.1　機能の種別

IFPUG 法では，計測する機能を 5 種類に分けて考える．(1) 内部論理ファイル (ILF)，(2) 外部インタフェースファイル (EIF)，(3) 外部入力 (EI)，(4) 外部出力 (EO)，(5) 外部照会 (EQ) である．これらを図で表すと図 1 のようになる．

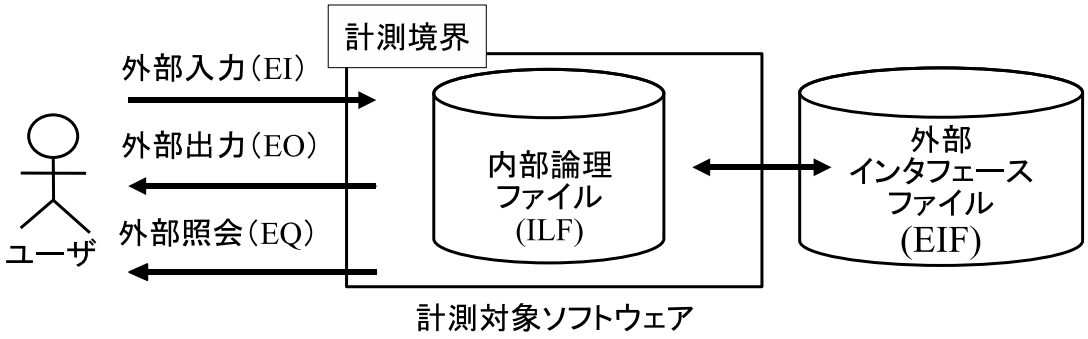

図 1: 計測モデル

これら 5 つの機能は 2 種類に大別される．ILF と EIF がデータファンクション (DF)，EI，EO，EQ がトランザクションファンクション (TF) である．データファンクションはデータのまとまりを表し，トランザクションファンクションは入出力処理を表す．ILF と EIF の区

別は計測対象内部で管理されるデータ (ILF) か否 (EIF) かでなされ，EO と EQ の区別は，要求に対しなんらかの処理を通す (EO) かそのまま返す (EQ) かによってなされる．

3.2.2 機能の重み付け

IFPUG 法において，各機能の重みは機能の種類とその複雑さによって決まる（表 2）．複雑さは高・中・低の 3 種類あり，表 1 で求められる．表 1 において，RET：レコード種類数，DET：データ項目数，FTR：関連ファイル数，である．

表 1: 複雑さの算出

データファンクション

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

外部入力

FTR \ DET	1〜5	6〜19	20〜
0〜1	低	低	中
2〜3	低	中	高
4〜	中	高	高

外部出力

FTR \ DET	1〜5	6〜19	20〜
0〜1	低	低	中
2〜3	低	中	高
4〜	中	高	高

外部照会

FTR \ DET	1〜4	5〜15	16〜
0〜1	低	低	中
2	低	中	高
3	中	高	高

表 2: 重みの算出

データファンクション

機能 \ 複雑さ	低	中	高
ILF	7	10	15
EIF	5	7	10

トランザクションファンクション

機能 \ 複雑さ	低	中	高
EI	3	4	6
EO	4	5	7
EQ	3	4	6

3.2.3 FP 値の算出

抽出された全機能に重み付けをして，合計した値が「未調整 FP」となる．この「未調整 FP」では性能・信頼性・ユーザインタフェースなどのシステム特性を評価していないため，14 項目のシステム特性を 6 段階評価して調整係数を算出し，「未調整 FP」を補正して FP 値を算出する．

3.3 実用上の問題点

FP 法には実用上，以下のような問題点がある．

- 一般的な計測ルールが述べられているだけで，測定者によって誤差が出る
- 基礎データとして必要な「過去の開発物の FP 値」を計測することが困難
  - 設計仕様書が残っていない場合がある
  - 設計仕様書と実装された機能が対応していない場合がある
  - 過去の開発物に対する見積りという余分なコストの発生
- FP 計測導入のための初期コスト（教育）が必要

4　ソースコードからのファンクションポイント自動計測

4.1　研究グループの目標とその有用性

前述した「FP 法の実用上の問題」を解決するため，私が所属する研究グループは，開発の成果物（ソースコード等）からの FP 自動計測というテーマで研究を行っている．

成果物から計測することで，設計仕様書がない場合や，設計仕様書と実装された機能が対応していない場合に対処し，自動で計測することで計測コストの問題を解消する．またこのような手法を実装したツールがあれば，過去の開発物に対する FP 値の計測だけでなく，開発前に仕様書から見積もった FP 値と，開発後にツールで見積もった FP 値とを比較する，といった用途にも効果が期待できる．

4.2　Java プログラム中のデータファンクション・トランザクションファンクション

私の所属する研究グループは，研究の第一歩として Java プログラムを対象とした FP 自動計測を目標とした．Java プログラムから FP を計測するためには，FP を構成するデータファンクション（DF）とトランザクションファンクション（TF）が，ソースコード上でどのように実現されているかを知る必要がある．

Antoniol らによる解析の結果，DF はクラスで実装され，TF はメソッド呼び出し群で実装されることが多い，という結論が得られた [6]．つまり，データのまとまりを表現するクラスが作られて，そのクラスに対する入出力処理がメソッド呼び出し群として現れる．また，このときデータ項目はデータファンクションに相当するクラス (DFC) のインスタンスフィールドとして実装されている．

4.3　Java プログラムからの FP 自動計測の提案手法

以上のことを踏まえて，私の所属する研究グループは以下のような「Java プログラムからの (未調整)FP 計測手法」を提案した．まず，以下のような状況を前提とする

- 計測対象プログラムは，実行履歴が取得可能
- 全ての TF を実行するテストケースが存在する
- DF に対応するクラスが存在し，ユーザによって DFC として指定される

このような前提の下で以下の手順に基づいて FP を計測する．

1. テストケースを全て実行し，実行履歴を取得
2. DFC のフィールド・メソッドからのデータフローを解析

3. 解析したデータフローを基に，実行履歴上から TF 候補を抽出

4. 抽出した TF 候補を分類し，TF 群を作成

5. TF の種別判定 (EI，EO，EQ) をする

6. TF が利用する DFC とそのフィールドから各 TF に重み付けする

7. DFC にもフィールドを基に重み付けする

8. 7.8. から未調整 FP を算出

本研究は,「4. TF 候補の分類」をテーマとした.

