反復型開発のためのユーザが理解しやすい進捗状況把握支援システム

吉田享子[†,††] 飯島正[†]
櫻井彰人[†] 山口高平[†]

本論文では，反復型開発においてユーザが理解しやすい進捗情報の提示方法を提案するとともに，その実現のためにツールとして作成した支援システムについて述べる．このシステムは，階層・包含・拡張関係が設定されたユースケースに表された機能と，分析クラスと実装クラスを関係付け，実装工程で逆に実装クラスからユースケースに変換することによって，ユーザにユースケースに表される機能から見た進捗情報を提供するものである．また，進捗状況を把握することが難しいとされる反復型開発においても，本手法を適用させることによって，ユーザのために，それぞれの反復の進捗状況に加えて，反復型開発全体の進捗状況を，機能を通して可視化させた．本支援システムに実際のソフトウェア開発の例題を適用し，反復的に開発する過程をシミュレーションさせ，本支援システムが有効であるか検討した．

User-oriented Progress Reporting System for Iterative Development

Kyoko Yoshida,[†,††] Tadashi Iijima,[†] Akito Sakurai[†]
and Takahira Yamaguchi[†]

In this paper, we propose a method and a support system which provide comprehensible progress reports for user. This system handles relationships among use-cases, such as generalization, inclusion, and extension, and holds links between a sentence of use-case and an implementation class. And the system provides the progress reports based on functions shown in use-cases, by converting from implementation classes to use-case. In addition, this system handles iterative development and visualizes the progress of iterative development as a whole. And then, we examined the effectiveness of the system by simulating an iterative process of software development.

1. はじめに

ソフトウェア開発では，初めに計画したスケジュールや予算のとおりに開発が進まず，スケジュール遅延や予算超過や品質低下を起こすプロジェクトが少なくない．米国においても，ソフトウェア開発の 1/3 から 2/3 が，スケジュール遅延とコスト超過になっていると報告されている[1)]．このようなプロジェクトでは，開発途中に，開始時のタイム，コスト，品質の制約条件を変更せざるをえない状況におちいることがある．これらの制約条件の変更は，開発の進捗状況と関係することが多く，発注者が最終的な判断に関わるためには，進捗を把握することが重要になる．たとえば，スケジュールが大幅に遅延し，稼動開始時期を遅らせる必要が生じたとき，発注者が開発の進捗状況を正しく理解していないと，誤った判断をくだす危険性がある．このような場合，開発者と発注者に対して，分かりやすい方法で開発の進捗状況を提示するツールがあれば，これを利用することによって，正しい判断を支援することができる．またこのツールを使って，両者が進捗情報を共有することができれば，進捗の遅れとして表面化することが多い開発途中の問題に対して，早期の対応が可能となる．

また近年，ソフトウェア開発の失敗を防ぐために，開発を成功に導くためのベストプラクティス（最良の実行原則）が集められている．そこでも，ユーザが開発の進捗を把握することの重要性が指摘されている[2)]．PMBOK（Project Management Body of Knowledge）においても，ユーザに対する進捗情報提供を重視することが強調されている[3)]．

本論文では，開発者から見て顧客にあたる，ソフトウェア開発の発注者をユーザとする．ソフトウェアの発注者も，ソフトウェアの開発に対する理解度はさまざまで，要求仕様だけが分かる人もあれば，設計や実装方法まで理解できる人もいる．ここでは，ソフトウェア開発で実現したい要求仕様については熟知している

† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
†† 平成国際大学
Heisei International University

が，設計や実装方法は分からない人，しかし，開発の進捗状況を把握する必要のある人をユーザとする．

進捗を把握する重要性が認識されるようになってきたにもかかわらず，ユーザがソフトウェア開発の進捗を簡単に把握できる環境は整っていない．そのため我々は，ユーザが理解しやすいように，要求仕様書に現れる機能を使って進捗状況を表す支援システムを提案した[4]．しかしこれは，簡単な要求仕様書について適用させた基本的なシステムであり，実際のソフトウェア開発に使用するには限界があった．今回報告するシステムでは，複雑な要求仕様書に使用できるように改善を加え，反復型開発にも適用できる支援システムに改訂した．

2. 進捗管理における問題点と提案手法

2.1 進捗管理における問題点

ソフトウェア開発の進捗管理においては，ユーザが進捗状況を把握することの重要性が強調されているにもかかわらず，ユーザが進捗を理解しやすいように考慮された進捗レポートは用意されていない．現在使われている進捗レポートは，開発の各工程の作業を洗い出したうえで，作業ごとの生産物の量によって進捗を表しているものが多い．特に，実装工程ではクラスやモジュールの完成度で進捗を測るのが一般的であり，進捗レポートもこれに基づいて作成されている．クラスやモジュールは，設計によって作成された成果物であり，要求仕様を実現するためのしくみにあたる．これは，ソフトウェアの設計や実装の方法を知らないユーザに適した進捗レポートとはいえない．

最近は，開発プロセスとして，ウォータフォール型だけではなく反復型開発[5]も適用されるようになっている（反復型開発は，分析・設計・実装・テストという一連の開発を繰り返して適用する開発方法論であり，その 1 回あたりの開発をここでは反復と呼ぶ）．反復型開発に適する例としては，要求仕様が確定できない場合や，新しい技術を使用するためその技術への不安を早い時期に解消しておきたい開発などがある．反復型開発は，小さいサイクルで一連の開発を複数回繰り返すため，リスクを早期に発見できるメリットがある．また，ユーザのフィードバックを取り入れながら開発できるため，ユーザの要求に近いソフトウェアを作成することができる．しかし，反復ごとの進捗管理に加えて，開発全体の進捗管理も必要となり，ウォータフォール型よりも管理が難しい面もある．また，反復型開発では，毎回の反復の作業が前回の作業結果の影響をうけ，開発途中で反復が追加されたり削除されたりすることもある．進捗管理システムも，この点を考慮したものが必要となっている．

近年では，進捗管理において，PMBOK に基づいて，earned value を統一的な尺度として，プロジェクトの予算やスケジュールを定量的に測定・分析し，一元的な管理を行う手法も提案されている[6]．しかしこうした手法も，開発側の立場から，時間と予算と品質を管理することに重点が置かれたものであり，ユーザの視点からのアプローチを試みているものではない．

