ソフトウェア工学 98－7
（1994．5．25）

オブジェクト中心型モデルからフェーズ中心型モデルへの変換と動的変更に対するリソース制約回避方法の検討

竹原昭浩　鈴木正人　片山卓也

北陸先端科学技術大学院大学 情報科学研究科

数多くのソフトウェアプロセス記述モデルの中でオブジェクト中心型モデル (OCM) 及びフェーズ中心型モデル (PCM) の 2 つはそれぞれオブジェクトの静的関係、アクティビティの実行順序を記述するためのモデルとして知られている。しかし現在提唱されている PCM はそのままでは開発過程の監視・管理のために使用することはできない。特に開発中に発生するリソースの欠如などの制約を回避するためのプロジェクト全体の再スケジューリングの方法等を記述できない。

本稿では動的なリソースの変更に対するプロジェクト全体の変更を見通しよく行うことが可能な PCM の構成方法を提唱する。具体的にはある制御ソフトウェアの開発を例としてオブジェクトの構造に基づいてフェーズを定義し、PCM を構成し、これに作業者 (リソース) の役割等を考慮した割り付けを行う。また動的リソース変更として作業者の増減を例にリソースの再割り付けおよびオブジェクトの縮退による対処方法を述べる。この方法により、オブジェクトの質の低下を最小限にしたままで作業全体の量を減らすことが可能になる。

Construction of Phase-Centered Model from Object-Centered Model and Management of Constraints for Dynamic Resource Changes

Akihiro TAKEHARA　Masato SUZUKI　Takuya KATAYAMA

School of Information Science

Japan Advanced Institute of Science and Technology

15 Asahidai, Tatsunokuchi, Ishikawa 92312, Japan

Many formal models are proposed for describing the different aspect of software processes. Object-centered model and phase-centered model are two famous models of them. The former is a model focusing on the static relationship among objects, and phase-centered focusing on *phase*, which is a collection of activity enactions based on object relationships.

One of the issue on this phase-centered aspect is that we have no scheme for changing the resource allocation and re-scheduling the project in unexpected situations, such as resource shortages. In this paper, we propose a formal method for constructing phase-centered model from the structure of objects, and guidelines for handling resource changes, more precisely, resource reallocation and activity reduction. With these procedure, we can reduce the amount of works without losing much qualities in products.

1 はじめに

ソフトウェアプロセスモデルにはその記述対象の違いからいくつもの種類があるが、中でもオブジェクト中心型モデル及びフェーズ中心型モデルの2つはそれぞれオブジェクトの関係、アクティビティの性質を理解するために有効であることが知られている [1]。しかし現在のところ、フェーズ中心型モデルを得るために必要となるフェーズの設定およびそれに基づいたアクティビティの形式化のための方法は十分に確立されてはいない。また現在のフェーズ中心型モデルではソフトウェア開発過程の管理、運用のために必要となるリソースに関する情報は表現されておらず、このためプロセスの実行中に発生した制約、あるいはリソースの欠如などの例外的事態に対応するためのプロセスモデルの変更方法の検討も十分に行なわれてはいない。

本稿ではこのような動的なリソースの変更に対してプロジェクト全体の変更を見通しよく行うことが可能なフェーズ中心型モデルの構成方法を提唱する。また実際のソフトウェア開発プロセス (無線機の制御ソフトウェア) を例として、オブジェクト中心型モデルからフェーズ中心型モデルを得るための変換方法としてオブジェクトの詳細度に基づいたフェーズの設定方法、リソースとその制約の記述、および動的変化に対するフェーズ中心型モデルの変更の方法について述べる。

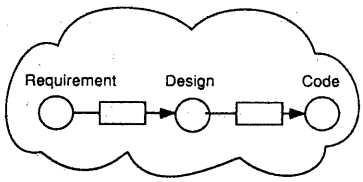
2 オブジェクト中心型モデルとフェーズ中心型モデル

最初にソフトウェアプロセスの2つのモデルに関する説明を行ない、以下で行なう論議の前準備として幾つかの用語の定義を行なう。なおここでいうフェーズの定義は [1] で導入されたものである。

