SEAMAIL

Newsletter from Software Engineers Association

Volume 4, Number 10-11 1989

目　次

事務局から　0

第1回　ソフトウェア・プロセス・ワークショップ特集

（1989年2月2-4日　於：機械振興協会伊豆研修所）

はじめに　片山　卓也　1

プログラム　2

参加者名簿　3

セッション・レポート　5

1．既存モデルの適切さ　松尾　正敏　6

2．プロセス・モデルの具体化　大筆　豊　11

3．プロセス・モデルのダイナミクス　中川　中　17

ポジション・ペーパー　21

付　録　89

プログラムと運営方針　90

参加者募集パンフレット　92

第2回ソフトウェアプロセスワークショップ参加者募集　94

第6回 ISPW Call for Papaer　96

事 務 局 か ら

今年の２月初めに，SEA としては初めて２つの学会（日本ソフトウェア科学会および情報処理学会）と共催で，プロセス・ワークショップを開催しました．やや遅くなりましたが，ここにその報告書をお届けします．

この話題については， ICSE でプロセス・プログラミングが話題になった直後のソフトウェア・シンポジウムに提唱者の Osterweil 教授を基調講演者としてお招きしたりして，国際的な動向をもっとも早くキャッチして追いかけてきたのは SEA であったと思います．

そうした努力が国際的にも認められて，来年の秋には，ソフトウェア・プロセスに関する国際ワークショップ ISPW-6 がついに日本で開かれることになりました（本誌９６ページの Call for Paper 参照）．具体的な場所としては，函館あたりがよいのではないかと， PC メンバーの岸田代表幹事がいま会場（ホテル）を物色中です．

このワークショップは，全部英語での高度にインタラクティブな討論で，参加者の人数も限定されていますが，より多くの人々にソフトウェア・プロセスの重要性を知ってもらおうという趣旨で，併設のシンポジウム（といってもチュートリアル的な色彩の濃いもの）が，多分東京または大阪でほぼ同じ時期に開催される予定です．御期待ください．そして、また大勢の方にまたボランティアとしての御協力をお願いすることになるでしょう．よろしく．

だんだん年の瀬も近づいてきました．お忙しいとは思いますが，すでにお手元に御案内が行っている２つのワークショップ（テクニカル・マネジメントおよびソフトウェア・プロセス）への御申込みと，ソフトウェア・シンポジウム'90 への論文の御応募もお忘れなきよう！

さて，年内に発行予定の次号は， SIGENV からのレポート，秋のセミナー・ウィークからいくつかのセッションの報告，そして前号に引き続いて大神原さんの力作「シリコンバレー・ガイド」など，盛り沢山の内容です．年明けには，秋に行われたイベント（教育WS や若手の会）の報告なども上がって来るでしょう．この分だと，来年はそれほど船便シンドロームに苦しまずにすみそうです（ホントカナ？）．

はじめに

ソフトウェアの開発や保守の過程を広い意味での計算プロセスとして捕らえ、その形式的記述や解析、あるいはその実働化メカニズムについて考えるソフトウェアプロセスに関する研究がソフトウェア工学研究者や実務家の強い関心を集めている。ソフトウェアプロセス研究の一つの目標はそれを通してソフトウェア開発過程の明確化や改良、ソフトウェアの品質向上などを計り、またその実働化メカニズムの解明によりソフトウェア開発環境やプロジェクト管理の方法論の望ましい姿を探ろうというということであろう。

ソフトウェアプロセスに関する色々な話題について形式に捕らわれずに自由な雰囲気の中で議論することを目的として第１回ソフトウェアプロセスワークショップが開催された。ソフトウェアプロセスに関する研究は、ソフトウェア工学国際会議（ＩＣＳＥ）でもここ数年の中心的話題のひとつであるし、また、国際的なワークショップ（ＩＳＰＷ）もこれまでに４回開催されている。ＩＳＰＷは米国で開催されるのに第５回に続き第６回が日本で行われることになっている。ソフトウェアプロセスに対する一般的興味に加えてこのような事情がソフトウェアプロセスワークショップを今回開催した背景である。

ワークショップは約３０人弱の参加者で正味２日（計４セッション）行われた。各セッションごとにテーマをきめてディスカッションを行ったが、レポータにはその概要を記録としてまとめていただいた。参加者には事前にポジションペーパを提出してもらったが、ワークショップ終了後にそれを書き直していただいたのが本報告書に載せられているものである。ワークショップのテーマは本年１０月に米国メイン州ケンネブンクポートで開催される第５回ＩＳＰＷのテーマと同じものとすることにしたが、これはに５ＩＳＰＷへの日本からの参加を促すを言う意味も込められていた。最初のセッションでは自己紹介をかねて参加者全員から短い話をしていただき、それに引続き第２セッションでは既存のプロセスモデルの適切さに関して、また、第３セッションではプロセスモデルからプロセスインスタンスへの具体化について、そして最後のセッションではソフトウェアプロセスのダイナミックスについての議論を行った。

ソフトウェアプロセスの研究はまだ始まってから間もなく、また、ソフトウェア開発現場でもそれがようやく明確に意識され始めた段階であり、参加者のイメージにもばらつきがあり、必ずしも同期のとれた議論ができたかどうか疑問の残る点もあるが、第１回目のワークショップとしてはまあまあの成功ではなかったのではないかと思われる。議論の内容についてはポジションペーパやセッションレポートを読んでいただくとして、私の全般的な感想を述べると、２日間の議論を通じてそれなりの問題意識の整理や分析やを行うことができ、有意義なワークショップであったと思っている。

実行委員長
東京工業大学情報工学科　片山卓也

プログラム

2/2（木）	13:00 - 13:30	受付
	13:30 - 17:00	オープニング・セッション 参加者全員のショート・プレゼンテーション（各５分） Chairman: 片山
	17:00 - 19:00	夕食休憩
	19:00 - 22:00	自由討論（１） Chairman: 久保
2-3（金）	08:00 - 09:00	朝食
	09:00 - 12:00	テーマ別セッション（１） －Suitability of Existing Models－ Chairman: 岸田，Reporter:松尾
	12:00 - 14:00	昼食休憩
	14:00 - 17:00	テーマ別セッション（２） －Instantiation of Process Models－ Chairman: 落水，Reporter:大筆
	17:00 - 19:00	夕食休憩
	19:00 - 22:00	自由討論（２） Chairman: 二木
2-4（土）	08:00 - 09:00	朝食
	09:00 - 12:00	テーマ別セッション（３） －Dynamics of Process Models and Model Instances－ Chairman: 片山，Reporter:中川

参加者名簿（５０音順）

No	氏　名	所　属	委員
1	阿草　清滋	京都大学 工学部 情報工学科	○
2	鯵坂　恒夫	京都大学 工学部 情報工学科	
3	井上　克郎	大阪大学 基礎工学部 情報工学科	
4	上原　憲二	三菱電機 情報電子研究所	
5	臼井　義美	日本電子計算 情報システム営業部	
6	大岩　　元	豊橋技術科学大学 情報工学系	
7	大槻　　繁	日立製作所 システム開発研究所 第２部	
8	大筆　　豊	東芝 システムソフトウェア技術研究所	
9	落水浩一郎	静岡大学 工学部 情報工学科	○
10	梶山　雄三	日立製作所 大森ソフトウェア工場	○
11	片山　卓也	東京工業大学 工学部 情報工学科	○
12	福田由紀雄	東芝 システムソフトウェア技術研究所	
13	岸田　孝一	SRA 専務取締役	○
14	北　　英彦	沖電気工業 総合システム研究所	
15	久保　宏志	富士通 システム本部 パッケージ企画統括部	○
16	熊谷　　章	PFU 研究開発部	○
17	小前　俊哉	三菱電機コントロールソフトウェア 技術統括部 開発課	
18	中川　　中	SRA ソフトウェア工学研究所（電総研に出向中）	
19	新田　　稔	SRA 関西支社（大阪大学に出向中）	
20	野村　敏次	日本電子計算 情報システム事業部	○
21	馮　　安	大阪大学 基礎工学部 情報工学科	
22	二木　厚吉	電子技術総合研究所 言語システム研究室	○
23	松尾　正敏	SRA ボールダー研究所	
24	松本　正雄	日本電気 C＆Cソフトウェア開発グループ	
25	松本　吉弘	京都大学 工学部 情報工学科（89年 1月から）	○
26	宮本　衛市	北海道大学 工学部	

セッションレポート

セッション－1　Suitability of Existing Models

セッション－2　Implementation of Proceses

セッション－3　Dynamics of Process Models and Instances

セッション - 1

- Suitability of Existing Models -

チェアマン：岸田 孝一
レポーター：松尾 正敏

このセッションは、『既存のモデルの適切さ』をテーマにしたものであるが、チェアマンは、議論を進めるためにテーマの表現を以下のようにかえることとなった。

現在行なっているソフトウエア開発に際しては、ツール、環境、組織、手法などを用いているはずである。その背後には、何らかのモデルが存在しているであろう。

それらが適切かどうか、つまり、現状のソフトウエア開発がうまくいっているか ？ このワークショップ参加者は、何らかの問題を認識しているであろう。

したがって、『現在使っているソフトウエア開発のモデルは、ガイドしたりコントロールするのにどの程度有効であると考えるか』を議論のテーマと考えられる。

セッション全体としては、以下のようなことがらについて活発な議論がなされたといえるだろう。

(1) 保守作業とプロセスモデル

(2)『硬い』アプリケーション領域でのプロセス・モデル

(3)『軟らかい』アプリケーション領域でのプロセス・モデル

以下は、上記のそれぞれについて、主要な意見をまとめたものである。

(1) 保守作業とプロセスモデル

この議論は、

『保守を explicit にガイドするモデルは存在するか？』

また、

『保守と言うことで特別にモデルを作成しているケースがあるのか？』という質問から始まっている。

ＯＳ担当部門と通常のアプリケーションの部門では、保守作業の必要性が異なっている。ＯＳ担当部門では、いわば保守作業的な性格が強いようである。

このＯＳ担当部門では、開発作業の繰り返しともいえる状況である。ここでは、再利用を行なうことがおおいといえそうである。

これは、スパイラル・モデルの変形のようなものであらわすことが出来そうである。ただし、作業をコントロールするという面では有効でありそうだが、作業をガイドするという面では、明確ではない。

作業の性格上、顧客との関係で開発作業のみしか行ってない組織と、メーカーのＳＥ部門、ユーザのＳＥ部門とでの作業の相違による、作業プロセスの違いがある。(開発作業中心 vs. 保守作業中心)

保守に関連したプロセス・モデルという意味では、 Lehman の E-Program は保守も含んでいることになろう。プログラムは（広義の意味で）アプリケーションにエンベッドされているのでアプリケーションの変化とともにプログラムも変化せざるを得ないという考え方である。

E-Program の考え方からすれば、最初は、『開発』作業で、二回目以降は『保守』という考え方はおかしいといえる。

Lehaman の E-Program の考え方が、世の中に浸透していないのは、多分、プログラムがアプリケーションにエンベッドされるということが、考え方としては理解されたとしても、そこまで考慮したプログラムの作り方がなされていないためではなかろうか。

Jackson のＪＳＤはアプリケーション全体をモデル化し、ソフトウエアで実現する部分を明確にしている数少ない例であろう。それまで、ファンクショナル・スペックを記述すれば済むという考え方から、アプリケーションの世界の中にエンベッドするソフトウエアとその回りの世界という考え方をしている。ＪＳＤは、E-Program の考え方の影響を受けているのかも知れない。

ソフトウエア・エンジニアリングのコミュニティとデータプロセッシング・コミュニティの違いが明らかに存在しており、ソフトウエア・エンジニアリングのコミュニティにおいては、問題が数多く存在している割には、保守のプロセスはフォーマルに考えられていないようである。

water fall モデルをもとに、稼働中のシステムの性格・情報を考慮してモデルを変更し、作業を行なっているケースもあるようである。保守の形態によって、トラブルシュートのやり方などがかわっており、 water fall というよりも、フラワー・モデルとでもいうべきものになっている。

ここでの議論から言えそうなことは、ここでは、開発作業と保守作業との対比であったが、作業性格の相違が、対応するプロセスモデルを異なったものにするのではないだろうか、と言うことである。

(2)『硬い』アプリケーション領域でのプロセス・モデル

『硬い』という表現は、たまたまケース・スタデイとして取り上げられた対象のアプリケーションが、プラント関係のソフトウエア開発であったからでなく、作業プロセスが比較的明確にされているということから、開発プロセスが固定されている意味である。

話題の中心となったのは、東芝における『Unit Workload』を用いた開発プロセスである。
基本となる考え方は、前提として、ドキュメントの体系が詳細に定められており、それぞれのドキュメントを生成するのを Unit Workload として定めている。つまり、スペックを与えてドキュメントを作成しており、ドキュメントは Unit Workload の産物である。

Unit Workload を実行した結果として現れる属性として、生産性があり、コストや LOC(Lines of Code)、また、管理的な面からは管理指標が設けられており、、生産性目標値、プログレス・スケジュール、制約条件、リソース(人、ツール、再利用ソフト)などである。これらを、一枚の用紙に記述し、作業担当者に渡すことになる。

原則として、一つの Unit Workload は一人の担当者に渡される。 Unit Workload を与えられる一人の担当者にとっては、全体像を知らなくても、作業が可能なようにしている。複数での作業には、他の Unit Workload とのコミュニケーションを行なうように指定してある。このような一つの Unit Workload は、1 週間から 3 週間程度の作業である。通常の作業では、 Unit Workload の数は約 50 程度であると思われる。

プロジェクト・プランニングは各プロジェクトのマネージャーに任されている。 Unit Workload プランニングがおこなわれ、Unit Workload ネットワークを考える。 (実際は、スタッフが行なっている) 全体の制約 (生産性など)は、与えられており、 PERT チャートを用いて計画を立てている。

Unit Workload のネットワークが作られることになるが、通常は、過去の経験から、このような開発作業の Unit Workload のネットワークはこうなるであろうというのがほぼわかっている。

この作業における計算機での支援は、 PERT 図作成、ガント・チャート出力 (出来高・予定、管理指標との差、個人の accomplishmentなどを含む)がある。PERT図が途中で変更された場合、整合性のチェックはあまりなされていないため、あまり変更がない場合はうまくいくが、大きな変更が生じた場合にはうまくいかない場合がある。

これと同様に、 Unit Workload をわければわけるほど、ドキュメントの修正や、ドキュメントとコードとの整合性などの問題、フィードバックなどに関しての対処が難しくなりため、このような事態が起こりにくいアプリケーションを対象にしている。

また、仕様変更や機能変更が頻発するのでは、このような方法は効果が薄く、比較的安定していなければ有効でない。したがって、要求定義の段階が非常に長い。例えば、2 年半のプロジェクトでは、約 1 年を費やしている。従って、Unit Workload を用いる時にはほとんど変更がない。

要求仕様レベルでの Unit Workload は下のレベルに比較して作業単位が大きくなっている。実際には、要求定義の場面では、おなじ Unit Workload を何度もくりかえしていることになる。例えば、I/O リストを作成する時には、数万点の入力があり、それらの上限値を決定したりするのにかなりの時間を費やしている。その場合には、いくつかの作業を行き来しながら行なっている。 PERT 図にこのような作業の繰り返しは、PERT 図には書かれていない。また、要求仕様レベルでの Unit Workload は下のレベルに比較して作業単位が大きくなっている。

以上が、『Unit Workload』を用いた開発プロセスの概要である。これまでにいたる経緯や考え方は次のようなものである。

従来は全て手作業だった。このようなことを手掛け始めたのは、昭和 35 年頃からであり、その歴史の中で現在の方法が浸透していき、計算機で支援するようになってきている。これがベストであるとは思っていないが、組織の中では適切であるという認識がある。

この方法は、今までのものを機械化したものであるだが、全体で約 2000 人程で作業を行なうには、何らかの機械化を行なうしかない。 100 人から 200 人からなるプロジェクトがいくつもあるという状況で工程管理を行なうには、このようなものがないととても出来ない。つまり、何らかの機械化を行なわないととても管理は出来ない。現状ではこのような方法を採っている

現在の運用のや方や課題に関しては、次のようなことといえそうである。

工程別に見た場合、関わっている人数は、要求仕様レベルでは、ＳＥの数も限られており、10 人前後であり、その後では、70 から 80 人程度である。要求仕様レベルでの Unit Workload は下のレベルに比較して作業単位が大きくなっていることから、このレベルでは、その後の作業と比較して、管理は弱いといえる。

また、比較的小規模の場合でも、例えば、 10 人前後の作業でも、 Unit Workload を用いて行なっている。

現在の方向としては、 Unit Workload の中の作業をどのように行なうかの手順書の作成を奨励している。

以前の手作業による状態と、計算機化された後の状態を比較してみると、 Unit Workload の切り方はかわらないが、チャート類を手で書いていたものが、計算機を用いて書くようになったりした。また、この背景としては、アプリケーションを作成するに当たって、それぞれの業種毎にインフラストラクチャー（プログラムの構造やモジュールに当たるもの）が存在している。原子力プラントや火力プラントでは、その完成度が高く、50 - 60 % の再利用率があり、鉄鋼では、これらよりも低く 20 - 30 % 程度である。作業プロセスが固まっている所からこのような方法が考えられるのであって、例えば、プロトタイピングが必要とするような場面では、まだ難しいと思われる。

『Unit Workload』を用いた開発プロセスとプロセス・モデルの関連については、以下のようなことがいえるだろう。

リクワイアメントにかなりの時間を費やしているということは、 Water Fall モデルに忠実に従っているともいえる。通常は、リクワイアメントを中途半端に作業して、あとで問題を起こしている。このケースでは、リクワイアメントに充分時間をかけて、手戻りがないようにしてうまくいっているようである。

先ほどの、Lehman の E-Program で、アプリケーションのドメインが人間を含んでいるかどうかの違いがある。この場合は、ドメインがかなりハードウエア中心である。また、プロセス・モデルは、どのようなドメインを対象にしているかで異なってくるといえるのではないだろうか。つまり、『硬いドメイン』と『軟らかい』ドメインがあり、それぞれ異なったプロセス・モデルがなければ対応出来ないのではないだろうか。

この例は、プロセスの規定はあるようだが、それらに対するモデルは次のように考えられるのではないだろうか。

まず、メタモデルとして {PERT, 標準ドキュメント} の組合せが考えられ、アプリケーションの特性を考慮して、それぞれのプロセス・モデルが考えられる。各プロジェクトに対しては、このプロセス・モデルをインスタンシエイトしたものとみなすことが出来るであろう。

経験を積むことにより、プロセスを比較的明確に定義出来ているケースであるだろう。したがって、Osterweil のプロセス・プログラミングの考え方を反映したプロセスの把握が比較的容易に出来る例であった気がする。