4.3.1 トランザクションファンクション候補分類の必要性

本研究でテーマとした，TF 候補分類の必要性について説明する．実行履歴上に存在する TF を使用した形跡（TF 候補）が 1 対 1 に TF と対応すれば問題はないが，実際はそうではない場合が多い．1 つの TF が全てのテストケースの中で複数回使用されることは通常起こり得るので，その場合，実行履歴上の TF 候補が 1 つの TF に対して複数存在することになる．これをそのまま複数回数えてしまうと計測値が大きく誤ってしまうことになる．よって，同一 TF から発生した TF 候補を検出してまとめる，という作業が必要になる．

5　トランザクションファンクション候補間の類似度

この節では，トランザクションファンクション候補（以下 TF 候補）を分類する材料として使用する，TF 候補間の類似度の計測手法について説明する．

5.1　トランザクションファンクション候補

TF 候補とは，FP 計測手法の手順 2 によって抽出された，実行履歴上における時間範囲のことである．具体的には，実行時に発生したあるメソッド呼び出しの開始時から，それ以降に発生したあるメソッド呼び出しの開始時までの範囲を表す．手順 2 によって TF 候補が正しく抽出されているならば，この範囲の中では，何らかの 1 つの TF が実行されていると考えられる．つまり理想的には，1 つの TF 候補には 1 対 1 で対応する TF が存在する．しかし実際には，手順 2 による抽出を常に完全に行うことは難しいため，本研究で対象とする TF 候補には TF と対応していないもの，あるいは，複数の TF と対応しているものなども含まれることとする．

5.2　メトリクスと類似度

メトリクスとは「尺度」のことであり，対象をある視点において評価する基準となるものである．例えば，人間を評価するなら「身長」「年齢」「性別」などといったメトリクスが考えられる．このようなメトリクスを用いて対象間の類似度を計測する場合，その計測値の差（距離）を非類似度として計測することになる．複数のメトリクスで測定された場合も，メトリクス分の次元を用意すれば各要素間の距離が間単に求められる（図 2）．この距離を用いた要素の分類手法は，数多く提案されている [7][8]．しかし，計測対象を「要素」ではなく「要素の組」としたメトリクスも考えられる．例えば，人間そのものでは無く人間の組を評価する「共通の友人の数」「共通の趣味の数」といったメトリクスである．このように，対象を「要素の組」とすると要素間の類似度に直結するメトリクスが設定できるというメリットがある．しかし，対象を「要素」とした場合と同じように分類すると,「要素の組」の分類になってしまう（図 2）．その為「要素」の分類をしたい場合，[7][8] のような既存手法が適用できないので，新しい分類方法を考える必要がある．本研究では，メトリクスの設定のしやすさからメトリクスの対象は「要素の組」とした．分類手法は次節で説明する．

5.3　類似度計測に使用可能な情報

本研究で TF 候補の類似度計測に使用できる情報には,「実行履歴」,「実行履歴中の DFC を起点・終点とする全てのデータフローグラフ」がある．これらについて説明する．

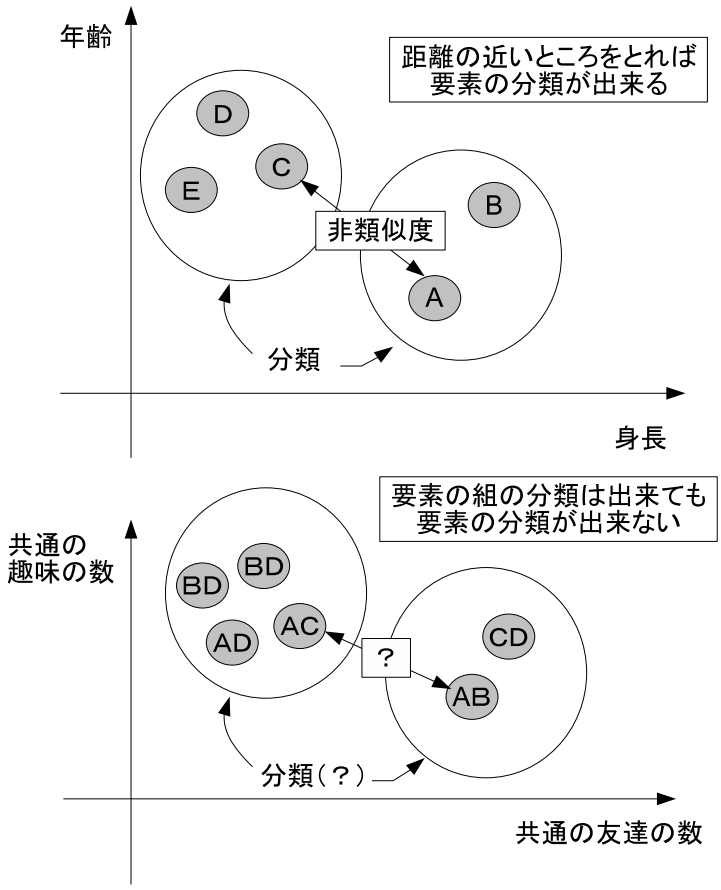

図 2: メトリクス対象の違いによる分類への適用

5.3.1 実行履歴

本研究で用いる実行履歴には，実行時に発生したメソッド呼び出し系列が記録されており，これを用いることで，どのようなメソッドがどのような順番で使用されたかを知ることが出来る．各列からは，メソッド名，パッケージ名，クラス名，引数の型，返り値の型などの情報が入手できる．TF 候補は，この実行履歴上におけるメソッド呼び出しの範囲として提供される．

5.3.2 データフローグラフ

本稿で述べるデータフローグラフとは，実行履歴中に記録されているメソッド呼び出しイベントと，そのメソッド内で発生していると考えられるフィールド参照・代入イベントを頂点とし，それらの間のデータの流れを辺として表現したグラフである．本研究では，DFC

のフィールド参照やメソッドの終了イベントを表す頂点から派生するデータフローを表すグラフ（forward-flow-graph）と，DFC のフィールドへの代入やメソッド呼び出しイベントを表す頂点に至るデータフローを表すグラフ（backward-flow-graph）の 2 種類のグラフを用いる．

なお，backward-flow-graph では，DFC へのデータ入力を開始点として，データのフローを逆に辿ることでグラフを作成してある．つまりどちらのグラフも，DFC へのアクセスが開始点になる．

さらに抽出されたグラフ群からは，以下の 2 種のフローによるグラフを除去してある

- forward-flow で利用されないデータを DFC にセットした backward-flow
- bakward-flow でセットされていないデータを利用する forward-flow

その理由を簡単に説明する．

データベースにデータを格納するようなシステムにおいては，DFC は中継点として利用される．このような場合，データのフローが DB ⇔ DFC ⇔外という形になる．このとき，DFC が起点・終点となるフローを全て抽出すると，データの利用とは本質的に関係のない DB ⇔ DFC のフローも抽出してしまう．このようなフローの多くは上記の 2 種に該当するため，このようなフローグラフの除去を行っている．

5.4 考案したメトリクス

「どのようなメソッド」を使って「どのような DFC」や「どのようなデータ項目」に「どのような手順」でアクセスするか，ということが TF 候補の特色を示し，これらが一致するほど TF 候補間の類似度が高いと考えた．そこで，TF 候補間の類似度を計測するメトリクスとして以下の 4 つを考案した．なお全てのメトリクスにおいて，計測値（類似度）は 0〜1 とする．

MG(x,y) ：TF 候補 x と TF 候補 y に関連するメソッド群の類似度

DC(x,y) ：TF 候補 x と TF 候補 y に関連する DFC 群の類似度

DI(x,y) ：TF 候補 x と TF 候補 y に関連するデータ項目群の類似度

DFG(x,y) ：TF 候補 x と TF 候補 y に関連するデータフローグラフ群の類似度

まず,「TF 候補に関連するメソッド群」とは，TF 候補が持つ実行履歴の範囲において出現するメソッド群である.「TF 候補に関連する DFC 群，データ項目群，データフローグラフ群」は，与えられたデータフローグラフ群から抽出する．どのように抽出するかを以下に説明する．まず，与えられたグラフ群から，以下の条件を満たすグラフを抽出する．

- TF 候補に関連するメソッドを通過している
- 以下の 2 条件のどちらかを満たす
    1. TF 候補範囲の 1 %以上の長さをフローしている
    2. 算術演算または文字列連結を受けている