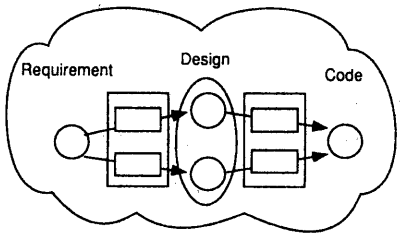
2.1 オブジェクト中心型モデル

ソフトウェアプロセスの記述において最初に注目すべきものはその実行の結果として作成または修正される生成物であり、オブジェクトと呼ばれる。生成物に注目する理由は，ソフトウェアプロセスにおいて最も基本的な作業単位の目的がこれらを生成することであり、ソフトウェアプロセスはその第1次近似としてオブジェクトの集合とそれらの間の静的な関係の集合によって抽象的に表現できると考えられるからである。このようなモデルは**オブジェクト中心型モデル** (以下、OCM) と呼ばれ、オブジェクト間の関係を成立させるための作業主体は**プロセス**、その構成要素は**アクティビティ**と呼ばれる。

OCMが得られるまでの概念図を1に示す。OCMを構成するためには最初にオブジェクトとその関係を抽出する。この時点では、すべてのオブジェクトを均等なものとして扱い個々のオブジェクトが持つ内部構造までは考慮しない。次に複雑なオブジェクトに対しては分解を行ない、似通った性質を持つオブジェクトの合成を行なう。この時、アクティビティの分解や合成も同時に行なわれる。図1の例では、オブジェクト design とそれを使用するプロセスは2つに分解されている。もし必要ならば、中間オブジェクトを導入することによってOCMはさらに階層的に分解することもできる。



(a) Extract objets and processes



(b) Decompose complex objects and processes

図1 オブジェクト中心型モデル

2.2 フェーズ中心型モデル

OCMではオブジェクト間の静的関係は記述されているが、一般にはそれらの関係を成立させるプロセスが十分に単純でなければプロセスを実行することはできない。また大規模で複雑なソフトウェア開発では、オブジェクトとプロセスに関する構造を最初からすべて把握することや、そのプロセスの入力となる全てのオブジェクトが同じ時期に生成されていることを期待できない。これはOCMがプロセスの実行順序を記述するのに十分なモデルではないということを意味している。

一般にOCMにおけるオブジェクトはその詳細度において異なり，徐々に詳細化されていく複数の段階のインスタンスを持っている。OCMにおけるプロセスとは、これら各段階のオブジェクトを作成する操作(アクティビティ)の集合であると考えることができる。同一の詳細度に属するオブジェクトを得るためのアクティビティの集合をフェーズと呼ぶ。主にフェーズの順序関係を記述することから、このモデルはフェーズ中心型モデル(以下、PCM)と呼ばれている。

図1に対応するPCMを図2に示す。OCMにおける単一オブジェクトはPCMでは複数の段階のインスタンスの集合体として扱われ，プロセスはオブジェクトの詳細度に対応したフェーズ $1,\ldots,n$ の集合体として扱われる。

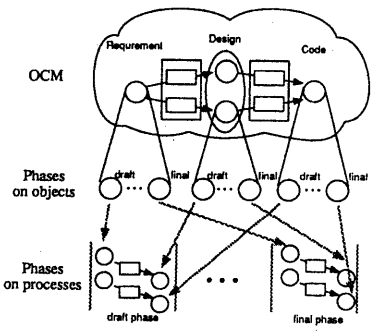


図2 フェーズ中心型モデル

3 OCMからPCMの構成

OCMからPCMを機械的に構成するためにはオブジェクトの段階およびフェーズの境界を決定する必要がある。オブジェクトの段階を定義するにはいくつかの方法が考えられるが我々はもっとも単純かつ有効な方法として、以下のようなオブジェクトの構造に着目した段階およびフェーズの決定方法を採用した。

1. オブジェクトの構造解析
2. オブジェクトの段階及びフェーズの設定
3. 同じフェーズに属するアクティビティのグループ化と実行順序の決定
4. 各アクティビティへのリソース割り付け

3.1 オブジェクト木とオブジェクトグラフ

オブジェクトの構造としては木構造を採用する。その理由として、構造の理解及びオブジェクトの作成進捗記述の容易さが挙げられる。各々のオブジェクトはそれらを構成する要素間において、「～から作られる」「～を改善する」といった関係で結ばれた木構造を形成する。この木はオブジェクト木と呼ばれ、木における各々のノードはオブジェクト要素と呼ばれる。

現実にはオブジェクト木で表現される"強い"関係の他にも別のオブジェクト木の要素を参照するといった"弱い"関係も考慮する必要がある。"弱い"関係はオブジェクト木をまたいで張られるため、オブジェクト木の集合はオブジェクトグラフを構成する。

