ソフトウェア工学 110－8
（1996．7．26）

マイルストーンを組み込んだ オブジェクト指向開発プロセス生成方法の提案

花川典子[†‡]　飯田 元[†]　松本 健一[†]　鳥居 宏次[†]

[†] 奈良先端科学技術大学院大学 情報科学研究科
〒630-01 奈良県生駒市高山町 8916-5
[‡] （株）日本コンピュータ研究所 関西支社
〒540 大阪市中央区大手前 1-2-15
E-mail:{noriko-h, iida, matumoto, torii}@is.aist-nara.ac.jp

オブジェクト指向開発の大きな問題点の１つは，開発の進捗把握，及び，制御が困難なことである．本報告では，オブジェクト指向開発手法をはじめとする様々なソフトウェア開発手法に対して，適切なマイルストーンを組み込んだソフトウェアプロセスを生成するためのフレームワークを提案する．提案するフレームワークは，対象となる開発手法における作業項目とプロダクトの関係に基づいて，開発フェーズとそのベースラインプロダクトを識別するアルゴリズムを提供する．また，ベースラインプロダクトに基づく進捗管理プロセスモデルも定義している．提案するフレームワークを Object Modeling Technique (OMT) に適用した結果，比較的容易にプロセスモデルが得られることが分った．また，作業項目とプロダクトの関係に制限を導入することで，フェーズの数や長さの調整も可能であることが分った．

A Framework of Generating Software Process including Milestones for Object-Oriented Development Method

Noriko Hanakawa[†‡]　Hajimu Iida[†]　Ken-ichi Matsumoto[†]　Koji Torii[†]

[†] Graduate School of Information Science, Nara Institute of Science and Technology
[‡] Computer Institute of Japan Corporation

[†] 8916-5 Takayama, Ikoma, Nara 630-01, Japan
[‡] 1-2-15 Otemae Chuoku, Osaka 540, Japan

This paper proposes a new framework which gives us a rigorous guideline for generating software process with relevant milestones for various kinds of software development methods, especially for object-oriented development methods. The framework provides algorithms for identifying development phases and baseline products based on relationships among activities and products of the development method. In addition, the framework defines a software process model to manage development progress in which milestones are established at the end of each phase in order to check the baseline product and establish goals for the following phase. Result of the application of the proposed framework show that the framework can generate a software process customized for well-known object-oriented development method in a systematic way.

1 はじめに

オブジェクト指向開発の大きな問題点の1つは, 開発の進捗把握, 及び, 制御が困難なことである. これまでに数多くのオブジェクト指向開発法が提案されているが, 「開発チームのメンバーが必要な作業を必要な時期に行っているかどうかが分かり, プロジェクトを制御することのできる方法論がどれであるか, 多くのプロジェクト管理者には分らない」と Yourdon は指摘している [14].

進捗管理のアプローチの一つは, 開発作業全体をいくつかのフェーズに分け, フェーズ毎にマイルストーンを設定することである [13]. マイルストーンとは, 開発過程で得られる特定のプロダクトの作成完了日, あるいは, その予定日である. ソフトウェア開発における典型的なマイルストーンの一つは, コーディングが完了し, コードレビューに合格する（予定の）日である. ソフトウェアプロセスモデルとして Waterfall モデルが広く用いられてきた理由の一つは, マイルストーンを設定しやすいことにあったと言える [12].

開発作業をいくつかのフェーズに分け, フェーズ毎にマイルストーンを設定することは, オブジェクト指向開発プロジェクトでも可能である. しかし, 要求分析, 設計, コーデイング, テストといった従来通りのフェーズ分けは, オブジェクト指向開発には適していない [3]. オブジェクト指向は新しい方法論であり, 従来とは異なる作業項目や表記法が用いられる. 例えば, データフローダイアグラムのバブル（プロセス）を作成する代わりにオブジェクトを発見する, といった違いである. 従って, 新しい作業項目や表記法に即したフェーズ分けやマイルストーンの設定が必要となる [14].

本報告では, オブジェクト指向開発手法をはじめとする様々なソフトウェア開発手法に対して, 適切なマイルストーンを組み込んだソフトウェアプロセスを生成するためのフレームワークを提案する. 提案するフレームワークは, 対象となる開発手法における作業項目とプロダクトの関係に基づいて, 開発フェーズとそのベースラインプロダクトを識別するアルゴリズムを提供する. ここで, ベースラインプロダクトとは, 対応する開発フェーズで作成され, そのフェーズのマイルストーンにおいて完成度をチェックされるべきプロダクトのことである. 更に, このフレームワークでは, 各フェーズにマイルストーンを設定し, ベースラインプロダクトの完成度をチェックすることによって進捗管理を行うためのソフトウェアプロセスモデルも定義している. 従って, 開発手法の定義に基づいて作業項目とプロダクトの関係を明確にするだけで, その開発手法向けにカスタマイズされた, 進捗管理可能なソフトウェアプロセスを得ることができる.