条件 1.2. の存在理由を説明する．仕様上利用されないはずのデータが，実装上ではデータベースから別のデータを取得する際にレコード単位で一括して取得されてしまうケースがある．このような本来利用しないデータのフローは除去して，実際にその機能において利用されているデータからのフローだけを抽出したい．そこで，

- ある一定以上の長さをフローしているデータはその機能において長い期間利用されるデータであり，一括して取得され実際には使用されないデータではない可能性が高い．
- 算術演算や文字列連結を行われるようなデータは，機能上重要な役割を果たすデータであり，ついでに取得されるようなデータではない

という考えを基に，条件 1.2. を加えた．実際この条件を付加することで，一括して取得されるが実際には使用されないデータのフローを効果的に除去できるという実験結果が研究グループにより得られているが，本研究で行ったことではない為省略する．

ここで抽出されたグラフ群の「各開始点におけるクラスの集合」を「TF 候補に関連する DFC 群」とし,「各開始点におけるクラスとフィールドの組の集合」を「TF 候補に関連するデータ項目群」とした．各データフローグラフの開始点は DFC のアクセスであることと，DFC のフィールドがデータ項目を表現しているという前提から，このような定義を行った．

さらに，TF 候補の範囲内でフローが納まる（TF 候補の範囲内でフローが始まり，終わる）グラフ群を抽出し，これを「関連するデータフローグラフ群」と定義した．これは，類似度計算する際にグラフの長さで重み付けするため，ただデータが後の方で使用されるだけの広域的なフローが類似度に大きな影響を与えてしまわないようにするためである．

5.5 各メトリクスでの類似度計算法

全てのメトリクス M(x，y) は「TF 候補 x から TF 候補 y への類似度」と「TF 候補 y から TF 候補 x への類似度」の平均として計算する．これは本研究の発展課題である，ある TF 候補が別の TF 候補に包含されるケースの検出（包含 TF 候補の検出）に拡張しやすいように配慮したものである（9 節参照）．以下，メトリクス M における TF 候補 x から TF 候補 y への類似度を M(x → y) と書く．

5.5.1 MG，DC，DI の計算法

前述したような方法で抽出した，TF 候補 x に関連する各項目（MG ならメソッド，DC ならデータファンクションクラス，DI ならデータ項目）の集合を $R(x)$ と置いて，$M(x \to y)$ を以下の式 (1) で定義した．

$$M(x \to y) \ = \ \frac{\mid R(x) \cap R(y) \mid}{\mid R(x) \mid} \tag{1}$$

5.5.2 DFG の計算法

グラフ N からグラフ M への類似度が $K(N, M)(0 \leqq \mathrm{K} \leqq 1)$ と与えられたとする．グラフ群 G からグラフ群 H への類似度 $S(G \to H)$ を式 (2) のように定義した．ここで $L(N)$ とはグラフ N の実行履歴上の長さであり，これを用いて各グラフ類似度に重み付けをした．

$$S(G \to H) \ = \ \frac{\sum_{N \in G} (L(N) \ \max_{M \in H} \ K(N, M))}{\sum_{N \in G} L(N)} \tag{2}$$

TF 候補に関連するグラフ群は，forward-flow と backward-flow の 2 種類ある．そこで TF 候補 x に関連するグラフ群を，forward：$F(x)$，backward：$B(x)$ と置いて $DC(x \to y)$ を以下の式 (3) のように定義した．

$$DFG(x \to y) \ = \ S(F(x) \to F(y)) \ + \ S(B(x) \to B(y)) \tag{3}$$

以下，グラフ類似度 K の計算法について説明する．

Tree Kernel

Collins らは，２つの木の類似度を計測する Tree Kernel を提案している [11]．本研究で扱うグラフは性質的に木と言えるので，この計測手法を基にグラフ類似度を考えた．Tree Kernel は類似度を「２つの木が共通に持つ部分木の数」と定義している．ノード n，m を根とする部分木中に含まれる共通部分木の数を $C(n, m)$ とすると，Tree Kernel は式 (4) で与えられる．

$$K_A(N, M) \ = \ \sum_{n \in N} \ \sum_{m \in M} C(n, m) \tag{4}$$

さらに Clooins らは，以下のような $C(n, m)$ の効率的な計算方法を提案している [11]．

- n と m の子ノードを導出する規則が異なるとき，$C(n, m) \ = \ 0$

- n と m の導出規則が等しく，n と m がともに前終端記号のとき，$C(n, m) = 1$
- n と m の導出規則が等しく，n と m がともに前終端記号ではないとき，式 (5)

$$C(n, m) = \prod_{i=1}^{nc(n)} (1 + C(ch(n, i), ch(m.i))) \tag{5}$$

ここで，nc(n) は n の子ノードの数を示し，ch(n, i) はノード n の i 番目の子ノードを示す．このとき，類似度を調べるノードの順番を後順序にすれば，式 (5) を計算する際，子ノードの C を再計算する必要がない．これにより計算時間のオーダーは，比較する木のノード数を n，m とすると $O(n, m)$ で抑えられる．

Tree Kernel の拡張

[11] で提案されている Tree Kernel は自然言語の解析を目的としており，比較する木も句構造木という特殊な木を対象としている．しかし，本研究では一般的な 2 つの木の部分木の数を求める手法として適用したい．そこで，$C(n, m)$ の計算アルゴリズムを以下のように変更した．

- n と m が異なる節点であるとき，$C(n, m) = 0$
- n と m が等しい節点であり，n と m がともに葉のとき，$C(n, m) = 1$
- n と m の等しい節点であり，n と m がともに葉でないとき，式 (6)

$$C(n, m) = \prod_{x \in cg(n)} \prod_{y \in cg(m)} (1 + C(x, y)) \tag{6}$$

ここで cg(n) は n の子節点の集合である．これにより，一般的な 2 つの木の共通部分木の数が求められる．例として，図 3 のような 2 つの木に対する各節点間の C を表 3 に示す．これら図・表において数字は説明のために付与したものであり，節点が等しいか否かの比較はアルファベットのみで行うこととする．

式 (4) より，これらの木の類似度は表 3 の全要素を合計した「22」となる．

高橋らは，$C(n, m)$ を使った木の類似度として，式 (4) で与えられる K_A の他に，以下の 2 つの類似度 K_B, K_C を提案している [12].

$$K_B(N, M) = \sum_{n \in N} \max_{m \in M} C(n, m) \tag{7}$$

$$K_C(N, M) = \max_{n \in N} \max_{m \in M} C(n, m) \tag{8}$$

表 3 で考えると，K_B は各行の最大値の和なので「21」，K_C は全体における最大値なので「9」である．3 つの類似度の特徴は，以下のように説明されている．

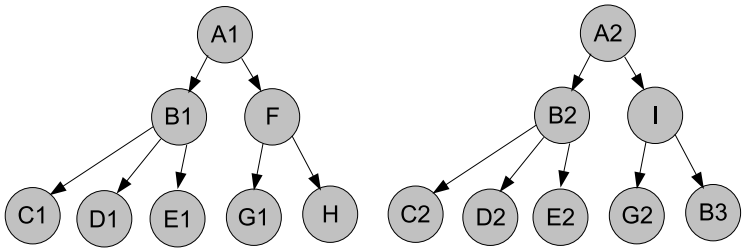

図 3: Tree Kernel の適用例

$K_A(N, M)$ ：共通する部分木を多く含む木ほど類似度が高い

$K_B(N, M)$ ：N の部分木が不足なく M に含まれるほど類似度が高い.