(3)『軟らかい』アプリケーション領域でのプロセス・モデル

次に、ここまでの議論の対象にしていたケースと異なり、経験にもとづいて、同じようなプロセスを繰り返すことが難しい場合が取り上げられた。

最初に、現状分析から作られるプロセス・モデルと何らかの本質をとらえて作り出したプロセス・モデルとの比較が出された。

プロセス・モデルを考える際に、二通りの方法が考えられる。一つは、現在行なっている作業を分析し、その中からプロセスを見つけ出す方法があるだろう。ここでの問題は、このようにして考えられたプロセス・モデルが本質をとらえたものになっているかであり、むしろ、現状にひきずられて過ぎているのではという疑問が生じる。二番目の例として、自動作曲を考えてみる。音楽自体は、人間にある種の快感を与えることが目的であるといえる。音源と人間に適した音の組合せなどをつかめば、自動化出来るモデルが作れるだろう。そうして、このモデルを用いて作曲した音楽を人が聞き、そこからのフィードバックを行なって、モデルを適切なものとすることができるであろう。このことは、ソフトウエア・プロセスを考える際にも、適用できるのではないだろうか。つまり、プロセス全体として、いわばあるべき姿や本質を探りだし、それをモデル化することである。

また、人間を含んだシステムを考える際にも、二通りの方法が考えられる。現状の人間の動きを正確に記述してモデルを作る方法、そして、目的に最適なモデルを考えて、人間の動きを従わせる方法があり得るであろう。

現実的には、現状の分析を行ない、組織や人間のかかわりを考慮しながら、人間と機械が組み合わされたものとして（自動化を進めると言う側面から）プロセスを変化させて行く方が良いのであろうという考えも示された。

『硬い』または『軟らかい』に関した議論から、『軟らかい』ものに関して（要求分析）、Ｋ－Ｊ法やＨＣＰチャートの有効性についての議論に変化した。また、プロセス・モデルに関しては、Water Fall モデルで充分であるとの考えも示された。

軟らかいドメインとはどんなアプリケーションが対象となっているのか、と言う問いに対して、制御系のアプリケーション開発のための開発支援用ツールの開発プロセスが議論された。このような開発では、『硬い』アプリケーション領域でのプロセス・モデルのように、かなり詳細に定式化するのは難しいようである。

例として出されたプロセスは以下のようなものであった。

(1)要求が発生しても、それが実際に何を要求しているかが不明確である。従ってツール開発者が求められているものを推測する。

(2)実際の仕事のやり方を見て、まずマニュアルを作り、要求とてらし合わせる。

(3)相手に渡して、レビューをさせる。この結果を参考にして、要求を固定する。

(4)仕事の組み立てをする。（作業とその順序を定める。GANT チャートを作る。）

(5)プロタイプを作る（または変更）。

(6)その後、デモを行ない変更希望をえる。

その後、開発者とユーザの間で(5)、(6)を繰り返し、ユーザと開発者との間で合意がとれれば、開発の完了となる。

一番最初に、プロトタイプ作成を行なわないのは、社内の手続きによるところが多いようである。組織として、このようなツール作成をオーサライズするためには、作業の内容がわかるようにせねばならず、それでマニュアル作成が最初の作業になっているようである（実際は、開発計画書が作られるが、開発の背景、ツールの効果等を明示しなければならない）。マニュアルの作成は、仮想プロトタイプと考えられるであろう。

『硬い』、『軟らかい』の相違は、納期がある場合とない場合の違いから生じるのかもしれない。

ここまでの議論で話題になったモデルとしては、ウォーター・フォール・モデル、スパイラル・モデル、フラワー・モデルなどであり、その他に、最近話題となっている、プロセス・プログラミングや Perry、Kaiser の環境モデル等であった。

(4) レポータの独りゴト

今回のワークショップは、ソフトウエア・プロセスをテーマにした最初のものであったためか、現実のプロセスとそのモデル化という点においては、まだ未成熟であるとの印象であった。界で行なわれていることは、あまりモデルを前面にだしたものではなく、むしろ、現実の作業を定式化し、作業の改善、自動化を試みる方向であるようだ。一方、アカデミックの世界では、現実のソフトウエア開発作業のモデル化の試みが、まだ始まったばかりであるようである。

セッションの結果、ケースとしては 2 例のみであったが、産業界での実際のプロセスが議論され、プロセス・モデルについての関心が高まったのではないかと期待している。

来年に予定されている、日本での ISPW に向けて、プロセス・モデルに関する議論が盛んになることを望み本レポートを終る。

セッション2

プロセスモデルの具体化　（チェアマン　落水浩一郎（静岡大学））

討議記録

リポータ　大筆　豊（東芝）

1．はじめに

セッションのテーマは、第5回ＩＳＰＷ（International Software Process Workshop ）のCall for Paperの内容

> プロセスの具体化：
> a．広く適用可能なプロセスモデルの特徴は何か？
> b．特定の組織やプロジェクト固有の要求を満たすため、どのようにモデルを選び、専用化を行うか？
> c．専用化や実行可能なモデルの具体化を行うには、どのようなメカニズムが必要か？

に沿って討議が進められた。

どのセッションでも共通のことであるが、討議の目的は、参加者全員のコンセンサンスを得るような玉虫色の結論を出すことではなく、言ってみれば、アイデアの（無責任な）出し合いの場であり、この記録も変なまとまりをつけることをさける。

まず、上記a．b．c．のサブテーマに関係のありそうなことを、ポジションペーパあるいはショートプリゼンテーションで述べた人達から、あらためてショートプリゼンテーションを行なってもらい、出てきた内容から討議を進めることになったが、さてどこに行きつくことやら？

2．広く適用可能なプロセスモデルの特徴は何か？

2．1　現状の具体的なモデル例

・ウォータフォールモデル：

組織的に開発するような大規模なシステムで、同種のシステム開発の経験があり、知識の蓄積されているものは、ウォータフォールモデルをベースにモデル化でき、実行もできる。

・スパイラルモデル：

開発支援ツールのように、比較的小人数で開発を行い、ツールの使用者（ユーザ）の要望などで何度か改良を重ねるような開発形態は、スパイラルモデルとしてモデル化できる。

・個人開発モデル：

ハッカ的な人が、ソースコードをタイプしながら、仕様を次々に変更させ、プログラムを完成させるような開発形態については、具体的なモデルの提案はない。強いて分類するならフラワーモデルとなろうか。

2．2　モデルの特徴

広く適用可能なプロセスモデルの特徴、（あるいはモデルに望まれる性質）についての意見を列挙する。前提条件（モデル化の目的，対象とするソフトウェアなど）については統一されておらず、相互に矛盾があるのは否めない。

・後戻りの記述：

ウォータフォールモデルを前提とすると各工程には後戻りが許されないが、現状にはそぐわない部分もある。モデルの記述上、あるは実行メカニズムで、現実的な解決が望まれる。

・要員のわりつけ：

開発プロジェクトがスタートしても、上流工程では、少数の人間だけが忙しく、ほとんどメンバーはひまである。また、ピーク時にはプロジェクト以外からの応援を必要とする。これら要員配分の問題を記述できる必要がある。

・コミュニケーションのしかけ：

管理者はプロセス内の各アクティビティの終了状態を把握し、管理行動を起す。常に各アクティビティをモニタリングするか、アクティビティからの応答を待つかにかわらず、モデルとして、そのコミュニケーション方法の記述を必要とする。

・保守プロセスも含める：

新規開発は、初期値零でライフサイクルを回すこと、保守は初期値として、実在のプロダクトを持ってライフサイクルを回すことと見なせる。現状では保守という名のもとで行われている開発が多いので、モデルの記述は、保守プロセスも含めたものにすべきである。

・プロセス自身をメンテナンスできること：

計画された通りにプロセスが進むことはまれであるので、実行時、プロセス自身をメンテできるメカニズムが必要である。

・ユーザとのインタラクション：

プロセスの進捗状況、プロダクトの状況など、ユーザとの確認をしながら進めることが多く、プロセス記述の中にユーザとのインタラクションを記述する必要がある。

・アクティビィティ間でのインタラクション：

あらかじめ想定できるインタラクションについては、モデルの中に記述できるが、実行時必要になるインタラクションの取り扱いについては記述できない。実行メカニズムで吸収するか、メタな記述（どのアクティビティにも共通なもの）として吸収するかが問題である。メタなルールとしてすら記述できないものは、やはり人間が介在せざるを得ないのか。

・明解な記述：

モデル化の目的にもかかわる項目であるが、モデルは明解に記述する必要がある。明解なという表現には二通りの意味があり、一方は、形式的で厳密な記述であること、他方は、簡明であること、である。簡明という意味は、モデルとしては何らかの抽象化が行われており、具体的に実行するとき、インスタンシェーションにより組織やプロジェクトの特性に適合させる。

2．3　モデル化の目的

順序が逆になったが、モデル化の目的について考えられるいくつかを列挙しておく。もちろんコンセンサスのとれたものではないし、またとれるものでもない。

・管理の手段：

多数の人間で組織的に納期、品質、コストを守りながら開発するためのガイドとなる。

・自動プログラミングへの過程：

自動プログラミングのを行うため、モデルから人間を除外し、人間系による管理を除外する立場をとる。人間を含めざるを得ない要求仕様の段階はモデルの範囲に含めず、厳密な形式仕様が記述されてからがモデル化の対象である。

・コミュニケーションの手段：

開発プロセスをモデルとして記述することにより、ノウハウの伝達が容易になり、かつプロセスの再利用が可能である。

・モデルの分析：

モデルを形式的に記述することにより、ソフトウェア・プロセスの分析が可能となり、プロセス改善の手段となる。

・開発環境の改善：

モデルの分析、実行を通じて、開発環境の改善につなげる。この場合環境とは広義な意味を持ち、設備環境、管理方法、ツールも含める。

・プロセスの自動実行：

プロセス・プログラミングの考え方であり、ソフトウェア・プロセスそのものをプログラムとして記述し、実行させる。

3．モデルの選択と専用化

特定の組織あるいは、プジェクトの要求に合うようなモデルをどのように選定し、専用化させるかについては、まだまだ深く議論するまでに至っていないのが現状である。

3．1　モデルの選択

各企業が制定している標準開発工程マニュアルは、現状ではほとんどウォータフォールモデルに基づいている。

論文の形で提案されているスパイラルモデルやフラワーモデルなどについては、実質的には実行されているが、まだその得失や運用方法については、明らかになっているわけではない。プロセスモデルをはっきり意識した試行あるいは、実施に関する報告が必要である。

ツール開発をしている人に、「あなたの行っている開発業務は、実はスパイラルモデルに従っているのですよ」と言っても、何のメリットもない。スパイラルモデルに従った開発なら、どういうメリットがあり、どこに気をつければ成功するかといった、もっと精度の高いカイドになってこそ効果が現われる。

どのモデルを選択するかは、現状では

- 開発対象となるシステム：分野，規模，期間
- 開発組織：経験，構成，人数
- システムへの要求：製品開発／研究試作，特注／パッケージ

などを考慮し、適切なものを管理者なりプロジェクトリーダが選択することになる。

ウォータフォールモデル以外では、まだモデルをはっきりと意識して開発された例が少なく、選択基準はどうであるかに論議する段階にはないようだ。

3．2　モデルの専用化

モデルを特定の目的に専用化するのは、現状では人手でやることになる。標準開発工程マニュアルが存在していても、プロジェクト毎に、その特性に合せてプロジェクト標準とするのが現実的なアプローチである。

プロセスモデルの記述が形式的に（あるいはたとえ自然言語であろうと厳密に）書かれており、個別のシステム開発の知識が蓄積されるようになれば、より自動的な方法が可能になろう。

４．専用化や実行可能なモデルの具体化を行うためのメカニズム

４．１　プロセスモデルとその実行可能性

ここで実行可能という言葉を定義する必要がある。無意識のうちに二つの使われ方をしており、一方は、コンピュータによる自動実行の可能性を意味し、他方は、人間系を含めた運用の可能性を意味する。前者は、オスターワイルのプロセスプログラミングの立場である。もちろんこの場合でも系の中に人間が実行単位（オブジェクト）として入りうるが、メインの流れ（上位の制御の流れ）は、コンピュータが握っていると考えている。

２．３で述べたプロセスモデルの目的との関係でいうと、自動的な実行可能性もモデル化の目的の一つであると考え方がよさそうであるが、ここでは

実行可能性＝自動実行可能性

として話を進める。

４．２　実行可能なプロセスモデルのメリット

言葉の定義をしだすときりがなくなるが、４．２の標題は厳密に言うと、「プロセスモデルを、記述言語で記述して、実行可能にしたプロセスプログラムのメリット」とでも言えば良いのか。

開発パラダイムとしてのプロセスモデルと、それを実行可能な言語で記述したプロセスプログラムの区別を認識せずに議論をすると発散することだけは事実である。プロセス記述言語の持つべき性質については別途考える必要はあるが、ここでは対象にしない。

もっとも、プロセスプログラムを考えると、プロセスモデルとして実行可能性を意識したモデルを仮定することになっているだろう。

実行可能なプロセスモデルのメリットは、管理面、個人的な面、チームの交信面から次のように考えられる。

- 管理的な面：進捗管理，品質管理，コスト管理など
- 個人的な面：成果物の管理，ツールの使用法や開発方法のガイド　など
- チーム交信面：交信情報の管理や作業の同期など

４．３　具体化のためのメカニズム

いろいろのアイディアが提案されたが、そのいくつかを記述する。

- プログラムモドキの実行：

  プロセスプログラムは実行時にも変更可能であるとする。ただし、変更の方法は、それ自身に書かれているが、その方法まで実行時に変えてはいけない。その条件を満たすものをプログラムモドキと呼ぶ。

  このプロセスモドキを解くべき問題（ソフトウェア開発）に適用すると、解としてのプログラムとともに、プロセスプログラムが変更されてゆき、開発終了後に具体化（インスタンシエーション）が完了するというアイデア。

・標準プロセスの専用化：

今までの経験から、すでに標準的なプロセスが問題分野に存在するとして、問題毎の特殊性に応じ変更を行うという立場。ごく現実的なアプローチであり、もちろんその専用化は人間が行う。

・ジェネリックなジェネレータ：

汎用のジェネレータがあれば、すべて解決するという立場。但し、そのジェネレータを作るための知識が十分あるか、あるいは、その費用が、個別にインスタンスを作るより安くなるか、については疑問が残る。これが可能なら、ジェネレータを作るためのジェネレータを作るというようにどんどんメタの世界に行くことになる。コンパイラ・コンパイラが可能なら、コンパイラ・コンパイラ・コンパイラを作れば良いというのと同じ議論。

・モデル＝インスタンス：

インスタンスを得ることとプロダクトを得ることの費用は同じくらいかかり、結局問題毎にプロセスモデルを作ることになるという説。

各人が、前提条件を揃えず議論するので、まとまりがつかない。

5．おわりに

初めから期待したとおり、結論にたどりつくことなく進んだが、新しいアイデア、陳腐な繰り返しなど、出るべき意見が出ており、考え方を整理する上で参考になろう。

議論のすべてをつくしておらず、またリポータの独断と誤解により忠実な記録でないことをことわっておく。また発言者の氏名を記さなかったが、気になる人は、各人のポジションペーパと対比し推測されたい。

ソフトウェアプロセスあるいはプロセスプログラミングの研究は、新鮮なキーワードをベースに（言い換えると新しい眼鏡をかけて）ソフトウェア開発を見直す機会である。開発過程をどうモデル化するか、モデルをいかに記述するか、いかに実行するかが研究課題であるが、対象の中には人間含まれるので簡単に解決する問題ではない。

２．１に記述したように、組織的なあるいはチームで開発するうなソフトウェア開発に対しては、善し悪しは別にしてモデル存在しているが、個人的な開発については、まだモデルがない。組織的な開発では、個人のふるまいが陽に現れないが、個人モデルでは、「人間」のモデル化が陽に現われることになり、面白いのだが、この議論でも提案はなかった。一つの突破口になるのではないかと思う。

最後に、本ワークショップでは、ポジションペーパに対する議論ではなく、テーマだけがあり、各人が、言った意見から次々に議論を展開するという、日本では珍しい方式が取られたが、成功したのではないかと思う。チェアマンの手腕によるところが大きい。

第一回ソフトウェア・プロセス・ワークショップ
テーマ別セッション (3)
— Dynamics of Process Models and Model Instances —
討論記録

レポータ：中川 中

1　議論の焦点

本セッションでは、ソフトウェアのプロセス・モデルやそのインスタンスのダイナミクス（動態）、とくに

1. プロセス・モデルの進化・適応の可能性
2. モデルの妥当性の判定法とモデルへのフィードバック機能
3. モデル・インスタンスの動的変更法

の3テーマを追究すべく議論がすすめられた。（以下、敬称略）

2　基調説明

本セッションの司会者である片山から、上記テーマに関し以下のような説明がなされた。

プロセス・モデルとモデル・インスタンスのダイナミクスを論ずるにあたり、以下の設問への解答の形で基本的認識を確認したい。

1. ダイナミクスとは具体的に何を意味しているか？
2. 如何なる理由でダイナミクスが必要であるのか？
3. ダイナミクスの実現には如何なるメカニズムが必要か？
4. ダイナミクスはどの程度まで実現できるか？

最後の設問には直ちに答えられるわけではないので、ここでは他の3問のみを順次みていくことにする。

2.1　ダイナミクスの具体的意味

ダイナミクスと呼ばれる概念は、次のような可能性を包含している。

1. 実行中のプロセスを修正する。
2. 実行中にプロセスの一部をインスタンシエイトする。
3. 実行結果に基づきプロセスを修正する。
4. 実行結果の分析を集成しプロセス・モデルを進化・成長させる。
5. 作成時には不明であった詳細部分を考慮しつつプロセスを実行する。

[注] 後の発言で、落水はさらに概念的な整理を行ない、

1. プロセス・モデルの進化・成長
2. モデル・インスタンスの可塑性
3. 後戻りを含む実行制御の柔軟性

の3つの視点を提案した。

2.2　ダイナミクスの必要な理由

少なくとも以下に挙げる理由から、プロセス・モデルやインスタンスの有効性を高めるうえでダイナミクスを採り入れることが望ましい。

(1) プロセスの作成時には特定しえない部分がある。すなわち、プロセスは実行 以前には不完全である。この認識を前提とするならば、プロセスの作成・実 行過程で

- 作成時に既知の部分はプロセス・スクリプトに従う。
- 作成時に具体値の決まらない部分はパラメータにしておく。
- 作成時に存在することさえ不明な部分は実行時に補完する。

