社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－SE－144　(13)
2004／3／18

ユースケースポイント計測支援ツールの実装とその適用

松川文一[†]　楠本真二[†]　井上克郎[†]
英　繁雄[††]　前川祐介[††]

ソフトウェア開発における開発規模および工数を, 開発プロセスの早期段階である要求分析段階で見積るための手法が求められてきている. そのための一手法として, 開発初期に作成されるユースケースモデルを用いて見積りを行うユースケースポイント法が提案, 研究されている. しかし, この手法を実際に適用して見積りを行うには, ある程度の熟練度が必要とされている. 本研究では, 経験の浅い者でも見積りが可能となることを目的として, ユースケースポイントの計測を行うツールを試作した. また試作したツールを, 実際に作成されたユースケースモデルへ適用し, ツールの評価を行った. 評価の結果, 本ツールの実用性を見出せる結果となった.

Implementation and Application of the Use Case Point Measurement Tool

Fumikazu Matsukawa[†], Shinji Kusumoto[†], Katsuro Inoue[†],
Shigeo Hanabusa[††] and Yusuke Maegawa[††]

Use case point (UCP) method had been proposed to estimate development effort based on use case models which are developed at early phase of software project. In order to effectively introduce UCP method to software project, it is important to develop a supporting tool to count UCP. This paper describes the idea to automatically calculate the UCP from use case model. We have also applied this tool to actual use case models and evaluated the applicability of it.

1. はじめに

ソフトウェア開発において通常, 開発規模や開発工数を予測するのに, まずソフトウェアの規模を見積り, 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. 近年では, ソフトウェアの機能要件だけを抽出し, 規模を定量的に計測するファンクションポイント法が世界的に普及しつつある. ファンクションポイント法は, 1979 年に A.J. Albrecht によって提案された[1].

しかしながら, ファンクションポイント法は, 計測を行う上で, データ項目やトランザクションが明確になっている必要があり, 効果的な結果を得るには, 開発プロセスを設計段階まで進めておく必要がある. 近年の受注競争の激化や, 設計に多くの時間を必要とするような複雑なソフトウェアの増加により, 開発プロセスのさらに早期の段階 (上流工程) で見積りを行なうための手法が求められてきている.

この流れを受け, 1993 年, G. Karner により, ユースケースポイント法が提唱されている[5]. ユースケースポイント法は, ファンクションポイント法をベースとし, 近年増えつつあるオブジェクト指向開発において, ライフサイクルの早期である要求分析の段階で作成されるユースケースモデルをもとに, 開発規模および開発工数を見積るための手法である. この手法の最大のメリットは, 開発プロセスの設計段階よりもさらに上流である要求分析段階での工数見積が可能となることにある.

ユースケースポイント法は, 具体的にはユースケースモデルに記述されるアクタおよびユースケースに対して重み付けを行なうが, この作業にはある程度の熟練度を必要とし, それゆえ同一プロダクトに対する計測でも, 計測者の経験度によって結果に誤差が生じる可能性がある.

本研究では, 見積り経験の浅い者でも見積りが可能となることを目的として, ユースケースモデルを入力とし, ユースケースポイント法に基づいて工数の見積りを行うためのツールを試作し, 実際のプロジェクトに適用することで, ツールの評価を行った. ユースケースポイントの自動計測にあたり, アクタとユースケースに対する重み付けの部分にいくつかのルールを設定し, モデル内の情報から計測を行っている. また, ユースケースポイント法における重み付けに必要な情報が欠けているユースケースモデルに対しても, 過去の情報を利用することで計測を行うことを可能とした. そして実際にユースケースモデルが作成されたいくつかのプロジェクトを用いて,

† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†† 株式会社 日立システムアンドサービス
Hitachi Systems & Services, Ltd.

ツールによる重み付けと, 経験者による手動での重み付けの結果との比較を行うことでツールの精度を評価した.

以降, 2 で本研究が対象とするユースケースモデルについて説明を行い, 3 節では, ユースケースポイント法について説明する. 次に 4 節でツール化へ向けて設定したルールについて述べ, 5 節で, これらのルールを実装した見積支援ツール U-EST に関して説明を行う. 次に 6 節で, いくつかのプロジェクトのユースケースモデルを対象にして適用実験を行ったその結果についての分析と考察を行う. 最後に 7 節で, 本研究のまとめと今後の課題について述べる.

2. ユースケースモデル

2.1 ユースケースモデル

本研究において対象とするユースケースモデルは, ユースケース図とユースケース記述の 2 つの要素で構成されている. 以下, それぞれについて説明を行う.