2.2 支援システムの概要

ソフトウェア開発では，ユーザと開発者が協力して要求仕様書を作成し，開発者が要求仕様を実現するためにプログラムを作成する．開発者は，要求仕様とプログラムの両方を理解しているが，ユーザは，要求仕様は分かるがプログラムは分からない．この点を考慮すれば，ユーザに対しては，要求仕様を使って進捗を表す方が理解しやすいと考えられる．本支援システムでは，要求仕様書としてユースケースを，プログラムとして実装クラス（実装すべきクラス）を使用する．

ユーザと開発者は，共同でユースケースを作成し，開発者が，本支援システムを使って，ユースケースの入力，分析クラスの抽出，実装クラスへの変換，実装における進捗情報の入力を行う．本支援システムは，ユースケースから実装クラスを作成する変換過程を保存し，逆にそれを利用して，実装クラスで集められた進捗情報をユースケースから見た進捗に変換して，ユーザ用の進捗レポートを作成する．実装クラスで集められた進捗情報は，そのまま開発者用の進捗レポートにも使用される．反復型開発においても，反復ごとにユースケースから実装クラスを作成する変換過程を保存することによって，実装クラスの作成を支援し，進捗状況が比較できるような進捗レポートを作成する（**図 1**）．

本支援システムは，先行研究であるユーザのための進捗管理システム[4]と比較して，以下の特色を持つ．

まず，多数の入り組んだ構造を持つユースケースを扱えるようにした．要求仕様書としてユースケースを作る過程では，最初に粒度の粗いユースケースを作成し，そのユースケースを詳細化していくという作業が行われる．また，ユースケースの中の共通な部分をまとめて，新しいユースケースとして独立させ，共有ユースケースとして扱うことや，ユースケースの一部を別のユースケースに拡張することもある．今回の支援システムは，ユースケース間でこれらの階層，包含，拡張関係を設定できるようにした．このため，多くのユースケースが関連しながらソフトウェアの仕様の全

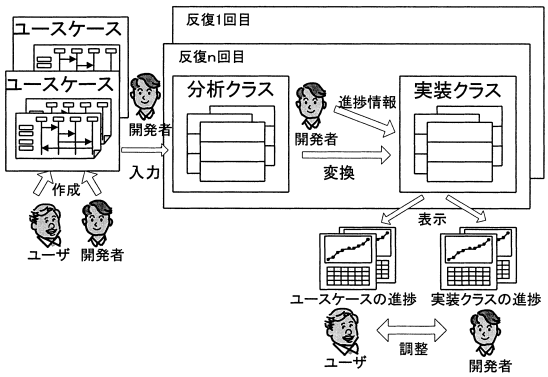

図 1 支援システムの概要
Fig. 1 System overview.

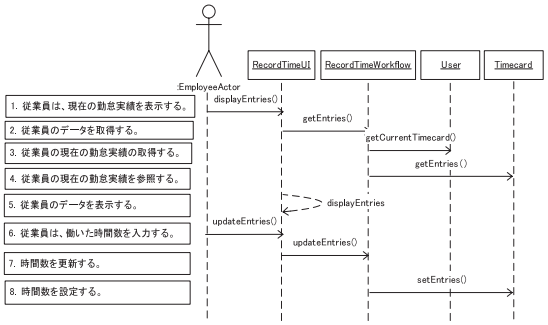

図 2 ユースケースの例
Fig. 2 Example of a use-case.

体を構築していく開発に使用することが可能となった．

また，反復型開発においても本支援システムを適用できるようにした．反復型開発では，新しい反復開始時には，それまでの反復で作成されているユースケースを修正したり，新しいユースケースを加えたりしなければならない．実装クラスについても，既存のものをそのまま利用する場合もあれば，修正や追加作成が必要になる場合もある．この作業を支援するために，前回と今回のユースケースを比較して，前の開発で作成された実装クラスが今回の開発で使えるかどうかを識別できるようにする．そのうえで，反復型開発においても，ユースケースを通して見た進捗レポートを提示する．

2.3 対象とするソフトウェア開発

本論文では，機能と構造との対応性が良いソフトウェア開発を対象とすることにする．たとえば，大きなソフトウェアではデータアクセス機能の完全性が問題になることがあり，またデータアクセス機能を使うときのワークフローが重要になることもある．こうした場合には，要求仕様書に現れる機能とソフトウェアの構造との対応は悪くなる．本研究では，これらのようなソフトウェア開発は対象にしていない．また，ユースケースで表現される機能に限定して，非機能要求については対象外としている．

今回は，機能と構造の対応を保ちやすい方法で開発することを考慮して，UML（Unified Modeling Language）[7] を使用したオブジェクト指向で開発することとした．

一方，ソフトウェア開発に対するユーザのかかわり方は，プロジェクトの規模や，ユーザ側の組織体制，作業に対する責任の切り分け方によっても異なってくる．本論文では，ユーザは，開発者と協力して要求仕様書の作成を行い，開発者がコンピュータ上に実装する作業を請け負う．実装終了後は，ユーザがその実施と運用に責任を負うケースを想定している．

2.4 使用するユースケース

UML では，ユースケース図についての定義はあるが，ユースケースの記述法については規定がなく，現在さまざまな方法が提案されている．本手法では，要求仕様書として **図 2** のユースケース記法を採用する．これは，ユースケースのイベントフローの文を左に記述し，その文と対応したシーケンス図を右に配置したものである．シーケンス図の上には，イベントフロー文から抽出されたアクタとオブジェクトが配置され，アクタとオブジェクトの間はイベントフロー文と対応したメッセージを記述し，矢印で結ぶ．イベントフローの文とシーケンス図のメッセージは 1 対 1 で対応させる．本論文では，イベントフローとシーケンス図をあわせたものをまとめてユースケースと呼び，イベントフローの各文をイベント文と呼ぶ．図 2 のユースケースはユーザと開発者が協力して作成する．

2.5 反復型開発の手順

反復型開発については，RUP（Rational Unified Process），XP（Extreme Programming），FDD（Feature-Driven Development）など，各種のプロセスが提案されている[8)~10)]．本論文が対象とする反復型開発は，調査工程の後，分析，設計，実装，テストの工程を繰り返すことによってソフトウェアを完成させる方法である．

まず調査の工程で，ユーザと開発者は，開発するソフトウェアの構成や仕様の概要を決定する．それに基づいて，1 回目の反復の分析工程で全体の要求仕様を確定する．この工程で主要なユースケースをすべて抽出し，その標準的なイベントフローを作成する．1 回目の分析工程は，要求仕様の全容を把握するため，2 回目以降の分析よりも長い時間をかけて行う．作成されたユースケースに基づき，1 回目の設計・実装の工

