修士学位論文

題目

ユースケースモデルを入力とした工数見積支援ツールの構築

指導教官

井上 克郎　教授

報告者

松川 文一

平成 16 年 2 月 12 日

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 ソフトウェア工学講座

平成 15 年度 修士学位論文

ユースケースモデルを入力とした工数見積支援ツールの構築

松川 文一

内容梗概

ソフトウェア開発における開発規模, 工数の見積りを行なう手法として, ファンクションポイント法が広まりつつあるが, この手法で効果的に見積りを行なうためのデータを得るには, 開発プロセスを設計段階にまで進める必要があるとされている. また近年, 設計段階での見積りでは受注競争に間に合わなくなってきており, プロセスのさらに早期である要求分析段階での効果的な見積り手法が求められている.

ファンクションポイント法をベースとし, オブジェクト指向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, 開発規模および開発工数を見積る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, モデル内の要素を重み付けする際にある程度の熟練度を必要とし, また同一プロダクトに対する計測でも, 計測者によって誤差が生じるといった問題点も存在する.

そこで本研究では, 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法に基づいて工数見積りを支援する計測ツールの試作を行った. また試作したツールを, UML モデリングツール「Describe」で記述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, ツールの評価を行った. 比較の結果, ツールの計測値は, 手動での計測値に近い値を算出したことを確認した.

主な用語

ソフトウェア見積り (Software Cost Estimation)
オブジェクト指向開発 (Object-Oriented Development)
ユースケースモデル (Use Case Model)
ユースケースポイント法 (Use Case Point Method)

目 次

1　まえがき

現代の高度情報化社会において, コンピュータシステムへの依存度が高まるにつれて, その主要な構成要素であるソフトウェアはますます大規模化・複雑化・多様化してきている. さらに, ソフトウェアの需要量も増大し, 開発期間の短縮化が求められるようになってきた. このような背景から, 高い品質を持ったソフトウェアを効率良く開発するために, 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必要性が高まってきている.

明確な開発計画を立てる上で, 開発中に起こるさまざまな事象を予測し, あらかじめ必要な手段を講じておくことは重要である. ソフトウェア開発に関して予測の対象となるものに, 開発規模, 投入する工数, 開発期間, 開発に使用される技術, 品質などがある. 中でも重要なものは, 開発工数と開発期間である.

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. 従来は, ソフトウェアの規模を表すのにプログラムの行数 (SLOC) が用いられていた. 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 [13]. 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.

この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, UML モデリングツールのひとつである, 「Describe」で記述された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. ツールとして自動に計測を行うにあたり, 各要素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, 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, またツールによって見積られた工数と実工数との比較を行うことで, 実用性の評価を行った.

以降, 2 節では, ソフトウェア開発における見積りについて述べる. 3 節では, 本ツールの入力となるユースケースモデルについて述べ, 4 節では, ユースケースポイント法について説明を行う. 次に 5 節でツール化へ向けて設定したルールについて述べ, 6 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. 7 節では, 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い, その結果についての分析と考察を行う. 最後に 8 節で, 本研究のまとめと今後の課題について述べる.

2　ソフトウェア開発における見積り

2.1　見積りの重要性

ソフトウェア開発を成功させる重要なポイントの 1 つは, 的確な生産管理の実施である [18]. ソフトウェアにおける生産管理とは, 高品質なソフトウェアを, 適正な価格で納期通りに開発するために, ソフトウェアのライフサイクル全体を通して開発状況をチェックし, 管理することである. 言い換えれば, ソフトウェアの生産管理とは, 開発計画時に設定した品質・コスト・納期の目標がキープできているか (あるいはキープできそうか) を, ソフトウェア開発の各工程において収集した実績データに基づいてチェックし, 問題があれば目標が達成できるよう, 対策の検討, 立案, 実施を迅速に行うことである.

初期のコンピュータプログラムは, 1 人のプログラマが 1ヶ月以内で開発できるくらいのものであり, それゆえ見積りの誤りの影響はさほど大きなものではなかった. しかしながら近年のソフトウェアの大規模化により, 1000 人もの技術者が 5 年以上かけて開発を行うようなものも出現してきている. そのような巨大なプロジェクトでは, 見積りの誤りは非常に深刻な結果をもたらしかねない [12]. 特に, 要求定義フェーズにおける見積りに重大な誤りがあったために, 納期遅延, 予算超過, キャンセルといった状況に陥るソフトウェア開発も少なくない. それゆえ, ソフトウェアの見積りは今や全ての開発現場において欠かせない重要な活動となっているのである.

2.2　ソフトウェアの規模とコスト見積り

ソフトウェアの見積りは, 「ソフトウェア計測」と「見積り」の 2 つからなる [14]. 「ソフトウェア計測」とは, ソフトウェアの持つ様々な要素に対して数的な評価を行うことで, ソフトウェアの規模を定量的に測定することである. 測定されたソフトウェアの規模から, 開発に費やす工数が見積られ, それに投入する時間や予算を決定することができるのである.

ソフトウェアの規模は, その量, 機能性, 複雑さから決定され, それらは静的に測定される. 量とは, 開発の各プロセスにおいて生成されるプロダクト (要求仕様書, 設計書, コードなど) の物理的な量を指す. 機能性とは, そのソフトウェアを利用するユーザから見た, 機能的要件の規模を指す. 複雑さとは, 効率性と問題の複雑さの両方を指す [9]. ソフトウェア見積りの手法についていくつかを以下に挙げる.

- ボトムアップ見積りとトップダウン見積り

  ボトムアップ見積りは, まず最も細かい単位のコンポーネントそれぞれに対して見積りを行い, それらを徐々に積み上げることで全体の積りとする方法である. 逆にトップダウン見積りは, まずプロダクト全体の見積りを行い, プロダクトを構成するコン

ポーネントそれぞれに全体の見積りから分配していく方法である [9].

- 専門家による評価

  これは, 見積りの熟練者が過去の経験に基づいて見積りを行う方法である. この手法は概ね正しい結果をもたらすが, その専門家の過去の経験に完全に依存してしまう. このような経験ベースの手法は, 定量的な実験的データが欠けているような場合に有効であり, その分野の専門家の知識に基づいたものであり [4], 見積りが偏ったものになってしまう可能性もある.

- 類推

  これは専門家による評価をより形式的な形で行う方法である. 見積者は対象のプロジェクトと, 過去のいくつかのプロジェクトとを比較し, 類似点及び相違点を識別し, 見積りの調整に用いる. まず見積者は対象がどのようなソフトウェアであるかを識別し, 初期見積りを行う. その後, 独自の範囲内で見積り結果の調整を行う. この手法による見積り結果の精度は, 過去のプロジェクトの情報の可用性に依存する.

- コストモデルの利用

  コストモデルとは, プロダクトの規模などといった入力から, 労力や時間を算出するための手法であり, ある数式によって算出する方法と, 何らかの参照テーブルを用いる方法がある [14]. またいくつかの調整要因を含んでいることも多い. この調整要因とは, 例えばプログラマの経験度といった, 生産性に影響を与えるようないくつかの要因を評価し, コストモデルによって算出された見積値に対して調整を行うものである. コストモデルの利点は, 定められた一定の方法により, 専門家や熟練者がいなくても見積りが可能な点にある. しかしながら, 対象ソフトウェアの開発方法に合わせて, そのモデルを適時変更する必要がある [14].

2.3 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, A.J. Albrecht によって 1979 年に提案された [1]. その後, これをベースに目的等に応じて様々な改良, 変更を行った手法が提案されてきており, 現在ではその方法は数十種類存在する. 例えば, IFPUG 法 [10], MkII 法 [27], COSMIC-FFP 法 [6] などが挙げられる.

このファンクションポイント法は, 利用者要求のうちの機能要求仕様の大きさを定量的に計測する手法であり, 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」という呼称が用いられている.

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, 種類数を加重合計した値を機能量とする. こうして得られた機能量の規模尺度としての最大の長所として,

- 規則に従って計測される値 (誰が計測しても安定した結果が得られる)
- 機能仕様にのみ依存 (開発環境や開発言語等の技術要件に依存しない)

が挙げられる. 本研究では, このファンクションポイント法をベースとして, G. Karner が 1993 年に提案したユースケースポイント法に着目する. ユースケースポイント法については, 4 節で詳しく述べる.

3　ユースケースモデル

3.1　UML について

UML は, オブジェクト指向開発のための仕様記述言語である. 1994 年, 代表的なオブジェクト指向開発技法である Booch 法 [17] を提案した G. Booch と OMT 法 [17] を提案した J. Rambaugh らが, 当時乱立していたオブジェクト指向開発方法論の統合を開始したのが始まりであり, 近年多くのソフトウェア開発の場において広く用いられており, 今後も普及が予想される. UML は 1997 年に UML1.1 が発表され [24], 2003 年には UML1.5 が発表された. 2004 年には最新の UML2.0 が正式リリースされる予定である [20]. UML では, 以下に示す 9 種類のダイアグラムが成果物として定義されている [24].