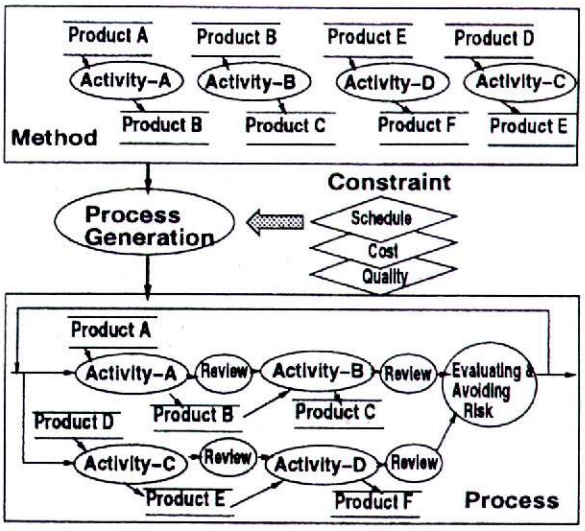

図 1: 開発手法, プロセス, 制約の間の関係

2 では, マイルストーンによる進捗管理の概要を述べる. 特に, 開発手法, プロセス, プロジェクト制約, 及び, マイルストーンの間の関係について論じ, 同時に, オブジェクト指向開発における進捗管理の問題点を明らかにする. 3 では, 提案するフレームワークにおけるプロセス生成の手順を述べる. 特に, 開発フェーズとベースラインプロダクトを識別するアルゴリズム, 及び, マイルストーンを組み込んだ進捗管理プロセスモデルを示す. 4 では, 提案するフレームワークをオブジェクト指向開発手法の一つである Object Modeling Technique (OMT) に適用した結果について述べる. 更に 5 では, 関連研究を示し, 6 においては, 本報告で論じたアイデア, 及び, 今後の課題を簡単にまとめる.

2 マイルストーンによる進捗管理

2.1 開発手法とプロセス

ソフトウェアプロジェクトの開発手法 (Method), プロセス (Process), 制約 (Constraint) の間の関係を図 1に示す. 開発手法とは, 特定の目標へと一歩ずつ近づくために予め定められた手続きのことである [6]. ソフトウェア開発における作業記述の多くは, 開発手法の記述ということが出来る. 開発手法の記述

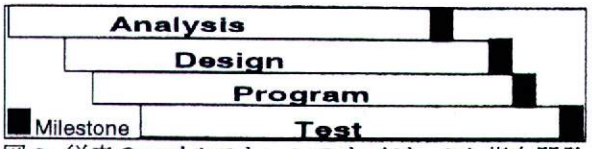

図 2: 従来のマイルストーンのオブジェクト指向開発

は抽象的なことが多いが, ソフトウェアを開発する上で開発者は何をどう考え, 判断すべきかが示されている. また, 開発手法の記述は, ソフトウェアアーキテクチャの概念にも基づいている [6].

一方, プロセスとは開発手法から生成されるものであり, 実際のソフトウェア開発プロジェクトにおける制約 (例えば, コスト, スケジュール, 品質など) を考慮した, より具体的な作業内容の形式的記述である. 例えば, 比較的短期間でソフトウェアを開発しなければならない場合, 同時実行可能な作業を並行化するようなプロセスが生成される. また, ソフトウェアに高い信頼性が要求される場合, レビューやテストといった管理的作業が多く追加されたプロセスが生成される. 更に, ユーザー要求が明確でなく不安定な場合, 要求変更のリスクを回避するために, 作業を反復するようなプロセス, 例えば, スパイラルプロセスが生成される.

2.2 マイルストーン

マイルストーンとは, 開発過程で得られる特定のプロダクト (例えば, 要求仕様書, 設計ドキュメント, プログラムコード, テストレポートなど) の作成完了日, あるいは, その予定日である [13]. ソフトウェア開発における典型的なマイルストーンの一つは, コーディングが完了しレビューに合格する (予定の) 日である [13].

プロダクトがマイルストーンに達したかどうか, 即ち, プロダクトの作成が予定通りに完了したかどうかは, チームメンバー, 管理者, 顧客によってマイルストーンにおいて行われる一連のレビューによって決定される. なお, マイルストーンにおけるレビューでは, プロダクト作成に要した作業量が予定よりも小さかったかどうか, ソフトウェアを予定通り出荷するために開発計画が変更されたかどうか, といった点もチェックされる [1]. また, マイルストーンにおいて作成が完了したと認められたプロダクトはベースラインプロダクト (Baseline product) となり, 以降の作業では原則として更新できなくなる*[11].