2.1.1 ユースケース図

ユースケース図は, ユースケースの概念を UML で視覚化して表現したものである. アクタを人間の形のアイコンで示し, ユースケースを楕円形のアイコンで表現する. アクタとユースケースとの関連は実線で表す.

2.1.2 ユースケース記述

ユースケース記述は, ユースケース図に記述されたユースケースそれぞれについて, 主体となるアクタの情報や, アクタとシステムとのやり取りなどといった, そのユースケースの機能を実現するために必要な詳細情報を記述したものである. ユースケース記述は確立したフォーマットはなく, 自由に記述が可能となっている.

ひとつの例として, 「注文を出す」というユースケースに関し, ユースケース図とユースケース記述をまとめたものを図 1 に示す.

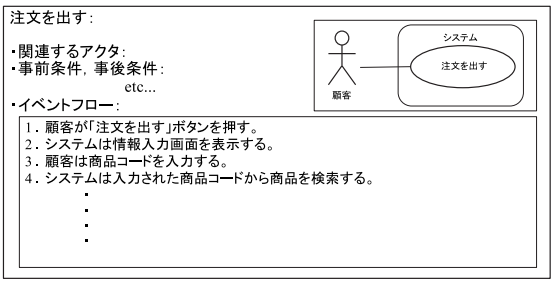

図 1 「注文を出す」ユースケース

このユースケース記述における, 「事前条件」「事後条件」は, ユースケースの開始時, 及び終了時にシステムがどのような状態でなければならないかを表すものである. この条件は必ず成り立っていなければならない. また, 「イベントフロー」は, ユースケースの手順を並べた一連の宣言文を指す.

本研究では, 区別のため, イベントフロー内の 1 つのパスを「イベント」と呼ぶことにする.

3. ユースケースポイント法

3.1 概　　要

ユースケースポイント法は, オブジェクト指向開発の早期段階における工数及びコスト見積手法が要求される中で, 1993 年に G. Karner によって提案された工数見積手法である[5]. この手法は, A.J. Albrecht によるファンクションポイント法をベースとし, 開発の要求分析段階で作成されるユースケースモデルに基づいて見積りを行う. 具体的には, ユースケースモデルに記述されるアクタ, ユースケースを重み付けしてその数をカウントし, 計測対象プロジェクトの技術的な要因や, 開発環境の要因をもとに調整を行い, 規模, そして工数を見積る.

3.2 計 測 手 順

3.2.1 アクタの重み付け

まず, ユースケースモデルに記述されているアクタについて, それがどのようなアクタであるかで分類を行う. 表 1 に詳細を示す. それぞれのタイプには重みが設定されており, 各タイプのアクタの個数に重み係数を掛け, 合計したものを, **アクタの重み** (AW : $Actor\ Weight$) とする.

表 1 アクタのタイプと重み係数

タイプ	説明	重み係数
単純	定義済み API を備えた別システム	1
平均的	プロトコル駆動のインタフェース (別システム), テキストベースのインタフェース (人間)	2
複雑	GUI を介する人間	3

単純なアクタというのは, 定義済みの API(Application Programing Interface) を備えた別システムを指す. 平均的なアクタは, TCP/IP などのプロトコルを介して計測対象システムと相互作用する別システム, もしくはテキストベースのインタフェース (ASCII 端末といったもの) を介して相互作用する人間を指す. 複雑なアクタは, GUI を介して相互作用する人間がこれに分類される.

3.2.2 ユースケースの重み付け

次に, ユースケースについても同様に複雑さの分類を行う. ユースケースの場合は, 副シナリオを含めたユースケースシナリオ内のトランザクションの数に基づいてその複雑さを決定する. この場合のトランザクションは, 原始的な一群のアクティビティと定義され, これはすべて完全に実行されるかあるいはまったく実行されないかのどちらか一方となる. また, include するユースケースや拡張ユースケースについては, 計測の対象としない. 分

類について表 2 に示す.

表 2 ユースケースのタイプと重み係数

タイプ	説明	重み係数
単純	トランザクション数が 3 個以下	5
平均的	トランザクション数が 4 個から 7 個	10
複雑	トランザクション数が 8 個以上	15

ユースケースについても, 各タイプの個数に重み係数を掛け, すべて合計する. これを**ユースケースの重み** ($UW : Use\ Case\ Weight$) とする.

3.2.3 未調整ユースケースポイントの算出

算出されたアクタの重みと, ユースケースの重みを合計すれば, **未調整ユースケースポイント** ($UUCP$: $Unadjusted\ Use\ Case\ Point$) が得られる.

$$UUCP = AW + UW \tag{1}$$

このようにして算出された未調整ユースケースポイントに対し, 計測対象のシステムの複雑さや, 開発に関わる人々の経験レベルといった要因を考慮した調整を行う.