- 静的構造図：
  - **クラス図**：モデル内に存在するクラスやタイプ, その内部構造, 他の事項との関連について示したもの.
  - **コンポーネント図**：ソフトウェアのコンポーネント間の依存関係を示す.
  - **配置図**：システムの物理的なアーキテクチャを記述する.
  - **オブジェクト図**：クラスから生成される個々の具体的なオブジェクト同士の構造を記述する.
- 振舞い図
  - **ユースケース図**：アクタとシステム内のユースケースとの関係を示す.
  - **ステートチャート図**：オブジェクトの状態, イベント, アクションを記述する.
  - **アクティビティ図**：ステートチャート図に分岐制御などが加わったもの.
  - 相互作用図
    - **シーケンス図**：オブジェクトの相互作用を時系列で並べたもの.
    - **コラボレーション図**：オブジェクト間の相互作用と相互間のリンクを示したもの.

本研究では, このダイアグラムのうち, 開発の早期で作成されるユースケース図について特に着目する.

3.2　ユースケースモデル

ユースケースモデルは, オブジェクト指向ソフトウェア開発の要求分析段階において作成されるモデルである. 開発するシステムについて, 「誰 (何) が」, 「何を」するのかといっ

た, システムの動作, 機能をユーザ視点で表現するものである. Jacobson らは, システムの開発に際し, その機能要求をユーザの視点で把握, 表現し, また次の設計段階へのベースとして, このユースケースモデルを用いる, 「ユースケース駆動開発」を推奨している [11]. この「誰 (何)」にあたるもの, つまりシステムを使用する実体を**アクタ**, そして「何を」にあたる, システムの動作部分を**ユースケース**と呼ぶ.

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.

3.2.1 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. 図 1 に例を示す.

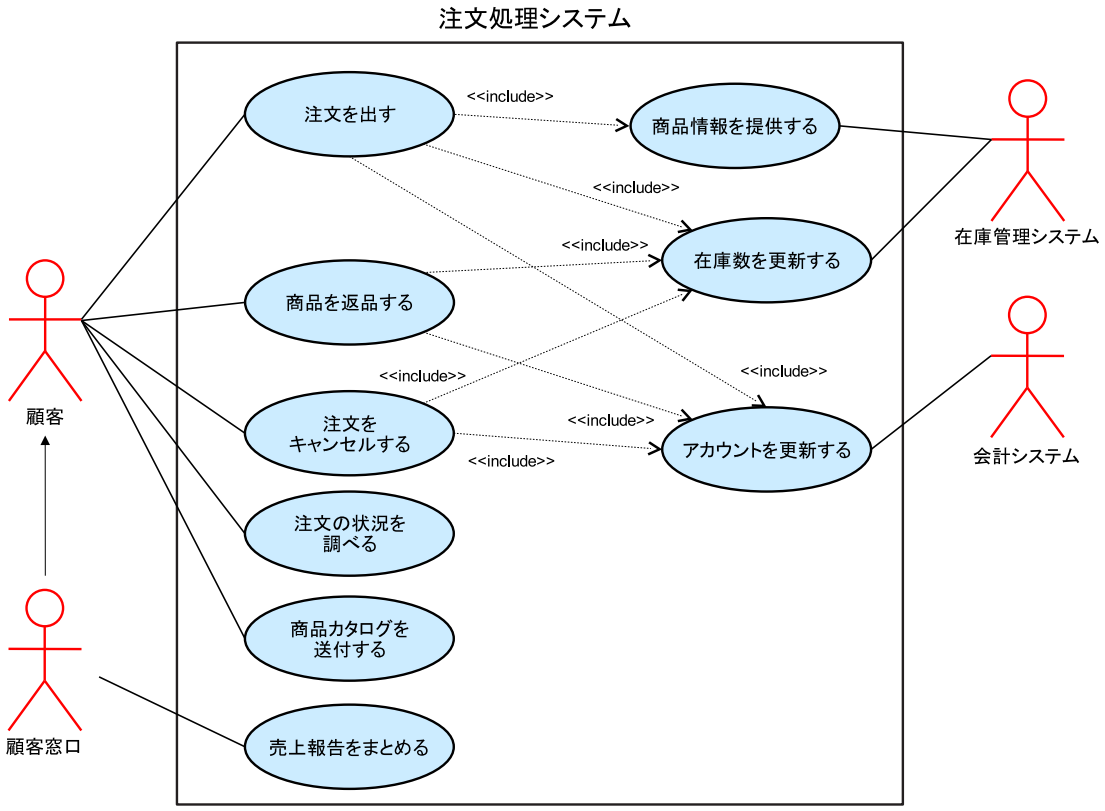


図 1: ユースケース図

また, アクタ間やユースケース間の特殊な関係について以下に示す.

- アクタ間の汎化

  あるアクタが別のアクタと同じ役割を果たす場合, アクタ間に汎化関係が成立する. 図 2 を例に考えると, アクタ「顧客窓口」は, ユースケース「売上報告をまとめる」と相互作用し, さらにアクタ「顧客」が相互作用している「注文を出す」「商品を返品する」「注文をキャンセルする」「注文の状況を調べる」「商品カタログを送付する」の 5 つのユースケース全てと相互作用することができる.

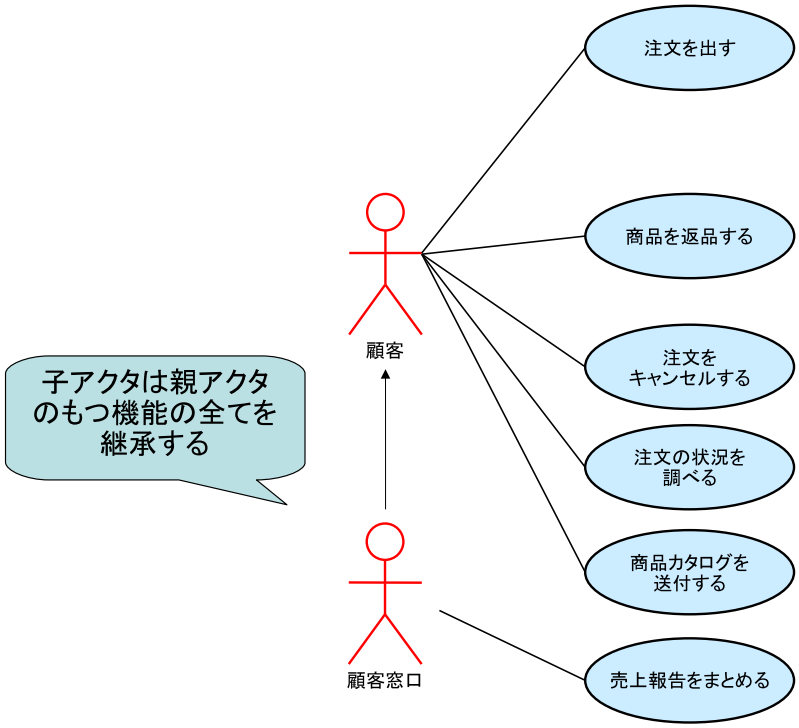


図 2: アクタ間の汎化関係

- ユースケースの包含

  複数のユースケースにおいて, 共通の振る舞いが存在する場合は, 包含関係を使って抽象化することができる. 図 3 を例に考えると, ユースケース「注文を出す」「商品を返品する」「注文をキャンセルする」の作業内容には, アカウントを更新するという共通の作業が含まれているため, それを 1 つのユースケース「アカウントを更新する」として抽出し, 包含する側から包含される側へ向けて, 破線の矢印でその関係を表す.

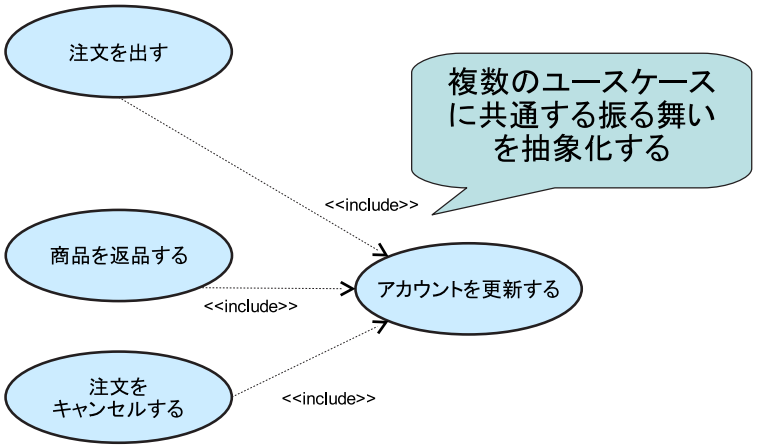


図 3: ユースケースの包含関係

- ユースケースの拡張

  あるユースケースに組み込みたいオプションのイベントシーケンスがある場合は, 拡張を用いてそれを表現する. 図 4 を例に考えると, 「注文を出す」ユースケースにおいて, ある決まった時期に特別割引をするといったことがある場合, 拡張ユースケース「期間限定特価」として記述し, そのオプションを追加する対象のユースケースへ破線の矢印を引く. 対象となるユースケースには, 拡張が許される地点を「拡張点」として記述する.

