論　文

オブジェクトモデルを用いたソフトウェアプロセスの表現方法

非会員　松　下　　誠　（ 大阪大学 ）
非会員　飯　田　　元　（奈良先端大）
非会員　井　上　克　郎　（ 大阪大学 ）

An Expression of Software Process with Object Model

Matsushita Makoto, Non-member (Osaka University), Iida Hajimu, Non-member (Nara Institute of Science and Technology), Inoue Katsuro, Non-member (Osaka University)

Most of process-centered software engineering environments and those languages focus on process-oriented software development process. However, recent software development tends to require to focus on product-oriented software development, because of emergence of various types of software developments; e.g., software reuse, component-based composition, and so on. In this paper, we propose a new process modeling method named "MonoProcess" to support the new software development approach with the idea of software process. MonoProcess consists of a set of objects which represent artifacts and resources in the software development. An object has attributes and methods, which represent characteristics and operations of the object, respectively. MonoProcess illustrates software development environment as it is, and provides a framework for software process description and software process management.

キーワード：ソフトウェアプロセス、ソフトウェアプロセス記述、オブジェクトモデル

1. はじめに

ソフトウェア開発作業を効率よく行い、かつ、高品質のソフトウェアを作成するために、ソフトウェアの開発プロセス[5]をあらかじめ記述し、それに基づいた開発作業を行うようになってきている[9][17]。ソフトウェアプロセスを表現するために、ソフトウェアプロセスのモデル化に関する研究が広く行われている。しかし、既存の多くのプロセスモデルは「ソフトウェアを作る際の作業手順」に着目した物であり[2][10][11][18]、それらに基づいたプロセス中心型開発環境もまた、開発作業に着目した物となっている[1][6][16]。

しかし、オブジェクト指向プログラミングやソフトウェアの再利用、ネットワークを活用したプログラム部品の開発と、プログラム部品の組み合せによるソフトウェア開発等、近年さかんに行われている開発形態は「ソフトウェア開発の際に作られるべき生成物」に着目した開発作業となっている。このような開発形態は従来とは大きく異なっているため、従来のソフトウェアプロセスモデルや、そのようなモデルに基づいたプロセス中心型開発環境は、そのまま適用できない可能性が高い。

このような開発形態におけるソフトウェアプロセスを考える際には、開発ごとに作成される各種のプロダクトをその要素の中心にする必要があることは容易に想像できる。例えば、PCTE[19]に代表されるプロダクト指向のCASE 環境を用いて、開発環境それ自身はもちろん、その環境で行われるソフトウェアプロセスを記述し、その実行方法、実行状況の把握等を行う開発環境を構築する方法が考えられる。しかし、このような開発環境は従来行ってきた開発環境を置きかえる物であり、導入時に大きな環境の変化を伴ない、かつ、利用者に対してある種の定型作業を強いるものとなってしまうと考えられる。

我々は、このような近年の開発形態に即した開発環境の構築に関する研究を行っている[12]。我々の提案する環境は、開発時に生成されるプロダクトや、開発環境中のさまざまな資源を構成要素とするソフトウェアプロセスモデルを用いることによって、プロセスの記述、プロセスの実行、実行された作業の計測と、計測したデータに基づく進捗状況の把握[14][15]を行うことを目的としている。

本敲では、我々が提案するオブジェクト中心型ソフトウェアプロセスのモデル化手法である MonoProcess について述べる。MonoProcess では上記で述べた目的を達成するために、開発環境中のさまざまな要素を構成要素として開発環境全体を表現することができる。また、記述されたオブジェクトを用いて、ソフトウェアプロセスを表現し、かつ管理するための枠組を提供する。

```
Object .Doc.Design def
  Attribute @Owner .Person.Matsushita;
  Attribute @Type "Design Document";
  Attribute @Filename "design.doc";
  Attribute @Input (.Doc.Specification
                    .Doc.Schedule);
  Attribute @Localtion .ShareDisk.Document
  Method &Edit def
    $editor = .caller&GetEditor(@Type);
    if ($editor) {
      &View;
      foreach $input (@Input) {
        $input.&View;
      }
      invoke($editor,
             "@Localtion/@Filename");
    }
  endMethod
  Method &View def
    $viewer = .caller&GetViewer();
    if ($viewer) {
      invoke($viewer,
             "@Localtion/@Filename");
    }
  endMethod
endObject
```