3.2.4 技術要因の評価

まず, 対象のシステムの技術的な要因を考慮した調整係数を算出する. これは**技術要因** (TCF : $Technical\ Complexity\ Factor$) と呼ばれる. システムの技術要因については, 表 3 に示す 13 の項目があり, 各要因には重み係数が設定されている. 計測者は, これら 13 の項目それぞれについて, 0 から 5 までの 6 段階で評価を行う.

表 3 システムの技術要因とその重み係数

要因番号	要因の説明	重み係数
T1	分散システムである	2
T2	レスポンスまたはスループットのパフォーマンス目標が設定されている	1
T3	エンドユーザの効率 (オンライン時) を重視	1
T4	内部処理が複雑	1
T5	コードが再利用可能でなければならない	1
T6	インストールしやすい	0.5
T7	使いやすい	0.5
T8	移植可能である	2
T9	変更しやすい	1
T10	並行性が必要	1
T11	特別なセキュリティ機能が必要	1
T12	第三者に直接アクセスを提供している	1
T13	特別なユーザトレーニングが必要	1

要因ごとの評価点と, 各要因に設定された重み係数を掛け, すべての数値を合計したものを $TFactor$ とする.

$$TFactor = \sum \{(\text{各要因の評価点}) \times (\text{重み係数})\}$$

最後に, (2) 式を用いて, 技術要因の調整係数を算出する.

$$TCF = 0.6 + (0.01 \times TFactor) \tag{2}$$

3.2.5 環境要因の評価

次に, 開発に関わる人々の経験や, 開発環境を考慮した調整係数を算出する. これは**環境要因** (EF : $Environmental\ Factor$) と呼ばれる. 環境要因については, 表 4 に示す 8 つの要因があり, 各要因に重み係数が設定されている. 計測者は, 各要因について 0 から 5 までの 6 段階で評価を行う.

表 4 環境要因とその重み係数

要因番号	要因の説明	重み係数
F1	採用する開発プロセスに精通している	1.5
F2	その分野での開発経験	0.5
F3	採用する開発方法論についての経験	1
F4	主任アナリスト (リーダー) の能力	0.5
F5	プロジェクトに対するモチベーション	1
F6	要求仕様の安定性	2
F7	兼任スタッフの存在	-1
F8	プログラミング言語の難しさ	-1

要因ごとの評価点と, 各要因に設定された重み係数を掛け, すべての数値を合計したものを $EFactor$ とする.

$$EFactor = \sum \{(\text{各要因の評価点}) \times (\text{重み係数})\}$$

最後に, (3) 式を用いて, 環境要因の調整係数を算出する.

$$EF = 1.4 + (-0.03 \times EFactor) \tag{3}$$

3.2.6 ユースケースポイントの算出

(1) 式で算出した未調整ユースケースポイントに, 各調整係数を掛け合わせることで, **ユースケースポイント** ($UCP : Use\ Case\ Point$) が得られる.

$$UCP = UUCP \times TCF \times EF \tag{4}$$

3.2.7 工数の見積り

(4) 式で算出されたユースケースポイントに対して, 工数への変換を行う. 変換にあたり, 手法の提案者である Karner は, 1UCP あたり 20 人時という係数を用いることを提案している[5]. しかしながら, 文献[9] では, 以下に示すような, 環境要因の評価について考慮した新しい係数が提案されている.

(1) 環境要因 F1〜F6 について評価が 2 以下である要因の個数をカウントする.

(2) F7 と F8 について評価が 4 以上である要因の個数をカウントする.

(3) (1) と (2) の個数の合計について

- (a) 合計が 2 以下：20 人時/1UCP
- (b) 合計が 3 または 4：28 人時/1UCP
- (c) 合計が 5 以上：プロジェクトの変更が必要 (失敗の危険性)

4. 提 案 手 法

4.1 計測方針概要

3 節で述べたユースケースポイント法に基づいて計測を行うツールを作成するにあたって,

- アクタの分類に必要な情報をどのようにして得, そして判断するか.
- ユースケースのシナリオからどのようにトランザクションを識別, カウントするか.

が考慮すべきポイントとなった. 次項以降, それぞれの計測方針について詳細を記す.

4.2 アクタの計測方針

Karner の定義によれば, アクタのタイプ分類の基準は, そのアクタが, システムに対してどのようなインタフェースで相互作用するかということである. しかし, モデルに記述されるアクタは, インタフェースに関する情報をそれ自身が明示的に持っているわけではない. そこで我々は, 以下に示す 2 つのステップにより, アクタの分類を行うことにした.