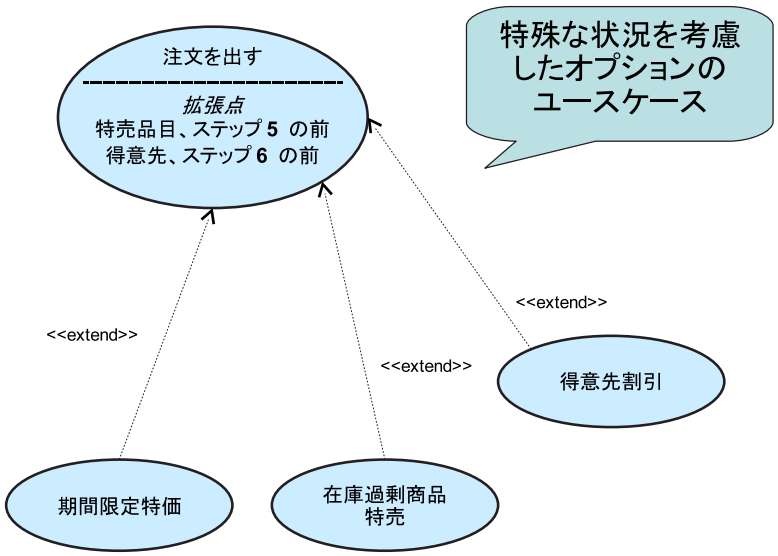


図 4: ユースケースの拡張

3.2.2 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のユースケース記述を図 5 に示す．

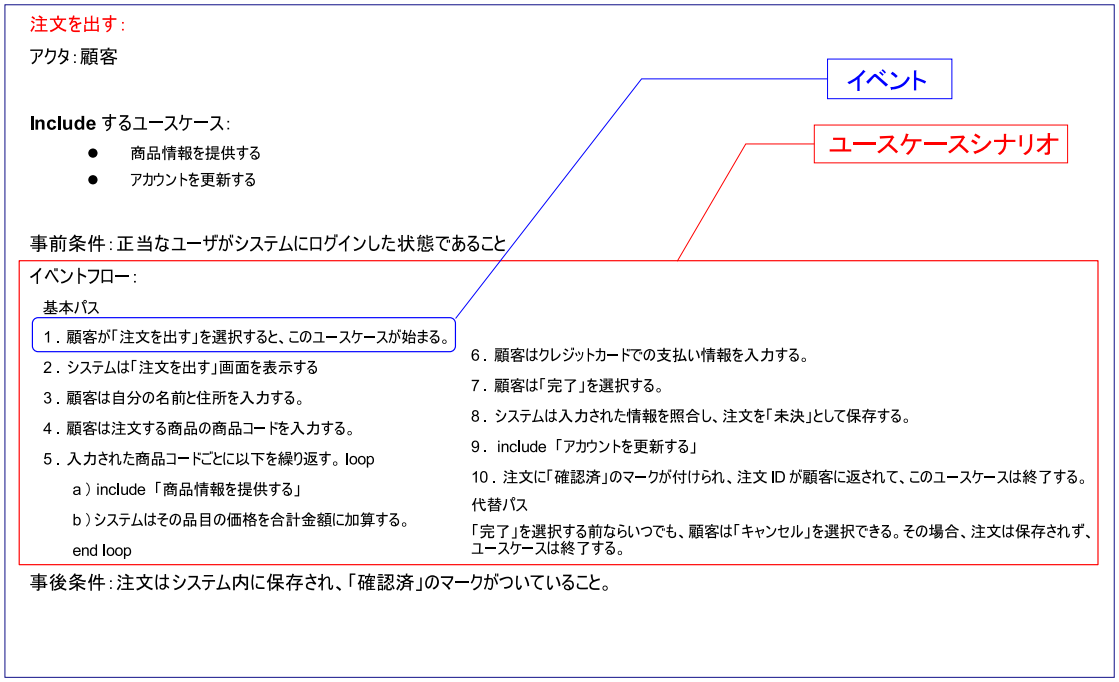


図 5: 「注文を出す」ユースケースのユースケース記述

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．イベントフローは「基本パス」と「代替パス」から成り，代替パスは，基本パスの代替手順を記述する．

本研究では，区別のため，図 5 に示すように，このイベントフローを「ユースケースシナリオ」，そして 1 つのパスを「イベント」と呼ぶことにする．

4　ユースケースポイント法

4.1　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 [13]. 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.

この技術の最大の利点は, ユースケースそれぞれのイベント数がある程度判明していれば, たとえユースケースが書かれる前であっても, 開発プロセスのごく初期の段階であまり正確ではないユースケースモデル調査を使って頭の中でも見積ができる点にある [30].

この手法はまだ確立されたものではなく, ファンクションポイント法などの他の見積技術と併用する必要があると言われている [25] が, 様々な関連研究, ケーススタディも報告されており, 注目すべき手法である.

4.2　計測手順

Karner が定めたユースケースポイントの計測手順を以下に示す.

4.2.1　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(Aplication 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.

4.2.2 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 (トランザクション数に基づく分類)

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.

また, システムの分析クラスを既に取り出しており, 個々のユースケースを実装するためにどのクラスを使うのかが判明している状態であれば, トランザクションの代わりにその情報を使ってユースケースの複雑さを決定することができる. このときには分析クラスだけを考慮し, 設計時に追加することになるクラスについては考慮しない. 分析クラスに基づく分類について表 3 に示す.

表 3: ユースケースのタイプと重み係数 (分析クラス数に基づく分類)

タイプ	説明	重み係数
単純	分析クラス数が 4 個以下	5
平均的	分析クラス数が 5 個から 10 個	10
複雑	分析クラス数が 11 個以上	15

しかしながら, 本研究では, 分析クラスを取り出していない段階を想定しているため, ユースケースの分類についてはトランザクションに基くものを採用している.

4.2.3 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.

4.2.4 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 4 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. 評価点が 0 であれば, その要因はプロジェクトには関係がないという意味であり, 5 であればその要因がプロジェクトに不可欠であることを示す.

表 4: 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.

$$TFactor = \sum\{(\text{各要因の評価点}) \times (\text{重み係数})\} \tag{2}$$

最後に, (3) に示す公式を用いて, 技術要因の調整係数を算出する.

$$TCF = 0.6 + (0.01 \times TFactor) \tag{3}$$

4.2.5 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 5 に示す 8 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.

F1 から F4 までの要因については, 0 はその項目に経験がないという意味, 5 は専門家であることを意味する. F5 については, 0 はモチベーションが低く, 5 は高いということを意味す

る．F6 については, 0 は要求が極めて不安定であり, 5 は要求が変わらないことを意味する．F7 は, 0 が兼任の技術スタッフが存在せず, 5 は技術スタッフが全員兼任ということになる．最後に, F8 は, 5 であれば非常に難しいプログラミング言語であることを意味する．

表 5: 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．

$$EFactor = \sum\{(\text{各要因の評価点}) \times (\text{重み係数})\} \tag{4}$$

最後に, (5) に示す公式を用いて, 環境要因の調整係数を算出する．

$$EF = 1.4 + (-0.03 \times EFactor) \tag{5}$$

4.2.6 ユースケースポイントの算出

(1) 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．

$$UCP = UUCP \times TCF \times EF \tag{6}$$

4.2.7 工数の見積り

(6) 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 [13]. しかしながら, 文献 [25] 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)

本研究における工数の算出については, この方法を採用している.

4.3 関連研究

これまでに, このユースケースポイント法, およびユースケースに基づく工数見積に関し, 様々な研究やケーススタディが報告されている. そのいくつかについて以下に紹介する.

- ユースケースモデルのファンクションポイント分析へのマッピング [8]
  オブジェクト指向モデルをファンクションポイント計測モデルへマッピングする手法を提案するもの. この中で筆者らは, ユースケースモデルに対していくつかのルールを設定し, マッピングを試みている. アクタの概念とファンクションポイントモデルの概念の広さの違いや, ユースケースの詳細さの違いから, 1 対 1 に対応づけるのは難解であるものの, 筆者らはオブジェクト指向モデルに対してファンクションポイントが適用可能であると結論づけている.

- ユースケースモデルに基づく規模予測 [26]
  これはユースケースから LOC(Lines Of Code) を見積り, 開発規模を予測するフレームワークについて記述されたものである. このフレームワークでは異なるカテゴリのシステム向けにユースケースのレベル, 規模, および複雑性の概念を考慮している. この手法はいくつかの見積ツールにおいて採用されている.