*予め定められた正式の手続きを経れば更新は許可される.

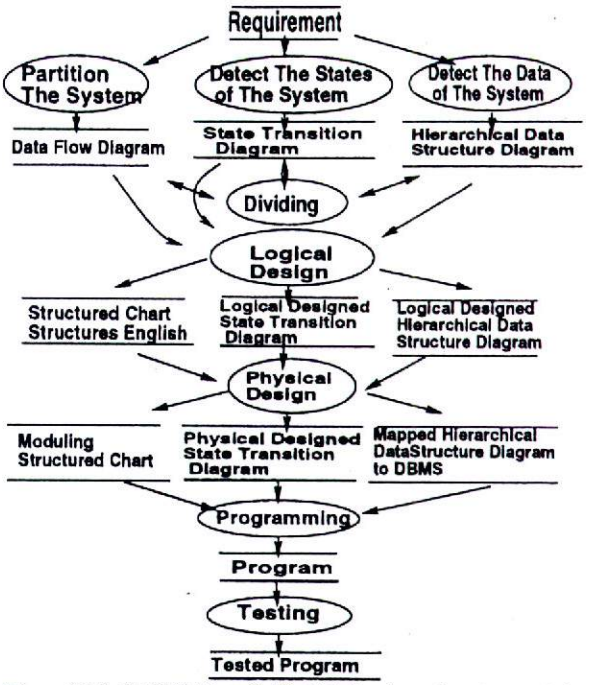

図 3: 構造化開発法の作業項目とプロダクトの関係

2.3 オブジェクト指向開発の進捗管理の問題点

オブジェクト指向開発は, 本来, 相互作用的でありシームレスである [9]. 相互作用的とは, 開発するソフトウェアシステムの同一箇所が, 開発中に何度も繰り返し作業対象になるという意味である. そのため, オブジェクト指向開発では, プロトタイピングによる開発や作業のフィードバックループが一般的である. 一方, シームレスであるとは, 分析, 設計, コーディングといった従来の作業区分 (フェーズ) に関しては, その境界が明確でないという意味である.

従来の作業区分 (フェーズ) に関してシームレスであるという性質は, 従来から用いられてきたマイルストーンがオブジェクト指向開発における進捗管理では効果のないことを意味している. 即ち, 要求分析, 設計, プログラム, テストといった従来通りのフェーズ分けをオブジェクト指向開発において行うと, ほとんどのマイルストーンは開発期間の後半に集中する (図 2参照). ただし, このことは, マイルストーンに基づく進捗管理がオブジェクト指向開発に適用できないということではない. オブジェクト指向開発に適したフェーズ分けとマイルストーンの設定が必要なだけである.

3 提案するフレームワーク

本報告で提案するフレームワークは, オブジェクト指向開発手法をはじめとする様々なソフトウェア開発手法に対して, 適切なマイルストーンを組み込んだソフトウェアプロセスを生成するためのガイド

ラインである．提案するフレームワークは，次の4つのステップから成る．

(1) 開発手法分析
(2) フェーズ識別
(3) ベースラインプロダクト識別
(4) プロセス生成

このフレームワークの特徴は，対象となる開発手法の定義に基づいて，フェーズとベースラインプロダクトをアルゴリズミックに識別する点である．即ち，プロセス生成においては，従来型の固定されたフェーズやベースラインプロダクトは用いず，開発手法毎にカスタマイズされたフェーズとベースラインプロダクトを用いている点である．以降，この章では，提案するフレームワークの4つのステップについて述べる．

(1) 開発手法分析

開発手法分析は，開発手法が定義する作業項目とプロダクトの関係を明確にする手続きである．本フレームワークでは，作業項目とプロダクトの関係をデータフローダイアグラム（DFD）[2] によって表す．ここで,DFD の「プロセス」は作業項目を，「データストア」はプロダクトを，それぞれ表す．また，プロセスからデータストアへのアークは，対応する作業項目によってプロダクトが生産されることを意味する．逆に，データストアからプロセスへのアークは，対応する作業項目によってプロダクトが参照されることを意味する．更に，プロセスとデータストアを結ぶアークのラベル（例えば，図6の"Association"）は，対応する作業項目で一時的に生産され，参照されるプロダクトを表す．

構造化開発法 [2] における作業項目とプロダクトの関係を DFD として表した例を図3に示す．この例の場合,8 つの作業項目と 12 のプロダクトで DFD は構成されている．また，作業項目とプロダクトの間の関係が比較的単純で直線的になっている．

(2) フェーズ識別

本フレームワークでは，開発作業全体をいくつかのフェーズに分け，それぞれのフェーズにマイルストーンを設定するものとする．ただし，マイルストーンにおいて作成が完了したと認められたプロダクトはベースラインプロダクト (Baseline product) となり，以降の作業では原則として更新できないものとする [11]．また，参照すべきプロダクトが作成されていない状態では，作業項目は開始できないものとする．従って，次の2点を満足するようにフェーズ分けを行う必要がある．