$K_C(N, M)$ ：両方の木のマッチする部分が大きいほど類似度が高い

本研究では，N から M への類似度という特徴を出したかったことから，K_B を採用した．さらに，グラフ類似度は 0 から 1 の間に収めなければならないという制約があるので，式 (9) を使って正規化した．なお，この正規化の方法は [12] で推奨されているものである．式 (2) に示したように，グラフ群の類似度計算の中で，N からグラフ群 H が含む全ての M への類似度を計算する．この正規化は，$K_B(N, N)$ を一度計算すれば，N からグラフ群 H が含む全ての M への類似度計算に対して使用できるので，この点からも K_B は本研究に適しているといえる．

$$K'_B(N, M) = \frac{K_B(N, M)}{K_B(N, N)} \tag{9}$$

表 3: 図 3 における $C(n, m)$ の行列

A1	0	0	0	0	0	0	0	9
F	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0
G1	0	0	0	0	1	0	0	0
B1	0	0	0	8	0	1	0	0
E1	0	0	1	0	0	0	0	0
D1	0	1	0	0	0	0	0	0
C1	1	0	0	0	0	0	0	0
	C2	D2	E2	B2	G2	B3	I	A2

状態爆発

式 (6) の計算法にはある問題点がある．それは，１つの節点に同じ子が複数ある場合，状態爆発を起こし類似度が跳ね上がるおそれがあることだ．例えば図 4 のような 2 つの木を比較するとする．このときの各節点間の C を表にすると表 4 のようになる．

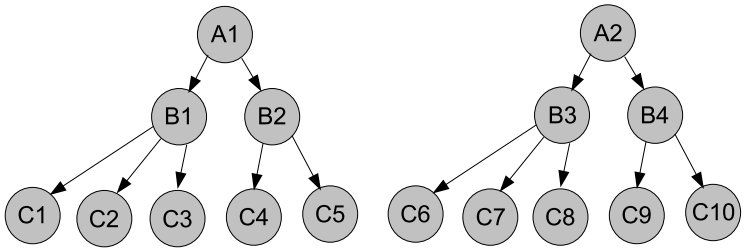

図 4: 状態爆発

図 3 の B1，B2 以下の木と，図 4 の B1，B3 以下の木は形は全く同じなのに，$C(B1, B2) = 8$, $C(B1, B3) = 512$ と大きくかけ離れている．$C(A1, A2)$ に至っては，比べ物にならないほどの差が出ている．ちなみに図 4 のような形の木では，複数の同じ子を持たない場合，最大（全く同じ木との比較）でも $C(A1, A2) = 32$ にしかならない．一見，正規化を行えば問題は解決しそうにも思えるが，図 5 のような木を比較するとき，部分木 E と部分木 F がどれだけ似ていても全体の類似度にほとんど影響しない，という問題は残る．

この対策として，あらかじめデータフローグラフに対し，１つの節点が同じ子を複数持たないように圧縮する処理を行った．アルゴリズムは，根から前順序でたどって行き，同じ子が複数あれば１つだけ残して他は消し，消した節点の子を残したノードへつなげるというものである．例を図 6 に示す．圧縮することで，例えば，図 4 の $C(B1, B3) = 2$ なり，もと

表 4: 図 4 における $C(n, m)$ の行列

A1	0	0	0	0	0	0	0	1,111,889,025
B2	0	0	0	64	0	0	512	0
C5	1	1	1	0	1	1	0	0
C4	1	1	1	0	1	1	0	0
B1	0	0	0	512	0	0	64	0
C3	1	1	1	0	1	1	0	0
C2	1	1	1	0	1	1	0	0
C1	1	1	1	0	1	1	0	0
	C6	C7	C8	B3	C9	C10	B4	A2

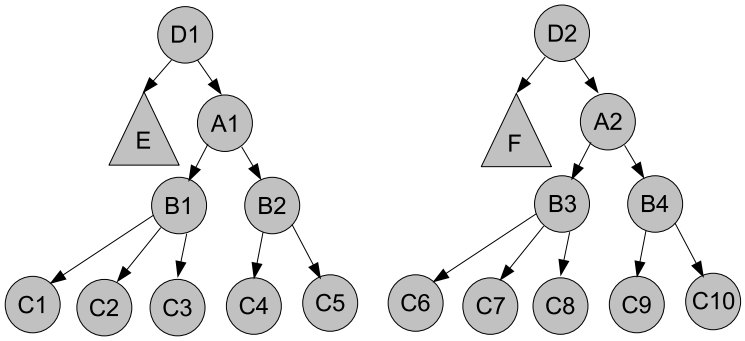

図 5: 状態爆発の弊害

もと同じ形である図 3 の $C(B1, B2) = 8$ よりも少ない値になってしまう，というようなことが起こるが，同じフロー（子の生成）が複数あってそれが一致するよりも，別のフローをしていて一致するほうが類似性が高いとも言える．よってこの圧縮後のフローの類似性と，圧縮前のフローの類似性の違いはあまり無いと考え，この対策を採用した．

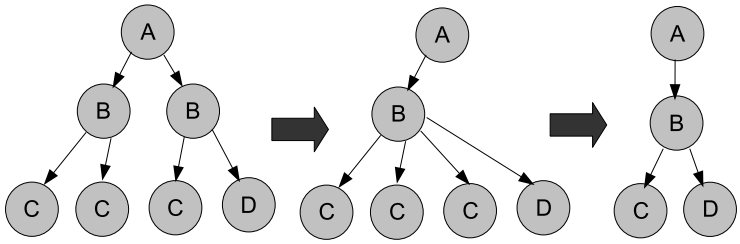

図 6: 圧縮処理手順

Tree Kernel を適用するグラフの削減

各グラフ比較の計算時間が良くても，本研究で比較するグラフの数は膨大なので，類似度計算するグラフは可能な限り削減しなければならない．そこで，グラフの開始点が違うグラフ同士の比較を削減した．グラフの開始点はデータファンクションへのアクセスである．全く違うデータファンクションに対する処理の比較は余り意味がないだろうという考えから，このような場合は類似度を計算することなく 0 とした．なお，このグラフの選別を行うことで実行時間が飛躍的に減少した．

グラフ類似度の計算法

結論として，式 (2) での $K(N, M)$ は以下のようにして計算される．

- N と M の開始点 (根) が違うとき，$K(N, M) = 0$
- N と M の開始点 (根) が同じとき，$K(N, M) = K'_B(N, M)$

6　メトリクスの評価実験

この実験では，全ての TF 候補の組み合わせに対する各メトリクスでの類似度を計算し，TF 候補が同一の TF の場合とそうでない場合のメトリクス計測値の振る舞いを比較する．メトリクスの振る舞いを評価する項目として，(1) 最大値，(2) 最小値，(3) 平均値，(4) 分散値の 4 種類を用意した．

6.1　実験対象ソフトウェア

メトリクスの評価実験に使用したソフトウェアは，ある企業で用いられている「ツール貸出システム」という Java で記述された Web アプリケーションで，コード行数は約 37000 行である,「ツール貸出システム」に含まれる TF を表 5 に示す．

表 5: ツール貸出システムに含まれる TF 一覧

利用中物品検索
利用中物品検索 (条件付)
物品検索
物品詳細
物品貸出
物品登録未承認案件一覧
物品新規登録 (一般向け)
物品登録未承認案件詳細
物品管理品一覧
物品管理品詳細
物品追加登録
備考表示
備考更新
アイテムグループ検索 (大分類)
アイテムグループ検索 (中分類)
コードテーブル検索