- ケーススタディ1[3]

  Arnold と Pedross による, スイスの主要銀行の大規模システムに対してユースケースポイントを適用した実験報告である. この実験において彼らが用いたユースケースポイント法は, Karner の手法を元に彼らが独自に開発したものであった. 彼らはこの実験から, ユースケースモデルから規模を見積ることが可能であることを示した.

- ケーススタディ2[2]

  Bente らによる 3 つのプロジェクトを対象とした実験. ここでは Karner の手法を用いている. 熟練者による見積りと, 実工数とも比較を行っており, どのプロジェクトにおいても熟練者の見積値に近い結果となり, そのうち 1 つは実工数にも近いものとなった. 彼らはこの実験から, ユースケースポイント法は他の見積手法と組み合わせて使うべきであることを示しているものの, 経験の浅い者などには受け入れやすい良いサポートとなると述べている. しかしながら, 今後より多くの実験を重ねていく必要があると述べている.

5　ユースケースポイント計測手法

5.1　計測方針概要

4 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. 本研究において計測の対象としているユースケースモデルは, Embarcadero Technologies 社 [32] が開発した UML モデリングツールである「Describe」でモデリングされたユースケースモデルである. 「Describe」が生成するモデルファイルは, XMI(XML Metadata Interchange) 形式 [21] で記述されており, アクタ, ユースケース, 関連などの様々な情報がこの形式で保存されている. そこから分類に必要な情報を機械的に抽出し, 計測を行うことにした. 次項以降, それぞれの計測方針について詳細を記す.

5.2　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が, 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.

5.2.1　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**

5.2.2 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 6 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**

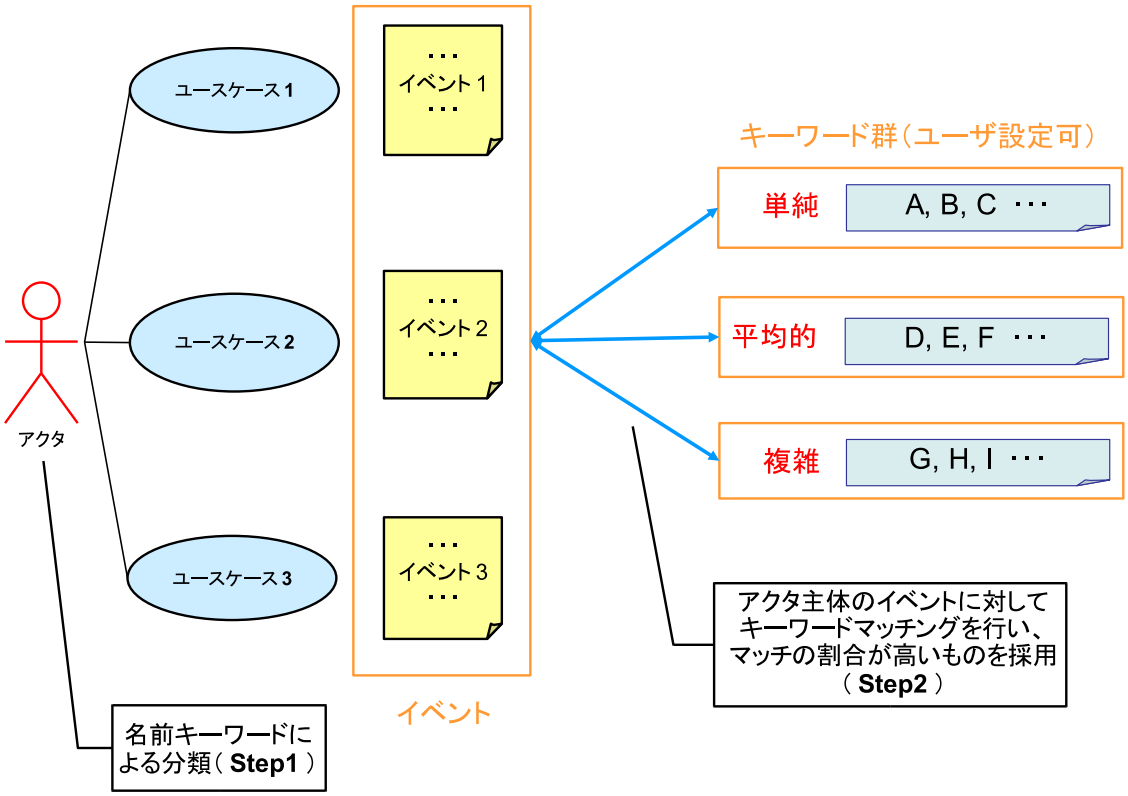


図 6: アクタの複雑さ決定イメージ

5.2.3 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, 複雑さの設定, 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**

5.2.4 汎化に対する処理

3 節で述べた様に, 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**

5.3 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, ユースケースの持つシナリオに着目する. 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, ユースケースシナリオ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, 3 節でも述べたとおり, 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, それはシナリオについても同様である. 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. 自由に記述可能なシナリオ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.

5.3.1 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, シナリオ作成者が 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, シナリオ中の全てのイベント記述に対し, 形態素解析, 及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.

解析をするにあたり, 奈良先端科学技術大学院大学情報科学研究科自然言語処理学講座が開発した日本語係り受け解析器「南瓜」[31] を採用した. 「南瓜」は, SVM(Support Vector Machines) に基づく日本語係り受け解析 [15] 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) [16] を採用しており, 効率の良い解析が可能となっている.

南瓜の実行例として, 「ユーザは、パスワードを入力し、OK ボタンを押す」という文章に対して解析を行った結果を, 図 7 に示す。

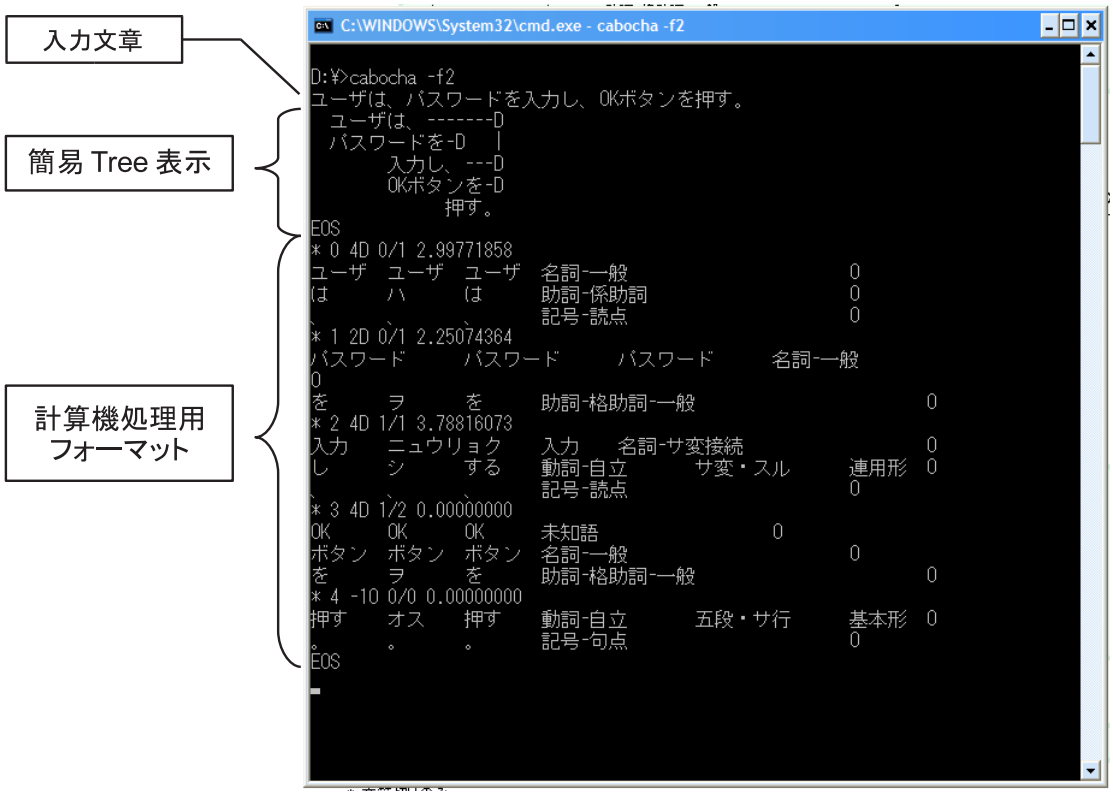


図 7: 南瓜の実行例

文節ごとの形態素, 及び係り関係の情報について, 簡易ツリーで視覚的に, また計算機処理用のフォーマットでも出力が可能となっている. 本研究では, この計算機処理用フォーマットで出力される情報を用いる.

5.3.2 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 [5] 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.