4.2.1 Step1: アクタ名による分類

表 1 を見ると, まずアクタが人間であるか, もしくは外部のシステムであるかで, タイプの候補を 2 つに絞ることができる. 外部システムであれば単純, もしくは平均的であり, 人間であれば平均的, もしくは複雑となる. そこで, 第 1 段階として, まず人間か外部システムであるかを判断する.

具体的には, アクタにつけられた名前に注目し, 外部システムであると判断できるキーワードを設定し, それを名前の語尾に持つアクタを外部のシステム, そうでなければ人間とする. キーワードには例として, 「システム」「サーバ」などが挙げられる. また, 見積を行う組織, 団体等で, ある特定の命名規則を設定する可能性も考慮し, このキーワードはユーザによる変更・追加も可能とする. 外部システムに関するキーワードとしたのは, 人間である場合の名前には無数のパターンが考えられ, 外部システムに関するそれよりも量が膨大であると判断したためである.

- **ルール A-1:アクタ名の語尾にキーワード (ユーザ変更可) を持つアクタは外部システム，それ以外を人間とする**

4.2.2 Step2: キーワードによる分類

アクタ名による第 1 段階の分類後, 絞られた 2 つの候補から 1 つに決定する第 2 段階の分類を行う. ここからは, どのようなインタフェースを持つかに依存するが, 先に述べた様にアクタ自身はその情報を陽には持たない. そこで我々は, 対象アクタが相互作用するユースケースのシナリオが持つイベントに着目し, インタフェースに関する情報がそこから得られないかと考えた.

具体的には, インタフェース情報に関するようなキーワード群を, それぞれのタイプで設定し, アクタが関連するユースケースのイベントから, 対象アクタが主体となるイベントを取り出し, そのイベント本体とのマッチングをとり, 複雑さを決定する. しかし, 例えばあるイベントに, 異なるタイプに設定したキーワードが重複して含まれることも考えられる. そこで, 各タイプのキーワード群において, キーワード全体数に対するマッチング数の割合の高いものを採用することにする (図 2 参照). また, このキーワードについてもユーザが変更・追加可能とする.

- **ルール A-2:対象となるアクタが関連しているユースケースのイベントを抽出し，タイプごとのキーワード群 (ユーザ変更可) に対するマッチングの割合によりタイプを決定する**

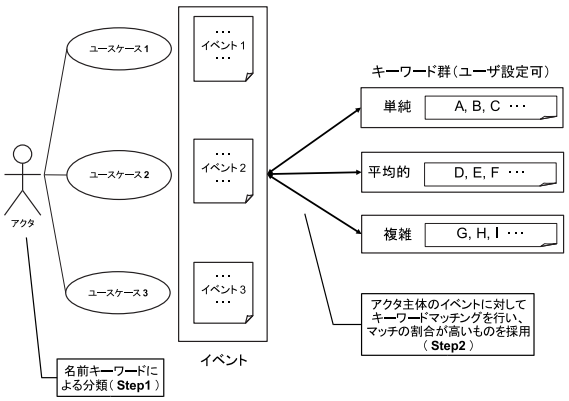

図 2 アクタの複雑さ決定イメージ

4.2.3 分類不可能時の処理

上記の方針によりアクタのタイプを決定するが, 関連するユースケースのシナリオに, 用意したキーワードに全くマッチしない場合も考えられる. このような場合, 支援手法として, 過去の情報を利用することを考える.

具体的には, まずそれまでに計測がなされたプロジェクトの様々な情報をデータベースとして蓄積する. その中のアクタの情報から, 分類が不可能なアクタに類似したアクタを検索し, 見つかった情報をユーザに提示する. 検索はアクタ名の部分一致による検索を行い, アクタ名, そしてそのアクタに対して過去に決定された複雑さ等の情報を表示する. ユーザはそれを受けて, 複雑さを手動で設定, 修正を行うことができる. 複数の類似アクタが見つかった場合は, その複雑さについて多数決で決定する.

もし, 過去の情報からも類似したアクタが見つからなかった場合は, デフォルトの複雑さとして, 外部システムであれば「平均的」, 人間であれば「複雑」とする.

- **ルール A-3:キーワードにマッチしないアクタについては過去の情報を利用する**
- **ルール A-4:ルール A-2 及び A-3 が適用できない**

場合は, デフォルトの複雑さ (外部システム:「平均的」, 人間:「複雑」) とする

4.2.4 汎化に対する処理

アクタ間には汎化関係が存在するが, Karner はこの汎化関係については特に言及してはいない. 我々はこの関係が存在する場合, 親アクタの機能を継承した子アクタについては, 自身が関連するユースケースを調べるだけではなく, 親アクタの関連するユースケースをも調べてマッチングをとるべきであると考える. また, もし親アクタに関連が全く存在していない場合は, 機能を継承した子アクタが既に存在するため, カウント対象に入れるべきではないと考える. このことから, 以下のルールを追加する.