- 同一プロダクトを作成，更新する作業項目は同一フェーズに属していなければならない．
- ある作業項目 A が参照するプロダクトを作成，あるいは更新する作業項目は，作業項目 A と同一フェーズ，もしくは，それ以前のフェーズに属していなければならない．

フェーズ識別のための入力データ構造，出力データ構造，及び，アルゴリズム *Algorithm1* は次の通りである．

```
Input:
D_act Dev_Method;
class List     //define Class List////
{   ........
  int GetLength() { Get this List length};
  void AddList(data){Add data to this List};
  void AddListAll(List data)
          {Add all list data to this List};
  void DeleteList(data)
          {Delete data from this List}; }
lass D_act: Public lisy
{//Activity and Product of method
    CPro Pro_in;//Input(Reference) Product
    CPro Pto_out;//Output(Work) Product
    Act_function();
      //Activity of development method }
class CPro : public List{String Name;...}
Output:
D_phase Dev_process;
class D_phase : Public List
{ //development process class
  CPro Pro_in;//Input(Reference)Product
  CPro Pro_out;//Output(Work)Product
  CAct_Funs Act_functions;//activities
  CNext Next_phase;//next phase pointer   }
class CAct_Funs : public List{
  D_act.Act_function() *function; }
class CNext :   public List{
  D_phase * next_ph;}
Algorithm1:
for (int i=0; i<Dev_Method.GetLength();i++)
{D_act M;
 M =  i-th member of the Dev_Method List
 D_phase new_ph=new(D_phase);//Add new phase
 new_ph.Act_functions.AddList
                      (M.Act_function());
 new_ph.Pro_in.AddListAll(M.Pro_in);
 new_ph.Pro_out.AddListAll(M.Pro_out);
 for(int j=i+1;j<Dev_Method.GetLength();j++)
 {D_act S;
  S = j-th member of the Dev_Method List;
  if((M.Pro_out ∩ S.Pro_out) ≠ φ)
  { new_ph.Act_functions.AddList
                    (S.Act_function());
```

```
    new_ph.Pro_in.AddListAll(S.Pro_in);
    new_ph.Pro_out.AddListAll(S.Pro_out);
    Dev_Method.DeleteList(S); } } }
for(int i=0;i<D_phase.GetLength();i++)
{D_phase P_M;
 P_M =i-th member of the D_phase List
 for(int j=i+1;j<D_phase.GetLength();j++)
 {D_phase P_S;
  P_S = j-th member of the D_phase List
  if((P_M.Pro_out ∩ P_S.Pro_in) ≠ φ)
     P_M.Next_phase.AddList(P_S);} }
```

ここで, 入力データクラス D_act は, 開発手法において定義された作業項目を表し, 作業項目で作成されるプロダクト, 及び, 参照されるプロダクトに関する情報を含む. D_act に格納する情報は, 開発手法分析で作成された DFD から容易に得ることができる. 一方, 出力データクラス D_phase は,*Algorithm1* で識別されたフェーズを表し, フェーズで作成されるプロダクト, 及び, 参照されるプロダクトに関する情報を含む. *Algorithm1* では, まず, 作成するプロダクトに基づいて作業項目を分類する. 即ち, 同一プロダクトを作成する作業項目を要素とする集合を作成しフェーズとしている. 次に, 各集合の要素である作業項目の参照プロダクトを調べ, 集合間の順序関係, 即ち, フェーズの実行順序を決定している.

Algorithm1 を構造化開発手法に適用した結果を図4に示す（ベースラインプロダクトについては後述する）. この例では,5 つのフェーズ Analysis,Logical Design, Physical Design,Programming,Testing が識別されている. また,5 つのフェーズの順序関係は単純な直列関係であり, 構造化開発法に対しては Waterfall プロセスモデルが適している, という従来の考え方と *Algorithm1* によるフェーズ識別は矛盾していないことが分る.