  – 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:ユースケースシナリオ内のイベント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**

5.3.3 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, ユースケースシナリオ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シナリオ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**

5.3.4 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 [13] 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, 23], 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, 自身のシナリオ内で包含 (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, その処理部分をトランザクションに含めない**

5.4 過去情報の蓄積

提案した方法で分類が不可能なアクタやユースケースに対して利用するために, これまでの計測情報をデータベースとして蓄積する. 蓄積する情報について以下に挙げる.

- プロジェクト情報
  - プロジェクト ID
  - プロジェクト名
  - 業種
  - 開発形態

  - 開発主言語
  - 認識されたアクタ数
  - 認識されたユースケース数
  - 見積もられた工数 (人時)
  - 開発開始日時
  - 開発終了日時
- アクタ情報
  - アクタ ID
  - アクタ名
  - 備考 (汎化関係など)
  - 決定された複雑さ
  - 業種
  - 使用されたプロジェクト ID
- ユースケース情報
  - ユースケース ID
  - ユースケース名
  - 備考 (包含, 拡張関係など)
  - 決定された複雑さ
  - 業種
  - 使用されたプロジェクト ID
- ユースケースシナリオ情報
  - イベント ID
  - 使用されたユースケース ID
  - イベント本体

6　工数見積り支援ツール U-EST

6.1　ツール概略

本ツール U-EST(Usecase-based Estimation Supporting Tool) は, UML モデリングツール「Describe」で記述されたユースケースモデルを入力とし, 5 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. 開発ライブラリとして JDK1.4.2[33] を使用し, モデルの解析には Apache[29] の提供する XML 構文解析ライブラリ Xerces2 Java Parser を使用した.

6.2　システム構成

本ツールのシステム構成図を図 8 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.

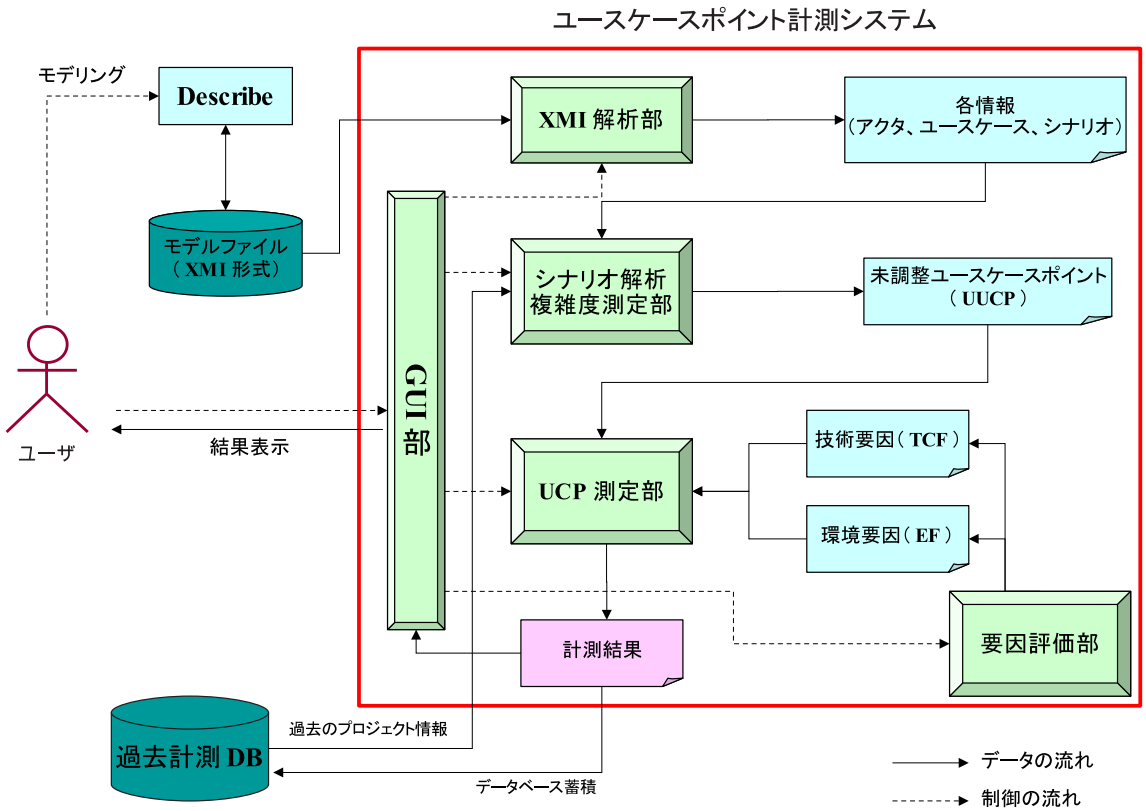


図 8: システム構成

6.2.1 XMI 解析部

ユーザが Describe を用いて作成したモデルファイル (XMI 形式で記述) を解析し, 記述されたアクタ, ユースケース, 及びそれらに関する情報を抽出する. 具体的な抽出方法については, 6.3 項で説明する.

6.2.2 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, 各アクタ, ユースケースのタイプ分類を行い, 未調整ユースケースポイントを算出する. 各分類結果, 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.

6.2.3 調整要因評価部

技術要因, 環境要因それぞれに対して, ユーザが画面上で評価を行い, 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.

6.2.4 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.

6.3 モデルの解析

Describe は, 独自の DTD 定義ファイルを用いた XMI(XML Metadata Interchange) 形式でモデルファイルを記述し, 本ツールはこのファイルを解析対象とする. 具体的には, Xerces2 を用いてモデルファイルの構文解析と DOM ツリーの構築を行い, そのツリーをトラバースしていくことでこれらの情報の抽出を行う. モデルファイルには, Describe で記述される UML 図の様々な情報がこの形式で記述されるが, 本ツールにおいて計測するために抽出する情報について表 6 に示す.

表 6: 抽出する要素

要素	属性	説明
UML:Project	xmi.id	プロジェクト ID
	name	プロジェクト名
UML:Actor	owner	親要素の ID
	xmi.id	アクタ ID
	name	アクタ名
	associationEnd	関連に関する要素 <UML:AssociationEnd> の ID
	specialization	汎化関係があれば, 汎化に関する要素 <UML:Generalization> の ID が記される
UML:UseCase	owner	親要素の ID
	xmi.id	ユースケース ID
	name	ユースケース名
	associationEnd	関連に関する要素 <UML:AssociationEnd> の ID
	includedBy	包含関係があれば, 包含に関する要素 <UML:Include> の ID が記される
	extendedBy	拡張関係があれば, 拡張に関する要素 <UML:Extend> の ID が記される
UML:AssociationEnd	xmi.id	自身の ID
	association	関連先のダイアグラムを探すキーとなる
	type	関連元のダイアグラム ID
UML:Generalization	xmi.id	自身の ID
	general	汎化における親アクタの ID
	specific	汎化における子アクタの ID

表 6: 抽出する要素

要素	属性	説明
UML:Include	xmi.id	自身の ID
	owner	包含するユースケースの ID
	target	包含されるユースケースの ID
UML:Extend	xmi.id	自身の ID
	owner	拡張するユースケースの ID
	target	拡張されるユースケースの ID
UML:TaggedValue	name	アクタやユースケースに持たせる属性の名前. ユースケースシナリオの場合, これが「documentation」となる
	xmi.id	自身の ID
	owner	この属性を持つダイアグラムの ID
	UML:TaggedValue.dataValue	この属性が更に持つ要素. ユースケースシナリオの場合, 実際のシナリオが記述される

6.4 ツールの実行例

本ツールを用いての計測の流れを説明する. ツール起動時のメイン画面を図 9 に示す.