- **ルール A-4:汎化関係のあるアクタについては, 子アクタは親アクタの関連するユースケースシナリオのイベントともマッチングをとる**
- **ルール A-4:親アクタに関連するユースケースが存在しない場合, その親アクタはカウントの対象としない**

4.3 ユースケースの計測方針

Karner の定義から, ユースケースの複雑さは, その内部の処理 (トランザクション) の数によって決まる. 故に計測に当たっては, ユースケースモデルにおけるユースケース記述内の, イベントフローに着目する. トランザクションの数をカウントする最も単純な方法としては, イベントフロー内のイベントの数をカウントすることが考えられる. しかしながら, ユースケース記述は定まったフォーマットが存在せず, 作成者が自由に記述することができる. それ故, 例えばトランザクションが 2 つ認識できる様な処理が, 1 つのイベントとして記述される可能性もある. 自由に記述可能なイベントフローから, いかにしてトランザクションを認識・カウントを行うかが課題であった.

4.3.1 形態素解析と統語解析

先に述べたように, 単純にイベントの数をトランザクションの数として複雑さを判断する方法では, 作成者が 1 トランザクションと認識すべき処理を 1 イベントとして意識して記述しない限り, 正しく計測を行うことができない. そこで我々は, 全てのイベント記述に対し, 形態素解析および統語解析を行うことで細かい分析を行い, その結果からトランザクションを認識し, カウントする方法を考えた.

両解析を行うにあたり, 日本語係り受け解析器「南瓜」[11] を採用することにした. 「南瓜」は, SVM(Support Vector Machines) に基づく日本語係り受け解析[6] 器であり, 統計的な日本語係り受け解析器として最も精度が高いとされている (89.29%) システムである. また, バックトラックを行なわない決定的な解析アルゴリズム (Cascaded Chunking Model)[7] を採用しており, 効率の良い解析が可能となっている.

4.3.2 トランザクションの識別

「南瓜」を用いて, 各イベントに対して解析を行うが, その結果を用いてどのように 1 つのトランザクションを認識するかが問題となる. 文献[4] では, ユースケースシナリオの効果的な記述方法としていくつかの指針を挙げている. その中の以下に挙げる 2 つに着目した.

- 単純な文法で記述する

  これは, 記述する文章の構造を単純にするということである. つまり, 主語, 修飾語, 目的語, 述語のセット, 少なくとも主語と述語だけでもきちんと記述するべきであるということ.
- 意味のあるアクションの集合を含める

  ユースケースの複雑さを決定するためには, 認識されるべきトランザクションは, ユーザあるいはシステムにとって意味のある内容である必要がある. Jacobson は, 相互作用について代表的な 4 つの部品として以下を挙げている[4].
  - 主アクタがシステムに要求とデータを送る.
  - システムが要求とデータを確認する.
  - システムがその内部状態を変更する.
  - システムがアクタに結果を返す.

  よって, このような処理内容を 1 つのトランザクションとして認識すべきであると考えた.

以上の事項から, ユースケースのトランザクションについて, 次のルールで計測を行う.

- **ルール U-1:ユースケース記述内のイベントフローに対して形態素解析を行い, 主語と述語のセットを 1 つのトランザクション候補とする**
- **ルール U-2:抽出された候補のうち, アクタの動作およびシステムのレスポンスであるものを 1 つのトランザクションとする**

4.3.3 分類不可能時の処理

Karner の定義に基づいてユースケースの複雑さを決定するには, イベントフローが記述されていることが前提となる. しかしながら, 実際の開発においては, ユースケース図が記述されていてもそのイベントフローまでは記述されないこともある. このような場合, アクタの場合と同様に, 過去の蓄積された情報を利用することを考える. 蓄積されたユースケースの情報から, 対象となるユースケースに類似したユースケースを検索し, 見つかった情報をユーザに提示する. ユーザはそれを受けて, 複雑さの設定, 修正を行うことができる. 検索はユースケース名の部分一致による検索を行い, ユースケース名, そして過去に決定された複雑さ等の情報を表示する. 複数見つかった場合は, その複雑さについて多数決で決定する.

もし, 過去の情報からも類似したユースケースが見つからなかった場合は, デフォルトの複雑さとして, 「平均的」とする.