図 1　Object 記述例
Fig. 1.　Example of object description

2. MonoProcess

本節では、我々の提案するソフトウェアプロセスのモデル化手法について説明する。最初に、MonoProcess の定義について述べ、MonoProcess によってソフトウェアプロセスがどう表現されるか説明する。また、MonoProcess の持つ機能についても紹介する。

〈2•1〉　概要　MonoProcess は、開発環境中に存在する生成物や資源を表現する「オブジェクト」を単位としたモデル化を行う。オブジェクトは属性とメソドから構成され、属性によってオブジェクトの持つ特性を、メソドによってオブジェクトに適用する操作を定義する。MonoProcess は任意のオブジェクトを集合化する機能を提供しており、これによって複数のオブジェクトを 1 つのオブジェクトとして扱える。オブジェクトの継承機構を用いることにより、複数のオブジェクト間で情報の共有を行える。また、属性の参照等のオブジェクトへのメッセージはオブジェクトの操作履歴として自動的に保存される。これらの履歴は作業状況の把握等を行う際に利用される。

〈2•2〉 オブジェクトの定義　オブジェクトはソフトウェア開発環境におけるさまざまな生成物や資源を表現する。図 1 は MonoProcess オブジェクトの記述例である。この記述は、`.Doc.Design` と名付けられたあるデザインドキュメントを記述している。

オブジェクトの名前（ラベル）は “.” （ドット）に続く 1 つの単語で始まり、必要に応じて複数の「ドット+単語」を繋げることによって構成される。オブジェクトは属性とメソドから構成される。属性はオブジェクトの特性を表現し、メソドはオブジェクトに適用される関数である。

属性にはどのような種類かを示すためにラベルが一意に付けられている。図 1 では 5 つの属性が定義されており、`@Owner`, `@Type`, `@Filename`, `@Input` そして `@Location` がそれぞれの属性に付けられている属性ラベルである。属性値としては次のような種類がある。

- 整数/実数 （例: 1, -3.5）
- 文字列 （例: “ABC”, “あいうえお”）
- オブジェクトラベル （例 .A, .X.Y.Z）
- 上記 3 つの型を要素として持つリスト

例えば、図 1 で定義されている属性 `@Owner` とそれに対する属性値 (`.Person.Matsushita`) は「このオブジェクトの管理者が `.Person.Matsushita` というラベル名で表現されるオブジェクトである」ことを意味する。

属性と同様、メソドに対しても、その操作内容を示すためにラベルが一意に付けられている。図 1 では 2 つのメソドが定義されており、`&Edit` や `&View` がメソドに付けられたメソドラベルであり、それぞれ、編集と閲覧の作業を表わしている。

メソドによって表現される作業の内容は、オブジェクトの集合内で閉じた演算であるメソド関数として記述される。メソド関数によって行われる操作には、次のものがある。

- オブジェクトの保持する情報への操作: 属性値の参照、属性値の変更、属性/メソド一覧の入手、メソドの実行。
- その他のオブジェクトに関する操作: 新規オブジェクトの生成、外部ツールの起動。
- 通常のプログラミング言語に見られる操作: 整数/実数の演算、文字列演算、集合演算。

また、MonoProcess ではメソド関数が持つべき機能のみを定義しており、その文法等は定義しないため、複数のメソド記述言語を用いることができる。なお、オブジェクトの記述を解釈する実行系は、これらのメソド記述言語を解釈するためのアプリケーションインターフェイスを持つ。新たなメソド記述言語を追加する場合には、それを解釈するモジュールを利用者が追加することによって行なう。図 1 では、現在試作中である Perl 風の簡単な記述言語を解釈するモジュールを用いてメソド内の記述を行っている。

〈2•3〉 オブジェクトを用いたソフトウェアプロセスの表現　部分オブジェクト Op を、あるオブジェクト O に対して定義する。Op は O の持つラベルを接頭語として持ち、O の持つ属性やメソドの部分集合を持つものとする。Op は対象となるオブジェクト O の部分情報を持っており、ある観点において興味のある特徴を抽出したも