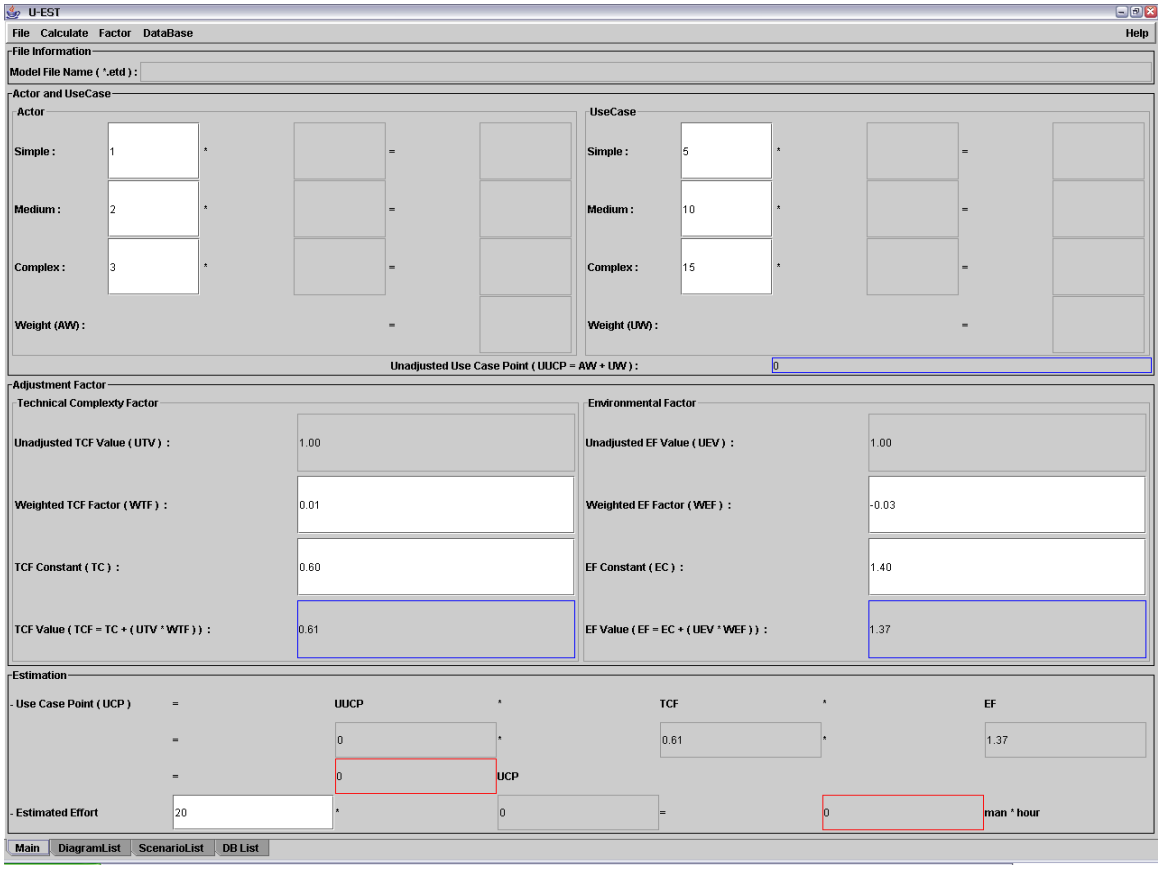


図 9: メイン画面

メイン画面では, 以下の情報を表示する.

- File Information
  入力となるモデルファイルのファイル名が表示される.

- Actor and UseCase
  アクタとユースケースに関する情報を表示

  – Actor
    各複雑さにおいて決定されたアクタ数, 及びその結果算出されたアクタの重みを表示.

  – UseCase

各複雑さにおいて決定されたユースケース数, 及びその結果算出されたユースケースの重みを表示.

- Unadjusted Use Case Point
  算出された未調整ユースケースポイント値を表示.

- Adjustment Factor
  調整要因に関する情報を表示.
  - Technical Complexity Factor
    技術要因について, 評価の結果 (TFactor), 算出式の因数及び定数 (ユーザによる調整可能), そして算出される調整係数を表示.
  - Environmental Factor
    環境要因について, 評価の結果 (EFactor), 算出式の因数及び定数 (ユーザによる調整可能), そして算出される調整係数を表示.

- Estimation
  算出されたユースケースポイント, 及び見積工数を表示する. 見積の際の工数変換係数は, デフォルトで 20 となっているが, ユーザによる変更が可能.

6.4.1 モデルファイルの読込み

メニュー「File」から「Set the Input File」を選択すると, ファイル読込みダイアログ (図 10 参照) が起動し, ユーザは計測対象となるモデルファイル (etd ファイル) を選択する. ファイル読込みに成功すると, 同時に解析を行い, ユースケースモデル中のアクタ, ユースケースに関する情報を抽出し, リストとしてユーザに提示する.

6.4.2 調整要因の評価

メニュー「Factor」から, 技術要因であれば「Evaluate Technical Factor」, 環境要因であれば「Evaluate Environmental Factor」を選択すると, それぞれの評価ダイアログが表示される. ユーザはそれぞれの要因に対して評価を行い, 「OK」ボタンを押すと, 評価の結果がメイン画面に反映される. 図 11 には, 技術要因評価画面を示す.

6.4.3 計測

モデルファイルの読込み, および調整要因評価を行った後, メニュー「Calculate」を選択すると, アクタ, ユースケースの分類, およびユースケースポイントの計測を行う. このとき,

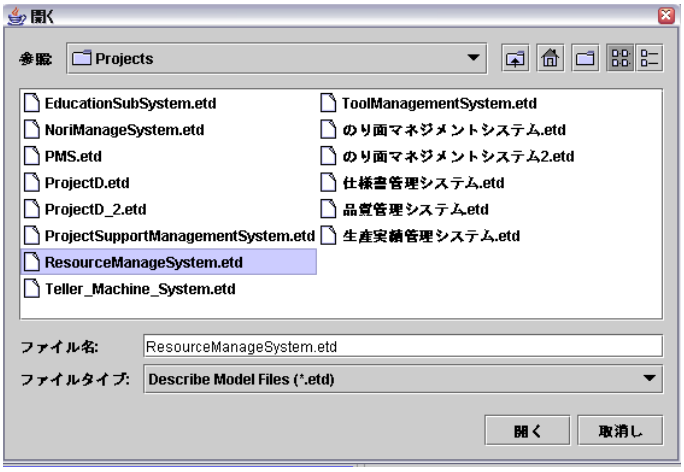
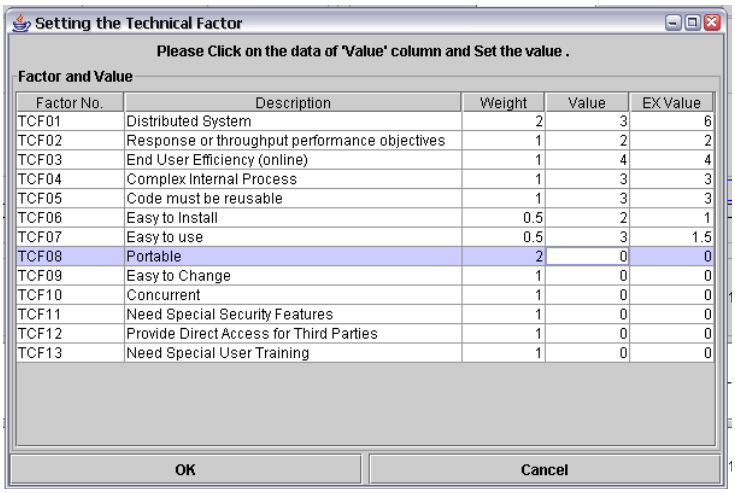


図 10: モデルファイル読込ダイアログ



Factor No.	Description	Weight	Value	EX Value
TCF01	Distributed System	2	3	6
TCF02	Response or throughput performance objectives	1	2	2
TCF03	End User Efficiency (online)	1	4	4
TCF04	Complex Internal Process	1	3	3
TCF05	Code must be reusable	1	3	3
TCF06	Easy to Install	0.5	2	1
TCF07	Easy to use	0.5	3	1.5
TCF08	Portable	2	0	0
TCF09	Easy to Change	1	0	0
TCF10	Concurrent	1	0	0
TCF11	Need Special Security Features	1	0	0
TCF12	Provide Direct Access for Third Parties	1	0	0
TCF13	Need Special User Training	1	0	0

図 11: 技術要因評価ダイアログ

別プロセスで「南瓜」が作動しているため, 処理には若干の時間がかかる.

計測が終わると, メイン画面に各結果が表示されるとともに, 「ダイアグラムリスト」画面 (図 12 参照) に各アクタ, ユースケースについて決定された複雑さ, トランザクション数などの情報が表示され, 「シナリオリスト」画面 (図 13 参照) には各イベントごとに認識されたトランザクション数が表示される.

また, 状況に応じ, ダイアグラムリストにおける各アクタ, ユースケースに対する複雑さ, およびシナリオリストにおけるトランザクション数のユーザによる変更が可能となっている. 変更結果は, すぐに各画面に反映される.