- **ルール U-3:シナリオを持たないユースケースについては過去の情報を利用する**
- **ルール U-4:ルール U-1 及び U-2, U-3 が適用できない場合は, デフォルトの複雑さ「平均的」とする**

4.3.4 包含や拡張に対する処理

Karner の提案した手法では, 包含や拡張ユースケースについては考慮しないとされている[5] が, このようなユースケースに, システムの本質的な処理が含まれる場合もあることが指摘されており[2),8)], 機能規模計測において無視できるものではないと考える.

包含や拡張の関係が存在するとき, それをひとつのユースケースとして処理を記述し, その処理を含むユースケースは, 自身のイベントフロー内で包含 (拡張) ユースケースを参照するイベントとして記述を行う. このことから, 我々は以下に示すルールを設定する.

- **ルール U-5:包含または拡張するユースケースは, 1 つのユースケースとして複雑さを決定し, 計測対象とする**
- **ルール U-6:それらを参照するユースケースは, イベントにおけるその参照処理をトランザクションに含めない**

5. 工数見積り支援ツール U-EST

5.1 ツール概略

本ツール U-EST(Usecase-based Estimation Supporting Tool) は, XMI(XML Metadata Interchange) 形式で記述されたユースケースモデルを入力とし, 4 節で述べた方針に基づいて, ユースケースポイントの計測, 及び工数の見積りを行う. 開発言語は Java であり, 規模は 42 クラス, 約 4.2KLOC である.

5.2 システム構成

本ツールのシステム構成図を図 3 に示す. システムは 4 つのサブシステムから構成されており, 以下に説明を記す. すべての処理はユーザが GUI を通して操作する.

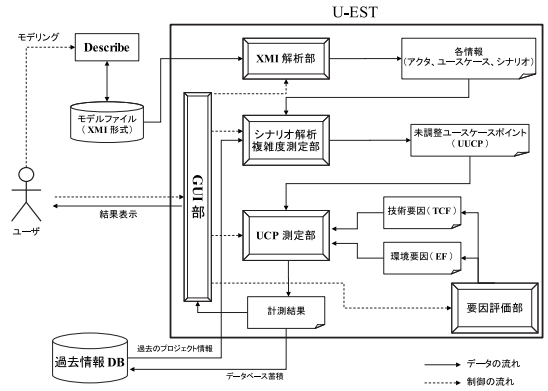

図 3 システム構成

5.2.1 XMI 解析部

XMI 形式で記述されたモデルファイルを解析し, 記述されたアクタ, ユースケース, イベント等, 計測に必要な情報を抽出する.

5.2.2 複雑度測定, UUCP 計測部

XMI 解析部において抽出した情報を元に, 4 節で説明した方針に従い, 各アクタ, ユースケースのタイプ分類を行う. またその結果から, 未調整ユースケースポイントを算出する. 各アクタ, ユースケースに対する分類結果, 及び未調整ユースケースポイントは, GUI を通してユーザに提示する.

また, 表示される結果に対し, ユーザは手動にて分類を再設定することも可能であり, 変更による各計測値への更新は随時自動で行われる.

5.2.3 調整要因評価部

技術要因, 環境要因それぞれに対して, ユーザがダイアログ上で評価を行い, 評価値を入力する. 入力された結果をもとに, 各調整係数を算出し, 画面に表示する.

5.2.4 UCP 測定部

複雑度測定部, および調整要因評価部の結果を受けて, ユースケースポイント値を算出し, 画面に表示する. また, その結果見積られる工数 (人時で算出) も同時に画面に表示する. ユースケースポイント値から工数値への変換係数は, ユーザによる変更が可能である.

6. 評　　価

6.1 評 価 概 要

試作したツールが行う分類が妥当なものであるかどうかを評価するため, 実際のプロジェクトにおいて作成されたユースケースモデルに対して, ツールを適用した. 今回用いたプロジェクトは, 小中規模の Web システム開発プロジェクトであり, 全部で 5 つある. 表 5 に, その規模等について記す.

対象となるプロジェクトのユースケースモデルにおけるアクタ, ユースケースの複雑さに関して, ツールによって決定された複雑さと, 経験者の手動によって決定された複雑さとの比較を行い, 4 節で述べた手法の正確性を評価する.

表 5 プロジェクトデータ

プロジェクト	開発言語	アクタ数	ユースケース数
A	Java	5	15
B	Java	5	14
C	Java, VB.NET	2	20
D	Java	5	28
E	Java	8	13

6.1.1 複雑さ分類の結果

アクタ, ユースケースについて, ツールと手動での分類結果について表 6, 表 7 に示す.

各表は, ツールによる分類と, 手動での分類での, 各複雑さに分類された要素の数を記している. タイプの一致割合という項目は, ツールで決定された複雑さと手動で決定された複雑さが一致した要素の数の, 全体数に対する割合を示したものである.