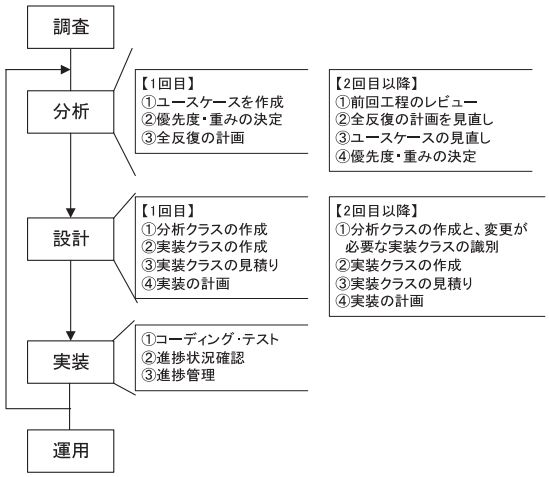

図 3 反復型開発の手順
Fig. 3 Iterative software processes.

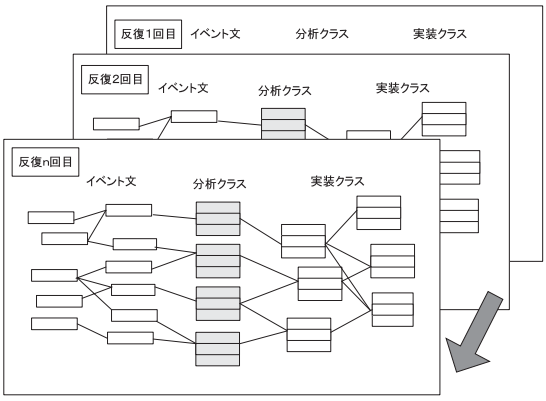

図 4 イベント文と実装クラスの関係
Fig. 4 Relations between events and implementation classes.

程と，それ以降の反復の計画を立てる．この反復型開発は，ユースケース駆動[8] の考えに基づき，ユースケースの優先度や重要度を考慮して，1 回目の反復では重要度の高いものや，リスクの高いものを作成し，2 回目以降は優先度の低いものや代替フローを作成していく．

設計工程で，開発者は，ユースケースから分析クラスとメソッドを作成し，そこから，実装のためのクラスを作成する．その後，実装とテストにかかると思われる工数を見積もる．また実装工程では，開発者は，実装クラスのコーディングとテスト作業の間，一定期間ごとに進捗状況を調査する．

反復型開発の反復開始時には，前回の反復の経験を次の反復に役立てるために，レビューを行う．そこで，実装クラスの見直しや，見積工数の再検討が行われた結果，開発計画の変更（反復の追加・削除など）がおこる場合もある．しかしこの開発は，要求仕様をほとんど固定しない状態で開発を開始し，少しずつ分析，設計，実装を繰り返すことによって仕様を確定していくという方法ではない．最初に全体の要求仕様を大部分見極めて，全開発の反復計画を作成したうえで，各反復においては要求仕様の調整を行いながら，開発を進めていくものである（**図 3**）．

3. 本支援システムの代表的な機能

3.1 ユースケースの関係の設定方法

本支援システムでは，ユースケース間に階層関係や包含関係や拡張関係を設定できる機能を作成した．

階層関係は，ユースケース間の記述の粒度が異なるときに使用し，包含関係は，1 つのユースケースが他のユースケースを使う場合に，拡張関係は，既存のユースケースに追加したいイベントフローがある場合に用いる．

これらの関係を設定すると，本支援システムは，関係元になるイベント文と，関係先のイベント文との間にリンクを張る．たとえば，階層関係では粗い粒度のイベント文が複数の細かい粒度のイベント文に分解されるため，1 対多のリンクが張られる．また包含では，包含元のイベント文から包含先のイベント文が参照されるため 1 対多のリンクが張られる．また，逆に，階層先・包含先のイベント文から見れば複数の階層元・包含元とリンクが張られる．このようにユースケースの関係を設定することによって，ユースケース間のイベント文には多対多のリンクが張られる．本支援システムは，イベント文の間にこれらのリンクを設定し，ユースケースの関係構造を保存する．**図 4** の反復 n 回目の左側のイベント文の間に張られたリンクがこれに該当する．

3.2 分析クラスの抽出方法と実装クラスへの変換

本支援システムは，ユースケースのシーケンス図を解析して，ソフトウェアの設計のためにクラスとメソッドを次の方法で抽出する．まず，シーケンス図の 1 つのオブジェクトは 1 つのクラスと考える．シーケンス図で 1 つのクラスから別のクラスにメッセージが送信されている場合，このメッセージは，送信を受けたクラスのメソッドになる．ただし，クラスからアクタへ送信されているメッセージは，送信したクラスのメソッドとする[11]．この方法で，まず，ユースケースのイベント文が 1 つの分析クラスのメソッドに関係付けられる．図 4 の反復 n 回目のイベント文と分析クラスのメソッド間に張られたリンクがこれに該当する．

表 1 クラスの変換の種類
Table 1 Types of conversion.

	変換の種類
C1	複数のクラスを合体して一つのクラスを作成
C2	一つのクラスから複数のクラスへの分解
C3	一つのクラスから複数のクラスへの部品化
C4	複数クラスからスーパークラスの作成
C5	複数のメソッドの合体
C6	メソッドの移動
C7	一つのメソッドから複数メソッドへの分解
C8	一つのクラスとメソッドから複数のクラスとメソッドへの分解
C9	クラスとメソッドの名前の変更

抽出された分析クラスを，最終的に実装される実装クラスに設計するためには，開発者による変換作業が必要になる．本支援システムには，分析クラスから実装クラスへ変換作業のために，クラスの合体，分解，部品化，スーパクラスの作成，メソッドの合体，分解などができる「変換エディタ」が用意されている（**表 1**）．この変換過程はクラスのメソッド間のリンクとして保存される．図 4 の反復 n 回目の分析クラスと実装クラスのメソッド間に張られたリンクがこれに該当する．

この変換では，表 1 の C7，C8 のように，メソッドの分解が行われた場合は，分析クラスのメソッドが，実装クラスのメソッドに分解されることになる．この場合は，分解されたメソッドが分解される前のメソッドの何割にあたるかという割合が入力される必要がある．このメソッドの分割の割合を構成割合と呼ぶ．