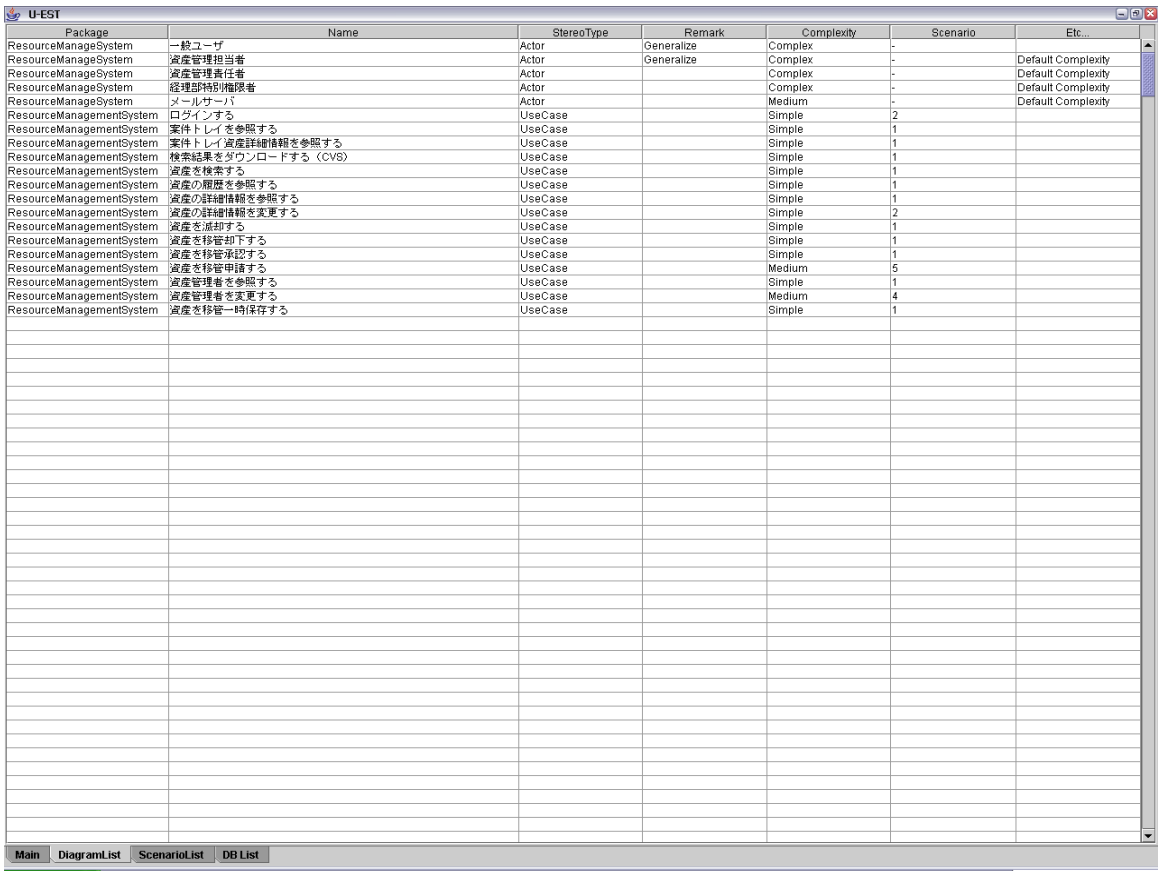


図 12: ダイアグラムリスト

ダイアグラムリストでは, 以下の情報が表示される.

- package
  この要素が含まれるパッケージ名

- Name

要素名

- StereoType
  アクタもしくはユースケース

- Remark
  汎化関係 (Generalize) および包含 (include), 拡張 (extend) があれば表示

- Complexity
  決定された複雑さ (Simple, Medium, Complex)

- Scenario
  ユースケースの場合, 認識されたトランザクション数

- Etc...
  備考欄 (データベース提示, 及びデフォルト表示)

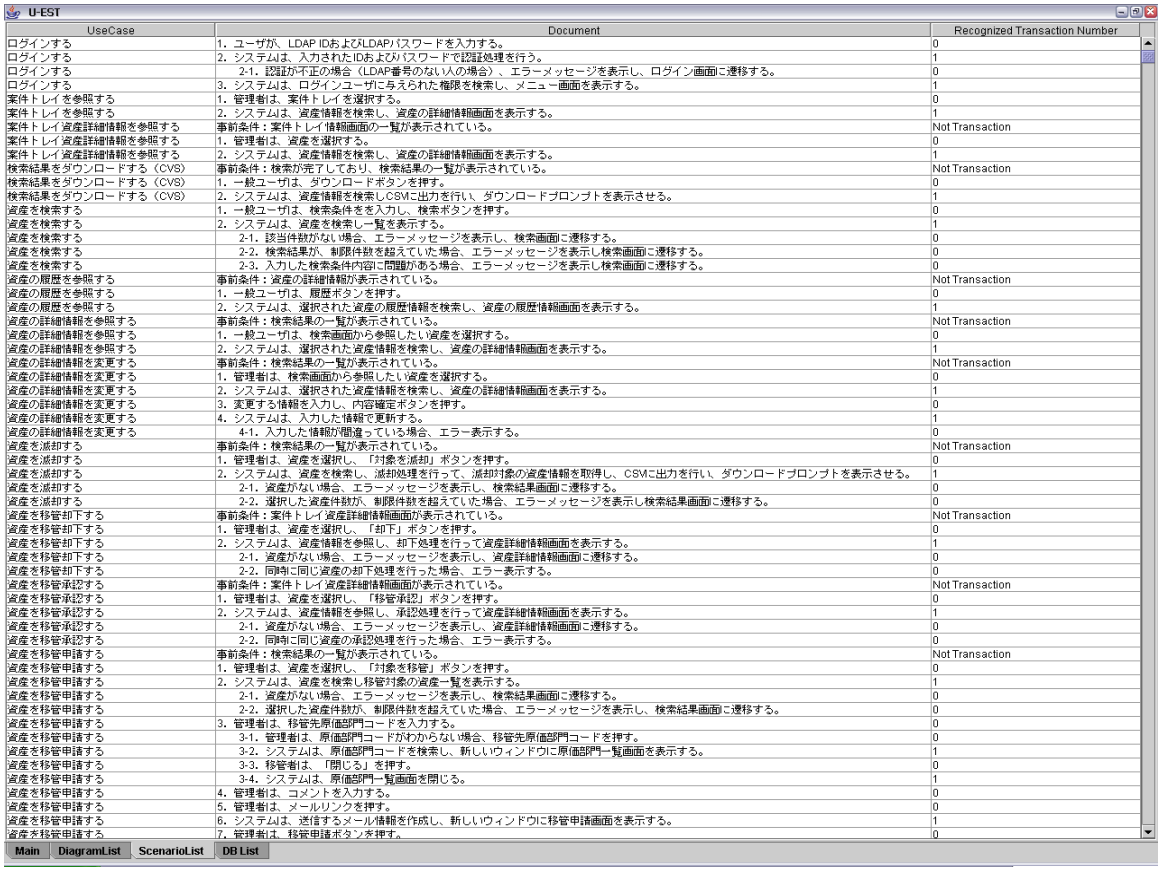


図 13: シナリオリスト

シナリオリストでは, 以下の情報が表示される.

- UseCase
  そのイベントを持つユースケース名

- Document
  イベント本文

- Recognized Transaction Number
  そのイベントに対して認識されたトランザクション数

6.4.4 データベース

過去の計測情報を蓄積するデータベースについては, 「新規登録」「検索」「更新」「削除」が可能となっている. メニュー「DataBase」から各操作を選択して行う.

- 新規登録

  計測を終えたプロジェクトを新たにデータベースに登録する。登録する情報は、「プロジェクト名」、「業種」、「開発形態」、「主となる開発言語」、「アクタ数」、「ユースケー

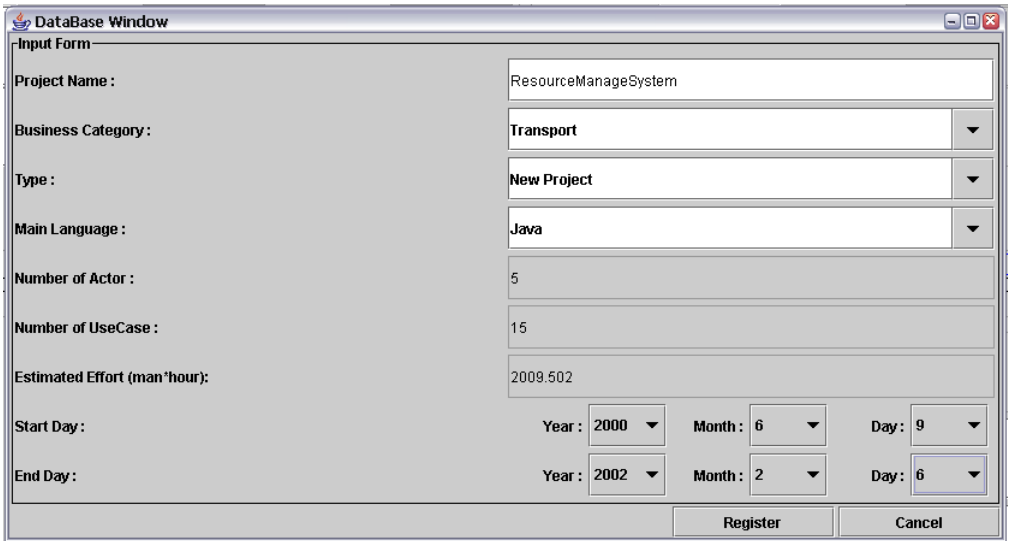


図 14: プロジェクト新規登録ダイアログ

- 検索

  データベース内の情報に対して、フリーワードによる検索を行う。プロジェクト、アクタ、ユースケース、シナリオの各情報から選択して検索を行い、ヒットした情報は各情報テーブルに表示される。

- 更新

  蓄積された情報に対して変更を行い、登録し直す。一旦読み出された全ての情報が各情報テーブルに表示されるので、それらに対して自由に変更を行うことができる。

- 削除

  蓄積されている情報の削除を行う。テーブルには、蓄積されているプロジェクト情報が表示され、ユーザは削除したいプロジェクトのチェックボックスに印をつける。「Delete」ボタンが押されると、印を付けたプロジェクト、および関連するアクタ、ユースケース、シナリオの全ての情報も同時に削除される。

7　評価

7.1　評価概要

試作したツールが行う分類, 及び計測するユースケースポイント値が妥当なものであるかどうか, そしてこのユースケースポイント法およびツールが実用性のあるもの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. そのユースケースモデルは全て「Describe」で作成されている. 表 7 に, その規模等について記す.