ここで, 今回の評価においては, アクタの分類を行うためのキーワード群として, 文献[9] をもとに, 以下のキーワードを用いた.

- 命名規則キー :「システム」「サーバ」
- 単純 :「リクエスト」「送信」「要求」「通知」
- 平均的 (システム) :「メッセージ」「メール」「送信」
- 平均的 (人間) :「コマンド」「テキスト」「入力」「画面」「表示」
- 複雑 :「入力」「画面」「ボタン」「押す」「表示」

表 6 複雑さの分類結果 (アクタ)

	ツールによる分類			手動による分類			タイプの
	単純	平均的	複雑	単純	平均的	複雑	一致割合
A	0	1	4	1	0	4	0.80
B	0	3	2	3	0	2	0.40
C	0	0	2	0	0	2	1.0
D	0	1	4	1	0	4	0.80
E	0	0	8	0	0	8	1.0

アクタの分類については, プロジェクト B においてその一致割合が低いものとなったが, 残りのプロジェクトについては高い一致割合を示した.

表 7 複雑さの分類結果 (ユースケース)

	ツールによる分類			手動による分類			タイプの
	単純	平均的	複雑	単純	平均的	複雑	一致割合
A	13	2	0	13	2	0	1.0
B	6	7	1	10	4	0	0.64
C	17	3	0	14	6	0	0.55
D	23	4	1	27	1	0	0.82
E	10	2	1	2	8	3	0.38

ユースケースについては, 一致割合が 0.38〜1.0 と, 広範囲に渡る結果となった. ユースケースの重みでみると, プロジェクト B, C, D で手動よりも高く, プロジェクト E においては手動よりも低いという結果になった.

6.1.2 アクタの分類に対する考察

手動による分類との違いが生じたアクタは, すべて人間ではなく, 外部システムであった. そこでユースケースシナリオについて詳しく分析すると, アクタが人間である場合は, 動作の方向がアクタからシステムに対するものであるが, アクタが外部システムである場合は, その動作の方向が主に計測対象のシステムからその外部システムに対するものとなることがわかった. つまりアクタが人間である場合は, そのアクタが動作の主体となるイベントが記述されているものの, アクタが外部システムの場合は動作の主体ではなく対象となり, それが主体となるイベントとして陽に記述されず, むしろ計測対象システムが主体となる動作における対象として記述されることが多いことがわかった.

実装の上では, 関連するユースケースのイベントから, 主にアクタが主体となる動作を抽出してキーワードのマッチングを行っていたため, 今回のケースでは, アクタが外部システムの場合, キーワードのマッチングを行うことができず, 過去の情報の利用による値, もしくはデフォルト値となってしまったことが原因であった.

6.1.3 アクタの分類に対する対策と再分類

前述の考察を受けて, 複雑さの違いが生じたアクタに対し, 関連するユースケースのイベントについて, 計測対象システムが主体となる動作に注目してキーワードマッチングを行い, 再分類を試みた.

その結果, キーワードマッチングを行うことはできたものの, 決定した複雑さとしては前回と変わらなかった. さらに分析を進める中で, アクタが外部システムである場合, そのインタフェースに関する情報をイベントから得るのは困難であることが判った. それゆえ, アクタが外部システムである場合, そのインタフェースに関する情報を, 別の要素から得る必要がある. 今後, インタフェースに関する情報の探索, そして多くのケーススタディを通して, キーワード設定の有効性を評価する必要がある.

6.1.4 ユースケースの分類に対する考察

まず, 手動計測よりも重みが高い結果となったプロジェクト B, C, D について, 各ユースケースの認識トランザクション数を分析してみると, その数が同じか, 手動よりも多く認識していた. その差は 1〜2 程度のものであったのだが, 複雑さの判断が手動と異なっていたユースケースは, そのトランザクション数の差が, 複雑さの分類の境界で起こっていたために, 手動よりも複雑さが高くなっていたのである.

この原因について更に分析を行うと, 手動計測者の計測方針として, システムが単に情報を表示するだけといった動作を, トランザクションとしていないことがわかった. シナリオではこの, システムの単なる情報表示も 1 つのイベントとして記述されており, 形態素解析, 統語解析を行った結果, トランザクション認識がなされていたため, 手動計測よりもトランザクション数が多くなったと考えられる.

次に, 手動計測よりも重みが低い結果となったプロジェクト E について同様に分析を行うと, 今度はトランザクション数を手動計測よりも少なく認識していた. そこでイベント 1 つ 1 つに注目してみると, このプロジェクトに記述されたユースケースシナリオは, 他のプロジェクトとは異なり, アクタのシステムに対する処理と, それに対するシステムのレスポンスが 1 つのイベントとして記述されており, しかもシステムの動作について, その主体

がシステムであることが陽に記述されていなかった. 前述の形態素解析による計測方針では, 主語と述語のセットを 1 つのトランザクションとするため, 主語が省略された記述であると, トランザクションとして認識されない. このことが, 手動計測よりもトランザクション数を少なく認識した原因であると考えられる.