(3) ベースラインプロダクト識別

アルゴリズム *Algorithm1* でフェーズ分けした場合, あるフェーズで作成されたプロダクトは, それ以降のフェーズでは作成も更新もされないことが保証されている. 従って, プロダクト全てをベースラインプロダクト, 即ち, マイルストーンにおいて作成完了をチェックするプロダクトとすることは可能である. しかし, 本フレームワークでは, 作成されたフェーズ以降のフェーズで参照されるプロダクトのみをベースラインプロダクトとする. ベースラインプロダクト識別のための入力データ構造, 出力データ構造, 及び, アルゴリズム *Algorithm2* は次の通りである.

```
Input://this is the phase List generated
```

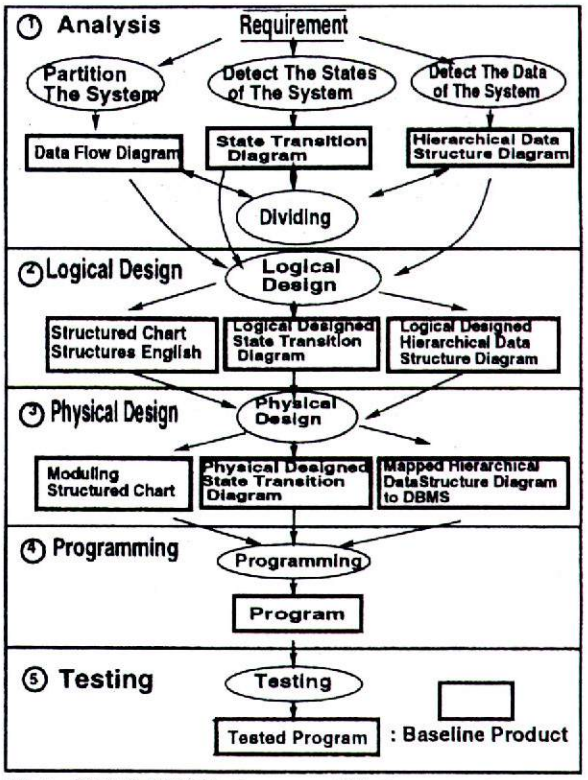

図 4: 構造化開発法のフェーズと Baseline Product

```
D_phase Dev_process;
Output:
D_milestone Dev_milestone;
class D_milestone{
   DATE check_date;//date of milestone
   CPro Pro_baseline;
   D_phase *check_phase; }
Algorithm2:
for (int i=0;i<Dev_process.GetLength();i++)
{D_phase M;
 M = i-th member of the Dev_process List
 for(int j=i+1;j<Dev_process.GetLength();j++)
 { D_phase S;
   S = j-th member of the Dev_process List;
   D_milestone new_mile = new(D_milestone);
   new_mile.check_phase =
     &(i-th member of the Dev_process List);
   if((M.Pro_out ∩ S.Pro_in) ≠ φ)
     new_mile.Pro_baseline.AddList
      (M.Pro_out ∩ (Member of S.Pro_in);}}
```

ここで, 入力データクラス D_phase は,*Algorithm1* の出力データクラスと同じである. 即ち,*Algorithm2* は *Algorithm1* の出力を入力として受け取る. 一方, 出力データクラス D_milestone は,*Algorithm1* で定められた各フェーズのマイルストーンを表し, プロジェクト計画時に設定された日付,*Algorithm2* で識別されたベースラインプロダクトに関する情報を含む. *Algorithm2* では, 全ての作業項目の作成プロダクトと参照プロダクトを比較することにより, 次フェーズ以降で参照されるプロダクトを特定している.

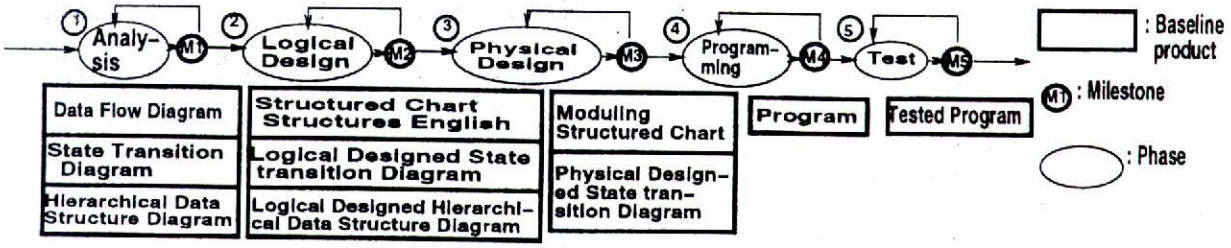

図 5: 構造化開発法向けプロセス

Algorithm2 を構造化開発手法に適用した結果を同じく図 4に示す．この例では，作業開始時に与えられるプロダクト "Requirement" を除く全てのプロダクトがベースラインプロダクトとなる．従って，構造化開発法に対しては Waterfall プロセスモデルが適している，という従来の考え方と *Algorithm2* によるベースラインプロダクト識別は矛盾していないことが分る．

(4) プロセス作成

プロセスの作成は，実際のソフトウェア開発プロジェクトにおける制約を考慮した，より具体的な作業内容の形式的記述を得る手続きである．本フレームワークでは，フェーズとマイルストーンに基づく進捗管理のための作業項目を開発手法に追加する．