表 7: 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

比較用データとして, 各プロジェクトに対し, 手動での計測結果とツールによる計測結果 (各要素の複雑さ, 未調整ユースケースポイント, ユースケースポイント, 見積工数), そして判明しているものについては, 実工数データを用意した. なお, 技術要因, 環境要因の評価については, 手動計測者のそれと同じ評価値を用いている.

以降, 用いたプロジェクトについて述べ, その後それらに対して行った評価方法とその結果, 及び考察について述べる.

7.2　ツールの精度

各要素の複雑さに関して, 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う. これにより, アクタの複雑さ判定のキーワード設定, および形態素解析を行った上でのトランザクション認識方法の正確性を評価する.

7.2.1　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 8, 表 9 に示す. 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 [25] を参

考にし, 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 8: 複雑さの分類結果 (アクタ)

プロジェクト	ツール			手動			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	0	1	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 9: 複雑さの分類結果 (ユースケース)

プロジェクト	ツール			手動			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	11	9	0	14	6	0	0.85
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.

7.2.2 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. 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, 外部システムの場合, 動作の主体ではなく対象となり, イベントとして陽に記述されていないことがわかった.

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, データベースからの検索, 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.

7.2.3 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, 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.

7.2.4 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, 形態素解析でも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, しかもシステムのレスポンスが, 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.

7.2.5 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 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, 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. その結果を表 10 に記す.

表 10: ユースケースに対する再分類の結果

プロジェクト	ツール			手動			タイプの
	単純	平均的	複雑	単純	平均的	複雑	一致割合
B	10	4	0	10	4	0	1.0
C	14	6	0	14	6	0	1.0
D	27	1	0	27	1	0	1.0
E	2	8	3	2	8	3	1.0

再分類の結果, 手動による分類に近い結果となった. 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認識する動作, もしくは認識しない動作をキーワード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 つの部品を考えると, 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ることもでき, 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より多くのケーススタディを通して議論する必要がある.

7.3 実工数との比較

ツールによって見積られた工数と, 手動計測によって見積られた工数, 及び実工数との比較を行う. 本ツールの最終的な出力は見積り工数であり, 実工数との比較を行うことで, 本ツールの実用性を評価する.

7.3.1 結果

結果について表 11 にまとめる. ただし, プロジェクト E に関しては, 現在進行中のプロジェクトであるため, 実工数が得られなかった. それゆえ, 工数の比較については比較対象外とする.

表 11: 工数データ (単位は人時)

プロジェクト	ツール見積工数	手動見積工数	実工数
A	20.8	20.6	18.6
B	13.1	10.2	4.5
C	29.8	26.8	16.0
D	23.9	20.5	21.2

ツールによる見積工数, 手動計測による見積工数ともに, プロジェクト B を除いて大きな差はみられなかったものの, どのプロジェクトに関しても, 実工数よりも大きく見積っている結果となった.

7.3.2 考察

まず, プロジェクト B に関して, 実工数との差が大きいものとなったが, 実際の開発において, このプロジェクトは自動生成部分の割合が高かったことから, 実工数が低くなったものと考えられる.

他の 3 つのプロジェクトについて, 実工数に対する見積工数の比率を調べてみると, ツールでは 111.83%〜186.30%, 手動計測で 96.87%〜167.79%の範囲であった. Bohem の主張によると, 要求仕様段階における実工数と見積工数との誤差は, 前後 60%以内に抑えるべきであるとしている [4]. 先に挙げた改良点なども考慮すると, 本手法及び本ツールは, 実用性が見出せるものであると考える. しかしながら, 今後もより多くのケーススタディを通して, 手法及びツールの改良が必要である.

8 むすび

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.

謝辞

本研究の全過程を通して，常に適切な御指導，御助言を賜りました 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 井上 克郎 教授に心より深く感謝致します.

本論文を作成するにあたり，常に適切な御指導，御助言を賜りました 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 助教授に心から感謝致します.

本論文を作成するにあたり，適切な御指導，御助言を賜りました 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 松下 誠 助手に心から感謝致します.

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 氏に深く感謝致します.

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 高橋まゆみ 氏に深く感謝致します.

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 生産技術部 英 繁雄 氏に深く感謝致します.

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 生産技術部 前川 祐介 氏に深く感謝致します.

最後に，その他様々な御指導，御助言等を頂いた 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 井上研究室の皆様に深く感謝いたします.

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] M. Arnold, P. Pedross, "Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department", *Forging New Links. IEEE Conput. SOC, Los Alamitos, CA, USA*, 1998, 490-493.

[4] B.W. Bohem, "Software Engineering Economics", *Prentice Hall*, 1981.

[5] 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.

[6] COmmon Software Measurement International Consortium, "COSMIC-FFP Measurement Manual", 2001.

[7] M. Demondran, A.N.E. Washington, "Estimation Using Use-Case Points", *The Proceedings of Information Systems Education conference*, 2002, 19:253d.

[8] T. Fetcke, A. Ablan, T. Nguyen, "Mapping the OO-Jacobson Approach into Function Point Analysis", *The Proceedings of 23rd Technology of Object Oriented Languages and Systems*, 1997, 192-202.

[9] N.E. Fenton, S.L. Pfleeger, "Software Metrics. A Rigorous and Practical Approach", *Cambridge University Press*, 1997.

[10] International Function Point Users Group (IFPUG), "Function Point Counting Practices Manual, Release 4.1", 1999.

[11] I. Jacobson, M. Christerson, P. Jonson, G. Overgaard "Object-Oriented Software Engineering. A Use Case Driven Approach", *Addison-Wesley Publishing Company*, 1992.

[12] C. Jones, 富野 壽, "ソフトウェア見積りのすべて", **共立出版**, 2001.

[13] G. Karner, “Use Case Points - Resource Estimation for Objectory Projects”, *Objective Systems SF AB(Rational Software)*, 1993.

[14] B. Kitchenham, L. Linkman, D.L. Desmet, “A methodology for evaluating software engineering methods and tools”, *Computing and Control Engineering Journal*, 1997.

[15] 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.

[16] 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.

[17] Lockheed Martin Advanced Concepts Center, “Succeeding with the Booch and OMT Methods : A Practical Approach”, *Addison-Wesley Publishing Company*, 1996.

[18] 真野 俊樹, 誉田 直美, “見積りの方法 - ソフトウェア開発における実践的見積り技法”, **日科技連**, 1993.

[19] 丸山 宏ほか, “XML と Java による Web アプリケーション開発 第 2 版”, **ピアソン・エデュケーション**, 2000.

[20] Object Management Group (OMG), “UML2.0 Superstructure Final Adopted Specification”, 2003.

[21] Object Management Group (OMG), “XML Metadata Interchange (XMI) Specification Version 2.0”, 2003.

[22] P. Pedross, “Software Size Estimation at the Credit Suisse Group: The Use Case Point Method”, *IFPUG Conference*, 1997.

[23] K. Ribu, “Estimating Object-Oriented Software Projects with Use Cases”, *Master of Science Thesis University of Oslo*, 2001.

[24] J. Schmuller, 長瀬嘉秀 [訳], “独習 UML”, **翔泳社**, 2001.

[25] 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.

[26] J. Smith, “The Estimation of Effort Based on Use Cases”, *Rational Software white paper*, 1999.

[27] C.R. Cymons, “Software Sizing and Estimating, MkII FPA”, *John Wiley and Sons*, 1991.

[28] ユースケースポイントによる見積講習会のまとめ,
http://objectclub.esm.co.jp/UseCasePoint.html.

[29] Apache, http://xml.apache.org/.

[30] @IT : The Rational Edge Dr. ユースケースの “ユースケース人生相談”,
http://www.atmarkit.co.jp/fjava/devs/redge10/redge10.html.

[31] CaboCha : Yet Another Japanese Dependency Structure Analyzer,
http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/.

[32] Embarcadero Technologies, Inc - Provider of database and application development,
http://www.embarcadero.com/index.html.

[33] Java Technology,
http://java.sun.com/.

[34] Hawk’s Nest@XMI に関する研究報告,
http://www.tom.sfc.keio.ac.jp/~hawk/xmi.html.