という役割分担を行なうべきである。

(2) 実行時にはじめてそのプロセスのエラーや例外性が判明することがある。 そのような場合、実行中に修正を加える必要がある。

(3) あるプロセス・モデルの利用を繰り返すあいだにその欠点や不都合が明らかになる。そうした経験をそのモデルの改良に生かすメカニズムが欲しい。

(4) ハードウェアをふくめた外部環境の変化に伴い、プロセスへの要求も変化しうる。プロセスはそうした変化への適応性をもつべきである。

2.3 ダイナミクス実現のメカニズム

2.1 に記されたダイナミクスを実現するためには、言語、実行系等に以下の機能が備わっている必要がある。

(a) プロセス記述言語

プロセス記述言語はパラメータ化とパラメータの具体値による置換メカニズム、高階性、リフレクション等の能力を備える必要がある。

(b) プロセス実行系

実行系はフィードバック機構の実現のためのモニタリングや実行履歴の記録・分析、プロセス・エディタ、割り込み処理を含めた柔軟な実行制御システム等を備える必要がある。

(c) その他

そのほかに、プロセスの良否を判断する検証・検定メカニズムが必要である。

これらの諸機能のうち、関数型言語などの分野で注目を集めているリフレクションの当否を特に検討してみる。プロセス記述言語Lでリフレクションが可能である（Lは反射性を持つ）とは、Lで記述されたプロセスの実行状態を認識し変化させる機構が自身に備わっていることにほかならない。このリフレクションによってダイナミクスを実現すれば、実行過程の制御をプロセスの一部として表現できることから、著しい柔軟性が得られる。しかし反面で、あるいはまさにそれ故に、プロセスの実行とその制御という通常は区別されている概念が混交し、静的解析やセマンティクスの付与に支障をきたす恐れが大きい。いずれにしても当面の研究課題としては非常に興味深い。

3 ダイナミクスの実現例

片山の基調説明に続き、松本（吉弘）、落水、井上からダイナミクスを実現した具体例ないしその実現構想について以下の説明が行なわれた。

[注] 新田も具体例を用意していたが、前セッションで報告され、本セッションではとくに取り上げられなかった。そのさい、新田の例はモデルとそのインスタンスの生成過程の文脈で論じられたが、むしろダイナミクスとの関連で扱われるべき例を提示していたように思われる。

3.1 松本構想

松本はプロセス・プログラムの生成段階、実行段階、および実行体（系）につき、試案を述べた。

(a) プロセス・プログラム生成フェイズ

このフェイズはメタコンパイラ、コンパイラ、およびプログラム・ジェネレータから構成される。メタコンパイラは問題分野（ドメイン）に関するメタレベルの定義を解析してコンパイラを生成する。次に、こうして生成されたコンパイラは具体的な問題分野の記述とプロセス・モデルの定義にしたがってプロセス・プログラム・ジェネレータを生成する。最後に、このジェネレータが資源その他に関するデータ記述にしたがって特定のプロセス・プログラムを生成する。

(b) プロセス・プログラム実行フェイズ

上記の過程を経て生成されたプロセス・プログラムは、プロセス・スクリプトを生む母体である。実行フェイズではプロセス・プログラムに諸々の意志決定が与えられ、手順としてのスクリプトが動的に生成される。

[注] 松本は時系列的に実行過程を捉えるが、これについて片山から並列性の考慮が不十分であるとの異議がだされた。この点につき、阿草はチェックポイントを中心に考える視点から、二木は実行例の結果としての観点から、松本の定型に理解を示した。

(c) プロセス・プログラム実行体

実行体のありうべき形態は以下のように考えられる。

- 実行体は複数の並列実行可能な単位実行体(CEU)から成る。各CEUは固有の状態をもち、その状態に関する前条件、後条件の成否により起動・終結される。
- CEUの実行はそれが持つ状態の遷移で表され、要素スクリプト(ES)の連接によりもたらされる。
- CEUの実行は長期（数日、数週間）に亘ることから、マシン・ダウン等に影響されない状態保存方法がとられる。これは通常のプログラムの実行と大きく異なる考慮事項である。
- ESの機能には、端末入出力、印刷、交信、同期、後戻り、ツール起動制御、ソフトウェア成果物の構成・バージョン管理等が含まれる。

[注] CEU間での交信機能につき、ブロードキャストや動的なチャネル設定を認めるべきか否かの議論があり、二木は否定

的立場で意見を述べた。記述と実行との関係、特に予測可能な実行過程をどこまで記述に盛り込むか、ならびに記述しえない実行過程はそもそも認められるべきか、は他のセッションを含めてしばしば論争の種であった。

3.2　落水の試行

落水は静岡大学で進められているSDAプロジェクトでの試行につき説明した。

既述のとおり、落水はダイナミクスを三様に分類するが、当面はそのうちの実行制御におけるダイナミクスに焦点をあてている。プロセス記述言語としてProlog、OPS83、Smalltalkをそれぞれ論理型(ないし手続き型)、ルール・ベース型、オブジェクト指向の例に採り、記述能力の比較を行なった結果は次のとおりである。

- Prologでは、後戻りの表現が容易である反面、繰り返しの制御や並列実行の記述方法に問題がある。並列実行や後戻り点を明示するために若干シンタクスを拡張しても、なお問題が残る。
- OPS83では、前条件や後条件をルール上に明示できる利点がある。欠点は、個々のルールの組み合わせや後戻りを明示できないこと、後向きの推論を記述しにくいことなどである。
- Smalltalkでは、オブジェクト指向ならではの再利用性や、手続き的知識やオブジェクト間の依存関係を表現しやすい点が魅力であるが、後戻りが記述できないという重大な欠点を持つ。

[注] なぜこれらの言語を採択したのかが問題とされた。結局のところ、処理系が手許にあったから、が理由のようであった。

以上の考察から、プロセス記述言語はルール・ベースのプロセス・マネジャの記述を目的とし、現実のソフトウェア開発上の諸現象に対応する以下の構文を含むべきである、と思われる。

- 試行錯誤に対応する、後戻りや複数のルールの適用可能性を表現する構文
- 適用候補である複数のルールの逐次検討に対応する、制御つきの繰り返しを表現する構文
- 多視点からの検討に対応する、並列評価を表現する構文
- 並列処理に対応する、同期や相互起動制御を表現する構文
- ルール選択の戦略に対応する、実行文脈や呼びだし条件を表現する構文

3.3　井上の試行

井上は大阪大学で開発中のPDL(Process Description Language)の説明を行なった。

PDLは関数型の言語で、「システム状態」というデータ型を用意したこと、開発過程をツールの起動やウィンドウの操作等の系列と捉えていること、並列性を記述できること、を主な特徴とする。実行制御の機構としては、不完全な記述の割り込みによる処理に特色があり、それはある意味でパラメータ化をも実現している。このPDLによってJSDを記述した。

記述にあたっては段階的詳細化の手順が踏まれた。手順の全体は

- JSDの初期記述S（既与とする）がある。
- Sに後戻り情報を加え、S'とする。
- S'における成果物へファイルを割り当て、S"とする。
- S"における各作業を基本機能の単位に分割したものをS"'とする。
- S"'の各機能に実際のツールを割り当て、スクリプトが完成する。

のように示すことができる。このうち、機能分割によるS"'の生成が詳細化による。この段階的詳細化では、任意のレベルの作業は前条件・本体・後条件に三分され、その各々が特定のツールに対応付けしうるまでさらに細分される。わかりやすい例でいえば、「云々を編集する」「云々の構文をチェックする」という作業は、「云々」がファイルのように十分特定可能な対象に対応するかぎりで、特定のツールに対応付けしうる基本単位であるといえる。

[注] 例示されたスクリプトを前にして、阿草と鷲坂はスクリプトの変更権限の所在を憂慮し、スクリプトの変更の可否は人的属性としても考慮されるべきである、と主張した。人間がその一部を分担するようなプロセス（ここで話題にされているのはまさにそのようなプロセスである）のなかで自主的判断による動的変更を認めること、あるいは認める程度、については他のセッションでしばしば論争の種であった。二人の基本的興味は人間を含まないプロセスにあるので、ここでの発言は人間を考慮するとした仮定を受け入れた限りでの感想である。

[注] ここで以下なる言語がプロセス記述に相応しいかが若干議論された。記述例

により経験的に導かれる結論以外に真実の解答は得られない問題である。出席者には片山をはじめとして関数型言語に近い立場が多く、上記井上の報告が包括的な記述例の唯一であったことも手伝って、関数型を基本とすべきであるという意見が圧倒的であった。

4　今後に向けて

3. で述べられた試行結果を受けて、片山と二木は以下のように今後の研究課題をまとめた。

4.1　片山の研究

片山は属性文法に基づく HFSP の近況を伝えた。HFSP は構文木のノードの属性として入出力仕様を与えることによりプロセスを定義する。副作用はない。プロセスの記述の重要な要件として読み易さ、階層性が挙げられるが、HFSP はアクティビティを構文木を構築する形で分解するという自然でかつ正確な表現でそれらの要件を満たしている。また、HFSP は後戻り制御よりも非決定性自体を重視し、非決定的なアクティビティの記述を可能にしている。

HFSP の構文木で記述されているのは特定のスクリプトではなく、それで表現しうるすべてのスクリプトの集合である。そして、HFSP に記述されたプロセスの実行とは、そうした集合の中から動的に特定のスクリプトを選択する過程にほかならない。

HFSP におけるリフレクションは構文木自身を属性として定義することにより実現される。プロセスの動的変更では構文木の書き換えが必要になるが、現在では部分木の計算アルゴリズムが知られており、効率上の問題は楽観的に捉えてよいと思われる。

> [注] 冒頭で片山が指摘したとおり、リフレクションは強力なるが故の危険性を持つ。ここで改めて、二木と片山がリフレクションの導入によって記述の透明性が失われることを懸念した。ただし片山の意見は、利用することの当否はともかく、機能として組み入れることを支持している。

4.2　二木の研究

二木はプロセス記述等を従来の概念に還元するという発想を示した。すなわち、

- プロセス・モデルは計算モデルである。
- プロセス記述は（環境を含めた）言語仕様である。
- プロセスの実行は言語を用いての実行のログである。

というものである。この立場では、言語の利用者マニュアルを記述することがプロセスを記述することに外ならず、たとえば OBJ* のような代数型の言語の仕様を形式的に（望ましくは OBJ* 自身で）記述する試みはプロセスをめぐる諸々の論点に対する一つの解答を提示する、といえる。

> [注] この大胆なテーゼは他の出席者には理解しがたかったようである。その当否が論ぜられることは残念ながらなかった。

5　おわりに：感想

本セッションはプロセス・モデルやそのインスタンスに関する一般的な議論を踏まえた上で、それらの動態を論ずる場である。したがって、中途半端な議論に終止したこれまでのセッションを振り返ると、果たして本セッションは成立するかという疑念が開始前にあった。予想以上に本セッションにまとまりがあったのは片山の進行の上手さによるところが大きい。とりわけ、ダイナミクス実現のメカニズムを抽象的に議論せずに、プロセス記述言語に望まれる機能という枠組で具体的例を中心に討論したことで焦点の確かなセッションとなったといえる。

ひとつ気になったことは、プロセス・モデルの進化にふれた議論がまったくなかったことである。これはこの問題が本質的にはプロセスの動的変化や実行制御と何ら関わりがないことに起因している。実現上の類似性、たとえばモニタリングのメカニズムやメタレベルの操作の必要性、は認められるにせよ、モデルの進化と個々のプロセスの可塑性・適応性とは同じダイナミクスという言葉で含意されるにはあまりに異質の概念である。より不遜には、今後はモデル論とプロセス論は峻別されるべきである、といえる。

それはともかく、プロセス・モデルの進化というテーマが避けられたのは、ソフトウェア・プロセスの話題を取り巻く現状では、タイム・スパンの長い進化のテーマを建設的に議論する素地はなかったというべきか。本セッションで取り上げられた実現例からはそう結論される。

もうひとつ気がかりな点は、用語の不統一である。あるいは、人により、用語の濫用であり、濫造である。この点はワークショップ全体に関わることでもあり、ここではさらに言を継がないが、本記録における用語上の粗放さはレポータの性格を反映しているばかりでないことを明言しておきたい。

ポジション・ペーパ　目次

阿草　清滋　22
鯵坂　恒夫　25
井上　克郎　31
上原　憲二　33
臼井　義美　38

大岩　　元　40
大槻　　繁　42
大筆　　豊　43
落水浩一郎　46
梶山　雄一　47

片山　卓也　48
福田由紀雄　52
岸田　孝一　57
北　　英彦　58
久保　宏志　59

熊谷　　章　60
小前　俊哉　61
中川　　中　65
新田　　稔　68
野村　敏次　71

馮　　安　72
二木　厚吉　75
松尾　正敏　78
松本　正雄　79
松本　吉弘　81

宮本　衛市　85

プロセスプログラミングとメタプログラミング

京都大学工学部　　阿草清滋

1．はじめに

プロセスプログラムはソフトウェアを対象とするソフトウェアであり、その意味でメタソフトウェアである。ソフトウェアという言葉は広い意味を持つので、ここではその中でプログラムを考えるということで、メタプログラムを考えよう。メタプログラムには多くの意味があると考えられるが、ここではソフトウェア開発の効率化を目的としたものを考える。

プログラム開発の効率化はプログラムを人手によって開発する部分を極力小さくする方法と、人手で開発するとしても開発の方式を定式化して属人性を排除し、いわゆる工業化を図る方法の二つの方法が考えられる。この両者は互いに関係しており、属人性の排除が可能ならば自動化への足掛かりを与えることができる。プロセスプログラミングはソフトウェア開発の作業過程の定式化を目指したものと言えるが、これはまたソフトウェア自動生成の一ステップと言える。

2．メタプログラミング

ソフトウェア開発の自動化システムを考えるとソフトウェアの開発をアルゴリズミックに与えるものと、知識工学的に知識とその処理系として実現するものと二つのアプローチが考えられる。一般に複雑なシステムほどアルゴリズミックな表現は難しく、我々の知っている事実を事実として提示することでシステムを実現できることが望まれる。もちろん計算機のプログラムはアルゴリズムの表現であり、アルゴリズミックな表現の方が効率もよいので、知識工学的なアプローチの経験からアルゴリズムを獲得することは必要である。

ソフトウェア技術者の能力として、経験と勘と度胸と言われるが、この中でも経験の重要さが強調される。経験は知識であり、その適用で新しいソフトウェアを開発していると言える。

どのような形で知識を蓄積すればよいかであるが、プログラム再利用技術に見られるように、標準的なプログラム部品の形での蓄積とプログラム設計過程など開発手順の蓄積の二つの方法があるとされている。これはプロセスプログラム的

なアプローチと知識をプログラム部品とするアプローチにほかならない。

どの計算機でもライブラリとしてすでに開発されているプログラムセグメントを利用することは行なわれている。この意味でプログラム部品の組合せで再利用効果を追及することはより現実的なものと言える。

プログラム開発の自動化のためには、プログラム部品の整備とともにそのプログラム部品の組み合わせ方をプログラムとして実現する。これにより、仕様記述を入力にして自動的にプログラムを生成できる。

このアプローチは一般の問題を考えようとすると非常に難しいものとなるが、ある特定のアプリケーション分野に限ることにより、このようなメタプログラミングは現実的なものとなる。これにはパラメタライズされたジェネリックなプログラムにカスタマイズパラメータを入力する方法と、生成系とプログラム核を用意し仕様記述に従って生成系がプログラム核をカスタマイズする方法が考えられる。

このようなメタプログラミングのための基礎を確立のためには仕様記述のためのモデルを与えること、そのモデルに基づいてプログラム核やプログラム部品を検索すること、カスタマイズのためのプログラム変換を行うこと等についての研究が必要である。これまでの我々の研究を基に仕様化技術、部品合成技術の基礎となる理論、および技法を明らかにし、メタプログラミングの手法を確立しなければならない。特に実用化に必要なカスタマイゼーションに伴って必要とされる最適化技法について研究が求められている。

3．メタプログラムと最適化

プログラムの自動生成には二つの方法がある。ひとつは厳密に定義された仕様記述言語を用いてプログラムの仕様を記述し、これに数学的な変換規則を繰り返し適用し目的とするプログラムを得る方法で、プログラム合成として知られている。この方法は変換規則に対象問を得る方法で、プログラム合成として知られている。この方法は変換規則に対象問題の知識がなく、仕様記述にすべての情報を与えなくてはならないので、仕様記述が複雑で記述困難なものとなりがちである。もうひとつの方法はプログラミング過程をプログラムとして与えることで自動化を目指すメタプログラミングである。メタプログラミングはプログラミングを処理の対象とするプログラムであり、プロセスプログラミングはこのひとつの例と考える。

設定された問題領域ごとにプログラム核と知識ベースに当たる部品群を用意し、

利用者からの要求仕様書を入力にプログラムを生成することを考える。プロセスプログラミングではプログラム一般を扱おうとしており実現性に問題がある。そこでソフトウェアの多用性を認め、また利用者の世界での言葉を用いた仕様記述を可能とするために問題領域を設定し、その限られた問題領域でのプログラム自動化を行なう。これらの実現性はプログラム言語処理システムやレポートプログラム生成で示されている。問題領域の知識を獲得、整理する方法としてのメタプログラミングが必要とされる。これまで仕様記述のモデル化、プログラム部品のデータベース化とその検索方式、選択された部品間の結合をカスタマイズする研究などを進められてきた。これらを基に実用化可能なメタプログラミングシステムとしての統合するための研究がなされるであろう。

ソフトウェアは与えられた環境でもっとも効率の良い解法を与えるものであるといえる。このために最適化技術は不可欠であり、ハードウェアの部品化が進んでいるのにたいしてソフトウェアの部品化が進んでいないのは、利用される環境に適するようにカスタム化が求められるし、またそれが可能であるからと言える。メタプログラムの一つの例として、パラメタ化されたソフトウェアの利用に際して、このカスタム化、最適化が不十分であると利用してもらえないと思われる。このためには部品もカスタム化がより容易な形で、すなわちソースに近い形で持つべきである。

この最適化の可能性を最大限なものとするためには、部品化の過程や部品作成の意図なども部品と共に蓄積されなければならない。このことはプロセスを計算機に記録し、それを利用しなければならないことを意味し、プロセスプログラミングへの道を進むことになると言える。

ソフトウェア・プロセスとしてのソフトウェア自動生成系

鯵坂恒夫（京都大学・工学部）

1．管理指向モデルと技術指向モデル

ソフトウェア・プロセス・モデルは，ウォータフォール型モデルと操作的モデルの二つに大きく分類できる．ひと口にいえば，前者は管理指向，後者は技術指向である．ウォータフォール・モデルというとシステム工学発祥以来の古典的モデルであるが，最近注目を集めているスパイラル・モデル[Boehm86] もこちらに属する．フェーズ分けされたステージワイズなモデルで開発過程の管理によってリスクを掌握し，生産性と品質の向上をはかるものである．一方の操作的モデル[Zave84]は，発展型プロトタイピングや実行可能仕様など比較的新しい研究の流れを汲む連続進行的なモデルで，その開発過程に含まれる個々のソフトウェア開発技術自体を革新することによって生産性と品質の向上をはかるものである．記号実行系やプログラム変換・生成系などの自動化ツールを導入し，開発過程全体に及ぶ自動化の研究につらなるものとして興味深い．