6.2　テストケースの説明

上記プログラムの機能を実行したいくつかの実行履歴を組み合わせてテストケースを作成した．

6.2.1 TestCase1

「ツール貸出システム」に対する 5 つの入力パターンから得られた実行履歴を利用したテストケース．この実行履歴群からは 26 個の TF 候補が抽出され，それらは 9 種類の TF に分けられる．実行履歴群に含まれる TF と，抽出された各 TF 候補がどの TF に対応するかを，表 6 に示す．抽出された TF 候補は A から Z のアルファベットで示す．

表 6: TestCase1 に含まれる TF と TF 候補の対応

抽出された TF	対応する TF 候補
利用中物品検索	A，D，H，I，L，P，V
物品検索	B，J
物品詳細	K
物品貸出＋物品詳細	C
物品登録未承認案件一覧	E，M，Q，W
物品新規登録 (一般向け)	G
物品管理品一覧	F，N，R，X
備考表示	O，S，U，Y，Z
備考更新	T

「物品貸出＋物品詳細」とした TF 候補は，実行履歴から抽出する際に 2 つの機能に対する処理が重なっていた為一つの TF 候補として抽出してしまったものである．本手法では，このような TF 候補も個別の TF に対応すると捉えて分類するが，最終的に FP を計測するにはこれらを分離する必要がある．この「複数の TF を含む TF 候補の検出と分離」は，今後の課題である．

6.3 実験結果

同一 TF 間，非同一 TF 間での，メトリクス計測値の振る舞いは以下のようになった．

表 7: TestCase1 におけるメトリクス計測値の振る舞い

同一 TF の組み合わせ (44 通り)

	最大値	最小値	平均値	分散値
MG	1.00	0.95	0.98	0.00
DC	1.00	1.00	1.00	0.00
DI	1.00	0.90	0.99	0.00
DFG	1.00	0.90	0.98	0.00

非同一 TF の組み合わせ (281 通り)

	最大値	最小値	平均値	分散値
MG	0.85	0.00	0.22	0.04
DC	1.00	0.00	0.26	0.15
DI	0.91	0.00	0.12	0.04
DFG	0.56	0.00	0.02	0.00

6.4 考察

表 7 から以下のようなことが読み取れる.

1. メトリクス全体において
    - 平均値が「同一 TF 間」において「非同一 TF 間」よりも非常に高い値を示す
    - 「同一 TF 間」において分散値が非常に低い
2. MG,DC,DI において
    - 「同一 TF 間」では DI が優秀な値を示す
    - 「非同一 TF 間」での最大値は MG が優秀な値示す
    - 「同一」,「非同一」での平均値を比較すると DI が優秀
3. DFG において
    - 「非同一 TF 間」において DFG が全ての値で最小
    - 「同一 TF 間」における DFG の値は他のメトリクスとほぼ同じ

これらから，以下のような判断をした.

1. 全てのメトリクスは TF の同一性に相関関係があり，ほとんどの同一 TF 間では平均値付近で安定している.
2. MG は適合率が，DC は再現率が高く，DI はその中間
3. DFG が最も優秀なメトリクスである.

さらにこの TestCase に対する各メトリクスの計測時間は，DFG：2 分弱，それ以外:1 秒程度であった.

7　トランザクションファンクション候補の分類手法

7.1　分類手法全体の方針

「メトリクスの評価実験」の結果，以下のようなことがわかった．

- 全てのメトリクスが分類に有用
- MG は適合率が，DC は再現率が高く，DI はその中間
- DFG が最も TF 候補との同一性に相関性が高いが，計算時間がかかる

このことから，MG，DC，DI で大まかに分類してその結果を組み合わせ，判断の難しいものを DFG で判断する方針を決定した．具体的には，以下のようなアルゴリズムで行う．

1. ３つのメトリクス (MG，DC，DI) で TF 候補を個別分類
2. ３つの個別分類結果の統合
3. 個別分類において判断の分かれた組み合わせを DFG で判断し，最終分類

以下，詳しい手順を説明する．

7.2　メトリクス個別による分類手法

基本的な方針は，類似度に閾値を設定して類似・非類似を判定し，類似 TF 候補同士を群にする，というものである．しかし，その際に気をつけなければならないことがある．それは「推移律」である．閾値を使って類似判定する以上,「A と B が類似」で「B と C が類似」なのに「A と C が非類似」となる可能性がどうしても存在する．この問題を解決するため，類似判定は以下のアルゴリズムによって行った．

1. 初期閾値を設定する
2. 閾値を超える類似度を持つ組み合わせを抽出
3. 抽出した組み合わせ集合が，推移律を満たす場合 6. へ，満たさない場合 4. へ
4. 推移律を満たすために必要な組み合わせを抽出
5. 4. で抽出した組み合わせの中で最も類似度が低い値を閾値に設定し，2. へ戻る
6. 抽出した組み合わせ集合に含まれる組を類似，それ以外を非類似と判定

推移律を満たせば，類似関係にある TF 候補を同じ群とするだけで分類できる．例として，図 7 のような 5 つの TF 候補が与えられた場合の分類手順を追う．A〜E が TF 候補，その間の数字が類似度である．初期閾値は 0.9 とする．

まず閾値 0.9 を超える類似度を持つ組み合わせを探す．この場合，{A，B}{A，C} が抽出される．この抽出結果は推移律を満たさないので満たすために必要な組み合わせを抽出する．この場合 {B，C} だけなので，{B，C} の類似度 0.8 に閾値を設定して，これを超える類似度を持つ組み合わせを探す．すると {A，B}{A，C}{B，C}{C，D} が抽出される．これらは推移律を満たさない．満たすために必要な組み合わせは {A，D}{B，D} なのでこれらの類似度のうち低い方の値 0.4 を閾値に設定する．この閾値を超えるものは {A，B}{A，C}{A，D}{B，C}{B，D}{C，D} となり，推移律を満たすので類似判定終了．あとは，A〜D が類似関係にあるのでこれらを同じ群として分類する．

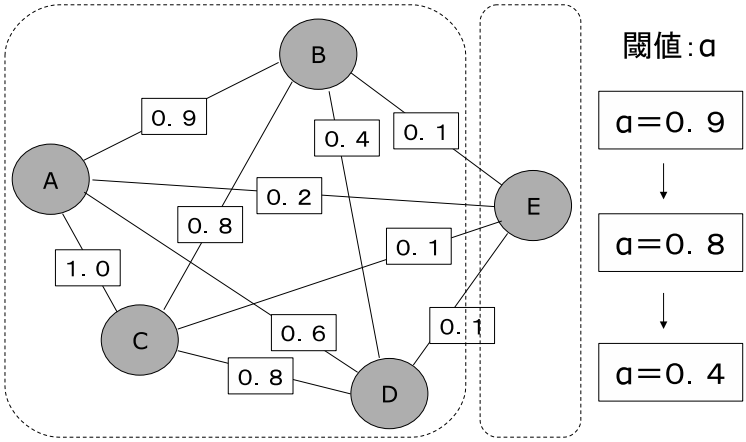

図 7: 個別分類の例

7.3 個別分類結果の統合

この手順で行うことは以下の二つである．

1. 「類似 TF 候補群として確定する群」を抽出する

2. 「DFG に判断させる組み合わせ」を抽出する

1. の「類似 TF 候補群として確定する群」とは，3 つの分類結果全てにおいて同じ群とされている群である．2. の「DFG に判断させる組み合わせ」とは，分類結果において同じ群かどうかの判断が分かれている組み合わせである．図 8 で説明する．なお，この図では「要素数 1 の群」は分類手順を通して必要がないため，表記していない．