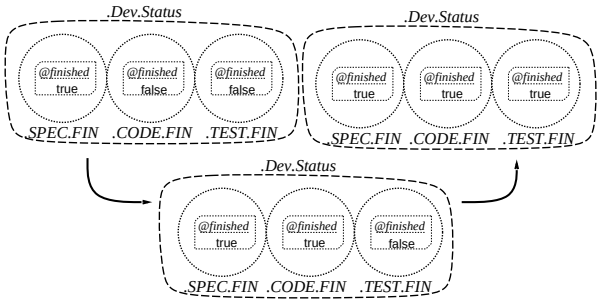


図 2　状態オブジェクトの遷移
Fig. 2.　State transition of status object

のである。

このようにして定義された部分オブジェクトを用いて、状態オブジェクト Os を定義する。Os は、ソフトウェア開発環境におけるある**状態**を表現し、部分オブジェクトの集合とする。我々は、開発環境におけるある状態は、開発環境中に存在する生成物や資源によって表現できると考えている。MonoProcess では、ソフトウェア開発プロセスをこれら状態オブジェクトの遷移系列として定義する。

簡単な例を用いて部分オブジェクトと状態オブジェクトを説明する。まず、.SPEC, .CODE, .TEST という 3 つのオブジェクト（それぞれ、仕様書、ソースコード、テスト結果、を表わす）を仮定する。これらの各オブジェクトは同一の属性ラベル @FINISHED を持ち、この属性によってそれぞれの生成物に対する作業が完了しているかどうかを表現している。

まず、.SPEC, .CODE, .TEST それぞれのオブジェクトから@FINISHED だけを抜きだした物を、それぞれ部分オブジェクトとして定義する。定義した部分オブジェクトをそれぞれ .SPEC.FIN, .CODE.FIN, .TEST.FIN とする。そして、これらのオブジェクトで表現される開発の進行状況を示す状態オブジェクト .Dev.Status を、上記 3 つの部分オブジェクトの集合として定義する。

この時、作業が進行するに従って .Dev.Status は次のような状態の変化を行う（図 2）。

（1） まず、開発開始時には.SAMPLEOBJ の持つ 3 つの状態オブジェクトが持つ @FINISHED 属性が全て “false” となっている。
（2） 仕様書の作成が終了した時点で、.SPEC.FIN の持つ属性 @FINISHED が “true” へ変化する。
（3） コーディングの作業が終了すると、.CODE.FIN の持つ属性 @FINISHED が “true” へと変化する。つまり、.TEST.FIN だけが “false” の状態である。
（4） 作業が終了すると、全てが “true” に変化した状態となる。

上記のような.Dev.Status の状態遷移は、開発の進行状況と直接対応するものとなっている。MonoProcess では、必要に応じて各状況に対応した状態オブジェクトを定義し、その遷移系列を定義することによってソフトウェアプロセスを表現することができる。

〈2•4〉 MonoProcess の機能　MonoProcess は、先に述べた開発環境の構成要素のオブジェクトによるモデル化とそれを用いたソフトウェアプロセスの表現の他に、プロジェクト管理、ソフトウェア開発、開発者の協調作業を支援するためにさまざまな付加機能を提供する。以下、MonoProcess の提供する付加機能について説明する。

オブジェクトの参照範囲とアクセス制御　一般的には、全ての各オブジェクトは他のオブジェクトから参照され、また他のオブジェクトを参照することができる。しかし、各オブジェクトはそれ自身の他からの参照範囲を自分自身の属性として定義することができる。実際には、この参照範囲は他のオブジェクトからのアクセス制御として定義している。従って、以下ではオブジェクトのアクセス制御についてのみ述べる。

アクセス制御は、オブジェクトが含まれるグループの指定と、そのグループに対する制御内容として示される。

- グループによる範囲の指定グループはグループの名前によって特定されるオブジェクトの集合である。各オブジェクトはあらかじめ決められたグループに所属できる。所属したグループの名前は、@GROUP 属性にて表わされる。@GROUP オブジェクトを持っていないか、持っていても値が定義されていないオブジェクトはどのグループにも属していないものとする。
- 範囲/範囲外から行える操作の指定@ACCESS 属性によって、オブジェクトに対して行える/行えない操作を定義する。操作には 4 つの種類があり、それぞれ「属性値/メソドの参照」「属性値/メソド関数の変更」「メソドの実行」「継承の許可」である。この 4 種類の操作について、「同一のグループに所属しているオブジェクトから」および「同じグループに所属していないオブジェクトから」行われた際に許可するかどうかを @ACCESS に保存する。これにより、参照範囲の制限は属性値/メソドの参照の可否によって行われる。