これら二つのモデルはどちらがより優れているというようなものではない．ただし，現実の生産現場ではウォータフォール型モデルが望ましいとされる開発対象，すなわち青写真がしっかり引けて建築的方法で進めるべき開発対象の方が，操作的モデルの推奨される対象，すなわち対象領域自体が明確でなく彫刻的な手法によらざるを得ないようなものより多いかもしれない（表１参照[Riddle84][Blum86]）．現実に膨大なバックログと戦う生産現場を抱えている企業では，少しでも効率的なソフトウェア開

	Waterfall Model	Operational Model
Process	Stagewise, Phased	Iterative
Metaphor	System Architectur	System Sculpture
Application	High Technical Risk	High Application Risk
Intent	Management oriented	Production oriented
Prototype	Throw-it-away	Evolutional

表１　２つのソフトウェア・プロセス・モデル

発のために管理指向のモデルを早急に確立する必要に迫られているだろう．これはすなわち，開発技術の革新が遅れているために，ソフトウェア開発が基本的に人海戦術であることを露呈しているものともいえる．いかに優れたソフトウェア・プロセス・モデルが確立され，工程管理，進捗管理等がスムーズに行えるようになっても，管理の対象となる生産過程そのものが技術の乏しさ故に遅々としていたのでは意味がない．技術革新にはある程度の長期的な構えが必要である．大学等研究機関としてはこの開発技術の研究・開発に重点を置くべきであると考えている．

2．ソフトウェア・プロセスの自動化

「ソフトウェア・プロセスもまたソフトウェアである」という命題に間違いはない．ただし，ソフトウェアには二種類のものが含まれている．ひとつはいわゆるドキュメント，もうひとつがプログラムである．ソフトウェア・プロセスというソフトウェアがドキュメントであるならば，それを解釈・実行するのは人である．いかにプログラミング言語風の言語で記述してあっても，人が読んで作業をするのならばそれはドキュメント・レベルのものである．もしそれが本当にプログラムであるならば，それを解釈・実行するのは計算機でなければならない．そして，ソフトウェア・プロセスとはソフトウェア開発プロセス[Wileden86]であるから，その全体としての実行結果としてソフトウェア製品が出てこなければならない．管理データはいわばプロセス・プログラム実行時のログとして得られる副産物である．

（もちろんこのログを解析するツールがあって管理者の使用に供することができる．さらに，出力データとしての管理データの他に，目標値など入力データとしての管理データもあって，それによってソフトウェア・プロセスを変化させることも考えられる．これは一見メタ・プロセスのようであるが，現実的にはドキュメント・レベルのソフトウェア・プロセスに含まれるパラメタになるだろう．）

ソフトウェア・プロセスもまた階層的な構造をもつ．最上位モジュールは要求を入力しソフトウェア製品を出力する．これを何段か詳細化したところではまだ「簡単に書けるが意味のないもの」の段階である．あるところで急に箱の詳細化が難しくなり，「ほとんど書けないもの」に出くわす．これが自動化ツールないし自動生成系として技術開発すべきものである．ある場合，とくに上流工程ではドキュメントとしてのソフトウェア・プロセスのなかでそのいくつかのオペレータ（演算子）だけがプログラム化され，すなわちツールとして提供されることになるだろう．そのツールには人がオペレータとして生成しなければならない（すなわちプログラム化されていない）データが対話的に入力される．一方，必要なデータ（仕様記述）がそろえばバッチ的に実行が進む自動生成系が開発できる場合もあるだろう．少なくともソフトウェア・プロセスの最下流であるコンパイラやリンカはこれにあたる．

3．自動生成系の試み

操作的モデルにも種々のバリエーションがあるが，一般に２分割プロセスモデルとして捉えることができる．すなわち，ソフトウェア・プロセスにおける最初の形式的仕様記述が獲得された時点をもって，それ以前の仕様化の過程（分析過程）とそれ以後のプログラム形成の過程（変換過程）に分けられる．初期仕様からプログラムまで単一の広領域言語[Goldberg86]によって記述するアプローチ，複数の言語変換によってプロセスを進めるＰＷモデル[Lehman84a,b]，変換過程においてジェネレータを作成するメタプログラミング[Levy86]，分析過程にほとんどの工数を注ぎ込み厳密な仕様記述からプログラムを自動生成する MODELアプローチ[Prywes77,83][Tseng86]など，いずれもこの２分割モデルのインスタンスである．（図１参照）

分析過程は問題領域に属する概念やその構造，及びそれに対する人間の思考法に強く依存するため，自動化の困難な過程である（表２参照）．したがって，初期仕様をできるだけ早期に獲得しやすいものとし，変換過程の守備範囲を広げてそこに設計自動化システム，プログラム自動生成システムを投入することによって，プロセス全体としての自動化率を高めるのが筆者のアプローチである（図１(5)）．現在，初期仕様としてデータ仕様を入力し，再利用技術に基づく設計自動化システムの出力する関数仕様を経て，パラダイム変換により手続き型プログラムを自動生成するトータルな変換過程自動化システムの研究・開発を行っている．（図２参照，詳細は[Ajisaka85, 86,87]）

２分割モデルのキーとなるのは初期仕様のあり方である．初期仕様は問題領域に依存する分析過程とインプリメンテーションに依存する変換過程の橋渡しをするものであるから，その双方から「初期仕様の仕様」に対する要求がありうる．構造変換，状態遷移，帰納法，生成ーテスト法など，問題のカテゴリからそのあり方を研究する必要があると同時に，およそあらゆるアプリケーションを記述できる現在の高級プログラミング言語のもつ概念要素を高レベル化することによる初期仕様へのアプローチも検討されなければならない．

参考文献

[Ajisaka85] 鯵坂恒夫，阿草清滋，大野 豊： 関数スキーマベースを用いたソフトウェア設計自動化，情報処理学会論文誌，26巻 2号，pp.304-311，1985.

[Ajisaka86] 鯵坂恒夫，阿草清滋，大野 豊： ソフトウェア設計自動化のための仕様記述言語Ｆ，情報処理学会論文誌，27巻 7号，pp.688-696，1986.

[Ajisaka87] T. Ajisaka, K. Agusa and Y. Ohno: An Automatic Program Generation System Based on Function Schema Base, Proc. 11th Ann. Int'l Computer Software and Applications Conf., pp.214-220, 1987.

[Blum86] B. I. Blum: Iterative Development of Information Systems: A Case Study, Software -- Practice and Experience, Vol.16, No.6, pp.503-515, 1986.

[Boehm86] B.W.Boehm: A Spiral Model of Software Development and Enhancement, Proc. Int'l Workshop on the Softw. Process and Softw. Environments, ACM SIGSOFT Softw. Eng. Notes, Vol.11, No.4, pp.4-5, 1986.

[Goldberg86] A. T. Goldberg: Knowledge-Based Programming: A Survey of Program Design and Construction Techniques, IEEE Trans. Softw. Eng., Vol.SE-12, No.7, pp.752-768, 1986.

[Lehman84a] M. M. Lehman: A Further Model of Coherent Programming Processes, Proc. Software Process Workshop, IEEE Comp.Soc.Press, pp.27-34, 1984.

[Lehman84b] M. M. Lehman, V. Stenning and W. M. Turski: Another Look at Software Design Methodology, ACM SIGSOFT Softw. Eng. Notes, Vol.9, No.2, pp.38-53, 1984.

[Levy86] L. S. Levy: A Metaprogramming Method and Its Economic Justification, IEEE Trans. Softw. Eng., Vol.SE-12, No.2, pp.272-277, 1986.

[Prywes77] N. S. Prywes: Automatic Generation of Computer Programs, Advances in Computers, Vol.16, eds. M. Rubinoffand M. Yovits, pp.57-125, Academic Press, New York, 1977.

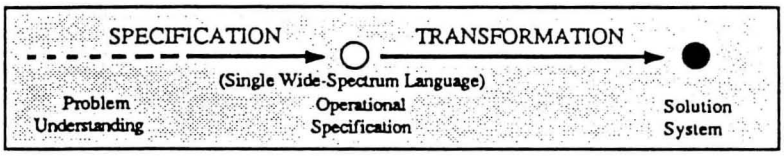
[Prywes83] N. S. Prywes and A. Pnueli: Compilation of Nonprocedural Specifications into Computer Programs, IEEE Trans. Softw. Eng., Vol.SE-9, No.3, pp.267-279, 1983.

[Riddle84] W. E. Riddle: Report on the Software Process Workshop, ACM SIGSOFT Softw. Eng. Notes, Vol.9, No.2, pp.13-20, 1984.

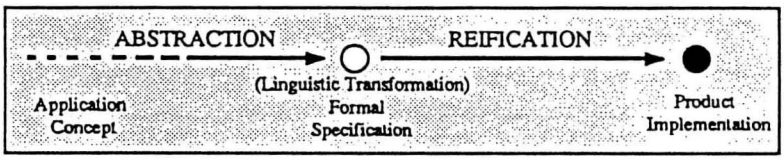
[Tseng86] J. S. Tseng, B. Szymanski, Y. Shi and N. S. Prywes: Real-Time Software Life Cycle with the Model System, IEEE Trans. Softw. Eng., Vol.SE-12, No.2, pp.358-373, 1986.

[Wileden86] J. C. Wileden and M. Dowson: Introduction, Proc. Int'l Workshop on the Softw. Process and Softw. Environments, ACM SIGSOFT Softw. Eng. Notes, Vol.11, No.4, pp.4-5, 1986.

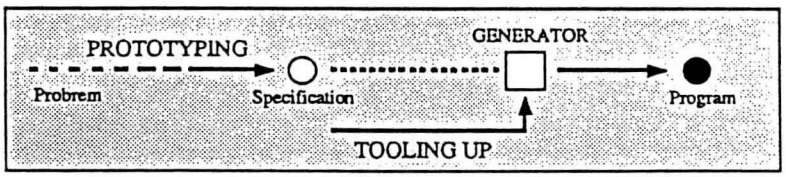
[Zave84a] P. Zave: The Operational versus the Conventional Approach to Software Development, Comm. ACM, Vol.27, No.2, pp.104-118, 1984.



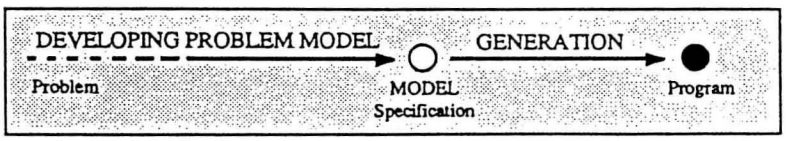
(1) Operational Approach



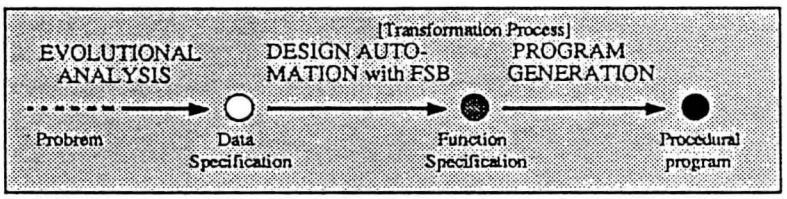
(2) PW Process Model



(3) Meta-programing



(4) MODEL Approach



(5) Fauna Approach

● Excutable
○ Not Executable
◍ Capable of Abstract Execution

図１　操作的モデルのインスタンス

	Software Process	
	Analysis Process	Transformation Process
Nature	Evolutionary Process	Rewriting of descriptions
Dependency	Human Characteristics (Way of Thinking & Communication)	Behaviour of Computer & Program
Result	Formal Specification	Program Code
Automating	Interactive Tools	Automatic Generator

表２　２分割モデル

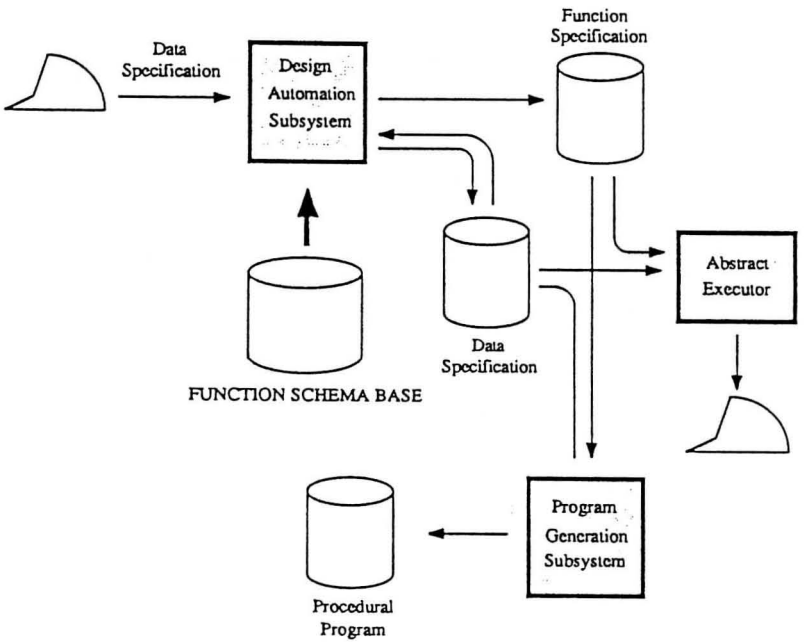


図２　ソフトウェア設計自動化・プログラム自動生成システム

第一回プロセスワークショップ（平成元年２月２日～４日、伊豆）

ポジションペーパー　　大阪大学基礎工学部情報工学科　　井上克郎　2/20/89

1.　発表内容

大阪大学鳥居研究室で開発を行なっているプロセス記述言語ＰＤＬ(Process Description Language)とその処理系について述べた。

これからやろうとしている開発作業や、すでにやられた開発作業の各工程をあいまいなく記述し、他人に誤解のないように伝えることは重要である。一人の開発者が一台のワークステーション上で要求分析、仕様書作成、プログラム構造図、コーディング、コンパイル、テストなどの開発作業を行なうことを前提として、その開発過程を記述するためにＰＤＬを設計した。ＰＤＬでは、どのような作業をどのようなツールを用いて、どのような順序で行なうかを記述する。ＰＤＬは、ツールの起動やウインドウ操作、メッセージ表示などの機能を備えている。ＰＤＬはいわゆる関数型言語であり、未定義関数を用いて抽象的な開発作業の流れを記述することができ、さらにそれに具体的な情報を段階的に付加していき、最終的に実行可能なスクリプト（プログラム）を得ることができる。この段階的詳細化手法は、Instantiation of Process Model と捉えることもできる。

ＰＤＬのインタープリタをすでに試作している。これはBSD UNIX上で稼働しており、いろいろなユーティリティプログラムの逐次実行、並列実行、ウィンドウのオープン、クローズ、メッセージの表示などをＰＤＬのスクリプトに従って行なう。また、段階的詳細化を支援する機能として、未定義関数を含むスクリプトも、実行開始後ににユーザーと対話しながら具体化していき、完全なスクリプトに変換して実行を完了させることができる。このシステムはＣで書かれており、現在ＳＵＮ－３，ＮＥＷＳ，ＩＢＭ　ＲＴ－ＰＣ上で稼働している。

ＰＤＬを用いて、ＪＳＤ（Jackson System Development）に基づいた開発過程のスクリプトやＪＳＰ（Jackson Structured Programming）に基づいた開発過程のスクリプトを段階的詳細化して作成し、実際に実行した。

2．ワークショップの感想

全体を通じていろいろな議論が出てきて、なかなか有意義なワークショップでした。そのなかで、主に議論されていたことが、いったいプロセスモデルはどのようなものがよいかとか、リフレクションを入れるとどうなるかとか、メタモデルはとか、すべての場合がうまく書けるかとかだったと思います。これらは、要するに今、行なわれている開発過程をどう書くかという問題です。これ自身非常に重要ですし、我々がやってきとこともこのようなことの一部といえるでしょう。

しかし、まあ、なんらかの方法でどうにかこうにか、開発プロセスが記述できるのなら、まあその辺の研究はもうええんじゃないかな？こっちが書き易いとか、あれはこのようなものが書けないとかの議論は、適当でいいんじゃないでしょうか？

それよりも、プロセスを書いてみて、そこから新たな強力な開発法なり、開発ツールなりを生み出すことに力を注ぐことが重要ではないでしょうか？バルザーがＳＤＥ３のときに発言した（？）ように、開発プロセスというのは、新たな方法やツール、考え方が導入されると根本的に変えなければならないようなモノではないでしょうか？だから、いかにうまく現状を表現するかというのではなく、現状把握から、いかに生産性や品質をあげるための改良をするかということについてもっと議論したほうがおもしろいと思います。

例えば家を建てるプロセズを考えてみましょう。この書き方では、建売住宅はうまく書けるが注文住宅は書けないとか、兎小屋は書けるがカハラの住宅はだめだとかよりも、プレハブ住宅や２×４にすれば生産性はこれだけ上がるよとか、いまの建て方はやたらと整地に手間暇がかかるからでっかいブルドーザーを使えばこんなに早く家が建つよとかの議論が重要と思います。

だからどうするの、と言われてもあまり良い知恵が浮かびませんが、何はともあれ、現実のプロセスを認識して、それを改善するような強力なツールや部品化の方法などを探っていきたいと思っています。

ソフトウェアプロセス記述におけるユーザ支援

上原　憲二
（三菱電機㈱情報電子研究所）

１．はじめに

ソフトウェアプロセスを規定することは、ソフトウェア開発を経験や勘にたよる職人芸ではなく、工業化するために必要不可欠である。工業化されたソフトウェアプロセスとは、基本的には作業、文書および使用する手法の標準があることである。更には、プロセスを評価するメトリクスがあり、そのためのデータ収集、分析に基づいてプロセスが管理され、また将来にそなえて改善される。現在、経験が多く蓄積されてきた応用分野において、ソフトウェアプロセスの規定が存在する。そこでは、具体的なプロジェクトが起ると規定に基づいて具体的詳細な作業が設定され、プロジェクトの各人が割り付けられた作業を果たすことで、プロジェクトが進行する。しかし、上記のようなメトリクスを持つものは少ない。更に、計算機の応用範囲が急激に広がるなかで経験の浅い分野では初歩的なソフトウェアプロセスしか存在しないことが多い。このような状況において、ソフトウェアプロセスの記述法を開発することは、記述されたソフトウェアプロセスに従ってプロジェクトを進め記録することができるので、既存のソフトウェアプロセスを改善したり、新たに規定する試みなどを効率的に行うために有用ある。