まず {C，D，E} はどの分類結果においても同じ群とされているので「類似 TF 候補群として確定する群」である．次に {A，B} は，MG・DC では同じ群とされているが DI ではされていない．よってこれは「DFG に判断させる組み合わせ」である．同様に {A，F} {B，F} {C，F} {D，F} {E，F} も「DFG に判断させる組み合わせ」である．

7.4 DFG を基にした最終分類

この手順では，DFG を使って「判断の分かれた組み合わせ」を分類していく．ここでも，推移律に注意する必要がある．アルゴリズムを以下に示す．

1. 設定された閾値を超える類似度（DFG）を持つ組み合わせを「判断の分かれた組み合わせ」から抽出する．
2. 抽出した組み合わせを類似度（DFG）の高い順にソートして，順に以下の操作を行う．
    - その組み合わせを「類似 TF 候補群」に追加しても推移律を満たすなら，追加して次の要素へ．
    - 満たさないならば，満たすために必要な組み合わせを「1. で抽出した組み合わせ」から探して，あれば全て追加して次の要素へ．

図 8 を例にとって説明する．ここで閾値は 0.7 と設定されたとする．

「判断の分かれた組み合わせ」から閾値 0.7 を超えるものを類似度が高い順に抽出すると，{A，B} {D，F} {B，F} {E，F} となる．まず {A，B} を類似 TF 候補群に追加することを考えると，推移律が崩れないことがわかるので，追加する．次に {D，F} を追加しようとすると，すでに {C，D，E} があるため推移律を満たすには {C，F} {E，F} が必要となる．しかし，DFG¿0.7 で抽出した組み合わせに {C，F} がないため，{D，F} は追加できない．同様に {B，F} も {A，B} があるため推移律のために {A，F} が必要だが，ないので追加できない．{E，F} も同様に追加できない．よって，最終的な分類結果は {A，B} {C，D，E} {F} となる（図では要素数 1 の群は表記していない）．

類似度の高い順に処理している理由は，先に追加した組が，後で他の組を追加する際に推移律の面で邪魔になるケースがあるためである．図 8 の例では，先に A，B が追加されたため {B，F} が追加できていない．類似度が同値の場合は，番号が若い順にソートしているので，その場合公平性に欠ける分類をしているといえる．例えば図 8 では，{B，F} を {E，F} より優先している．この場合どちらも追加できていないが，{A，B} {D，F} のような {B，F} の追加の邪魔になる組が同一 TF 群に先に存在しなければ，{B，F} だけ追加されることになる．この点は改良の余地があるが，こういった状況の発生率が低いことと時間の関係上，本論文の段階では対処していない．

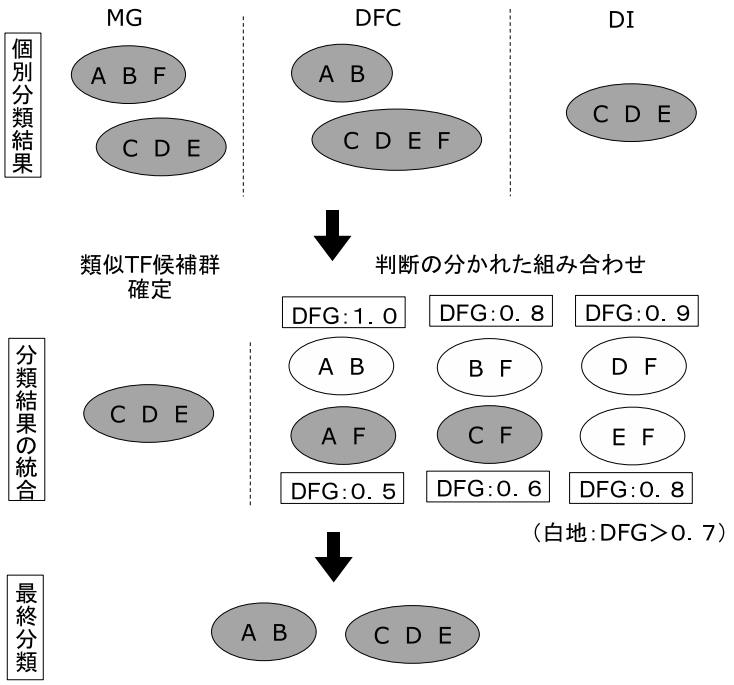
MG
DFC
DI
個別分類結果
A B F
C D E
A B
C D E F
C D E
類似TF候補群確定
判断の分かれた組み合わせ
DFG:1.0
DFG:0.8
DFG:0.9
分類結果の統合
A B
B F
D F
C D E
A F
C F
E F
DFG:0.5
DFG:0.6
DFG:0.8
（白地:DFG＞0.7）
最終分類
A B
C D E

図 8: 分類手順の例

8　分類手法の評価実験

8.1　実験対象ソフトウェア

分類手法の評価実験に使用したソフトウェアは，メトリクス評価実験でも用いた「ツール貸出システム」と，学生グループが授業で作成した「図書管理システム」の 2 種類である．どちらも Java で記述された Web アプリケーションで,「ツール貸出システム」のコード行数は約 37000 行,「図書管理システム」のコード行数は約 3600 行である.「ツール貸出システム」に含まれる TF は表 5 に既に示した.「図書貸出システム」に含まれる TF を表 8 に示す.

表 8: 図書貸出システムに含まれる TF 一覧

図書一覧表示
ウォッチリスト表示
図書追加
図書詳細表示
貸出中図書表示
画像表示

8.2　テストケースの説明

メトリクスの評価実験で使用したテストケースに加え，２つのテストケースを作成した．

8.2.1　TestCase2

「ツール貸出システム」から得られた 1 つの実行履歴を基にしたテストケース．31 個の TF 候補を持ち，それらは 12 種類の TF に分けられる．実行履歴に含まれる TF と，抽出された各 TF 候補がどの TF に対応するかを，表 9 に示す．抽出された TF 候補は 1 から 31 の数字で示す.

表 9: TestCase2 に含まれる TF と TF 候補の対応

抽出された TF	対応する TF 候補
利用中物品検索	1，4，10
利用中物品検索 (条件付)	11
物品検索	2
物品詳細＋物品貸出	3
物品登録未承認案件一覧	5，8，12，18，21，24，27，30
物品新規登録 (一般向け)	7
物品登録未承認案件詳細	20
物品管理品一覧	6，9，13，19，22，25，28，31
物品管理品詳細	23, 29
物品管理品詳細＋物品追加登録	26
備考表示	14，16，17
備考更新	15

8.2.2 TestCase3

「図書貸出システム」から得られた実行履歴を利用したテストケース．実行履歴からは 91 個の TF 候補が抽出され，これらの TF 候補は 6 種類の TF に対応する．TF 候補の数が多いため TF とそれに対応する TF 候補の数を表 10 に載せる．

表 10: TestCase3 に含まれる TF と対応する TF 候補の数

抽出された TF	対応する TF 候補
図書一覧表示	7 個
ウォッチリスト表示	1 個
図書追加	1 個
図書詳細表示	2 個
貸出中図書表示	1 個
画像表示	79 個

8.3 閾値の設定

今回の 3 つのテストケースに使用した閾値は，TestCase1 のメトリクス評価実験から設定した．

まず，MG，DC，DI メトリクス個別の類似判定での初期閾値は,「同一 TF の組み合わせ」での平均値｛MG：0.98，DC：1.00，DI：0.99｝とした．また，DFG で最終分類する際の

閾値は,「同一 TF の組み合わせ」での最小値と「非同一 TF の組み合わせ」での最大値の平均を取って 0.77 とした.