MonoProcess が一般的なオブジェクト指向の考え方に従っているならば、これらの機能はクラス構造やクラス間の関連付け等によって実現されるであろう。しかし我々は、時間の経過と共にその構造が変化し続けることが前提となるソフトウェアプロセスのモデル化にそのようなオブジェクト指向の方法をそのまま適用するのは困難ではないかと考えている。MonoProcess では、ソフトウェア開発環境中の要素をオブジェクトとして定義することができる。このため、開発環境の変化に対応して、自由にその構成を変更することが可能である。また、オブジェクトは単純な構造を用いており、より複雑な要素についても複数のオブジェクト等の組みあわせとして表現することが可能となっている。これにより、MonoProcess を

用いることにより、複雑でかつ多様な形態を取る実際の開発環境をより正確に記述できると考えられる。

オブジェクトへの操作履歴　属性値の参照等、オブジェクトに対する操作は、対象となるオブジェクトに対してメッセージを送り必要な処理を要求することによって行われる。MonoProcess では、全てのオブジェクトに対する操作は、履歴を保持する属性に記録される。一般に全ての操作に対する履歴の収集は非常に膨大かつ頻繁に行われる物となるが、MonoProcess では、各オブジェクトの粒度がある程度大きい単位 (ファイル等) であり、オブジェクトの持つ属性、メソドを単位とした履歴の収集を行うため、このような履歴の収集は現実的であると考えている。履歴の収集は、オブジェクトに対する変更操作や、ユーザ側で動作する履歴収集等を目的としたエージェント等によって行われる。

属性およびメソドに対する操作の履歴として、以下のような情報が記録される。

- 属性に対する操作操作したオブジェクト、操作した時間、操作の内容。
- メソドに対する操作操作を行ったオブジェクト、操作の開始時間、操作の終了時間、実行結果の戻り値。

また、これらの履歴は自動的に該当する属性へ記録される。すなわち、ある属性 `@X` の値の参照を行う際には、暗示的に `@X.HISTORY` という属性に対しての参照履歴の追加操作を伴ない、あるメソド`&Y` を実行する際には、その操作内容が属性 `&Y.HISTORY` に追加される。

オブジェクトの生成と継承　オブジェクトの生成は `.OBJECT` というあらかじめ定義されたオブジェクトの雛型か、既に存在しているオブジェクトからの属性/メソドの継承によって行われる。全てのオブジェクト生成はオブジェクトからの継承と等価であるため、以下の説明ではオブジェクトの継承に着目して説明する。

既に存在しているオブジェクト `.A` から、新たにオブジェクト`.B` を継承によって作成する場合、まず `.B` を `.A` の複製として作成する。この時に、`.A` にて設定されたアクセス制限によって、継承を禁止された属性とメソドについては省かれる。次に、`.B` の持つ属性とメソドに対して新たに追加および変更する必要があればその操作を行う。つまり、MonoProcess ではオブジェクトの継承は既存のオブジェクトの複製を作成することによって行う。

なお、オブジェクトの継承は、オブジェクトの雛型で定義されている `&FORK` メソドの実行によって行われる。このメソドは引数として新しく生成されるオブジェクトで用いられる追加定義の情報を取る。また、`&FORK` を再定義することによって、あるオブジェクトに固有の継承方法を定義することができる。

3. 考察

〈3•1〉 プロセスモデルとしての特徴　ソフトウェアプロセスモデルや、それを元にするソフトウェアプロセス環境にはさまざまな目的がある[7]。Curtis らは、プロセス記述、プロセスモデル、プロセス中心型開発支援環境には、プロセスの理解から自動実行といった幅広い分野において、5 つの基本的な使われ方があるとしている[5]。それぞれは次のように纏めることができる。