図3はオブジェクト木とオブジェクトグラフの例を示している。黒矢印はオブジェクト木内の要素の依存関係を表し、灰矢印はオブジェクト木間の要素の参照関係を表している。例えば、criteriaはdesignの一部であると同時に、testpkgの一部からも参照されている。参照関係は順序を規定する。例えばconditionを作成する場合には、criteriaを事前に作成しておく必要がある。

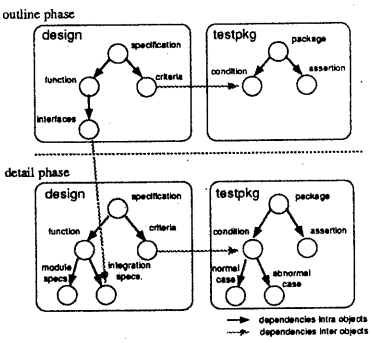


図3　オブジェクトグラフ

3.2　オブジェクトの段階とフェーズの決定

次にオブジェクト木を基にしてオブジェクトの段階及び各段階のオブジェクトを作成するためのフェーズの境界を決定する。オブジェクトの詳細度はオブジェクト木を構成する要素の枝の深さによって定義される。すなわち第 n 段階のオブジェクトを表すオブジェクト木の深さは n 以下である。図3において、function,criteria は共に深さが2であるから、第2段階のオブジェクト木中で初めて出現することになる。

OCM において、同じ段階のオブジェクトを作成するためのアクティビティをグループ化してこれをフェーズとし、必要なら適当な名前を与える。例えば図3では、フェーズ1,2を合わせて「概要フェーズ」、フェーズ3を「詳細フェーズ」と呼んでいる。

3.3　実行順序の決定

フェーズは $1,\ldots,n$ の順序にしたがって実行され，その過程を通じてオブジェクトは粗いものから細かいものへ詳細化されていく。実際のアクティビティの実行順序は、各フェーズにおいてアクティビティの実行順序によって決定されるが、そのときオブジェクト要素間の依存関係と矛盾するものであってはならない。

図4において，概要フェーズでは、c→g及びb→gという2つのアクティビティが存在するため、cからgへの依存関係の存在がこれらの実行順序を決定する。詳細フェーズでは、d→i,e→jという2つのアクティビティが存在するが、これらの間には依存関係が全くないことから実行順序は任意である。

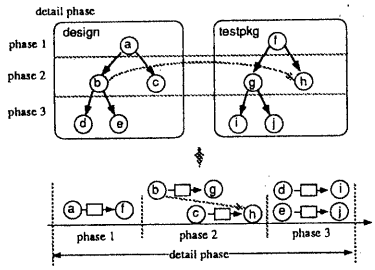


図4　アクティビティの実行順序

3.4　リソース割り付け

リソースには作業者・時間・予算・開発機器など様々であるが、ここでは典型的かつ一般的であるという理由で作業者のみを取り扱っていく。各々のアクティビティに対して作業者を割り付ける前準備として、各々の作業者には以下に示す属性を考える。

- **役割**：各々の作業者のプロジェクトにおける働きを表す
  「設計技術者」、「品質管理者」など
- **階級**：作業者の立場や能力、責任の違いを表すもので「主開発者」と「副作業者」がある。同じ役割を担った複数の作業者のグループには必ず1人の主作業者が存在するものと仮定する。

リソース割り付けは以下に示す条件を満たす必要がある。

1. 作業者は可能な限り他の作業者と独立して作業を行なっていく。これにより、コミュニケーションの量を最小限に押えることができる。
2. 各々の作業者における作業負荷が著しく異ならないようにする。

これらの条件を考慮しながら、以下の手順で割り付けを行う。

1. オブジェクト木において本質的に重要となる要素を選択する。この項目をオブジェクト木における**幹要素**と呼び、主作業者に割り付ける。

2. 主作業者は作業の負荷が極端に大きい場合、副作業者に作業の一部を担当させることができる。この場合、その作業が物理的に分割可能であると言う条件があり、かつ作業者間のコミュニケーションが最小となるように作業を分割しなくてはならない。
3. 幹要素に対する割り付けが終了したら、それ以外の要素に副作業者を割り付けていく。この時も作業者の作業負荷がなるべく均等になるように割り付けを行なう。

4 実行時におけるリソースの動的変更への対応

この節ではソフトウェアプロセス実行中に利用可能なリソースの変化が生じた際に割り付けに関する条件を満たしままフェーズ中心型モデルを見通しよく変更するための方法について論議していく。変更には大きく分けて (1) リソースの再割り付け、(2) オブジェクト要素数の縮小 (オブジェクトの縮約) の 2 種類がある。比較的軽微な変更に対しては前者で対応可能であるが、より重大な変更には後者が必要である。このとき、最終生成物の質をなるべく損なわないようにする必要がある。