この小論では、このようなソフトウェアプロセス記述のうちユーザ支援に関する記述に注目する。ソフトウェアプロセスにおいては管理の側面が強調されがちであるが、実際に存在し効果的に運用されているものは、個々の作業者すなわち設計者、プログラマ、試験者、保守者などに対する支援の側面ももつ。ソフトウェアプロセスは、管理のためだけでなく、むしろ作業者にとって、どんな入力生産物から、どんな手法・ツールを用いて、どんな新たな生産物を作るかが明確になり、自身の工程を見積り、自己管理し、更には改善できるための特性をもつ必要がある。本小論では、このような意味において、ユーザのスキルを記述し、ソフトウェア開発環境の中に格納する枠組みを提案する。この枠組みを適用したプロジェクトの経験を通して、ユーザ個々のスキルを蓄積し、その中から抽象化されより一般的で効果的なものが発見されれば、そのスキルはより多くのユーザで共有することができる。更に、それを規定に反映することによってソフトウェアプロセスを改善、進化させることができる。

２．想起支援

よく定義されたソフトウェアプロセスでは、作業分割、それぞれの作業で扱う生産物、使用する手法・ツールが明確になっている。しかし、実際の作業に当たり、例えばUNIX*ワークステーションを用いた場合、生産物のファイル群を格納するディレクトリ構造によっては、ディレクトリ移動やファイルリストのコマンドの多用、類似のコマンドの多用、複雑なコマンドの入力などの問題が起り、結局作業者本来の創造的な活動を阻害すること

* UNIXは米国AT&T社のBell研で開発されたOSです。

になる。これは、開発環境中への生産物の格納法、構造化法、検索法及びツールの使用法、組合せ法などが示されていないからである。このような方法は、作業者のスキルに係わることであり、作業者の創造的活動を支援するという立場から、プロセス記述に含める必要がある。

ソフトウェア開発という創造的活動を作業者の立場から分析すると、次のような図式が得られる。

ソフトウェア開発という創造的活動　＝　把握　＋　創造　＋　表現

ここで、把握とは対象である様々な情報を構造化することである。創造とは、真に人間の能力として、把握した情報から新たなものを構成することであり、現在直接的・効率的な計算機支援はできない。表現とは、創造した結果を、例えば、設計図式の編集系などのツールを用いて開発環境内に入力することである。

このような活動に対する支援は、情報を把握し、それを創造へ効果的に結びつけ、創造したものを容易に迅速に計算機入力させることである。すなわち、情報の構造化を支援し、それによって個々の情報、情報間の関係の理解を助ける。創造に際しては、情報の簡単な取り出し（検索）を支援し、又は検索の糸口を提供する。ツール使用などの表現に当たっては、容易迅速で誤りなく操作ができるよう支援することである。ここで注意すべきことは、構造化された情報、検索法、操作法は作業者の知識であり、作業者の経験や新たに学習した知識はコード化され格納できることが重要である。それによって、知識は改善、発展が可能になる。

このような支援を想起支援と呼ぶことにする。これは、上の図式において二つの「＋」を残して「創造」と「表現」を取り除いたものである。すなわち、次の図式となる。

想起支援　＝　「把握　＋　□　＋　□」の支援
　　↓　　↓
　　創造　　表現

3．作業の分析

作業者の想起支援をするために、その作業の進め方を分析する。作業者は、作業全体を幾つかの小さな作業（単位作業と呼ぶ）に分解し、それらを順次、繰り返し、選択的に、行きつ戻りつ、並列に、時には関係のないものを間に行いながら進める（図１参照）。

単位作業は一つの目的をもっており、通常それは一つの生産物（又は同種で関連の強い生産物群）を作ることである。生産物はファイル又はディレクトリ（ファイルの集り）で構成される。また、生産物を作るための操作では、その生産物に依存して、頻繁に使うコマンドが数種類に限られたり、コマンドのオプションも固定になるなどの傾向がある。

単位作業の集りは複合作業を形成する。そこでは、作業者は関連する複数の単位作業間を遷移（単に作業遷移と呼ぶ）しながら全体の作業を進める。作業遷移の契機は、異なる単位作業の生産物間の関係、作業者の思いつきなどがある。

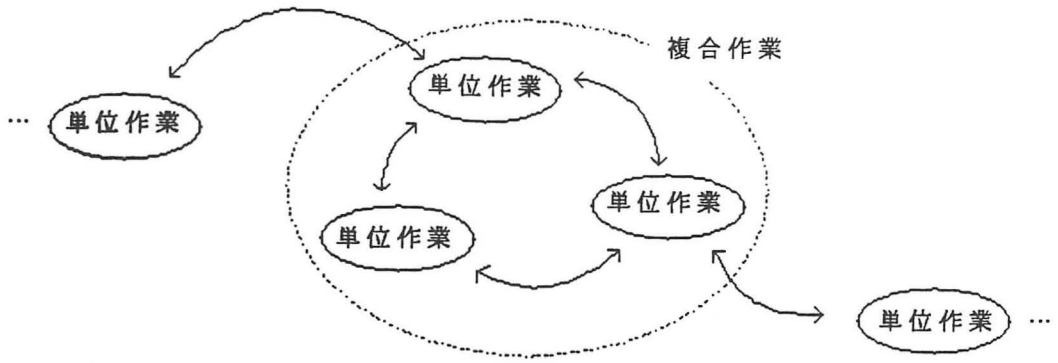


図１．作業の進め方

４．単位作業の想起支援

単位作業は一つの目的をもつことによって、前章に示したような局所性、反復性をもつなど、その創造的活動も一定の型をもつ。したがって、その想起支援はその型の特性を支援するものでなければならない。以下では、単位作業の想起支援の記述の枠組みを単位作業環境と呼び、その満たすべき性質について記す。

単位作業は少なくとも入力生産物、作成する生産物を含む。これらは、生産物自身の特質、その上の操作の特質、作業者個人に適した情報整理法などに依存して構造化され格納される。単位作業環境はこのような格納場所、構造を記述する。操作に関しては、頻繁に使うコマンドに対し、オプションの固定化、オペランドのパラメタ化を行った上で、生産物に対するその操作の意味を直接表わすような名前を付け、単位作業環境に記述する。これによって、作業者は適切な名前で操作を取り出して実行することができる。その他に、単位作業の状態、生産物の構造や操作に関するガイド、操作履歴などが記述される（図２参照）。

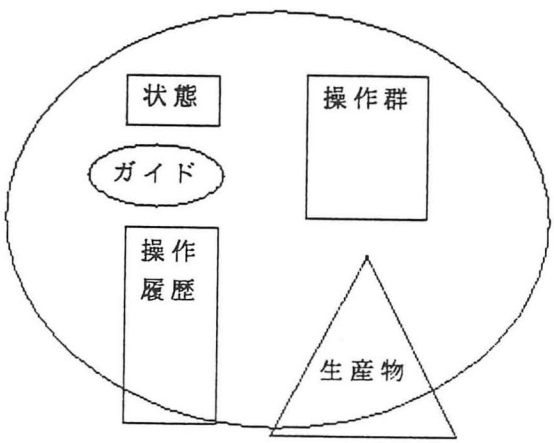


図２．単位作業環境

単位作業環境に基づく作業者支援のユーザインタフェースにおいて、次のことに気を付けねばならない。例えば、ワークステーションを用いて作業をしている場合、画面上の中心的な表示は、創造のため参照する生産物の内容、又は表現のために使用しているツールの画面である。作業者は、単位作業環境に記述されている情報を容易に検索できるが、そのための表示が上記中心的な表示を妨害してはならない。ポップアップメニュや小さなサブウィンドウを用いて表示し、不要になったらすぐに消失し、もとの画面に戻るようにしなければならない。

５．複合作業の想起支援

複合作業では、作業遷移があり、生産物間の関係、ある操作が他の単位作業を引き起こすといった単位作業間の関係がある。一つの複合作業に含まれる単位作業は並列に行われ、非決定的である。しかし、無秩序ではなく、複合作業としての目的、個々の単位作業の目的に従って、遷移先及びその契機に傾向がある。このような傾向を単位作業環境に記述することが複合作業の想起支援につながる。生産物間、特に他単位作業の生産物との間には成立すべき条件がある。これらには、その単位作業の前提となるもの、単位作業の終わりに成り立つものがある。作業遷移の契機や条件は評価が自動的でなくとも、単位作業環境に記述することは有用である。むしろ想起支援の意味からは、作業者の望む時に評価すべき契機、条件を検索できることが重要である。幾つかの他単位作業に影響する操作を行った場合、直接又は間接的にいずれはそれら単位作業に遷移し必要な操作を行うことになる。そこで、それら単位作業に必要な操作を事前に通知することは、複合作業を混乱なく進める一助となる。このように、他単位作業へのメッセージを単位作業環境に記述することは有用である。（図３参照）

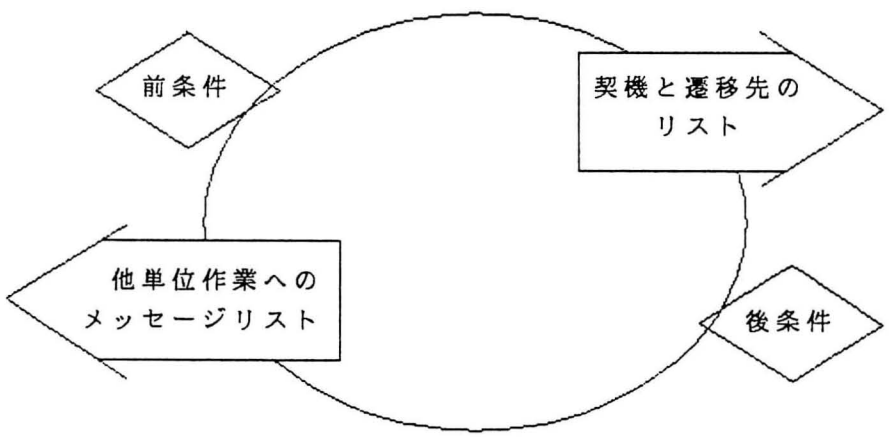


図３．複合作業の想起支援

複合作業の想起支援のために単位作業環境に記述した情報の検索表示においては、前章で述べたのと同様の注意が必要である。

６．おわりに

既存のソフトウェアプロセスの経験によれば、それが効果的に運用されるための特性として、個々の作業者の支援が重要である。この立場から、将来のソフトウェアプロセスをより効果的にするため、ソフトウェアプロセス記述に想起支援を含めること、そのための枠組み単位作業環境を提案した。

今後、このような枠組みによるソフトウェアプロセスの記述実験を通して、枠組みの評価を行う。また、このような記述に基づく支援をより強力にするための、生産物を格納するデータ管理への要求を明らかにする必要がある。これらを考慮して、枠組みを記述する記述言語の設計が必要である。

最後に、枠組みを作業者の構造化された知識としてとらえたとき、その柔軟性、すなわち知識の追加、発展、更には変革のための機構の検討も重要である。

参考文献

〔1〕 Riddle,W.E.,Software Designer's Associates: A Preliminary Description, Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences, 1987.

〔2〕 藤岡他、ソフトウェア開発用インタフェース：ｓｉｆ、情報処理学会ソフトウェア工学研究会５８－１７、１９８８年.

〔3〕 Schank,R.C.(黒川他訳)、ダイナミック・メモリ、近代科学社、１９８８年.

ポジション・ペーパー

日本電子計算（株）　臼井　義美

Q１．あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

以前、長岡の実践的ソフトウェア開発環境シンポジウムで、プロセスプログラミングが紹介された後、 Osterweil 氏の講演を聞き、 SDA の報告や SDE の発表などを聞かせてもらって、プロセスのモデル化に非常に興味を持っている。現在、ソフトウェア・アプリケーションを開発しているので、ユーザの要求仕様の確認から、テスト、信頼性の実証はもちろん、工程管理、コスト管理、要員管理などを含めて、プロジェクトマネジメントそのものも機械化を検討すべき時期に来ていると感じている。

Q２．あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

Williams の示したスパイラルモデルのように、各ステージごとにレビューや次の工程の計画をするのは、少々かったるい感じがする。しかし、リスク管理なども取り込んでいるところなど、利用する局面では、かなり現実的なアプローチかも知れない。いずれのモデルについても、人間の関係する要素をいかに解決するかということ、実現したとき、どれくらい快適な環境が提供されるかというところが私の興味の対象である。

Q３．あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

私は、ビジネス関連のソフトウェア・アプリケーションを開発する部門のマネジメントをしている。現行のアプリケーションの開発プロセスは、主に次のような手順である。まず、新しいアプリケーションを開発する場合、ユーザとマルチクライアントの形式で共同研究を行う。次に、特定ユーザを対象として、プロトタイプを開発し、そのアプリケーションを利用したユーザからの評価に基づいて、機能の付加や削除を行って標準システムを作る。この標準システムに対してカスタマイズすべきユーザとの話合いを行った後、要求に差異のある部分を抽出し、それをどのようにシステムに反映するかを考える。ユーザの数が増えてくると、共通部品のストックが増えて、部品を選択することにかなり多くの部分を比較的容易にカスタマイズすることができる。なお、システムは、ほぼ JSP に準じて作成されている。

さて、私は、上述の標準システムに対して、ユーザ要求を取り入れて、カスタマイズするためのプロセスを、極めて単純な方法でシステム化してみたいと考えている。本当は、もっとベーシックな部分からじっくりやってみたいのだが、現状のプロジェクトの置かれた立場では、あくまで現実の業務の改善というスタンス以上の取り組みは困難であるから、やむを得ない。また、プロトタイプを作るまでの作業と、カスタマイズするための作業とは、当面区別して考えたい。これは、前者の作業は創造的で、後者の作業は製造的であるからだが、現実的には、製造的なプロセスの方が不確定要素が少なく、効果的であろうというに過ぎない。実現の方法は、現在手作業でやっている作業プロセスを、コンピュータの支援のもとに置こうということである。それも極めて簡単で、すぐに成果が期待できる方法で。そのために、まず第１ステップとして、現実に使用している次のような資源を開発プロセスにおけるコンピュータ支援に役立てることにしたい。

・標準システム解説書及び変更部分の要求定義書
・標準システム及び変更用ソフトウェア部品
・ソフトウェア部品の組み込み手順書
・ソフトウェア部品と組み込み作業のコスト分析書

・ユーザのオリジナル仕様に対する作業手順書

・作業の進捗管理表

これをプロセス・プログラミングとして実現するとすれば、次のようになろう。

・モデル＝？

・プロダクト＝（INPUT）作業指示データ

（OUTPUT）実行システム、ドキュメント

・ツール＝標準システム、ソフトウェア部品、手順書の類。

・プロセス＝ここでは一般の手続き的言語でプロセスを記述する。

・環境＝ UNIX

具体的には、標準システムと蓄積したソフトウェア部品を SE が従来手作業で行っていたプロセスに基づいて組み合せ、やはり過去に蓄積したテストデータから必要なデータを抽出し、テストを行い、作業工数、コスト、スケジュール等を出力する。また、標準部品の用意されていない変更作業は、その手順を画面に表示し、その作業を支援するメッセージを順次提示することによって、 SE に対する作業を指示する。

入力データは、現在使用している手順書やマニュアルに沿って画面に質問を表示し、それに回答していくことによって与える。ただし、当面は、既に対処の方法が分かっているものしか取り扱えないが、現在自動化している部分も多少存在する。例えば、利用するデータ項目やコードの登録、計算式の入れ替え、出力帳票の選択、入出力媒体の変更、標準機能モジュールの挿入などである。このほか標準開発スケジュールに基づいた調整作業などは、比較的容易であろう。また、完成したソフトの見積書作成、契約書の作成、テスト結果報告書、ユーザへの作業依頼書、バージョン管理などの作成も役立つと思う。

以上、私のプロジェクトで、すぐに実験できそうな内容を述べた。特に、プロセス・プログラミングと呼ぶほどのこともなく、既に、この程度の開発環境はかなり実用化しているかも知れないが、実現すればかなり効果があるように思う。

要求分析／機能分析のプロセス・モデルとしてのＫＪ法とＨＣＰチャート

竹田尚彦・河合和久・大岩　元
豊橋技術科学大学　情報工学系

１．はじめに

我々はソフトウェア・プロセスの中で、ウォーターフォール・モデルの上流工程のプロセス・モデルとしてＫＪ法とＨＣＰチャートに着目し、前者を要求分析、後者を機能分析の過程に用いることを試みると同時に、この過程の計算機支援を試みている。

２．ソフトウェア設計におけるＫＪ法の使用とＨＣＰチャート

ソフトウェア開発における下流工程については、構造化設計／プログラミングから、最近のプロセス・プログラミングにいたるまで、多くの方法が試みられ、それなりの成果をおさめている。しかし上流工程に関しては、そのモデル化が困難なことから、従来積極的な方法論の提案はほとんど無かった。

発想法としてのＫＪ法を上流工程に用いることは、個人的にはかなり試みられたものと思われるが、積極的な提案が成されたのは最近のことである［１］。我々はＫＪ法のエディターを作ることから出発し［２］、これをＨＣＰチャートと組合せることによって、ソフトウェアの基本設計を行なうことを提案した［３］。

基本設計には、発注者の要求をボトムアップに分析する工程と、その結果を実現する機能分析の工程に分けられる。前者は、特にＫＪ法の考え方と一致するところが多い。

ＫＪ法によって要求分析を行なうと、Ａ型図解の形式で作成するソフトウェアの目的が明確化される。ＫＪ法ではこれに続いてＢ型文章化を行なうが、ソフトウェア設計の場合には、この代りにＨＣＰチャートによる階層化設計を行なう。ＨＣＰチャートは目的をどのように実現するかというプロセスを階層的に（目的の実現手段が次の階層の目的となり、その実現をその階層で行なう）繰り返し行なうので、ボトムアップに行ったＫＪ法の結果との整合性がよい。

ＫＪ法のＡ型図解は、要求の構造が２次元的に表現されているため、１次元的に表現された通常の仕様書に比べてはるかに理解が容易である。また、目的が実現手段に展開される様子が日本語で書かれたＨＣＰチャートも、それ自体が理解容易なばかりでなく、作られたプログラム・コードも、その意図する目的がチャートにより明らかにされているため、理解し易い。

３．支援環境

ＫＪエディターを我々は PC-9801 上に実現した。パソコンを用いたのは経済的な理由が大きいが、結果としてこれが成功の大きな鍵となった。Ａ型図解は大きな画面を必要とするが、これは実現不可能なため、代用として画面のスクロール機能を用いなければならない。我々はこれをパニング操作と呼んでいる。ゲーム用計算機として出発した pc-9801 は、テキスト画面のスクロールを高速に行なうことができる。これを利用して、カード操作を快適に行なうことができた。

操作が高速に行なえることは、設計作業に集中することに大きな影響力を与えることが、ＫＪエディターの使用経験から分った。また、当然のことであるが、エディターの使用によって、設計プロセスのやり直しなど容易に行なえるようになったので、質の高い設計結果が得られるようになった。

ＨＣＰチャート・エディターについても、我々はパニング操作に重点を置いてＫＪエディターと同様の概念で製作中である。

参考文献
[1] 宮沢修二：情報システム開発のニーズ分析におけるＫＪ法の活用、日経コンピュータ1987.6.22号.
[2] 小山雅庸　河合和久　大岩　元：発想支援ツールとしてのエディタ、昭和61年電気関係学会東海支部連合大会（1986）.
[3] 大岩　元　河合和久　竹田尚彦　小山雅庸　塩見彰睦：ソフトウェア基本設計のためのソフトウェア・ツール、日本ソフトウェア科学会第４回大会論文集（1987）.