- プロセスの理解や伝達を容易にする
- プロセスの進化を支援する
- プロセスの管理を支援する
- 自動的なプロセスの誘導を行う
- 自動的な作業の実行を行う

我々の提案するソフトウェアプロセスのモデル化では、これらの点を次のような形で実現している。

- 開発環境はオブジェクトの集合として表現されており、開発者も管理者も共通の表現を利用している。各オブジェクトに関する情報はそのオブジェクト内部で閉じている。
- MonoProcess で用いるオブジェクトはオブジェクト指向分析等を用いて分析を行うことができる。このため、オブジェクト指向分析の結果から導出されるオブジェクトの構成等に関する改善点を適用することによって、開発環境の構成をより効率的な形へ進化させることが可能である。
- 開発作業にて行われた操作は、履歴として保存されている。これらを利用することによって、進捗管理等を行う枠組を提供する。
- 各オブジェクトの内部状態を属性として管理することにより、そのオブジェクトに対して現在行える作業を明示できる。
- メソド記述により、定型的な作業については自動的に実行させることができる。非定型的な作業についても、その中に含まれる単純な作業を自動的に実行させることが可能である。

以上から MonoProcess は、ソフトウェアプロセスを表現、記述、動作するために必要な機能を全て持っていると言える。

〈3•2〉 オブジェクトを用いた記述　MonoProcess はソフトウェア開発環境における制御/データ統合を行うために、オブジェクトという単一の構造に情報と作業の両方を記述する。記述されたオブジェクトを用いて「開発作業中で現在何が行われているのか」を表現し、行われた作業を履歴として管理することによって、開発管理等にも用いることができる。これは、他のプロセス中心型ソフトウェア開発環境[6][8]で用いられている言語に見られる「何をしなければならないのか/どうしなければいけないのか」に着目して「予想されない例外に対してどう対処を行うかを記述する」こととは異なる物である。

我々は開発プロセスを開発時の結果として捉えており、それはさまざまな観点から充分に記述することは非常に困難であるという立場を取っている。MonoProcess では、多様な観点から捉えられる作業それ自身ではなく、作業

を構成する作業の断片をメソドを記述している。これにより、開発環境中で行われる内容をより正確に、かつわかりやすい単位で記述することが可能となっている。また、メソドを単位として行われた作業の結果を操作履歴として残すことができるため、開発プロセスをより詳細に表現することが可能となっている。これらのことから、MonoProcess はプロセス記述を行う際に現実的な基盤となり得ると考えられる。

あるオブジェクトの持つ内容を継承したオブジェクトを定義することは、MonoProcess 以外のプロセス記述言語でも行うことができる。しかし、それらは全体構造が木構造になることを仮定している物が多い[18]。このため、多重継承を表現することや、全体におけるあるまとまった一部分を変更することが困難である。Marvel は strategy によって作業をグループ化することができるが、strategy 自体は Marvel の持つデータ構造の定義に依存しているため、自由なグループ化が行えるとは言えない。

MonoProcess では、作業の分類は MonoProcess の持つグループ機能によって任意の作業に対して行える。また、継承機能を用いることによって、複数のオブジェクトで利用する属性やメソドを共有できる。さらに、さまざまな粒度における資源や作業等を、同一の属性/メソドラベルを用いることによって、一貫した記述が行える。

〈3・3〉 開発作業の記述 APPL/A[18] 等の手続き型のソフトウェアプロセス記述言語では、ソフトウェア開発作業は構造化に基づいたプロセスの定義が行われる。すなわち、プロセスを記述する際には、作業の構造を静的に決定する必要がある。Marvel[3][4][10] などの、ルールベースのソフトウェアプロセス記述では、作業の記述を行う場合に何が前条件や後条件となるのかを決定する必要がある。 MonoProcess においては、記述されるオブジェクトをオブジェクト指向分析等を用いて分析を行うことができるため、オブジェクト指向分析の結果から導出されるオブジェクトの構成等に関する改善方法を用いてプロセス記述をより効率よく行なうことができる。また、開発環境中に存在する文書、ツール、計算機資源、開発者等の要素を、それぞれオブジェクトとして集中的に記述することが可能である。

MonoProcess を用いた記述では、開発手順はオブジェクトにおけるメソドとなるため、ある一連の作業の流れが複数のオブジェクトの複数のメソドとして記述されてしまう可能性がある。これによって一連の記述が分散してしまい可読性を損ねる危険がある。しかし、MonoProcess の持つグループ化の機能を用いることにより、複数のオブジェクトに分散した情報を単一のオブジェクトへ纏めることが自由に行える。これにより、単一のオブジェクトに着目するだけで必要な情報を集中して扱うことができ、内容を参照する場合にも、単一のオブジェクト内で一連の作業が完結しているため理解しやすくなると考えられる。またこの機能により、参照される範囲を適切に限定することができる。