4.1 リソースの再割り付け

リソース再割り付けは、他の部分の縮約などにより余剰人員が発生した時にのみ適用される。このとき物理的に独立しており、複数の作業者で分割実行が可能なアクティビティには余剰な作業者を再割り当てすることによって作業者 1 人当たりの作業量を減らすことが可能である。このようなアクティビティの例としては、複数のモジュールのコーディング・単体テストなどがある。またリソースの再割り付け時の追加作業者は副作業者として扱われる。

4.2 オブジェクトの縮約

オブジェクトの縮約は、オブジェクト要素数を減らすことによりそれを生成するアクティビティの数およびその実行に要する時間を軽減することを目的とする。縮約には削除と合成という 2 つの方法がある。削除はアクティビティの実行を省略する方法であり省略されたアクティビティに対するオブジェクトは生成されない。合成は 2 つ以上のアクティビティをまとめて実行し結果を同時に得るものある。どちらの場合も最終的なプロダクトの質の低下を最小限にするために削除、合成されるオブジェクト要素を慎重に選択する必要がある。

4.2.1 オブジェクト要素の削除

オブジェクト要素の削除は中間段階のプロダクトや付帯するドキュメントなど重要度が比較的小さいオブジェクト要素に対して適用される。削除対象となるのは以下の条件を全て満たすものである。

- 幹要素でない
- 作業者が割り付けられていない
- 他のオブジェクト要素への依存関係を持たない

図 5においてノード a,c,e,f,h,i,j,k は幹要素として扱われている。またノード d は j を作成するのに必要となるノードであることから、削除できない。結果として、b,g のみがオブジェクト削除の対象となる

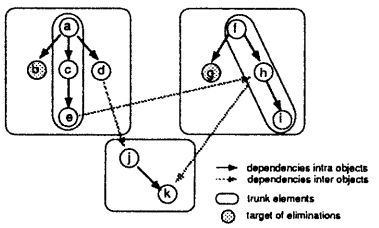


図 5 削除対象の決定

4.2.2 オブジェクト要素の合成

合成に関しては、図 6に示すように 2 つの方法がある。直列合成は同一のオブジェクト木もしくは異なる木の間の 2 つの要素に対して、並列合成は同一のオブジェクト木でかつ同じ深さの 2 つの要素に対して適用される。

異なるオブジェクト木間において直列合成を行なう場合の合成後のノードは、合成前のノードのオブジェクト木内での位置がルートに近い方の木へ組み込まれると定める。例えば図 2において e と

h の合成を行なう時は e のルートからの距離が 2 であるのに対して、h は 1 であるから、結果 eh は h の位置に組み込まれ、c から eh への依存関係を表す矢印が新たに加えられることになる。合成後のオブジェクトグラフを図 7 に示す。

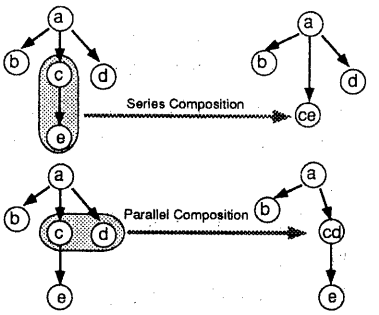


図 6　合成の種類

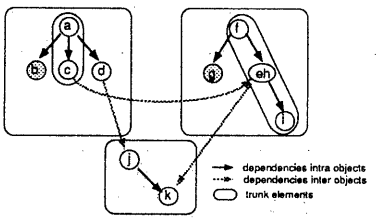


図 7　合成後のオブジェクトグラフ

オブジェクトグラフ変更後は、これに基づいてフェーズの設定および変更を行なうことで新しい PCM を得ることができる。

5　実プロセスを対象とした変更の実行例

この節では無線機ソフトウェア開発プロセス中の設計プロセスを対象にして、リソースの変化に対するオブジェクト要素の削除及び合成を行なった例を示す。

5.1　OCM と PCM の獲得

設計プロセスには基本設計書・概略設計書・詳細設計書・検査仕様書という 4 つのオブジェクトおよびこれらを作成するプロセスが存在している。これらのオブジェクトの構造を解析し、構成する要素間の関連を表したオブジェクトグラフを図 8 に示す。網かけは幹要素を、黒矢印はオブジェクト木内内部の依存関係を、灰矢印はオブジェクト木間の参照関係を表している。