分析クラスから実装クラスへの変換作業の工程は，機能を一定に保ってクラスの構造を改善するという観点からは，リファクタリング[12] の作業としてもとらえることができる．

3.3 反復型開発への適用

反復型開発は，ソフトウェアシステムの機能をよりよくするよう改善しながら開発していく方法と，とらえることができる．つまり，新規の反復では，新しいユースケース（またはイベント文）の追加，既存のユースケース（またはイベント文）の変更または削除が行われる．そのため，新規の反復が始まるときには，以下の作業が必要になる．

(1) ユースケース間の関係の再構築
(2) 追加，変更，削除されたユースケース（またはイベント文）と関係する実装クラスの抽出
(3) 追加，変更，削除されたユースケース（またはイベント文）と関係する実装クラスやメソッドの再見積り

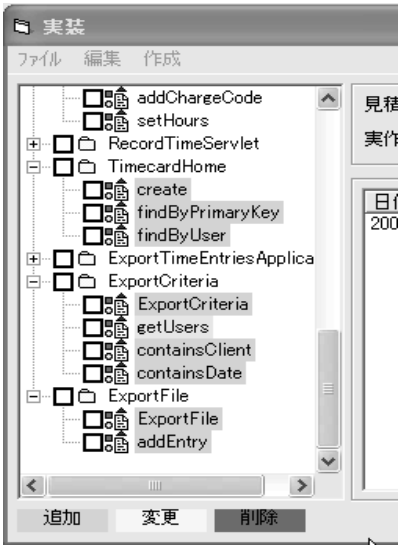

図 5 実装クラスの候補表示
Fig. 5 Candidates for implementation classes.

これらの作業のためには，新規の反復にともなうユースケース（またはイベント文）の追加，変更，削除によって影響をうける実装クラスやメソッドと影響をうけないものとに分別する必要がある．本支援システムは，前回の反復によって，図 4 の反復 1 回分にあたる，イベント文，分析クラス，実装クラスのネットワーク関係を作成している．新規の反復が開始されると，ここに新しいネットワーク関係が 1 回分追加作成される．前回と新規のイベント文を比較することによって，新規の反復で追加・変更・削除がおこると予想される実装クラスを表示することができる．

たとえば，実装クラスやメソッドの追加が必要となる場合は，“追加候補” として表示する．またユースケース（またはイベント文）の変更や削除があった場合は，そこからリンクしているものを “変更候補” や “削除候補” として表示する．画面には，追加・変更・削除の種類が分かるように候補が色分けられて表示される．**図 5** では，TimecardHome のクラスで Create と findByPrimaryKey と findByUser のメソッドが追加候補とされている．この機能によって，2 回目以降の反復において，開発者が既存のクラスやメソッドを修正したり，新しい実装クラスやメソッドを作成したりする作業を支援する．

3.4 進捗度の計算方法と例

イベント文，ユースケース，ユースケースから見た開発全体の進捗度は次の式で定義する．
ユースケース $i(i = 1 \cdots n)$，イベント文 $ij(j =$

$1\cdots m$) 実装クラスのメソッド $ijk(k = 1\cdots l)$ を $Mijk$ とすると

イベント文 ij の進捗度（%）

$$= \sum_{k=1}^{l} \left\{ Mijk \text{ の実装成否} \times \frac{Mijk \text{ の見積工数}}{\sum_{k'=1}^{l} Mijk' \text{ の見積工数}} \right\} \times 100 \quad (1)$$

ユースケース i の進捗度（%）

$$= \sum_{j=1}^{m} \left\{ \text{イベント文 } ij \text{ の進捗度} \times \frac{\text{イベント文 } ij \text{ の重要度}}{\sum_{j'=1}^{m} \text{イベント文 } ij' \text{ の重要度}} \right\} \quad (2)$$

ユースケースから見た開発全体の進捗度（%）

$$= \frac{\sum_{i=1}^{n} \text{ユースケース } i \text{ の進捗度}}{n} \quad (3)$$

式 (1) は，ユースケース i のイベント文 ij を実装するために必要な実装クラスのメソッドが，見積工数（人日）で何%終了したかを測る式である．これをイベント文 ij の進捗度とする．実装成否は，1 と 0（終了か否か）の 2 値のみを使用する．

式 (2) は，ユースケース i を構成しているイベント文 ij の進捗度のイベントごとの重要度付きの総和であり，ユースケース i が何%終了したかを測る式である．これをユースケース i の進捗度とする．ここで重要度とは，ユースケースのイベント文ごとに本支援システムで設定できるもので，1〜10 まで入力し大きい方が重要度は高い．

また，式 (3) のユースケースから見た開発全体の進捗度は，全ユースケースの進捗度の平均値とした．

たとえば 1 回目の反復で，ユースケース 1 が 2 つのイベント文 11，イベント文 12 から作成されるとする．イベント文 11 は 3 つの実装クラスのメソッド M111，M112，M113 からなり，その見積工数は 1 人日と 2 人日と 5 人日とする．ここで，1 つ目の M111 だけが終了しているとすると，それぞれの実装成否は，1，0，0 となる．また，2 つ目のイベント文 12 は 1 つの実装クラスのメソッド M121 で実装され，この見積工数は 4 人日で，実装は終了しているとする．この場合の各イベント文の進捗度とユースケース 1 の進捗度は以下の式で求まる（ただしこの例では，重要度はすべて 1 と設定している）．

イベント文 11 の進捗度（%）

$$= \left(1 \times \frac{1}{1+2+5} + 0 \times \frac{2}{1+2+5} + 0 \times \frac{5}{1+2+5}\right) \times 100 = 12.5$$

イベント文 12 の進捗度（%）

$$= \left(1 \times \frac{4}{4}\right) \times 100 = 100.0$$

ユースケース 1 の進捗度（%）

$$= 12.5 \times \frac{1}{2} + 100.0 \times \frac{1}{2} = 56.25$$

さらに，イベント文 11 とイベント文 12 が作成され 1 回目の反復が終了した後，2 回目の反復が行われたとする．2 回目の反復では，イベント文 11 の実装クラスのメソッド M113 の修正が必要になり，イベント文 12 はそのまま使い，新しいイベント文 13 は追加作成が必要になったとする．イベント文 13 は 1 つの実装クラスのメソッド M131 から作成される．M113 の修正の見積工数は 3 人日，M131 の追加作成の見積工数は 5 人日とする．2 回目以降の反復では，開発者は反復回数などを考慮した見積工数を入力する．この場合，2 回目の反復が始まる前のイベント文の進捗度とユースケース 1 の進捗度は以下の式で求まる（ただし，1 回目の反復で終了しているメソッドについては，見積工数の代わりに実際に作業した工数を使用する．今回は見積工数と実際に作業した工数は同じとする）．