ポジション・ペーパー

（株）日立製作所　大槻　繁

Q１．あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

実用的なソフトウェア生産技術（仕様化、設計、プログラミング技法等）を開発していくという視点から、ソフトウェア・プロセスの形式的モデル化に興味がある。例えば、従来のウォータフォール型の品質管理技術をベースにして、これに、最近の自動プログラミングや、形式的仕様記述に基づく開発技術を融合化していくためには、その背景になる理論が必要である。この意味で、現状の開発プロセスをどのように記述するか、というよりも、種々の開発プロセスをどのように体系化できるかというメタ理論を期待しているといった方が正しい。

Q２．あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

プロセス・モデルは、スパイラル・モデルのように、どれだけ語りえない事柄までをも対象としているかが、現実性の指標と考えられる。しかし、語りえる世界（形式的仕様）を明確に境界付け、その思想的枠組みを打ち出した点では、私は、リーマンのステップ・モデルを高く評価している。

Q３．あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

私は、以前に情報蓄積型のソフトウェア・システムの実用的な開発技法を提案した（「データモデルを導入した仕様化技法」情報処理学会　ソフトウェア工学研究会　11 月 1987）。そこでは、プロセス・モデルをステップ・モデルを基にして、さらに、これを最適化及び仕様の厳密性の観点から拡張を施している（RPC ライフサイクル・モデル）。ただし、ここでは、開発に関わる行為の対象としているものは、ソフトウェアの仕様、あるいはプログラムの生産行為のみであって、管理的な側面は、扱わないこととして排除していた。この点からも、プロセス・モデルに対する研究に期待するところ大である。

ソフトウェア・プロセス研究への期待

東芝システム・ソフトウェア技術研究所　大筆　豊

1．はじめに

現在、ソフトウェア生産支援システムの研究開発を担当しており、広義にはソフトウェア・プロセス研究に関係していると言えるが、今はやりの（？）プロセス・モデルあるいはプロセス・プログラミングに直接的に携っていると言えないので、エッセー風に記述する。

このワークショップに参加した目的は、プロセス・モデルあるいはプロセス・プログラミングという新しい概念（あるいは新しいキーワード）に対して、いろんな人が（特に日本を代表する研究者達が）どういう考えで取り組んでいかを見極めたい、あるいは、共通のコンセサンスを持っているかを知りたい、という好奇心からである。

結果から言うと、新しい眼鏡をかけてはいるが、それぞれの人がそれぞれの立場で行っていた研究の延長をやっており、言葉の定義すらまだ確立していないことを確認できたことかな、という次第である。

2．何故プロセス・モデルか？

ハードウェアの世界からのアナロジーで、ソフトウェア開発の世界に移転されたライフサイクルの考え方（いわゆるウォータフォール・モデル）が、どうも現実には適さない、無理にこのモデルで開発すると、どこかにひずみが出る、何か新しいモデルがないか、というのが直接的な動機であると考える。

間接的には、今までソフトウェア工学で生み出された多くの概念や手法あるはツールは主として、開発されるプロダクトに直接関係するものであったが、より効果をあげるためには、プロダクトに直接関与する技術のみならず、開発のためのインフラストラクチャ（狭義の環境）や開発方法論や管理方法まで含めた、人間系を含んだソフトウェア開発過程全体を見直す必要があるとの認識があるものと考える。

3．プロセス・モデルの目的

プロセス・モデルの目的は２つある。第一は、現実の（あるいは理想的な）ソフトウェア開発過程をモデル化し記述すること、第二には、記述されたモデルをある意図を持って使用することである。

第一の目的（記述）については、記述方法として形式的な言語を用いるか自然言語を用いるか、あるいは特別な記述法を用いるかは別として、個々の具体的な開発業務をトレースするのでなく、何らかの抽象化（モデル化）がなされるかどうかが鍵である。

第二の目的（使用）では、その使用目的により分類する必要がある。まず、そのモデルにそって、実際の開発業務を行うこと、第二はモデルを分析して、よりよい開発モデルを得る手助けになるかということになる。特に、モデルに則してソフトウェア開発を行うとき、何がメリットとなるかを明確にする必要がある。

4．プロセス・モデルの観点

モデル化を行う場合、その目的によって強調点（観点）が異る。このワークショップでもいろんな観点から提案されている。

- 管理面：　企業内で多数の技術者の協同作業により開発するモデルとしてとらえる。ライフサイクルモデル、フラワーモデルなどがこの範疇に入ろう。納期、品質、コストをいかに守るように管理できるかが、モデル化の目的である。
- 形式化：　プロセス・モデルの形式的な記述。形式性を追求すると、（何をやるか判らない）人間をいかに繰り出すかが重要となろう。形式化の目的は、プロセスの検証あるいは自動実行が目的となる。記述法とモデル化とは直接的なつながりを考えることはないものと言えるが。
- プロセスの実行：プロセス・プログラミングは、実際にコンピュータにより実行することを考えている。人間あるいは、自動化ツールをも体系に含み、かつ上位レベルでは、コンピュータによる実行を考慮しているので、厳密な記述を必要とする。

私は、企業にいる立場から、プロセス・モデルは管理モデルとの観点から見ざるを得ない。規模の大小、応用分野の特殊性、応用分野についての知識の蓄積の違いなどから、社内で統一的な標準モデルはありえないが、個々に対応できるようなからくりを作りながら、やはり標準化の観点からのモデル化を考えざるを得ない。

5．管理モデル

研究開発的なプロジェクトと製品開発プロジェクトでは異なるが、後者のウェイトが圧倒的に多く、モデル化への興味は後者になる。また、現状では、ウォータフォールモデルをベースにしたもの以外には考えられない。

現状では、ウォータフォールモデルをそのまま実行できるもの以外、「特例」や「運用」と称して、その場その場でしのいでいる場合が多い。

ウォータフォールモデルの最大の仮説は、「前工程が完全に終ってから次工程に入る（あるいは入れる）」ことであるが、現実には次工程に入ってから前工程以前の不具合により後戻りが起ることがある。特に、仕様の変更による後戻りが無視できない。「プロトタイピングによる仕様の早期確定を行なえば良い」という解決案は、案として完全に近いかもしれないが、現実には、実用に供しうるプロトタイピングツールは使えるものが少ない。

企業では、標準化を図るため、開発標準という形で規程化し、守らせることが必要である。部門毎、あるいはプロジェクト毎に、カスタマイズあるいはインスタンシエーションし、部門標準あるいは、プロジェクト標準の形で文書化し、実行することになる。

後戻りなどの特例については、個別に記述するか、共通のメタルールとして記述する。記述しきれない突発事故はその都度管理者が判断し、処置している。もし、プロセスプログラミングが完成したとき、予想されうる例外はメタルールとして記述されるので割り込

み処理的に飛び出せるが、突発事故に対しては、「管理者が判断せよ」とでも出力されるのか？

6．プロセス・モデル研究への期待

現状では開発への適用という観点から、管理モデルを中心に考えている。ウォータフォールモデルが現状に合わないといって、新規な（あるいは新奇な、もしくは珍奇な）モデルを提案しても、その有効性が実証されない限り、現状では動けない。だから、ウォータフォールモデルを現状に合うように変更し、改良することが着実なアプローチと考える。

しかし、付加価値のあるソフトウェアを開発するには、今のモデルで良いと考えているわけではない。研究者、あるいはハッカー的なプログラマが行っている、開発過程のモデル化ができるなら、創造性のあるソフトウェアを開発するモデルとなりうるか？もし可能性なら素晴らしいのであるが。ハツカー的なプログラマに対しては、開発環境を中心としたモデル化となり、人間の行うところは、自由にやれるようになるというのが解であろう。

形式記述については、ソフトウェア開発の自動化につながるテーマであり、プロセス・モデルあるいはプロセス・プログラムの研究とは関係があろうがなかろうが大いに期待している。モデルの形式記述は、プロセス分析や評価の客観的な裏づけとしても期待が大きい。

7．おわりに

ワークショップ参加の感想文に近くなってしまったが、我々が現在取り組んでいるのは現実ベースの改善であり、プロセス・モデル研究者の興味の対象からずれがあるのを認識している。現状のモデルでも、再利用あるいは分散開発というキーワードを追加するだけで、大巾に変化しうるので、モデルの記述法およびその（人間を含む）実行系については研究を進めていく。

ポジション・ペーパー

静岡大学　落水　浩一郎

現在、SA/SD、FSM、OOP 等の各種方法論を使用して、ソフトウェア設計を行う際のソフトウェアプロセスに関する情報の収集実験をすすめている。

実験の特徴は、以下の通りである。

1. 実際の設計活動に沿ったプロセスの収集

単なる教科書の要約ではなく、 LIBRARY、HEATING SYSTEM、LIFT の３例題を SA/SD、FSM、OOP の３種の方法論で被験者に記述してもらい、計９個の設計過程を実施している。

2. 実験データのツールによる収集

ビデオ・テープやテープ・レコーダーによる収集・解析のやり方に対して、オブジェクトを取り扱えるお絵書きツールを被験者に与え、設計過程を克明に記述してもらっている。

3. プロセスのルールベース記述

OPS83 と prolog を利用して、2. で得られるプロセス生データを整理記述し、「実行可能なプロセス記述」に関する研究もあわせてすすめている。

現在、研究は中間段階にあり、何らかの知見が得られる段階までは至っていないが、3. で得られたプロセス記述を他の被験者に適用することにより、プロセス記述の問題をも整理してみたいと計画している。

ポジション・ペーパー

(株) 日立製作所　梶山　雄三

Q1. あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

現在進めている仕事に適用していけるものは取り込みたい。

Q2. あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

勉強不足で、どのプロセス・モデルが良いか、コメントできない。その点を含め、この workshop で知識を深めたい。

Q3. あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

現在、個別のユーザ・プログラムを開発しているが、設計の上流工程（ある程度、仕様が決っている）での　break down、仕様をメーカの立場で規定する手順なり各サブ・プロセスでの成果物として、どういう規定の仕方をすれば良いかに興味を持っている。なかなか思うようにはいかない。

Mechanisms for Software Process Dynamics

Takuya Katayama, Masato Suzuki

Department of Computer Science
Tokyo Institute of Technology

1 Introduction

Dynamics is one of the characteristic points where software processes differ very much from the usual computational processes and it is essential for describing and enacting the software processes. It refer, in general, to the fact that we need dynamic and flexible control over the preparation and enaction of software processes. Specifically, it should allow that process scripts could be created, changed or refined incrementally during process enaction, or the state of process enaction can be changed, internally or externally, for directing a portion of the enaction history to be repeated for fixing the effects of mistaken operations or suspending/resuming enaction of the processes to allow flexible human intervention.

One reason we need dynamics and flexibility for software processes is the inadequacy of our understanding and capturing them. We may have to enact them even if we do not have enough knowledge of the processes, and we sometimes have to start with incomplete, partial and even erroneous process scripts and refine, augment and evolve them while or after they are enacted. Another reason also comes from human factor. As we often fail in completely and correctly performing activities done by invoking tools or mental activities such as decision making, some parts of processes have to be repeated for making these activities done satisfactory even if we have the complete process scripts to be enacted. We need more dynamic and flexible mechanisms for such redoing than is needed for the execution of usual computational processes. In this paper, we discuss how dynamic can be handled and propose *redoing* and *reflection* concepts for process dynamics. We also discuss briefly how redoing is introduced into the process model HFSP.

2 Requirements and Mechanisms for Software Process Dynamics

Among the requirements of software process models with respect to dynamics, understandability will be the most important, as software process scripts have to be read and changed during their enaction and this requires that processes have to be described in a highly understandable model.

Of course, this is essential too for dynamic and flexible control over their enaction, as we need the enaction status to be represented in a clear and understandable manner so that we can capture the status easily and take proper actions based on it.

As software process models consist of several, almost orthogonal, aspects such as relationship among products, object and tool access, activity scheduling, user interface, resource management and organizational/co-operative support, it is important that these aspects can be described separately and independently as much as possible. This will considerably decrease the complexity of creating and understanding process descriptions. It is also desirable that exceptional cases can be described separately from the main and straightforward cases as this will be effective in eliminating spaghetti of feedback loops for handling exceptions and will be, of course, desirable for realizing dynamics and flexibility.

Expressive power of the process model is, of course, essential. It has to be provided with enough features for dynamics such as changing enaction status, redoing a part of process script already executed and creating and changing process scripts while they are enacted. Though some features found in conventional programming languages, such as backtracking, iteration, conditional and parameterization/instantiation might be helpful to some extent, they are not enough in expressing the dynamics and flexibility of software processes. This is not, however, a surprise if we consider that the conventional programming languages are designed on the assumption that primitive operations always correct and come up with the same results when executed in the same environment and that complete programs could be prepared prior to their execution. Two operations, *redoing* and *reflection*, are proposed in this paper to cope with process dynamics.

Redoing is an operation to repeat a portion of a software process when unsatisfactory results have been observed which are caused by that portion. That is, it allows to go back, in the computation history, to the point where a mistaken or imperfect operation has been done whose effect is detected at some later point. It, then, enables to replace that sub-computation polluted by the mistake with a new backup computation. If the new computation succeeds to produce correct results, the computation resumes from the point where the error has been detected with the results which are produced by the backup computation.

Note that, in redoing, we completely ignore the results from the wrong sub-computation, though these results stored in objectbase could be referenced by human operators to produce new results in the backup computation. At first sight, redoing might look like the conventional iteration, backtracking or recursion, however, it is completely different from them in that redoing is, in a sense, a discontinuous operation as it abandons the previous results from the wrong computation, whereas these conventional operations tries to increase the accuracy of the results by refining them and they define continuous computation. This kind of discontinuity will be essential for handling computation involving error-prone elements and uncertainty.

Redoing can be issued not only from the inside but also from the outside of the processes. When we can know in advance the places where mistake will happen and how to fix them, we can write scripts for handling backup process. This will serve for making the entire process script clearer and more understandable if redoing is properly introduced into the process model. When it is difficult to pre-determine these things, we have to resort to the execution of redoing issued by human operator when he find the problem. In this case, scripts for backup computation will be usually prepared by him on-the-fly and this is a way for achieving incremental creation of pro-

cess scripts and their modification. Of course, process environment has to support such activities as suspending/resuming process enaction, browsing computation history and editing process scripts.

Reflection could be a very strong operation for treating dynamics. Roughly speaking, a computation model is *reflective* if the model has features for knowing about and changing the environment in which it is embedded[1]. Though the expressive power of reflection will be dependent on what are the changeable parts of the environment, it could be applied to software process dynamics if the enaction status and process scripts are changeable objects in the process environment, that is, if the process model is provided with operations for manipulating them. As it will enable the process enaction status to be obtained, analyzed, computed and changed from the inside and outside of the processes, we can take very flexible control over their enaction. Redoing we have proposed can be considered as a special case of applying reflection. Much study will have to be done for its effective and safe usage in process dynamics.

3 Dynamics in HFSP

HFSP is a software process model derived from the hierarchical and functional computation model HFP[2] and attribute grammar[3]. The primary concern of HFSP is the product-oriented aspect of the software processes which characterizes them in terms of their input/output relationships and it models the processes , in the first approximation, as a set of activities which realize the input/output relationships. The activities are decomposed into subactivities when their tasks are complex and cannot be performed by simply invoking software tools. HFSP combines this product-oriented view of the software process with other aspects such as objectbase access, tool interface, user interface and resource management[4].

Hierarchical and functional description style makes the introduction of features for dynamics clearer and more understandable than other process models which are based on process behaviors, as (1) computation history in HFSP is represented by an attributed tree of activity decomposition, on which every product to be referenced or generated in the process is explicitly represented as an attribute of a node, and status handling operations are expressed very clearly by tree operations like pruning and grafting, and (2) dependency analysis needed for detecting the activities redone and redoing them, for example, is easily performed by the incremental attribute evaluation technique[5]. Redoing correctly performed activities has to be avoided very carefully in process enaction as, in general, the same results will not be produced by their re-execution and the re-execution is costly for activities involving human operation.

Redoing is introduced into HFSP taking this advantage[6]. The problems to be considered in introducing redoing in HFSP are (1) how to invoke redoing activities, (2) how to prepare their input attributes, and (3) how to deliver their output. In HFSP, an activity is activated only when it is generated from another activity by decomposition (except the initial activity) and its input/output attributes are defined locally in the decomposition, so it is difficult and not desirable to activate redoing activities from out of the enaction tree. Redoing has been introduced through an *redoing decomposition*. When an exceptional or erroneous condition is detect for the input attribute values of an activity, its normal decomposition is suspended and, instead, a redoing decomposition is selected and applied to the activity, which takes a form of

$$A \Rightarrow \text{redo } A_0 \text{ when } C$$

This means that, when the condition C for redoing is true for the input attributes of A, the activity A is decomposed into A_0 for redoing computation starting from an occurrence of the activity A_0 in the process enaction history to remove the effect of errors or mistakes detected at A. The input attributes of newly introduced A_0 is taken from the that occurrence of A_0 and, if the redoing computation and its succeeding computation will come up with the correct output attributes of the new A_0, they are given to the original A_0. The most important fact about the redoing in HFSP is that it does not destroy the functional property of HFSP. That is, even if there occurs a situation that a part of already performed computation has to be canceled and redone, overall computation after it was redone looks like a normal computation in which output attributes of the activities in the enaction tree depend only on their input attributes.

So far, we considered the case that redoing is specified in the process script, that is, redoing decompositions can be prepared beforehand and put into the process script. Though this is the most desirable, this is not alway possible as we may not have complete knowledge of the process. Redoing can be issued also from outside the process, probably by an process operator. We can use this external redoing as a fundamental mechanism for incremental process creation and modification. Suppose we have an incomplete process script in which the description of an activity A is missing, possibly by the reason that its details are not known at the time of writing the process script, and it is temporally represented by the empty decomposition. Enacting the whole process will request external redoing to occur at the activity A because of its empty decomposition, and the operator is asked to give an appropriate real decomposition. If he could provide it, the enaction of the process resumes from A. On-the-fly process modification could be handled in the same way , where redoing is triggered by the operator when he detects something wrong about process enaction. For external redoing to be effective, enaction environment has to provide enough supports for keeping track of enaction history and editing/debugging process scripts.

References

[1] B.C. Smith. Reflection and semantics in lisp. In *Proceedings of 11th ACM Synposium on Principles of Programming Languages*, pages 23–35, 1984.