8.4 実験結果

ここでは，提案した分類手法に則って実際に TF 候補を分類し，正しい結果と比較する.

TestCase1

TestCase1 の TF 候補に対する分類過程と解答との比較を図 9 に示す．分類手法で例として示した図 8 と同じ形で表記したため，図の見方は当該部分を参照してほしい．なお「判断の分かれる組み合わせ」において，DFG の値を載せていない理由は，DFG $>$ 0.77 となった組み合わせが全て類似 TF 候補群に追加できたため，類似度の高い順に処理を行う意味が無かったからである.

TestCase2

TestCase2 の TF 候補に対する評価実験の過程を図 10 に示す．この分類においても TestCase1 と同様に，DFG $>$ 0.77 となった組み合わせの中に類似 TF 候補群に追加できなかったものがなかったため，図に DFG の値は載せていない.

TestCase3

TestCase3 の TF 候補に対する評価実験においては，TF 候補の数が多すぎるため図は省略して結果のみ述べる.

- 画像表示に対応する TF 候補 79 個の内，最も早く実行された一つの TF 候補が，他の 78 個の TF 候補と別 TF と判断された
- その他は全て正しい分類が出来た

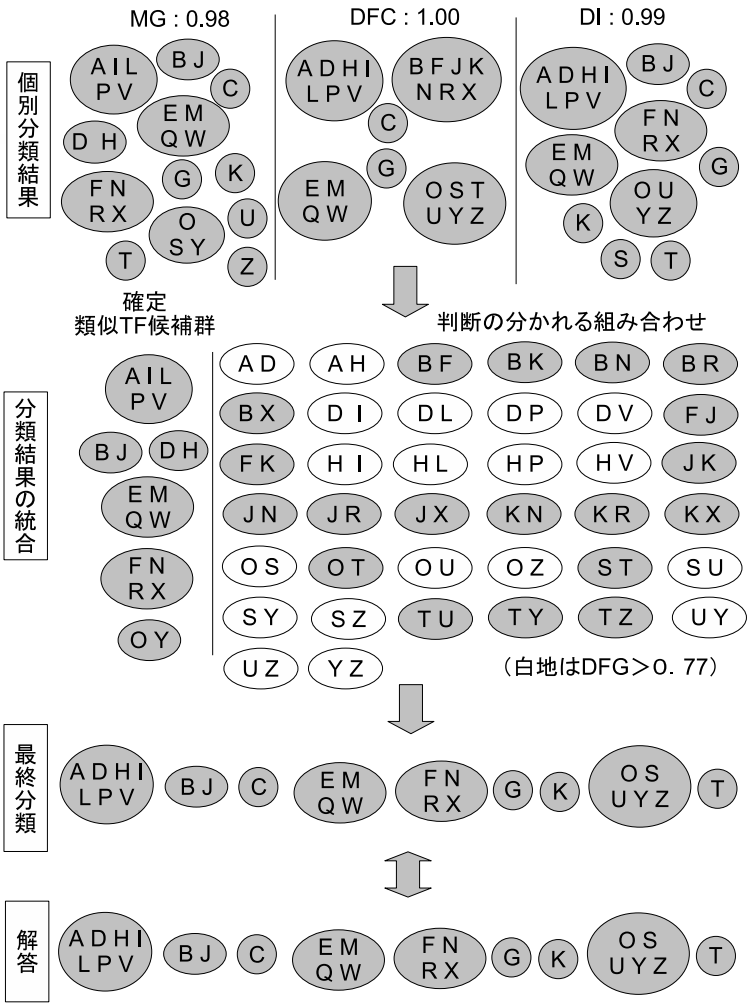

図 9: TestCase1 における分類過程と解答との比較

8.5 分類手法に対する考察

各テストケースに対する TF 候補分類手法評価の実験結果から考察を与える.

8.5.1 TestCase1

解答と完全に一致する分類が出来た.

分類方針や閾値の設定を TestCase1 のメトリクス解析を基に設定しているが，この結果はある程度手法の健全性を保障していると考えられる.

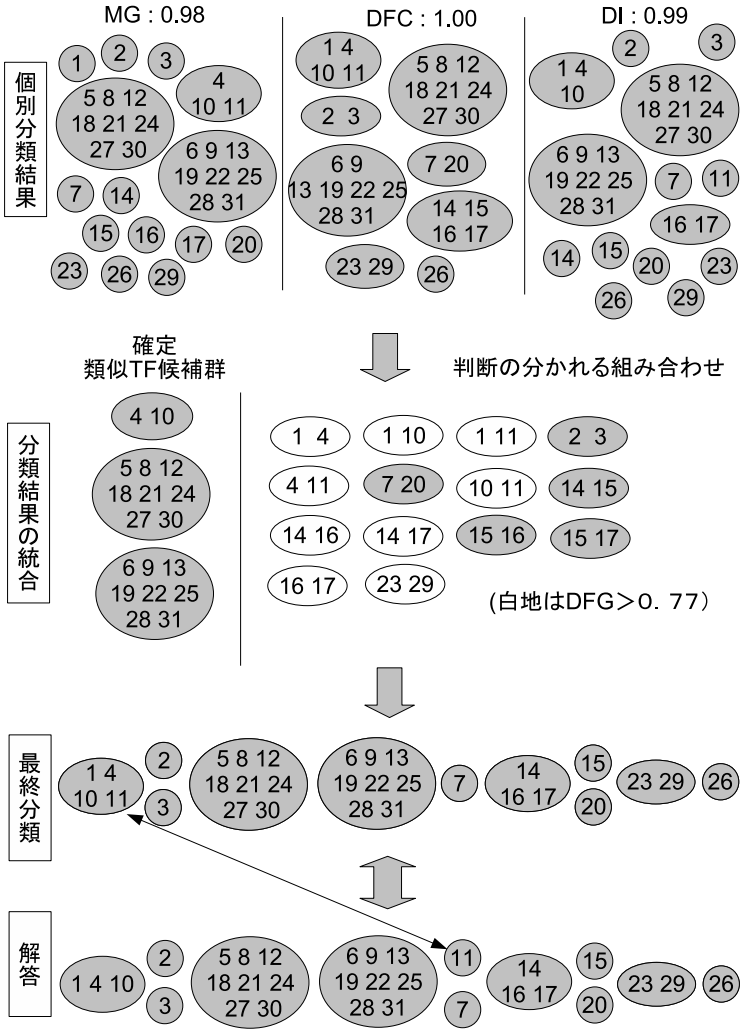

図 10: TestCase2 における分類過程と解答との比較

8.5.2 TestCase2

解答とほぼ一致する分類が出来た.

この分類結果により，分類手法に対する信頼性が向上したと言えるが，TestCase1 と基になるソフトウェアが同一であるため，このソフトウェアに依存した手法である可能性もある.

以下，分類が正しく行えなかった組み合わせに関する考察を述べる．正しい分類ができなかったのは「別の TF である {1，4，10} と {11} が同じ TF であると判断された」という点である.