イベント文 11 の進捗度（%）

$$= \left(1 \times \frac{1}{1+2+3} + 1 \times \frac{2}{1+2+3} + 0 \times \frac{3}{1+2+3}\right) \times 100 = 50.0$$

イベント文 12 の進捗度（%）

$$= \left(1 \times \frac{4}{4}\right) \times 100 = 100.0$$

イベント文 13 の進捗度（%）

$$= \left(0 \times \frac{5}{5}\right) \times 100 = 0.0$$

ユースケース 1 の進捗度（%）

$$= 50.0 \times \frac{1}{3} + 100.0 \times \frac{1}{3} + 0.0 \times \frac{1}{3} = 50.0$$

上例の 1 回目の反復の終了時点では，その反復の開始時に設定した開発目標であるイベント文 11 と 12 をすべて実装完了している（M111，M112，M113，M121 の実装）ので，「ユースケースから見た進捗度」も「実装クラスから見た進捗度」も 100%となっている．2 回目の反復の開始時には，あらためて開発目標として，1 回目の反復ですでに実装済みのイベント文 11 と

12 に加えて，イベント文 13 を追加している．そのため，2 回目の反復では，1 回目の反復で完了した実装の多く（M111，M112，M121）を無修正で使えるが，一部（M113）の修正と新規の実装（M131）が必要となっている．2 回目の反復では，その開始時に再設定した開発目標全体を 100%と見なすが，開始時点ですでにそのうちの 50%に相当する部分（M111，M112，M121）については，1 回目の反復での実装結果が無修正で使えるため，作業が完了していることとなる．

3.5 支援システムの構成

本支援システムの開発は，1 つのプロジェクト（開発全体）で反復を繰り返しながら，各反復でユースケースの実装を行っていくという方法である．このため **図 6**（実線は制御とデータの流れ，破線は単なるデータの流れを表す）のように,「プロジェクトの管理」が「反復の管理」を,「反復の管理」が「ユースケースの管理」をコントロールすることになる．ユーザは,「プロジェクトの管理」でプロジェクト情報（プロジェクト名，開始日，終了予定日）を,「反復の管理」で反復情報（反復名，開始日，終了予定日）を,「ユースケースの管理」でユースケースファイルとユースケース情報（ユースケース名，ユースケース間の関係）を入力する．ユースケースファイルは，開発者とユーザが Microsoft Visio の UML 関係のステンシルで描画し，その Visio ファイルを用いる．

入力されたユースケースファイルは「ユースケースの解析」によってイベント文に分解される．また,「分析クラスの抽出」でイベント文から分析クラスを抽出する．開発者は「変換エディタ」を使用して，分析クラスを実装クラスに変換する作業を行う．実装クラスを作成していく過程は，実装クラス間のリンクとして保存され，開発者が入力する実装クラス情報（実装クラス名，メソッド名，構成割合）とともに「実装クラスの管理」が管理する．また開発者は,「見積工数の管理」に実装クラスのメソッドの見積工数を入力し,「進捗状況の管理」にメソッドの進捗情報（実装成否，開始日，終了日）を入力する．これらの情報から「進捗計算」で作成された結果を使って,「進捗状況出力」がユーザと開発者それぞれにユースケースと実装クラスを基にした進捗レポートを作成する．

本支援システムでは,「ユースケースの解析」「分析クラスの抽出」「変換エディタ」は文献 4) のものを使い，その他の部分は，新規作成または大幅に修正を行った．

3.6 支援システムの画面

図 7 の画面は，図 6 の「ユースケースの管理」においてユースケース間に包含関係を設定する画面である．包含関係を設定するためには，まず包含元のユースケースのイベント文を選択する．ここでは,「ログインする」ユースケースの「従業員は，ログイン画面を表示する」というイベント文が選択されている．次に，メニューの「ユースケース（U）」から「ユースケースの包含」を選ぶと，包含先のユースケースを選択する画面が表示されるので，そこで包含先を決定する．イベント文に包含関係が設定されている場合は，包含元のユースケースのイベント文をクリックすると，包含

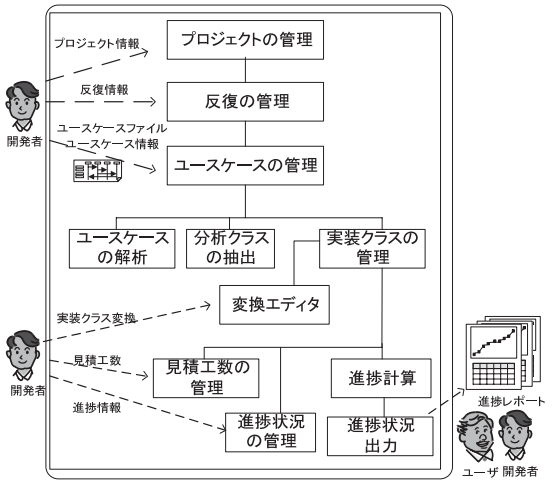

図 6 システムの構造
Fig. 6 System structure.

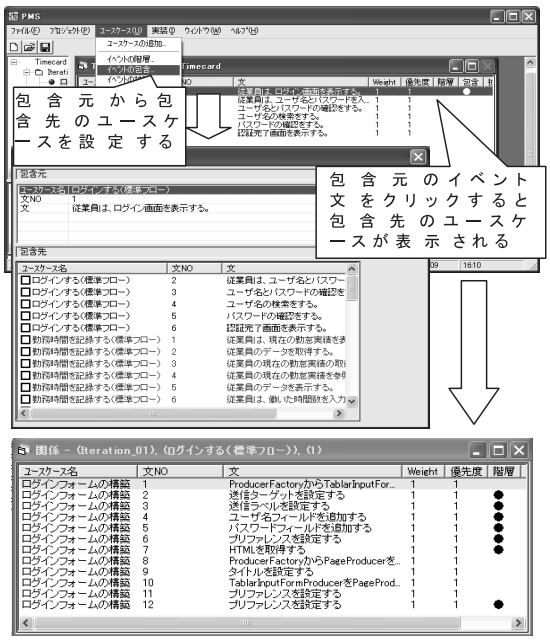

図 7 包含関係の設定と表示画面
Fig. 7 Typical screens with include relationships.