[2] T. Katayama. HFP: a hierarchical and functional programming based on attribute grammar. In *Proceedings of the 5th International Conference on Software Engineering*, pages 343–353, 1981.

[3] D.E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[4] T. Katayama. A hierarchical and functional software process description and its enaction. In *Proceedings of the 11th Internaltional Conference on Software Engineering*, pages 343–352, 1989.

[5] T. Reps. *Generating Language Based Environments*. MIT Press, 1984.

[6] M. Suzuki and T. Katayama. Redoing: a mechanism for dynamics and flexibility of software processes. 1990. Submitted to 12th Internaltional Conference on Software Engineering.

ソフトウェア・プロセス・モデルの要件

福田　由紀雄
株式会社　東芝　　システム・ソフトウェア技術研究所

［１］　ソフトウェア・プロセスのモデル化へのアプローチ

企業において、ソフトウェア・プロセス（開発過程）を管理することは、納期やコストの管理とともに品質を各工程毎に作り込むという品質保証の考え方を実行するための実現方法と考えている。

このような考え方を具体的に実現していくために、標準開発手順をＱＣＰ（Quality Control Process chart）として表現し、開発プロセスの状況を定量的なメトリクスで把握していく方法が進められている。[1][2]

プロセス・モデルの研究テーマとしては、プロセス・モデルの記述法とともにメトリクスの導入がある。例えば、工数・成果量等の見積・スケジューリング・ステータス（進捗等）・成果物の品質基準・作業自身の質の基準等である。

［２］　現状のソフトウェア・プロセス・モデルに対する考え方

ソフトウェア・プロセス・モデルとして、ウォータ・フォール・モデル、スパイラル・モデル、フラワ・モデル等いくつかのモデルが提案されているが、我々の対象とするソフトウェア開発においては、ウォータ・フォール・モデルが現実的と考えている。

企業におけるソフトウェア開発には目標（納期・品質・コスト）があり、その目標をキープするという意識が重要である。ウォータ・フォール・モデルは、各工程の切れ目が納期確保のマイル・ストーンであると同時に品質保証活動のマイル・ストーンでもあり仕事のけじめがはっきりするという意味で優れたモデルであると考えている。

ウォータ・フォール・モデルの欠点は、工程の後戻りの表現である。例えば、試験工程においてエラーが発見された場合、設計工程まで戻って修正をするという考え方はモデル上で説明できても、現実の仕事の中でははっきりしない概念である。

工程の考え方として、(1)工程は時間に沿って進もので後戻りしない、(2)各工程内には、予定作業（作業とは、管理上の最小実行単位として定義されたプロセスである）と予定作業とがあるという考え方でみると、いわゆる後戻り作業とは予定外作業ということができ、このように考えた方が管理のモデルとしてわかり易い。つまり、各工程の最初にその工程内の予定作業に対して詳細計画を立案し、計画の実行途中で予定外作業が発生した時点で必要により計画を修正するという考え方である。

このように考えることにより、各工程における予定作業と予定外作業の比率は上流工程でのＱＡ活動の成果が反映されていると仮定すれば、ＱＡ活動の効果を定量的に評価するためのメトリクスの１つになると考えている。

［３］　ソフトウェア・プロセス・モデルおよびその支援系の要件

ソフトウェア・プロセス・モデルおよびその支援系の要件について、以下に述べる。

（１）　ソフトウェア・プロセス・モデルは、ソフトウェア生産活動での種々の形態や開発条件に応じて、モデル内のプロセスのうち必要または重要なプロセスがのみが活性化されるようなモデルであること。

即ち、新規開発・改造開発・保守・移植などの形態と、対象ソフトウェアに要求される信頼性・開発メンバの技術力・納期などの開発条件に応じて、モデルをカスタマイズ・オプティマイズしたプロセス・モデルのインスタンスを各開発毎に生成または選択できること。

プロセス・モデルは、統一されたモデルであることが好ましいが現実的には、開発形態毎にカスタマイズされたプロセス・モデルを記述することから始まると

考えられる。

（２）　プロセス・モデル内の各プロセスは、

(1)　人間の実行するもの

(2)　機械の実行するもの

(3)　人間と機械が分担して実行するもの

から構成されること。

（３）　プロセス・モデル内の各プロセスは、

(1)　予定されているもの（標準：通常の開発で実行されるもの）

(2)　予定外のもの　　（例外：通常の開発では必ずしも実行されない、または、或る条件下でのみ実行されるもの）

の両方を含みかつ、(2)については、記述しやすいこと。

（４）　プロセス・モデル内の各プロセスは、

(1)　実行開始条件（作業開始条件）

(2)　実行終了条件（作業終了条件）

(3)　アウトプットとしてのプロダクト（成果物）

(4)　利用／参照すべきプロダクト

(5)　プロセデュア（手順）（含むツール・手法等）

(6)　達成基準（成果物の内容チェック・リスト等）

等が定義されていること。

また、各プロセスのインスタンス間の公式なコミュニケーション（例えばデザイン・レビューなど）もプロセスとして明記されていること。

（５）　プロセス・モデルのインスタンス内のプロセスのインスタンシエーション（実際の開発ジョブでの作業の具体化）においては、プロセス・モデルのインスタンス内の各プロセスに対して、対象システムの設計構造（サブシステム・機能群・タスク・モジュールなど）に基づいて生成されること。

即ち、開発中に設計が詳細化・具体化されるに伴って、プロセス・モデルのインスタンスが成長していくこと。

例えば、ソフトウェア開発管理（作業・成果物の管理）での標準開発工程に基づいてソフトウェア構成要素を具体化するにつれて、作業名・成果物名を生成するしくみ等。[5][6]

（６）　プロセス・モデルのプロセス・インスタンス内のプロセスのインスタンスに基づいて作業工数の見積やスケジューリングなどの計画作業が可能なこと。

（７）　プロセス・インスタンスのステータスによる開発の進度把握、成果物（プロダクト）の品質把握、これらのデータをもとにした開発の生産性把握などが可能なこと。

（８）プロセス・インスタンスのステータスやプロダクトの解析結果に基づくデザイン・レビューでの判定用情報の作成と判定の自動化が可能なこと。

例えば、　成果物の品質メトリクス値（ＥＳＱＵＴ[3][4]の基準等）

（９）プロセス・インスタンスのステータスやプロダクトの解析結果に基づく開発ジョブの診断・予測が可能なこと。

例えば、(1)　開発のリスク分析

(2)　管理者へのアクションアイテムの提示など

（１０）開発途中での不測の事態に対して、プロセス・モデルのインスタンスを変更できること。

即ち、開発途中に、開発開始時に想定した開発形態や開発条件が変化した場合に、プロセス・モデルから、必要なプロセスを活性化しプロセス・モデルのインスタンスを変更する。

（１１）　ソフトウェア・プロセス・モデルは、初心者の担当者に対して、仕事の手順書として使えること。

（１２）　ソフトウェア・プロセス・モデルのプロセスの内、人間と機械が分担して実行するものについては、ノウハウの蓄積・伝授として、使えること。

（１３）　開発を通じて、プロセス・インスタンスに関するデータ収集機構を備えていて、これらのデータを活用できること。

例えば、計画値の設定基準、開発中の異常の検出、開発の能率や質の評価などに活用できること。

（１４）　未知の分野での開発に関して非決定的・不完全な開発過程をモニタし、プロセスのインスタンスやプロセス・モデルのインスタンスのデータ蓄積が可能であるとともに、これに基づいてプロセス・モデルの改良可能なこと。

［４］　おわりに

ソフトウェア・プロセスのモデル化は、人間の知的活動の解明の要素を含んでいるため、どの程度の詳細度まで記述出来るのかという大きな課題がある。

我々は、ソフトウェア・プロセス・モデルを管理（開発上で、制御可能要素を制御することによって目標とした品質・納期・コストを達成するための活動で技術面を含む）の立場から研究・実践していくことが重要である。

［謝辞］

本稿の作成に際しシステム・ソフトウェア技術研究所・研究第三部　大筆　豊部長附、金子　信一主任研究員に謝意を表する。

［参考文献］

（１）　綾、他：「開発ランクを利用したソフトウェア開発管理の一手法について」，情報処理学会第３５回全国大会，１９８７

（２）　綾、他：「ソフトウェア開発管理手法とその評価　ＱＣＰにそった開発のしくみの実験」，第８回ソフトウェア生産における品質管理シンポジウム，１９８８

（３）平山、他：「モジュール設計段階における品質評価の一手法」，情報処理学会第３６回全国大会，１９８８

（４）平山、他：「ソフトウェア設計品質メトリクスの検証　ＩＭＡＰシステムにおける品質保証」，情報処理学会ソフトウェア工学研究会６４－５，１９８９

（５）　栄、他：「ソフトウェア構成管理システム（１）－（３）」，情報処理学会第３０回全国大会，１９８５

（６）　栄、他：「ソフトウェア開発管理用統合データベースの開発」，情報処理学会第３３回全国大会，１９８６

設計プロセス・モニタリングの概念とその支援環境

岸 田 孝 一
SRA, Inc.

ICSE-9 における Leon Osterweil と M.M.Lehman との論争には，まだ決着がついていないように思われる． Leon の提唱するプロセス・プログラミングのアイデアは，きわめて魅力的ではあるが，Manny が指摘するように，ソフトウェア・プロセスの肝心な部分はほとんど人間の直観に依存していることはたしかであり，したがって，プロセス・プログラムは「簡単に書けるがしかし意味のないものか，ほとんど書けないものか，の２極に分類されるだろう」というかれの予想は，もしかしたら当たっているかもしれない．

現在考えられているプロセス実行への支援形態には，次のような種類がある：

（１）作業の自動的実行
　　例：　Make in Unix
（２）プロセス・スクリプトによるガイド
　　例：　特定の設計技法ガイド
（３）モニタリング
　　統一的な枠組みによるプロセスの記録・蓄積・分析・開示

わたし自身が以前から興味を持っているのは，「プロセス・モニタリング」の概念である．いくつかの試行を自分でもやってみた結果，効果的な再利用を可能にするような形でプロセスを記述するのは，かなりむずかしいことがわかっている．そこで，次善の策として，具体的な開発プロセスの進行過程を忠実に記録し，それをあるパースペクティブのもとに整理して提示し，そのプロセスがこれから進むべき方向をガイドしたり，あるいは，将来発生するであろう類似のプロセスへの参考にすることはできないだろうか．

そのためには，単純で無味乾燥なプロセス・ステップのログではなく，もう少し意味のある構造を持ったなんらかのフレームワークが必要であろう．

たとえば，MCC で試作され試用されている gIBIS は，Hypertext を利用したグラフィカルなデスカッション・ボードの１種であり，グループによる設計意志決定プロセスを記録する枠組みのプロトタイプとしては，多くの示唆に富んでいる．

開発の上流工程（要求分析や設計段階）を考えると，プロセスの細部までを事前に記述することにはかなりの危険または非現実性がつきまとう．したがって，未完成なプロセス記述を何らかの形で稼働させ，なおかつオンザフライで，修正追加しながらダイナミックに実行して行く必要が生まれる．

比喩的にいえば，それは「白地図」（過去のプロセスの記憶にもとづく仮想的な地図）を片手に見知らぬ街を旅するようなものである．いずれかの機会にその支援メカニズムの具体的なイメージを提示してみたい．

ポジション・ペーパー

沖電気工業（株）　北　英彦

Ｑ１．あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

形式的モデル化は、ソフトウェア開発を一貫して計算機に支援させるためには、不可欠と思う。現実への適用は、新しいプログラミング・パラダイムの採用と同時に行われることになると思う。

Ｑ２．あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

形式的仕様、あるいはプロトタイプを重視する新しいパラダイムに基づくプロセス・モデルに強く関心を持っている。ユーザあるいは設計者が、最初に完全な仕様を作るというのは無理で、形式的仕様による仕様の検証あるいはプロトタイピングにより、仕様のレベルでのフィードバックができるようになる。

Ｑ３．あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

研究所における開発プロセスは、作成するソフトウェアが実験的なものであることが多く、プロトタイプモデルによりモデル化されそうである。問題点は、いつ、プロトタイプでなく、製品になるのか明確でない（明確にしていない）ことにある。

ポジション・ペーパー

富士通（株）　久保　宏志

Ｑ１．あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

ソフトウェア・プロセスに関する情報交換、討論のフレームワークとして、形式的モデルがあると望ましい。

Ｑ２．あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

IEEE の Software Quality Metrics Standards は、フレームワークになりうる。 Process 改善の Process をも含んでいるところが、気に入っている。

Ｑ３．あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

PDCA という万能の有効なモデルがあるのに、このモデルの適用さえなされていないのが現在の姿である。 PDCA モデルの完路を定着させるのが最優先課題と心得る。

ポジション・ペーパー

富士通（株）　熊谷　章

Q1. あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

どのような記述言語が適切か？その言語の処理学の形態は、どのようなもので、どのように作成すればよいか。現実の適用の仕方の MMI はどうなるか？

Q2. あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

システム設計のモデルが一番興味深い。何故なら、SE 活動の中で、最も困難であり、ノウハウが必要とされ、現実社会のトラブルの原因になっているからである。この難関を突破するのが、プロセスモデルの狙いであると理解している。

Q3. あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

Hyper Text や Hyper Media 風のモデルで記述できそう。ビデオを使ったスナップショットなんかも効果があるかな（補助的に）。改善を要する問題点は、生のデータやシーンから意味のある表現をどのようにして表現するかである。これが、うまくできると 1. で答えた記述ができる。

ソフトウェア・プロセス・モデルの適切さと具体化

三菱電機コントロールソフトウエア株式会社
技術統括部開発課
小前俊哉

1. はじめに

プロセス・モデルには、プロセス・プログラミングに代表される「数学的」モデルと、管理モデルやウォータ・フォール・モデルに代表される「抽象的」モデルの2つのビューが存在しているように思う。現状、企業における生産環境や管理方式を検討する場合、後者の「抽象的」モデルが現実的である。今後「数学的」モデルに期待するところは非常に大であるが、ここでは実際の業務を前提とした「抽象的」なプロセス・モデルの適切さと具体化について、現状の考えのアウトラインを整理する。

2. モデルの必要な機能

モデルは少なくとも以下の5項目を含むソフトウェア・プロセスを記述したり規定したりできなければならない。

○広い（計算機）環境範囲

実際のソフトウェア生産の環境は、生産用計算機と実機による環境の分散（散乱が正しい）が起こっている。例えばUNIXのようなOSがオールマイティなOSにならない限り、この散乱状態は続く。散乱した環境の中には「満足な」コミュニケーション（ネットワーク）機構やプロダクトの編集ツール等が用意されていないことがある。この現実からすれば、プロセスの記述や実行を計算機上で行なうのは非常に困難であり、より抽象度の高いモデルが望ましいことになる。

○グループ／組織

ソフトウェア・プロセスは、複数の人間や組織の間で散乱する（プロセス/プロダクトの分岐と散乱）。「一人の人間が自分のためだけに作るソフトウェア」はモデルの対象から外す。

○ヒューマン・ファクタ

ソフトウェア生産は、人間の創造作業と具現化作業である。それはベルトに乗ってやってくる部品のネジを締める方法のように明確ではなく、陶芸家の作業のように芸術めいている。また人間の能力には当然個人差があり、一人の人間でさえ一つの「状態」に対してコンスタントではない。よってできる限り人間の知的活動を意識した（作業者の側に立った）モデルが必要である。

○突発性（あいまい性）プロセス

ソフトウェア・プロセスとプロダクトは、人間の思考／行動に左右され易く、要求や価値観、表現方法がコンスタントでない。そこで突発性（あいまい性）の吸収とプロセス変更に柔軟に対応する機構を有し、かつプロセスが発散しないモデルが必要になる。

○管理

管理は生産プロセスにおける各状態の活性化とフロー制御のために必要である。モデルの価値は、その管理機能の高さにあると考えている。ここで言う管理とは、「プロセス/プロダクトの分岐と散乱」「ヒューマン・ファクタ（人間の知的活動）」「突発性（あいまい性）プロセス」をコントロールしながら、プロセスを完了に導く機構のことである。

3. ソフトウェア・プロセスの状態遷移

ヒューマン・ファクタと突発性（あいまい性）プロセスを考慮すると、「ソフトウェア・プロセスの状態遷移」（図3.1）について考える必要がある。

ソフトウェア・プロセスをいくつかの「状態」に分け、そこで何をするのかを決めることは容易にできる。しかし各状態の参照プログラクトと各状態の実行順序は、現在のウォータ・フォール・モデルがそうであるように、一意（厳密かつ正確）に決められない。なぜならソフトウェア・プロセスは、図3.1の様に人間の創造作業と具現化作業の2つの状態を行き来しながら進んでいるからである。従来のモデルだと、創造作業によって生成（出力）されるプログクトを具現化作業の入力にすることは容易だ（許可する）が、その反対に具現化から創造への「状態遷移」に費やすエネルギーは非常に高いものが要求される。元々両状態の間に実行順序は無いのではないか。もちろん全体のプロセス収束のため（管理のため）、具現化の状態から創造の状態への遷移時の約束ごとは必要である。つまり双方向遷移を禁止するモデルよりも、明確な双方向遷移規則をモデルに組み込むことが重要であると思われる。

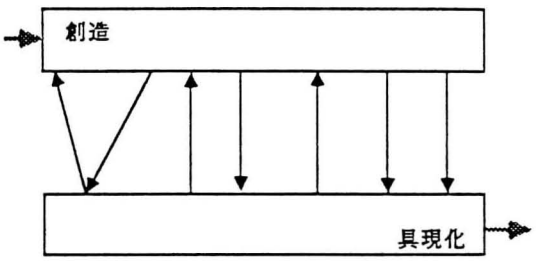


図3.1 ソフトウェア・プロセスの状態遷移

4. モデルがカバーする範囲

マクロに見た場合、プロジェクトの発生から完了までは図3.1に試験の状態を付加したイメージ（図4.1）になる。（ウォータ・フォール・モデルは、図4.1の特殊なものである。図4.2 参照）

しかし実際のプロジェクトは、多くの人間と複数の環境上で実行される。そこで各人間は役割を与えられ、それぞれ小さな状態遷移の「ノード」を持つと考える。各ノードはネットワーク状に分散される（図4.3 参照）。

この時ネットワーク上の各ノード(状態モデル)間には当然コミュニケーションが必要になる。各ノードが同じレベルのコミュニケーション機構を有していればコンピュータ支援されたコミュニケーション機構やプログクトのモデル化が可能だが、それでは余りにも適用範囲が狭くなるため、もう少し抽象化したモデルが必要になる。

またプログクトの管理がプロセス・モデルのキーポイントなので、共通プログクトやノード特有プログクトを一元管理し、プログクトの散乱を押さえる機構（規則）が必要不可欠である。なぜならプロセスのフロー制御や状態遷移は適切なプログクトの検索／検証と生成／修正によって行なわれるからである。つまりプログクトが正しく見えなければ、プロセスは正しく前進しない。各ノード/状態に適切なプログクトを示すプロセス-プログクト・マップのようなものがあれば良いと考える。