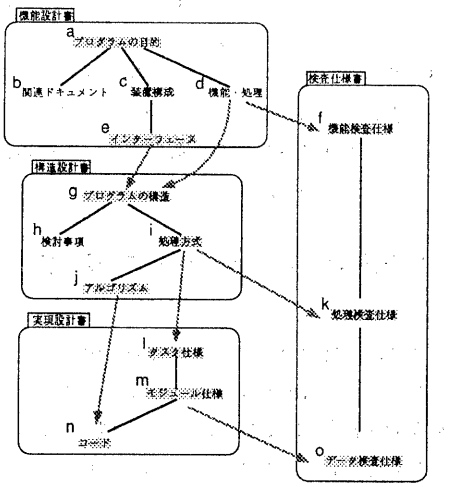


図 8　設計プロセスに関係するオブジェクトグラフ

設計プロセスでは 4 つの各々のオブジェクトにおいて 3 つの段階を設定する。また検査仕様書については他の 3 つのオブジェクトの作成プロセスと並行して実行されることから、第 1 段階は機能設計書、第 2 段階は構造設計書、第 3 段階は実装設計書の作成と同等のフェーズで作成されるものとして扱うことにする。

フェーズおよび実行順序を決定したものを図 9 に示す。この例ではフェーズは基本設計、概要設計、詳細設計の 3 つであり、黒矢印はその先のオブジェクト要素を作成するアクティビティを表している。同じ列に縦に並んでいるオブジェクト作成 (=アクティビティ実行) 順序は任意である。なおこれ以降は生成されるオブジェクト要素でアクティビティを表現する。

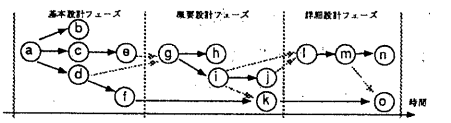


図 9　設計プロセス中の実行順序

これに割り付けられる作業者の役割はソフトウェア設計を行なう「ソフトウェア設計作業」とソフトウェアの検査を行なう「ソフトウェア検査作業」の2つとし、以下の3人を仮定する。

Dc ソフトウェア設計作業における主作業者

Da1 ソフトウェア設計作業における副作業者

Tc ソフトウェア検査作業における主作業者

次に役割を基にしたオブジェクト要素の分類を行なう。設計プロセスにおけるソフトウェア設計作業の要素は a,b,c,d,e,g,h,i,j,l,m,n の 12 個であり、ソフトウェア検査作業の要素は f,k,o の 3 個であることがわかる。そして分類したオブジェクト要素においてオブジェクト木の幹を決定していく。ここではソフトウェア設計作業における幹要素を a,d,g,i,j,l,m,n とし、ソフトウェア検査作業における幹要素を f,k,o とする。ソフトウェア検査作業の幹要素決定に関しては、作業者が 1 人であることから自動的に全てのオブジェクト要素が幹要素となる。

以上の作業者をオブジェクトグラフに割り付ける。幹要素には、Dc,Tc が担当作業者として割り当てられる。ソフトウェア設計作業におけるオブジェクト要素において Dc が割り付けられていないものは自動的に Da1 の作業として割り振られる。なお、オブジェクト要素 g,i,j,l,m,n に関しては物理的に分割可能であり、Dc の作業負荷が重い場合には一部を Da1 に割り付けるものとする。結果は表 1のようになる。

作業者	作業
Dc	a,d
Da1	b,c,e,h
Dc と Da1	g,i,j,l,m,n
Tc	f,k,o

表 1　リソース割り付け

5.2 動的リソース変更への対応

5.2.1 リソースの再割り付け

最初に Da2,Ta1 という 2 人の作業者を追加する場合を考える。現在 3 人いる作業者が 5 人になるので作業者一人当たりの作業量を減少させるように割り付けを行う。ソフトウェア設計作業において複数の作業者を割り付けることが可能なオブジェクト要素は、g,i,j,l,m,n であることからこの 6 つの要素を Dc,Da2 が共同で担当することになる。一方ソフトウェア検査作業におけるオブジェクト要素 f,k,o は全て物理的に分割可能であることから、これら 3 つの要素は Tc,Ta1 が共同で担当することになる。