MonoProcess の枠組では、メソドの記述に関して特定の言語仕様を仮定しておらず、その機能のみを定義している。既存のプロセス記述言語は、言語の機能に強く依存した構造であり、作業の表現方法に何らかの制約を課してしまう形となっている[11]。MonoProcess をメソド記述言語から独立して定義することによって、メソド記述の際、目的に応じて言語を選択することが可能となる。

〈3・4〉 作業履歴 MonoProcess の持つ作業履歴の管理は、他には見られない特徴である。既存のオブジェクト指向プログラミング言語やデータベース言語においても、メソドの呼び出し回数や実行を中断して内部状態を観察するための同様の機能が提供されている。しかし、それらは外部ツール等によって実現されているものが多く、言語自体に含まれているわけではない。MonoProcess では、記録されている履歴は属性として扱われているために、他の属性を扱う時と同様にして作業履歴を扱うことができる。

作業履歴の収集は、キーボードやマウスに対する割り込みを利用したり、オペレーティングシステムやアプリケーションに対する追加操作として収集することも考えられる。しかし、このようにして収集される履歴は非常に細かな粒度のデータになってしまうため、これらを用いて「作業者がどの生成物に対してどのような操作を行ったのか」を分析するのは非常に困難であろう。MonoProcess はオブジェクト中の属性やメソドを単位とした履歴の収集を行うため、オブジェクトを収集したいデータに応じて定義することによって、履歴の解析を行いやすいデータを効率よく収集することができる。

4. まとめ

本稿では、オブジェクト中心型ソフトウェアプロセスモデルのモデル化手法である MonoProcess について述べた。MonoProcess はオブジェクトを用いて開発作業環境中にある生成物や資源を表現する。MonoProcess ではソフトウェアプロセスはオブジェクトの状態遷移として表現される。MonoProcess を用いることにより、ソフトウェアプロセスをよりわかりやすく表現することが可能であり、作業履歴を用いることによってプロセス管理をより効率良く行うことができる。

我々は現在、MonoProcess に基づいたソフトウェア開発環境 MonoProcess/SME の設計と試作を行っている[13]。本環境は、オブジェクト記述を保持するリポジトリ、リポジトリを操作するためのユーザインターフェイス、そしてメソド記述を解釈するメソドエンジンから構成され、分散開発環境において、開発者に対する開発支援や管理者に対する進捗状況の提供を容易に行うことを目的としている。今後、試作環境を実際の開発環境で運用することによって本モデルの機能に関する評価を行う予定である。具体的には、数人の開発者と管理者で構成される開

発チームに利用してもらい、オブジェクトの数やその内容の記述状況、システムの利用パターン等の記録を行い、実験終了後にその結果を分析し、開発者等に対する聞き取り調査を実施する予定である。また、より抽象化されたメタオブジェクトを本モデルに導入することにより、本モデルをソフトウェアプロセスの改善を行う際の枠組として発展させる予定である。

（平成 10 年 4 月 1 日受付，同 10 年 6 月 8 日再受付）

文　献

(1) S. bandinalli, E. D. Nitto and A. Fuggetta: “Supporting Cooperation in the SPADE-1 Enviornment”, IEEE Transaction on Software Engineering, Vol.22, No.12, pp.841–865 (1996)

(2) S. bandinalli, A. Fuggetta and S. Grigolli: “Process Modeling in-the-large with SLANG”, In Proceedings of the Second International Conference on the Software Process, pp.75–83 (1993)

(3) N. Barghouti: “Supporting Cooperation in the MARVEL Process-Centered SDE”, In Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments, pp.21–31 (1992)

(4) I. Ben-Shaul, G. Kaiser, and G. Heineman: “An Architecture for Multi-User Software Development Environments”, In Proceeding of 5th ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments, pp.149–158 (1992)

(5) B. Curtis, M. Kellner, and J. Over: “Process Modeling”, Communication of the ACM, Vol.35, No.9, pp.75–90 (1995)