{1，4，10} は「利用中物品検索」という TF に, {11} は「利用中物品検索（条件付）」という TF に相当する．これらの TF 候補間では全メトリクスで非常に高い類似度を示して

おり，提案しているメトリクスで {1，4，10} と {11} を分離するのは不可能である．

しかしソースコードを調べてみると,「利用中物品検索」と「利用中物品検索（条件付）」は，ソースコード上ではほぼ同一の機能として実装されていたことが確認された．

FP の値を基に開発工数が見積もられることを考えると，同一コードを利用するということは，開発者側にとっては工数の削減につながるため望ましいことであるが，ユーザ側にとっては余計な開発コストを要求されることになる場合もある．

逆に，同一の TF と考えていた機能が，別の機能として実装される場合も考えられる．この場合は，開発者側にとって不利なことになる．

設計時に見積もられた FP 値とソースコードの FP を比較することにより，上述のように初期段階におけるコスト見積の妥当性について評価をすることが可能であると考えている．

8.5.3 TestCase3

解答とほぼ一致する分類が出来た．

TestCase3 の基になったソフトウェアは，TestCase1,2 の対象ソフトウェアとは別のものであるため，提案手法が「ツール貸出システム」に対してのみ正しく動作するというわけではないことがわかった．しかし「ツール貸出システム」と「図書管理システム」には，Struts を用いた Web アプリケーションであるという共通点があるため，もっと様々なソフトウェアを対象に適用実験を行う必要がある．

以下，誤った分類に関する考察を述べる．

誤った分類とは,「画像表示」という TF に対応するべき 79 個の TF 候補の内，最も早い段階で実行された 1 つが，他の 78 個と別 TF であるとしてしまった点である．

この原因となったのは,「クラスを初期化する際に実行されるメソッド」である．このメソッドに関する処理が，最も早い段階で実行された部分から抽出した TF 候補には含まれているが，他の TF 候補には含まれていないため，これらが別 TF と判断されてしまったというわけである．

この初期化処理をどう除去するかは今後の課題である．

8.6 閾値に関する考察

今回３つのテストケースに使用した閾値は，全て TestCase1 のメトリクス評価実験から設定したものである．しかし，前述したように今回適用した２種のシステムは同一のフレームワークを用いた Web アプリケーションであるという共通点があるため，この閾値がどのようなシステムに対しても適用できるかはわからない．この点でも，様々なソフトウェアに対する適用実験が必要となる．

8.7 対象言語に関する考察

今回は Java を対象言語としたが,「TF 候補の分類」に関して言えば特に Java に特化した作業を行っていないため，入力フォーマットさえ変わらなければ，言語に対する制約はないといえる．とはいえ，クラスやフィールドといった概念が必要なため，オブジェクト指向ではない言語で入力フォーマットを用意するのは難しいと考えられるので，対象言語はオブジェクト指向である必要がある．

また，研究グループとしても Java プログラムからの FP 自動計測の目標を達成すれば，対象を他のオブジェクト指向言語にも拡張することを考えている．

9 あとがき

本研究では，ソースコードからのファンクションポイント自動計測ツールの実現への一環として，実行履歴上から抽出したトランザクションファンクション候補群に対する分類手法の提案と実装を行った．そして，提案手法を 2 種のシステムから抽出したトランザクション候補群に対して適用したところ，精度の高い分類結果が得られた．

今後の課題としては，以下のような点があげられる

- 複数の TF を持つ TF 候補の検出.
- クラスの初期化処理の除去.
- 様々なソフトウェアを基にしたテストケースに対する適用実験.
- 対象言語の拡張

複数の TF を持つ TF 候補の検出に関しては，その中の 1 つの TF に対応する TF 候補との類似度を計測し，何か特徴がないかを調査中である．

謝辞

本研究において，常に適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 楠本 真二 教授に深く感謝致します．

本研究において，常に適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 岡野 浩三 助教授に深く感謝致します．

本研究において，常に適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 山口 弘純 助手に深く感謝致します．

本研究において，評価実験に関してお手伝い頂いた株式会社 日立システムアンドサービス 芝本 俊久 氏に深く感謝致します．

本研究において，評価実験に関してお手伝い頂いた株式会社 日立システムアンドサービス 英 繁雄 氏に深く感謝致します．

本研究において，貴重なコメントを頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 谷口 考治 氏，森岡 佑 氏に深く感謝致します．

最後に，その他様々な御指導，御助言を頂いた大阪大学 大学院情報科学研究科コンピュータサイエンス専攻 楠本研究室の皆様に深く感謝致します．

参考文献

[1] Barry W. Boehm: ”Software Engineering Economics” Prentice Hall PTR, Upper Saddle River, NJ, 1981

[2] A.J. Albrecht: ”Function Point Analysis” Encyclopedia of Software Engineering, Vol.1, pp. 518-524, 1994

[3] IFPUG: ”Function Point Counting Practices Manual, Release 4.2” International Function Point Users Group, 2005.

[4] ソフトウェア・エンジニアリングセンター: ”IT ユーザとベンダのための定量的見積りの勧め～見積り精度を向上する重要ポイント～” オーム社, 2006

[5] ソフトウェア・エンジニアリングセンター: ”ソフトウェア開発見積りガイドブック～IT ユーザとベンダにおける定量的見積りの実現～” オーム社, 2006

[6] G.Antoniol, C.Lokan, G.Caldiera, R.Fiutem: “ A Function Point-Like Measure for Object-Oriented Software ”Empirical Software Engineering, vol.4, No.3, pp.263-287, September, 1999.

[7] クラスタリングとは (クラスター分析とは). http://www.kamishima.net/jp/clustering/

[8] 竹内光悦，酒折文武: ”多変量解析入門” ソフトバンク クリエイティブ株式会社，2006

[9] クラスター分析. http://aoki2.si.gunma-u.ac.jp/lecture/misc/clustan.html

[10] 階層的クラスタリングアルゴリズム.
http://www.neurosci.aist.go.jp/˜kurita/thesis/thesis/node36.html

[11] Collins, M. and Duffy: ”N. Convolution Kernels for Natural Language.” In Proceedings of NIPS 2001，2001

[12] 高橋哲郎, 乾健太郎, 松本裕治: ”テキストの構文的類似度の評価法について” 情報処理学会自然言語処理研究会，NL-150-7，2002

[13] 箱田 慶太，市川 宙，橋本 泰一，徳永 健伸: ”構文的類似度を用いた文の検索” 言語処理学会第 12 回年次大会. pp.1131，2006

[14] 森岡 祐: ”ファンクションポイント計測のための実行履歴からトランザクションファンクションを抽出する手法の提案”，2007

[15] 森岡 祐，谷口 孝治，楠本 真二，井上 克郎，英 繁雄，前田 憲一，津田 道夫: ”プログラム実行時情報を用いたトランザクションファンクション抽出手法” 電子情報通信学会技術研究報告, SS2006-79, Vol.106, No.523, pp.1-6，2006

[16] 谷口 考治, 石尾 隆, 神谷 年洋, 楠本 真二, 井上 克郎: ”Java プログラムの実行履歴に基づくシーケンス図の作成” ソフトウェア工学の基礎ワークショップ (FOSE2004), pp.5-15，2004

[17] 谷口 考治, 石尾 隆, 神谷 年洋, 楠本 真二, 井上 克郎: ”Java 実行履歴からのシーケンス図生成ツール” 組込みソフトウェアシンポジウム 2004 論文集,pp108-111, 2004