結果を表 2に示す。リソースの再割り付けではオブジェクトグラフの構造及び PCM は変更されない。

作業者	作業
Dc	a,d
Da1	b,c,e,h
Dc,Da1,Da2	g,i,j,l,m,n
Tc,Ta1	f,k,o

表 2　作業者追加後のリソース割り付け

5.2.2 オブジェクトの縮約

今度は Da1 が何らかの理由により開発プロジェクトから外れた場合を考える。この場合、オブジェクト要素の削除及び合成が必要となり、オブジェクトグラフ及び PCM の変更が行われる。

要素 b,h は幹要素でないこと、及び最終フェーズ中に存在しないことから削除対象として扱うことができる。合成に関しては、オブジェクト要素の持つ内容の類似性から 3 つの直列合成 c と e,j と n,k と o を実行する。合成後のノードは合成元のノードがあった場所がよりオブジェクト木のルートに近い方、すなわちノード ce は c の位置に、jn は距離がいずれも 2 であるが l,m との依存関係から n の位置に、ko の位置も同じ理由により o の位置に置かれる。

以上をまとめたオブジェクトグラフを図 10に、

これに基いた新しいPCMを図11に示した。

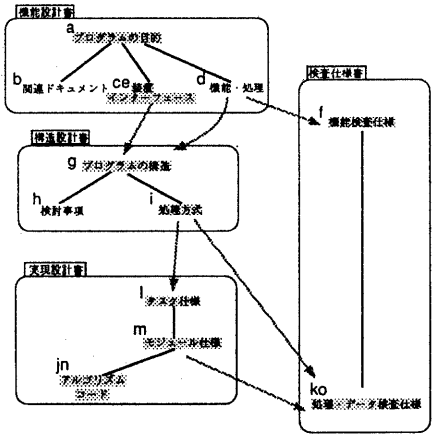


図10　変更後のオブジェクトグラフ

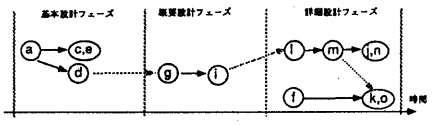


図11　変更後のPCM

オブジェクトの削除と合成により当初12個あったアクティビティは6個にまで減少している。またオブジェクトの質の低下もb,hの省略によるもののみである。必要ならばさらにjnとmを合成することも可能である。

6　おわりに

本報告では、OCMとPCMという2つのプロセスモデルを示し、OCMからPCMへの変換方法及び動的リソース変更発生時のPCM変更方法を示した。OCMはオブジェクト間の静的関係を解析することにより作られ、PCMはオブジェクトの段階及びアクティビティのフェーズを定義し、更に実行順序及びリソース割り付けを行なうことで作られる。そして動的リソース変更はPCMにおけるオブジェクト要素を減少させることにより扱うことができる。この方法により、オブジェクトの質の低下を最小限にしながら、動的なリソースの変化に対応できる新しいPCMを得ることができる。

実際のソフトウェアプロジェクトを解析したところ、オブジェクトの木構造が個々のオブジェクトやプロダクトに複雑に関係していることが判明した[2]。様々なプロジェクトに関するデータが収集できれば、今後実行しようとしているオブジェクトの大きさが与えられた時に、オブジェクトの評価を行なえるようになる。これにより、典型的なプロジェクトの構造獲得及びプロジェクトの再利用といったことができるようになる。本稿で提唱した方法をより効果的で実用的なものにするために、典型的なプロジェクトで用いられるオブジェクトのテンプレートを収集し解析する必要がある。

本報告ではオブジェクトの構造を基にしてフェーズの境界を決定する方法を述べたが、他にアクティビティの意味により決定する方法も存在する。例えば設計プロセスの中にはプロダクトの正確性や実現可能性を検査するためのアクティビティが存在するが、検査に失敗したときの不要なフィードバックを避けるためにフェーズの境界はこうしたアクティビティ検査の後に設定するのが自然であるといえる。もしプロダクトがこうした検査を通過したならば、次のフェースへ移行する、逆にいえば満足するプロダクトが与えられるまではそのフェーズを再実行する。再実行を考慮したPCMの構成方法において要素の不確定及び実行時間の増加をどのように扱うかは今後の課題である。

参考文献

[1] 望月 純夫, 山内 顯, 市村 英昭, 片山 卓也, 鈴木 正人. ソフトウェアプロセスの分析および評価. 情報処理学会第*41*回全国大会予稿集, 1990.

[2] S. Mochizuki A. Yamauchi T.Katayama. Describing and Evaluating Fundamental Design Process of Checkout System for Artoficial Spacecraft by Process Model HFSP. *Transaction of Information Processing Society of Japan*, 33(5):691–706, 1992.