先のユースケースのイベント文が表示される．

4. 評　　価

4.1 ソフトウェア開発のシミュレーション

本支援システムを実際のソフトウェア開発の一例として文献 13) にある例題を適用し，反復的に開発する過程をシミュレーションした．使用した例題は，従業員が勤務した時間を担当するプロジェクトごとに記録し，請求システムに勤務時間を出力するタイムカードアプリケーションシステムである．この例題は，ユースケースの作成と分析クラスから実装クラスへの変換の過程と，最終のソースコードが提供されている．このソフトウェア開発を，本支援システムを使用しながらシミュレーションしてみた．

ユースケースとしては,「ログインする」「勤務時間を記録する」「勤怠実績を外部出力する」の基本的なものが 3 つある.「ログインする」には標準フローが 1 つと代替フローが 2 つ,「勤務時間を記録する」には標準フローが 1 つと代替フローが 1 つ,「勤怠実績を外部出力する」には標準フローが 1 つ含まれている．また，Web ページ作成用のユースケースが別に 4 つ作成されている.「ログインする」「勤務時間を記録する」のユースケースは，これら 4 つのユースケースとの間に階層や包含の関係がある．

今回は 2 回の反復でこのシステムを開発すると計画した．初めの反復では,「ログインする」「勤務時間を記録する」の標準フローを作成する．2 回目の反復では,「ログインする」「勤務時間を記録する」の代替フローと「勤怠実績を外部出力する」の標準フローを作成する．1 回目と 2 回目の反復は，それぞれ 2 人で 23 日と 9 日で開発するとして計画を立てた．各メソッドの見積工数は，すでに作成されているソースのステップ数を用い，そのステップ数に比例する数値とした．重要度は，すべて 1 を設定した．

4.2 ユースケースの進捗度と実装クラスの進捗度

表 2 と **図 8** は，1 回目の反復のコーディングにおける実装クラスのメソッドの進捗度（以降は，実装クラスの進捗度という）とユースケースから見た開発全体の進捗度（以降は，ユースケースの進捗度という）の表とそのグラフである．実装クラスの進捗度とユースケースの進捗度はほぼ同じ傾向で推移しているが，日程の全般において実装クラスの進捗度がユースケースの進捗度より早く完成に近づいている．特に，5 月 19 日は，実装クラスが大きく進んだがユースケースの進捗度はそれほど進んでいない．この日に注目して調べると，実装クラスのメソッドの見積工数の大きなものが終了したが，それによって完成するイベント文は少なかった．これが，実装クラスとユースケースの進捗度の差を広げたと考えられる．

今回の実験結果のように，実装クラスの進捗度がユースケースの進捗度より大きい状況が続くと，ユースケースから見た進捗は遅いにもかかわらず，開発者は，進捗は遅くないと誤解する可能性がある．たとえば，図 8 においては，5 月 19 日に実装クラスの進捗度とユースケースの進捗度の差が大きくなり，その差が 5 月 25 日まで累積され，5 月 25 日には，実装クラスの進捗度はユースケースの進捗度より 17%程度大きくなっている．この差が生じた理由としては，5 月 19 日〜25 日の開発期間において，1 つのイベント文が複数の実装クラスに関連し，実装クラスの実装を進めても，イベント文（ユースケース）レベルの進捗はなかなか進まないという状況が発生しているためである．

表 2 1 回目のコーディングの進捗度
Table 2 Progress of coding in the first iteration.

日付	進捗度（%）			1 日あたりの進捗度（%）	
	実装クラス（C）	ユースケース（U）	（C）-（U）	実装クラス	ユースケース
04/5/10	2.17	0.00	2.17	2.17	0.00
04/5/11	7.61	2.78	4.83	5.43	2.78
04/5/12	13.04	5.56	7.48	5.43	2.78
04/5/13	17.39	10.38	7.01	4.35	4.83
04/5/14	21.74	14.32	7.42	4.35	3.94
04/5/17	23.91	16.17	7.74	2.17	1.85
04/5/18	26.09	19.15	6.94	2.17	2.98
04/5/19	34.78	23.08	11.70	8.70	3.94
04/5/20	39.13	25.40	13.73	4.35	2.31
04/5/21	41.30	25.40	15.90	2.17	0.00
04/5/24	43.48	27.48	16.00	2.17	2.08
04/5/25	52.17	34.94	17.23	8.70	7.46
04/5/26	56.52	40.27	16.25	4.35	5.32
04/5/27	60.87	45.36	15.51	4.35	5.09
04/5/28	65.22	49.99	15.23	4.35	4.63
04/5/31	69.57	55.57	14.00	4.35	5.58
04/6/1	73.91	62.28	11.63	4.35	6.71
04/6/2	78.26	68.96	9.30	4.35	6.68
04/6/3	82.61	73.13	9.48	4.35	4.17
04/6/4	86.96	80.59	6.37	4.35	7.46
04/6/7	91.30	87.10	4.20	4.35	6.51
04/6/8	95.65	92.16	3.49	4.35	5.06
04/6/9	100.00	100.00	0.00	4.35	7.84

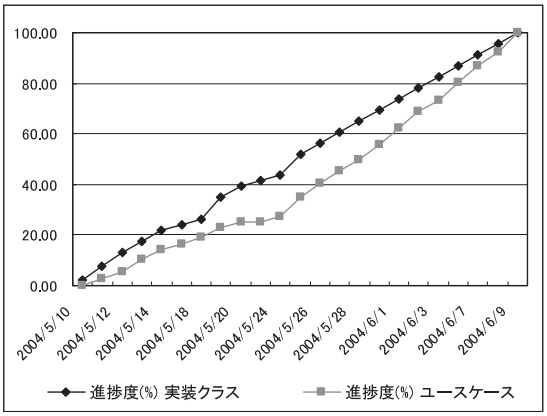

図 8 1 回目のコーディングの進捗度
Fig. 8 Progress of coding in the first iteration.