6.1.5 ユースケースの分類に対する対策と再分類

前述の考察をもとに, プロジェクト B, C, D, E について, 単なる情報表示をトランザクションとして除外し, さらにプロジェクト E については, 主語の省略された動作についてトランザクションとして含め, 再分類を行った.

その結果, 手動による分類と完全に一致する結果となった. このことから, 今後のツール改良点として,

- トランザクションとして識別すべき動作, もしくは識別しない動作をキーワードなどの情報として予め用意する
- 主語との対応が取れなかった等, 識別できなかったイベント情報をユーザに知らせる

などが考えられる. しかしながら, 先に述べた Jacobson による 4 つの意味のあるアクションの集合の部品を考慮すると, システムによる情報表示は, アクタへ結果を返す動作であるとみなすこともでき, トランザクションとして含めるべきであるとも考えられる. ゆえに, このことについては今後より議論する必要がある.

7. おわりに

本研究では, 近年注目されている早期工数見積手法であるユースケースポイント法に基づいて, ユースケースモデルから機能規模を計測し, 工数見積りを支援するツールの試作を行った. また, 試作したツールを, 実際に開発された 5 つのプロジェクトにおいて, 実際に作成されたユースケースモデルへ適用し, 経験者による手動での計測結果と比較することで, ツールの精度の評価を行った.

比較の結果, ツールの計測結果は, 手動での計測結果に近いものとなったことを確認し, 実用化の可能性を見出せる結果となった.

今後の課題を以下に挙げる.

(1) 今回評価に用いたユースケースモデル中のユースケース図は, トップレベルのものを対象としたが, ユースケースを階層化し, 更に詳細なユースケース図が記述される場合もある. この場合の分類, 及び計測方法について考慮する必要がある.

(2) より多くのプロジェクトに対するケーススタディを行うことで, アクタ分類のキーワードの精度向上, 及びツール, 手法の実用性の向上を目指す必要がある. その上で, 定まった記述形式の存在しないユースケースモデルについて, 最適なテンプレートの提供も課題となる.

(3) 既存の多くの見積手法 (ファンクションポイント法, COCOMOII など) との比較を行うことで, 手法の精度向上を測る. また, 他手法との併用による効果も調査の価値がある.

(4) 更に, 今回の評価に用いたプロジェクトは全て Web システム開発プロジェクトであったが, 他の様々なタイプ, 規模のシステムにも適用実験を行い, ユースケースポイント法の正確性, 及び可用性の評価を行う必要がある.

8. 謝　辞

本研究において, 御協力を頂きました 株式会社日立システムアンドサービス 産業・流通システムサービス事業部 津田 道夫 氏, 高橋 まゆみ 氏に深く感謝致します.

参 考 文 献

1) A.J. Albrecht, “Function Point Analysis”, *Encyclopedia of Software Engineering*, 1994, 1:518-524.
2) B. Anda, H. Dreiem, D.I.K. Sjoberg, M. Jorgensen, “Estimating Software Development Effort based on Use Cases - Experiences from Industry”, *Fourth International Conference on the UML*, 2001, 487-504.
3) B.W. Bohem, “Software Engineering Economics”, *Prentice Hall*, 1981.
4) A. Cockburn, 山岸耕二ほか [訳], “ユースケース実践ガイド - 効果的なユースケースの書き方”, 翔泳社, 2001.
5) G. Karner, “Use Case Points - Resource Estimation for Objectory Projects”, *Objective Systems SF AB(Rational Software)*, 1993.
6) 工藤 拓, 松本 裕治, “Support Vector Machine による日本語係り受け解析”, 情報処理学会自然言語処理研究会報告, 2000, NL-138.
7) 工藤 拓, 松本 裕治, “チャンキングの段階適用による日本語係り受け解析”, 情報処理学会論文誌, 2002, 43:1834-1842.
8) K. Ribu, “Estimating Object-Oriented Software Projects with Use Cases”, *Master of Science Thesis University of Oslo*, 2001.
9) G. Schneider, J.P. Winters, 羽生田栄一 [訳], “ユースケースの適用:実践ガイド”, ピアソン・エデュケーション, 2000.
10) J. Smith, “The Estimation of Effort Based on Use Cases”, *Rational Software white paper*, 1999.
11) CaboCha : Yet Another Japanese Dependency Structure Analyzer, `http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/`.