(6) C. Fernstrom and C. G. Innvation: “PROCESS WEAVER: Adding Process Support to UNIX”, In Proceedings of 2nd International Conference on Software Process, pp.12–26 (1993)

(7) A. Fugetta: “Functionality and architecture of PSEEs”, Information and Software Technology, Vol.38, No.4, pp.289–293 (1996)

(8) H. Iida, K. Mimura, K. Inoue, and K. Torii: “HAKONIWA: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model”, In Proceedings of 2nd International Conference on Software Process, pp.64–74 (1993)

(9) 井上克郎：「ソフトウェアプロセスの研究動向」、ソフトウェア科学会研究報告、SP-2-1、pp.1–10 （1995）

(10) G. Kaiser, P. Feiler, and S. Popovich: “Intelligent Assistance ofr Software Development and Maintenance”, IEEE Software, Vol.5, No.3, pp.40–49 (1988)

(11) T. Katayama: “A Hierarchical and Functional Software Process Description and Its Enaction”, In Proceedings of 11th International Conference on Software Engineering, pp.343–352 (1989)

(12) M. Matsushita, M. Oshita, H. Iida, K. Inoue: “Conceptual Issues of Object-Centered Process Model”, In Proceedings of 1997 Asia-Pacific Software Engineering Conference and International Computer Science Conference, pp.519–520 (1997)

(13) M. Matsushita, M. Oshita, H. Iida, K. Inoue: “Distributed Process Management System Based on Object-Centered Process Modeling”, In Proceedings of Worldwide Computing & Its Applications '98, pp.108–119 (1998)

(14) 松下誠、大下誠、飯田元、井上克郎：「オブジェクトモデルを用いたソフトウェアプロセス記述用言語 MonoProcess」、情報処理学会研究報告、97-PRO-14-17、pp.97–102 （1997）

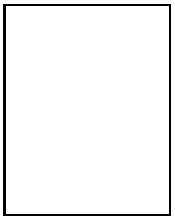
(15) 大下誠、永山裕之、山本哲男、松下誠、楠本真二、井上克郎：「オブジェクト指向モデル MonoProcess を用いたソフトウェア開発管理システムにおけるユーザインターフェース部」、電子情報通信学会技術研究報告、SS97-86、pp.71–78 （1998）

(16) P. Tarr and L.A. Clarke: “PLEIADES: An Object Management System for Software Engineering”, In Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp.56–70 (1993)

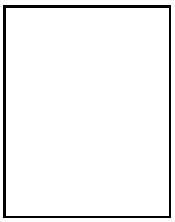
(17) 落水浩一郎：「ソフトウエアプロセスに関する研究の概要」、情報処理、Vol.36、No.5、pp.379–391 （1995）

(18) S. Sutton Jr., D. Heimbigner, and L. Osterweil: “APPL/A: A Language for Software Process Programming, ACM Transactions on Software Engineering and Methodology”, Vol.4, No.3, pp.221–286 (1995)

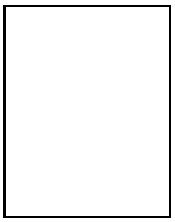
(19) L. Wakeman and J. Jowett: “PCTE The standard for open repositores”, Prentice Hall International (1993)



松　下　　誠（非会員） 平成 5 年大阪大学基礎工学部情報工学科卒業。平成 7 年同大学大学院博士前期課程修了。同年同後期課程入学、現在に至る。ソフトウェアプロセスの研究に従事。



飯　田　　元（非会員） 昭和 63 年大阪大学基礎工学部情報工学科卒業。平成 2 年同大学大学院博士前期課程修了。同年同後期課程入学。平成 3 年大阪大学基礎工学部助手。平成 7 年奈良先端大情報科学センター助教授、現在に至る。工学博士。ソフトウェア開発プロセスおよび開発支援環境、協調作業支援技術、分散オブジェクト処理技術などの研究に従事。



井　上　克　郎（非会員） 昭和 54 年大阪大学基礎工学部情報工学科卒業。昭和 59 年同大学大学院博士課程修了。同年大阪大学基礎工学部助手。同年 8 月ハワイ大学マノア校芸術・科学学部助教授。昭和 61 年大阪大学基礎工学部助手。平成元年同講師。平成 3 年同助教授。平成 7 年同教授、現在に至る。工学博士。ソフトウェア工学の研究に従事。