4.3 反復型開発の進捗レポート

本支援システムは，ユースケース，反復（イテレーション），プロジェクトごとに進捗レポートを作成する．**図 9** 上のユースケースレベルでは，ユースケースの進捗度と，イベント文と重要度（Weight），優先度，コーディングの進捗度などを表示する．図 9 中の反復レベル（イテレーションレベル）では，反復の進捗度と，ユースケース名とそれぞれの進捗度を表示する．図 9 下のプロジェクトレベルでは，反復ごとの進捗度を表示する．それぞれの画面の下部にはユースケース，反復（イテレーション），プロジェクトの進捗度がグラフで表示される．グラフの縦軸はユースケースから見た進捗度（%）で，横軸は実装成否が入力された日付である．

プロジェクトのグラフでは，前回の反復の後に続いて今回の反復の進捗状況を表示している．図 9 では，1 回目の反復の作業が 100%終了した後，次の 2 回目の反復の作業が 80%から開始している．2 回目で完成させなければいけないユースケースの全体を 100%とした場合，1 回目の反復によって 80%の部分がすでに作成されていることを意味している．1 回目で終了した作業全体が 2 回目の反復作業において占める割合を示していると考えることもできる．また，反復ごとの進捗を続けて表示しているため，各回の反復の開発状況を直感的に比較することができる．

一方，見積工数が誤っている場合やユースケースと実装クラスのリンクが正しく設定されていない場合は，実装クラスの進捗度がユースケースの進捗度に正しく反映されない．本支援システムは，見積工数と構成割合と実装クラスのメソッド数による 3 通りの方法で進捗度を計算している．見積工数に誤差が見られるときには，実現にかかわる機能の割合としての構成割合や，イベント文に関係する実装クラスのメソッドの数で計算した進捗度を使用することができる．

4.4 支援システムの考察

今回は，実際のプロジェクトで本支援システムを使用することはできなかったため，ユーザに本支援システムのデモンストレーションを行い，評価を聞いた．ユーザは 4 人で，ソフトウェアの作成方法についてはほとんど知らない人が 2 人，ソフトウェア開発の経験はあるが現在はユーザ側になった人が 2 人である．評価の高かったコメントとしては，「ユーザに進捗を見せようとする考えが良い」「過程がグラフになっているのが直感的で良い」「開発途中でソフトウェアができているのかどうかを知るためのツールとして使える」というものがあった．一方「情報が多すぎる」「最終的にできあがるのかどうかが分かる機能がほしい」「開発者側としてはここまで見せたくないのではないか」というコメントも出されたが，それらは本システムの有用性を根本的に否定するものではなく，基本的な有用性は認めたうえで，より改善すべき点として指摘されたものであった．以上のことから，開発の進捗状況をユースケースでユーザに分かりやすい形で見せると

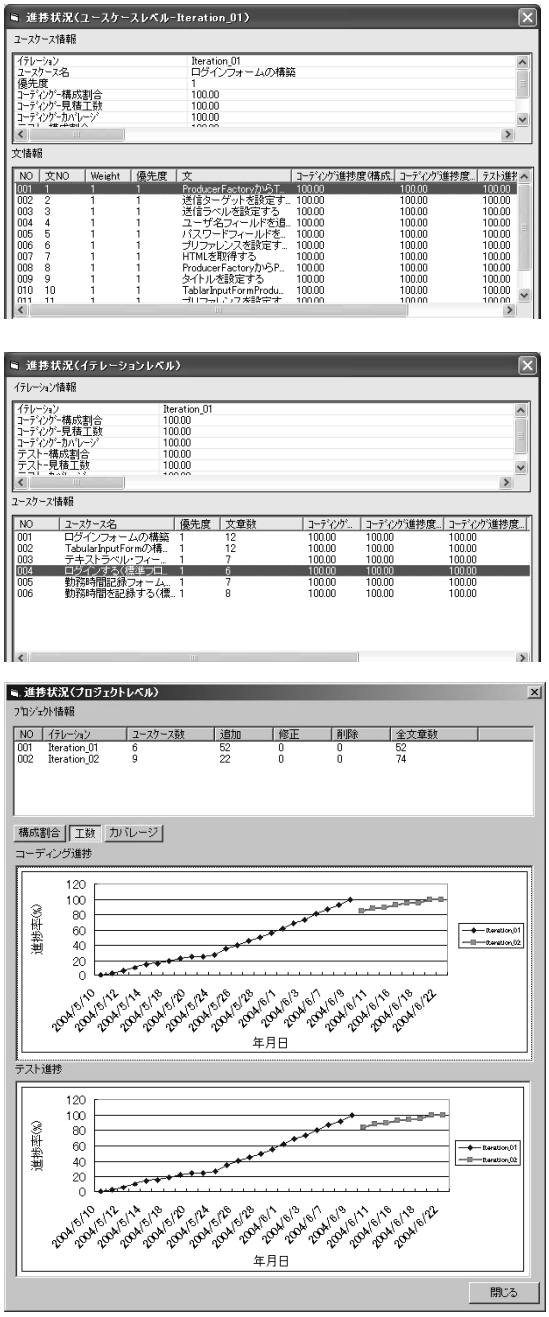

図 9 進捗レポート画面
Fig. 9 Typical screens of the progress reports.

いう本支援システムのコンセプトは，ユーザには評価されたと考えている．

また，今回は簡単な例題で評価を行ったが，現実の事例では，もっと複雑性が増すと考えられる．複雑性の問題としては，ソフトウェア開発の規模が大きくなる問題と，ソフトウェアの機能と構造の関係が悪くなる問題が考えられる．規模が大きくなる問題では，ユースケースが多数作成され，ユースケース間の関係が多く設定されることになる．単にユースケースの数が増える場合やユースケース間の関係が多くなる場合は，本支援システムが，ユースケースと実装クラスのリンクを維持する機能を基本としているため，対応可能である．また反復型開発においても，反復の数が開発途中で追加されたり，削除されたりする場合にも対応できる．一方，機能と構造の関係が悪くなる問題には，設計工程で機能に対して構造が与えにくいことから，ユースケースが正確に進捗度を表せない可能性は増す．また，反復型開発においても，最初の反復において機能がうまく構造に切り分けられないと，次の反復でも構造に切り分けられない部分が増大するという問題も生じる．両方の問題があいまってよけい複雑になり，本支援システムの信頼性が下がることが予想される．こうした問題に対しての対応は今後の課題である．

また一方，開発者にとっては，顧客の立場であるユーザが進捗管理に関与することによって，ユーザ側からの管理が強くなるのではないか，仕様追加や変更が頻繁におこるのではないかとかいう不安も生じる．提案手法は，進捗状況の共有化をポジティブなフィードバックとして使うことを目的にしており，ユーザ側からの管理を強化することをねらうものではない．むしろ提案手法によって，ソフトウェア開発の難しさを理解してもらい，無理な仕様追加や変更の要求を防ぐことを目指している．