構造化開発法に対して作成したプロセスの概略を図 5に示す．この図には，フェーズの実行順序と各フェーズのマイルストーンでチェックされるプロダクトが記されている．また，現段階では，マイルストーンにおけるレビューに合格しなかった場合には，対応するフェーズを再実行する簡単なモデルとなっている．

このプロセスを実際の開発プロジェクトで実行するためには，より詳細なプロセス記述が必要となる．本フレームワークでは，フェーズとマイルストーンに基づく進捗管理のより詳細なプロセス記述を OMT ドキュメント，即ち，オブジェクト図，イベントトレース図，状態図，データフロー図として提供している [4]．

4 適用例

本章では，提案するフレームワークを Object Modeling Technique (OMT)[10] に適用した結果を示す．ただし，作成されるプロダクトはいずれも，モジュールやファイルといった更に小さい単位には分割されないものと仮定する．

OMT の定義に従って作成した DFD を図 6に示す．図 6には，*Algorithm1*，及び，*Algorithm2* で識別したフェーズとベースラインプロダクトを合わせて示している．また，フェーズの実行順序のみを図 7に示す．更に，OMT 用にカスタマイズされたプロセスを図 8に示す．

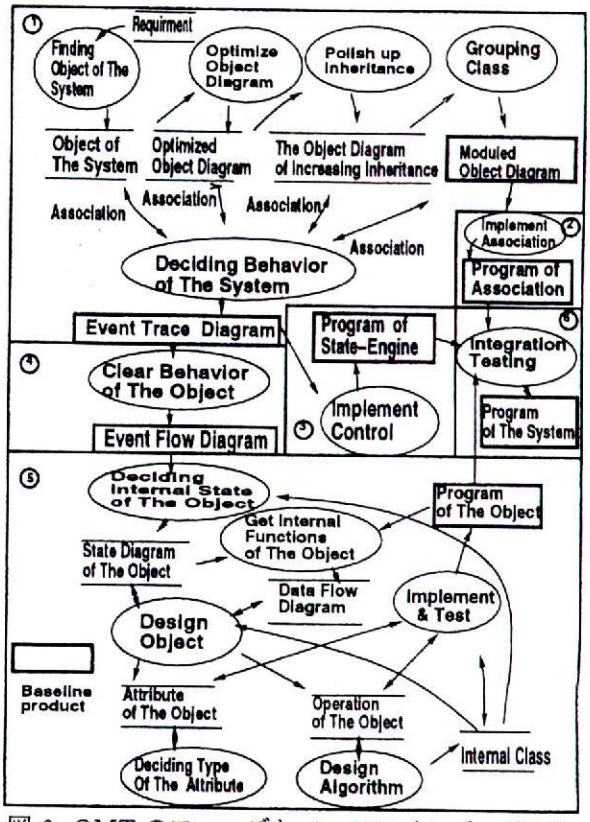

図 6: OMT のフェーズとベースラインプロダクト

図 7: OMT のフェーズ実行順序

(1) 構造化開発法に適用した場合との比較

構造化開発法に適用した場合と比較すると，作業項目もプロダクトも多く，両者の間の関係も複雑になっている（図 6参照）．その結果，構造化開発法では全てのフェーズが直列に実行されていたのに対し，OMT では 6 つのフェーズのうち 4 つが並行実行可能となっている（図 7参照）．また，構造化開発法ではほとんどのプロダクトがベースラインプロダクトとなっていたが，OMT では 16 個のプロダクトのうち 7 個のみがベースラインプロダクトである（図 6参照）．従って，構造化開発法に比べると，OMT には Waterfall モデルが必ずしも適していないことが

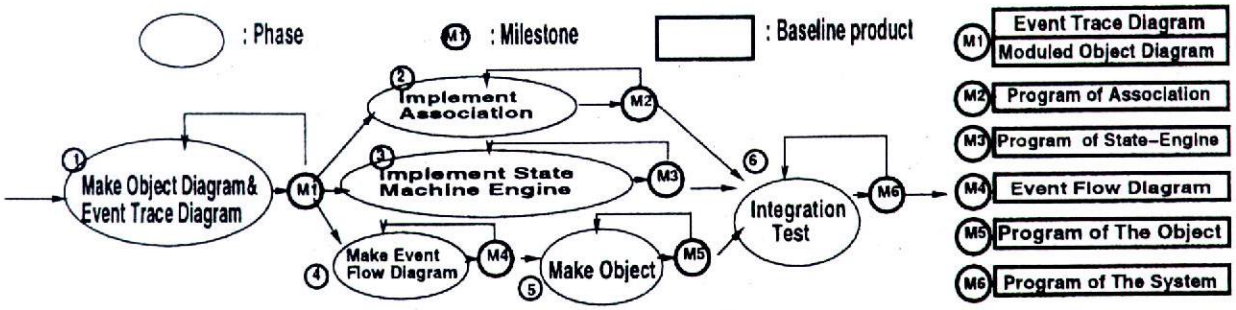

図 8: OMT 向けプロセス

分る.