5. おわりに

近年のソフトウェア開発においては，開発者だけでなくユーザも開発作業の進捗を把握することが求められていると考え，ユーザが機能を通して進捗を把握することができる手法を提案した．また反復型開発にも適用させて，そこで使用する支援システムを作成した．これを，ソフトウェア開発の例にあてはめて，反復型開発をシミュレーションすることによって検討を行った結果，ユースケースと実装クラスの進捗度に差が発生することが分かった．また，ユーザに本支援システムのデモンストレーションを行い評価を聞いた結果，本支援システムの有用性を確認できた．

今後の課題としては，このシステムをユーザと開発者間のコミュニケーションツールとして使用できるように改善を加えることである．たとえば，ソフトウェア開発がコストオーバや時間超過などの危機に陥り，作業の再見積りや，機能の切り分けが必要になったときに，ユーザと開発者が機能と構造を通して問題点や作業量の認識を共有できるツールが必要になる．また，大規模ソフトウェア開発においては，ユースケースの進捗度と実装クラスの進捗度の乖離が大きくなると予想され，本ツールの有用性が発揮できると考えられる．このような状況のもとで利用するための機能を追加していくことも課題である．

謝辞 本研究を進めるにあたっては，故永田守男先生に多岐にわたりご指導をいただきました．ここに，衷心より感謝の意を表します．

参考文献

1) McConnell, S.: *Software Project Survival Guide*, Microsoft Press (1998). アルテア・ジャパン（訳）：ソフトウェアプロジェクトサバイバルガイドマイクロソフト公式解説書，日経 BP ソフトプレス．
2) 日経コンピュータ（編）：動かないコンピューター情報システムにみる失敗の研究，日経 BP (2002).
3) Project Management Institute: *A Guide to the Project Management Body of Knowledge 2000 Edition*, Project Management Institute (2000).
4) 吉田享子，永田守男：ユースケースを利用したユーザのためのプロジェクトマネジメントシステム，電気学会誌，Vol.123, No.4, pp.707–713 (2003).
5) Jacobson, I., Booch, G. and Rumbaugh, J.: *The Unified Software Development Process*, Addison Wesley Longman (1999). 藤井 拓（監訳）：UML による統一ソフトウェア開発プロセス，翔泳社．
6) 能澤 徹：国際標準プロジェクトマネジメント（PMBOK と EVMS），日科技連 (1999).
7) Booch, G.: The Unified Modeling Language User Guide, Addison Wesley Longman (1999). オージス総研オブジェクト技術ソリューション事業部（訳）：UML ユーザガイド，ピアソン・エデュケーション．
8) Kruchten, P.: *The Rational Unified Process: An Introduction*, Addison Wesley Longman (1999). 藤井 拓（監訳）：ラショナル統一プロセス入門，ピアソン・エデュケーション．
9) Beck, K.: *Extreme Programming Explained: Embrace Change*, Addison-Wesley (1999). 長瀬（監訳）：XP エクストリーム・プログラミング入門—ソフトウェア開発の究極の手法，ピアソ

ン・エデュケーション.
10) Palmer, S.R., Palmer, S. and Felsing, J.M.: *A Practical Guide to Feature-Driven Development* (*Coad Series*), Prentice Hall (2002). 今野ほか（訳）：アジャイル開発手法 FDD—ユーザ機能駆動によるアジャイル開発，ピアソン・エデュケーション.
11) Larman, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall (1998). 依田光江（訳）：実践 UML—パターンによるオブジェクト指向開発ガイド，プレンティスホール.
12) Fowler, M.: *Refactoring: Improving The Design of Existing Code*, Addison Wesley Longman (1999). 児玉ほか（訳）：リファクタリング，ピアソン・エデュケーション.
13) Arrington, C.T.: *Enterprise Java with UML*, John Wiley & Sons (2001). ウルシステムズ株式会社（訳）：UML によるエンタープライズ Java 開発，翔泳社.

(平成 16 年 6 月 21 日受付)
(平成 17 年 1 月 7 日採録)

吉田　享子（正会員）

1977 年慶應義塾大学大学院工学研究科修士課程修了.（株）日本科学技術研修所,（株）アーク情報システムを経て，平成国際大学法学部助教授．現在，慶應義塾大学大学院理工学研究科後期博士課程在学中．電子情報通信学会，日本ソフトウェア科学会，プロジェクトマネジメント学会各会員.

飯島　　正（正会員）

1991 年慶應義塾大学大学院理工学研究科博士課程単位取得退学.（株）東芝勤務を経て，慶應義塾大学理工学部助手．現在に至る．著書（共著）に『分散オブジェクトコンピューティング』,『エージェント技術』（共立出版，1999 年）等．訳書（分担訳）に『コンピュータのための数学—論理的アプローチ』（日本評論社，2001 年）等がある.

櫻井　彰人（正会員）

1975 年東京大学工学部計数工学科卒業，77 年同大学院情報工学研究科修了．77 年日立製作所入社那珂工場配属．78 年 Dept. of Computer Science, Univ. of Illinois （MsCS 取得，1979）．89 年同基礎研究所，96 年同中央研究所，98 年北陸先端科学技術大学院大学教授を経て，2002 年慶應義塾大学理工学部教授となる．博士（工学）（東京大学 1993）．人工神経回路網，機械学習を専門とする．1987 年ソフトウェア科学会高橋奨励賞，94 年，98 年人工知能学会研究奨励賞を受賞．ACM，INNS，AAAI，人工知能学会，日本ソフトウェア科学会，応用数理学会，電気学会各会員.

山口　高平（正会員）

1957 年生．1979 年大阪大学工学部通信工学科卒業，84 年同大学院工学研究科博士後期課程修了．同年大阪大学産業科学研究所助手．89 年静岡大学工学部助教授．97 年同大学情報学部教授．2004 年より慶應義塾大学理工学部教授．工学博士．知識システム，知識発見，セマンティック Web，オントロジ，知能ソフトウェア工学に関する研究に従事．1991 年，1998 年度人工知能学会研究会奨励賞．1992 年度人工知能学会全国大会優秀論文賞．2002 年度人工知能学会研究会優秀賞．人工知能学会，電子情報通信学会，日本認知科学会，知能情報ファジィ学会各会員.