(2) フェーズの数と長さの調整

図7から分るように, 本フレームワークをOMTに適用すると, 6つのフェーズのうち4つまでが並行実行可能となっている. しかし, 図6を見ると, 並行実行可能な4つのフェーズはいずれも1つの作業項目と1つのプロダクトのみから構成される比較的小さなフェーズである. 一方, 並行実行可能でない2つのフェーズ "Make Object Diagram and Event Trace Diagram" と "Integration Test" は, どちらも5つの作業項目と6つのプロダクトから構成されている. このように, 多くの作業項目やプロダクトから構成されるフェーズの実行に要する期間は一般に大きくなる. 従って, マイルストーンによる進捗管理の効果が小さくなる恐れがある.

フェーズに含まれる作業項目やプロダクトの数が増える原因の一つは, 特定の作業項目が多くのプロダクトを参照, または, 更新しており, それらプロダクトを他の作業項目が作成, または, 参照していることにある. 例えば, フェーズ "Make Object Diagram and Event Trace Diagram" の作業項目 "Deciding Behavior of The System" は,5つのプロダクトを作成, 更新しているが, そのうち "Event Trace Diagram " を除く4つのプロダクトは, 同一フェーズに属する他の4つの作業項目がそれぞれ作成したプロダクトである (図6参照).

従って, いくつかの作業項目に対して, 作成, 参照, 更新できるプロダクトに制限を加えれば, より小さなフェーズへの分割が可能となる. 例えば, 作業項目 "Deciding Behavior of The System" に対して, "Event Trace Diagram" と "Moduled Object Diagram" を除くプロダクトへの更新を禁止すると, フェーズ "Make Object Diagram and Event Trace Diagram" は, より小さな4つのフェーズに分割することが出来る. 制限を加えた結果得られるフェーズとフェーズ実行順序を図9,10にそれぞれ示す.

こうしたフェーズの数や長さの調整は, プロジェクト管理の観点からは妥当なものである. しかし, 作業項目とプロダクトの間に, 開発手法では定められていない制限を導入することになる. 従って, 開発手法の原理や原則に反することにならないか, 開発者に受け入れられるか, といった点についての検討は必要である.

5 関連研究

ソフトウェアプロジェクトの管理プロセスの評価やモデル化を目的とした方法やツールはいくつか提案されている. 片山らは,Hierarchical and Functional Software Process (HFSP) モデルを提案し, 実際の開発プロジェクトの設計プロセスからフェーズを抜き出すことに成功している [8]. しかし, プロジェクトからフェーズを抜き出す手続きの一般化は行っていない. Kellner は, モデル化ツール STATEMATE で記述されたソフトウェアプロセスの評価方法を提案している [7]. STATEMATE は, ソフトウェア開発のスケジュールを立て, 仕事量を見積ることもできる. しかし, ベースラインプロダクトの識別は考慮されていない. 平山らは, 品質, コスト, 開発期間の観点からソフトウェアプロセスを評価するためのプロジェクト階層モデルを提案している [5]. 彼らのモデルの属性は, ソフトウェア開発の現在の状態を表すために用いることが出来る. しかし, ソフトウェア開発の現在の状態に基づく進捗管理のための具体的手続きは論じられていない.

6 まとめ

本報告では, オブジェクト指向開発手法をはじめとする様々なソフトウェア開発手法に対して, 適切なマイルストーンを組み込んだソフトウェアプロセスを生成するためのフレームワークを提案した. OMTへの適用の結果, 比較的容易にプロセスモデルが得られることがわかった.

また, 作業項目とプロダクトの関係に制限を導入することで, フェーズの数や長さの調整も可能であることが分った. 今後の課題を以下に簡単にまとめる.

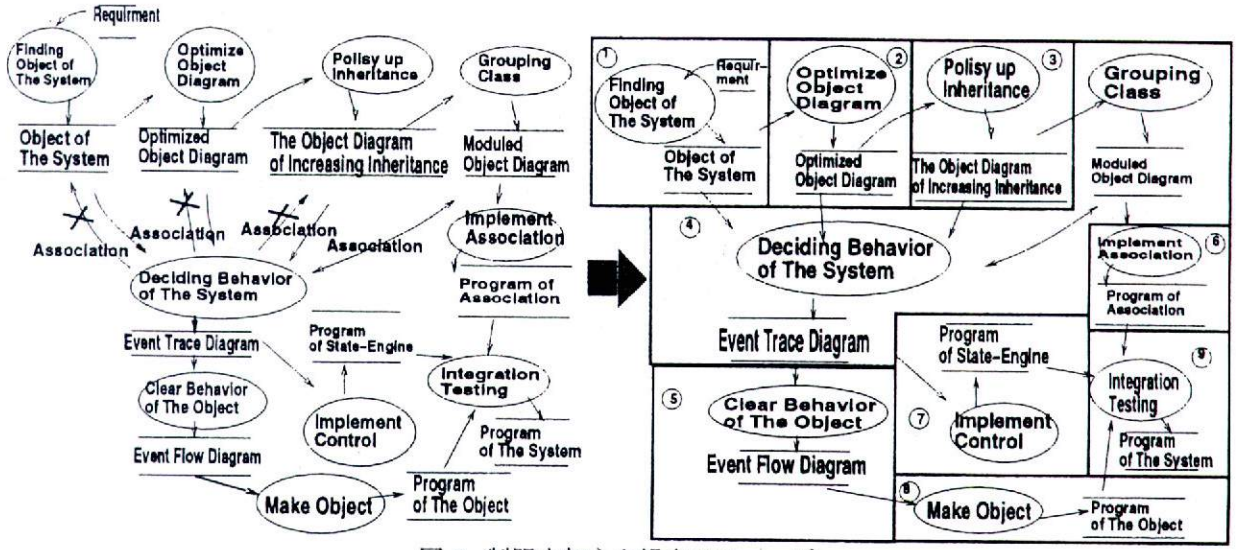

図 9: 制限を加えた場合のフェーズ

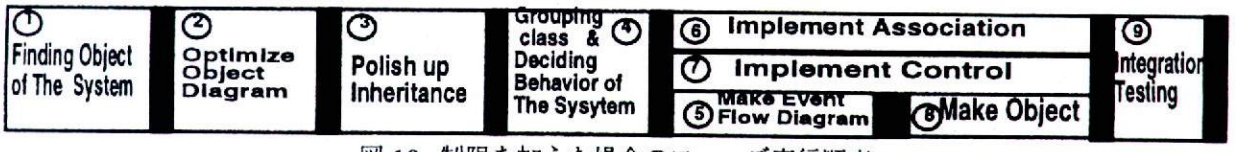

図 10: 制限を加えた場合のフェーズ実行順序

(1) 生成したプロセスの有用性を実際のソフトウェア開発で評価する.

(2) フェーズの長さのばらつきを小さくするアルゴリズムを開発する.

(3) プロダクトをより小さな単位 (モジュール, ファイル) に分割することを許す管理プロセスモデルの開発.

参考文献

[1] Clapp, J. A. : "Management Indicators," in Encyclopedia of Software Engineering, J. J. Marciniak, ed., pp.637-644, John Wiley and Sons, 1994.

[2] DeMarco, T. : "Controlling Software Projects, " Youdon Inc., 1982.

[3] 花川典子, 飯田元, 松本健一, 鳥居宏次 : "多様な開発形態に対応するプロジェクト管理モデルの構築", ソフトウェア科学会研究会報告,SP-96-3-10 (平 8-2).

[4] Hanakawa, N., Iida, H., Matsumoto, K., and Torii, K. : "A Framework of Generating Software Process including Milestones for Object-Oriented Development Method," Submitted to 1996 Asia-Pacific Software Engineering Conference.

[5] Hirayama, Y., Mizuno, O., Kusumoto, S., and Kikuno, T. : "Hierarchical project management model for quantitative evaluation of software process," Proc. of International Symposium on Software Engineering for the Next Generation, pp.40-49, 1996.

[6] Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. : "Object-Oriented Software Engineering," Addison-Wesley, 1992.

[7] Kellner, M. I. : "Software process modeling support for management planning and control," Proc. of the 1st International Conference on Software Process, pp.8-28, 1993.

[8] Mochizuki, S., Yamauchi, A. and Katayama, T. : "Analyzing and evaluating fundamental design process of checkout system for artificial spacecraft," Proc. of the Fifth Annual International Computer Software and Applications Conference, pp.507-514, 1991.

[9] Northrop, L. M. : "Object-Oriented Development," in Encyclopedia of Software Engineering, Marciniak, J. J. ed., pp.729-737, John Wiley and Sons, 1994.

[10] Rumbaugh, J., Blaha, M., Permerlani, W., Eddy, F. and Loresen, W. : "Object-Oriented Modeling and Design," Prentice-Hall, 1991.

[11] Schach, S. R. : "Software Engineering," Richard D. Irwin Inc. and Aksen Associates Inc., 1990.

[12] Sommerville, I. : "Software Engineering," 4th Edition, Addison-Wesley, 1992.

[13] van Vliet, J.C. : "Software Engineering - Principles and Practice," John Wiley and Sons, 1993.

[14] Yourdon, E. : "Object-Oriented System Design," PTR Prentice-Hall, 1994.