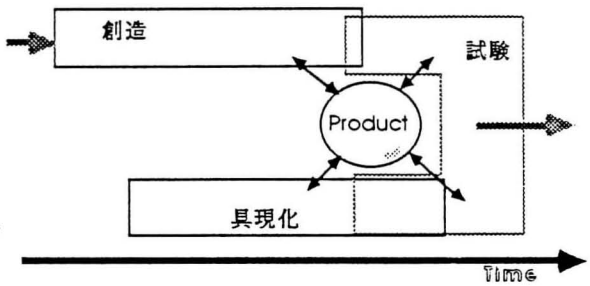


図4.1 状態遷移を考慮したプロセス・モデル

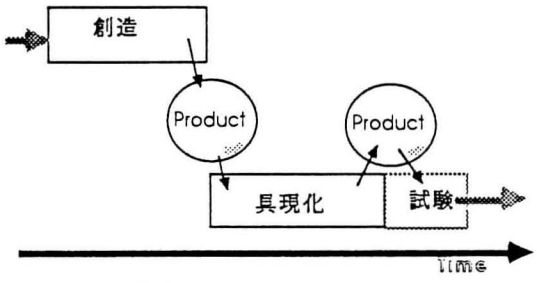


図4.2 ウォータ・フォール・モデル

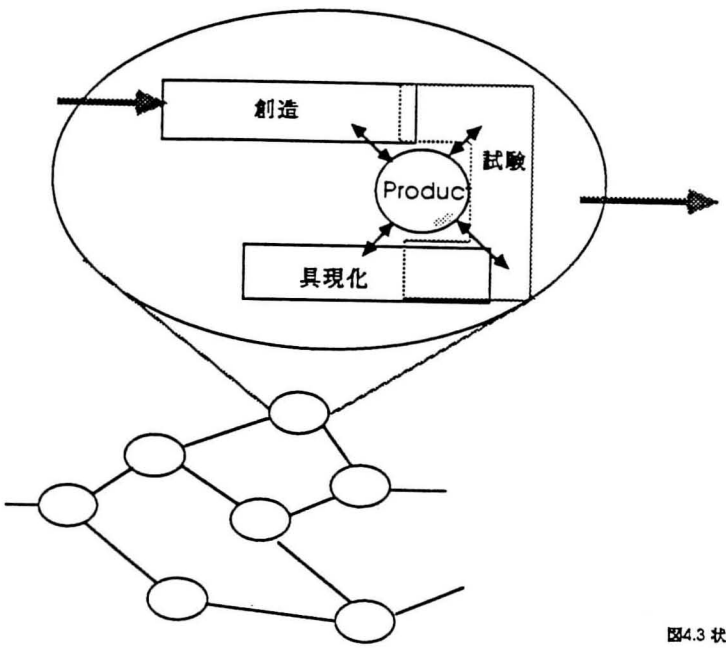


図4.3 状態ノードのネットワーク

5. ソフトウェア・プロセスの管理

プロセスの管理は、ソフトウェア・プロセスを（予定コストで）完了に導き、一定品質のプロダクトを生産するために、ノード内の状態遷移の管理と、全ノードのネットワーク形態やノード間フローの制御、そしてプロダクトの検証を行なう。

5.1 ノードのネットワーク形態や遷移状態を制御する管理

各ノードは管理が無い時点では散乱しており、お互いに規則性のない通信を行なうか全く行なわないかのどちらかである。まずソフトウェア・プロセスの管理機構は、散乱しているノードのネットワーク形態の固定化とフロー定義を行なう（図5.1）。この時ノードのネットワーク形態を、ウォータ・フォール型にするかスパイラルlike型にするかを決めることができる。

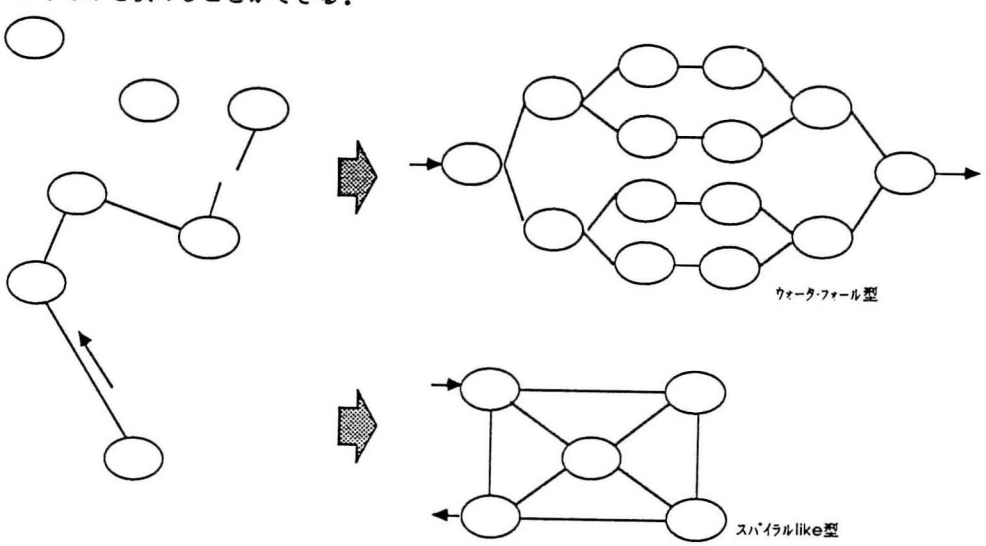


図5.1 ネットワーク形態の固定化とフロー定義

次に、各ノードのネットワーク形態と過去の経験（プロセス・ログ）を参照してプロセス-プロダクト・マップを生成する。これでソフトウェア・プロセスはReady状態になる。

なお、ネットワーク形態／フロー定義／プロセス-プロダクト・マップは、参照／修正／保存／流用可能でなければならない。（一種のプロセス記述言語にする。）

プロセスが実行を開始したときから、管理モデルは、各プロセスとプロダクトのモニタリング／ロギングを開始しプロセス-プロダクト・マップとの整合性をチェックする。そして必要に応じ、ネットワーク形態の修正、各ノードへのメッセージ／命令を行ないソフトウェア・プロセスを完了に導いていく。

5.2 ノード内の状態遷移の管理

状態の遷移回数制限等、ノード内のプロダクトとプロセスに関する「命令」を与える必要があるので、プロセス-プロダクト・マップから各ノードに対するプロセス指示(記述）を生成する。

6. おわりに

自分が現在考えているプロセス・モデルの適切さと具体化について「むりやり」アウトラインを整理してみた。しかし、状態遷移の自由性とプロセス管理の共存には大きな問題と矛盾が多い。あと少しの間は不自然な管理モデル(ウォータ・フォール)が一般的であると思われるが、別途以下のような自然で単純な管理モデルと人間の知的作業に合ったプロセス・モデルの実験的環境を検討している。

○創造と具現化の状態遷移を妨げないようにするドキュメンテーション環境。

○既存の管理手法によるプロセスの記述と、プロセス-プロダクト・マップの生成機構。

第一回ソフトウェア・プロセス・ワークショップ 参加報告

中川 中

1　事前の期待

本ワークショップに参加する直前まで、私は殊ソフトウェア・プロセスの話題に関しては真の無知を自認していた（尤も、他の多くの話題に関して同じことがいえるのであるが）。僅かに「ソフトウェア・プロセスもソフトウェアである。」という標語のようなものを耳にしていたのみである。そのまま頭が白紙の状態で参加するのは、他の参加者に対して礼を失することでもあり、前日約半労働日を費やして予備知識を得た。教材は事前に事務関係者に渡された[1]資料、昨年の国際ソフトウェア工学会議予稿集に偶然にみかけたL.Williamsの投稿文、および一昨年の日本におけるソフトウェア・シムポジウムの予稿集に同じく偶然にみかけたL.Osterweilの論文の転載、のみである。

そうした事情のもとで、私はソフトウェア・プロセスに関する議論は他の多くのソフトウェア工学における議論と同様に胡散臭いという仮説をたてていた。これは次のような無形式で無頓着な論理のようなものにしたがった結果である。

1. 上記の標語は、ソフトウェア生産過程[2]が機械的（必ずしも機械によらないが）に「実行」可能であるという事実を表現しているように思われる。
2. であるならば、それは直感的に誤りである。
3. この直感を覆す事実を私は知らない。
4. 上記のL.Osterweilの論文で1.の印象は強まった。
5. M.LehmanがL.Osterweilに反論している。その主張の詳細は 知らないが、これによりさらに2.の直感は勇気づけられた。
6. 上記のL.Williamsの論文でL.Osterweilの主張に反対しつつもソフトウェア・ プロセスを興味ある話題としている多くの人間がいることが判明した。
7. L.Williams自身が示している方法は、しかし、プロセスの記述法の代案に すぎない。目的において、また認識において、彼にも1.のような印象をもつ。
8. したがって、私が知り得たソフトウェア・プロセスにおける議論にはすべて2.が成立する。たとえ「すべて」が二つでしかないにしても。
9. 反証がない限り直感が正しいとすると、8.、2.および3.により上記の仮説が 成り立つ。

私が本ワークショップに期待していたのは、上のような事情を反映して、次の点である。

- ソフトウェア・プロセスに関する研究が目指している目標を己にとって 明らかにする。
- ソフトウェア・プロセスに関する議論の帰趨を己にとって明らかにする。
- もって先の仮説を検証する。

もとより本ワークショップで聴いた議論は参加者のそれであって、ソフトウェア・プロセス論者一般のそれを聴いたわけではない。しかしながら、以下では参加者がすべてのプロセス論者を代表していたと仮定する。逐次限定を付す労を避けるためである。

2　ソフトウェア・プロセス論の認識

本章では私の得た認識を事実の形で記す。これは事実たる確信があるためではなく、主観を示す文末辞を逐次付加する労を省くためである。

(a) すべてのソフトウェア・プロセス論を包括する共通点は、ソフトウェアの生産過程を記述する試みを有意義と認める点である。

(b) その試みの目的は大別して2つあり、また2つに限られる。すなわち、

1. 生産過程の管理を容易にする。
2. 生産過程を技術上の意味で合理化する。

ここでは、「研究対象としておもしろい」というような非当事者[3]の観点は無視する。また、管理の側

[1] 当稿では敬称および敬語を使用しない。事実を客観的に記述したいが故である。

[2] ここではそれが着想されてから廃棄されないし消滅するまでにソフトウェアに加えられるあらゆる生成・変更活動を総称して「生産」と呼んでいる。

[3] 生産過程に直接または間接に利害関係を持つ人間を当事者、それ以外の人間を非当事者と呼ぶ。

面は私の管轄外であり、かつ管理の側面と技術の側面を一括することは議論を混乱させるので、目的は2.のみとする。「合理化」とはあい昧な概念であるが、非形式的にはつぎのような意味である。

- 生産性や効率といった、可測性を想起させる概念を含みうるが、それらと はあえて区別される、より広い概念である。
- 人的な要素を排除する方向に向かう。ただし、これはあくまで付随的な属 性であって、合理化の直接の意味付けと解釈してはならない。
- 生産過程に可視性（別の言葉では透明性）をもたらす。
- と同時に、またそれゆえに、予測可能性をもたらす。
- おそらくすべての場合において、生産過程を構造化する。「構造化」につ いては後述する。
- ツールを組織的に（よりポレミカルには、有機的に）統合した成果としての 環境を構築する試みは合理化のひとつの具現である。

(c) すべてのソフトウェア・プロセス論は、生産過程の構造を明示しようとする。すべての場合において、これは生産過程の分解によっている。各種プロセス・モデルはこの分解方法を与えあるいは正当化するために利用される。分解の方法・程度はさまざまであるが、分解して得られるのは一様に副生産過程（または活動）である。

(d) 大部分のソフトウェア・プロセス論は、プロセス記述を機械的に実行することを目標としている。そのために最低限必要なことは機械的に解読可能な記述方式を確立することである。これは適当なプロセス記述言語の模索という研究テーマに結実している。ひとつ以上のレベルにおけるメタ記述言語を探究する試み、記述言語の生成系を考える試みはこのテーマの（評価上中立な意味であるが）亜流である。ところでこの目標を掲げる場合、上記の分割は特定のツール（の組み合わせ）にマッピング可能な単位過程（活動）を得るまで続けられることがしばしばである。

(e) ソフトウェア・プロセス論はまた、活動が人間によって行なわれる可能性を前提に据えている。ダイナミズムが枢要な役割を果たすと考えるのはこのためにほかならない。人間の活動は外生的要因に左右され、それがプロセスの安定性を損なう。それはおそらく組み込まれた（embedded）ソフトウェア・システムと同じように、予期せぬ事態にどこまでどうやって対処するか、あるいは無理に予期してしまうか、という問題を余儀なくするのである。この立場をとらないソフトウェア・プロセス論は自動プログラミングという別の分野に退避すべきである。

3　ソフトウェア・プロセス論の評価

前章の認識に基づいたソフトウェア・プロセス論に対する暫定的な私論を以下に示す。前章で述べたのと同じ理由により、事実の陳述の形式による。

(a) 生産過程を構造化することはソフトウェア工学においてもっとも重要な課題である。ところで、つとにプロセス・モデル論は構造化を取り上げた。ソフトウェア・プロセス論は構造化自体よりむしろそれを既与としたときの明示のしかたを問題とする。これは明示すること、換言すれば記述すること、によりその対象に対する理解が深まり、議論の焦点が定まり、もって問題点の把握と解決に大きく貢献する、という経験的（実証済みか否かを私は知らない）真理にしたがうならば、正当な方法である。

(b) しかしながら、ソフトウェア・プロセス論の大勢が実行可能なプロセス記述を目指している点は問題である。ここにはおそらく避け得ない二律排反がある。すなわち、一方で、実行と看なしうる程度までに機械的に処理しうるよく知られた最下層の活動はプロセス論を待つまでもなくほぼ自動化されている、あるいは少なくとも容易に自動化しうる。他方で、プロセス論が生産過程において役立つのはよく定義されていない活動に取り組むときである。実行を狙う実例がソース・ファイル云々を扱うプロセス論をまつまでもない活動や窮屈過ぎるという批判に耐えることのできないJSDの手法に限られているのは、研究が足りない所為ではなく、その方向に対するこの疑問に答える術をもたないからである。

(c) さらに、ソフトウェア・プロセス論で使われる「実行」という言葉自体が、誤解と混乱を招き易い。プログラムの実行との対比からは、プロセス記述の実行が可能であるということは、ある活動を開始し終了する条件、その活動の結果、および各活動間の関係（あるいは全体の制御）、が機械的に判断できることである。この定義を緩め、それらの一部が機械的に判断できることを実行可能というのは簡単である。問題は、定義のどの部分をどう緩和したのかが一向に示されていない点にある。漠然と明らかなのは、L.Osterweilに向けられた批判の合唱にも拘らず、単に制御を行なえることをもって「実行」と呼んでいる習慣である。

(d) この意味であい昧にされているのが「実行」の意義である。活動結果やその開始・終了の妥当性の判断が人間のし意的決定[4]によるのであれば、それを「実行」と呼ぶのは強弁にすぎる。このごう慢さは単純な機械による支援、たとえばあるファイルの有無を知らせるというような、をもってしても償いがたい。もちろん、そうした「実行」も有意義であ

[4] 人間の判断であっても、常に十分に拘束性のある指針にしたがう場合には機械的といってよい。もちろん、拘束性をどう測定するのかが問われようが。

る。ただし、古来パート・チャートが能くしてきたところと本質的に同じ意味で有意義なのである。

(e) 「実行」に関する批判は別としても、この活動結果等を評価する方法は重要である。生産活動の構造化に向けて如何なるアプローチをとろうとも、活動単位に定義を与えまたその属性を調べる手段を確立できる度合と構造化が役立つ度合とは間違いなく比例関係にあるからである。活動単位を行為の面から (behaviourally) 定義すべき点にはほぼコンセンサスがあるといってよい。典型的な例では、入出力仕様と前・後条件がある活動単位を規定する。ところで、そこで気がつくのは、そうした定義は非常にしばしば諸成果物ないしそれらの間の関係として記述されることである。それ以上に簡潔かつ正確な方法がありそうもないという理由から、また生産過程の良否を決するのは究極のところ成果物である意味で、これは当然である。

(f) しかしながら、ではなぜ成果物自体を単位として生産過程を考えないのか、という疑問が起きる。生産過程を構造化する単位に活動を据えるのは一見自明である。しかし、再び一般のプログラミングと対比してみるとき、それは選択のひとつにすぎないことがわかる。すなわち、データ、つまり成果物を単位とする視点がありうるということである。Hoare 流のプログラム意味論と抽象データ型に基づくプログラム意味論を比較するとき、簡潔さや理解の容易さにおいて後者に分があることからの単純なあるいは単純すぎる類比では、活動単位のプロセス記述と成果物単位のプロセス記述では、後者のほうにより期待がもてそうである。

(g) 実は、この成果物を単位とするプロセス・モデル、およびそれに基づくプロセス記述（言語よりも広い意味での）方法こそが私のあたためているところである。これはソフトウェア・プロセス論とは無関係に探究してきたことで、むしろ反プロセス論の立場から始まった。今の段階では、反プロセス論というより滅プロセス論に近いのではないかという感想を持っている。基本的な狙いは生産過程の構造を諸成果物、それらの状態の遷移、およびそれらの間の関係をベースとして記述することであり、活動単位やその効果は諸成果物の状態の変更として、また各状態の諸性質として表現される。「実行」は眼中にはない。人間の意志決定の当否の判断が（人工的な知能と呼ばれる分野の研究の副産物などとして）相当程度自動化されない限り、そしてそれは予見しうる将来に起こりそうもないのであるが、[5] プロセスの実行など目指すことはもとより論じることさえ無意味である。意味があり、そして意義があると思われるのは、人間のし意的な意志決定をすべて既与としてなおかつ整然たる生産過程が可能になるような枠組を考えることである。

4　結語

ソフトウェア・プロセス論は胡散臭いという仮説は、議論が「実行」に集中しているという（正しい）事実認識に基づいたものであることが判明した。今のプロセス論の潮流が変わらない限り、この仮説は証明されたといわざるをえない。

管轄外として避けてきた管理の側面にひとこと言い添える。技術上の要請を妨げないようなプロセス管理モデルを考えねばならない。私が思い付くよい方法は、技術者がよしとするモデルを無条件に受け入れることである。

[5] 分野を限定すればある程度のことは直ちに可能である。いわゆる 4 世代目のプログラミング言語なるものの真似をすればよいのである。

第1回 SEA プロセス・ワークショップへのポジション・ペーパ

ソフトウェア開発プロセス・プログラムはプログラム・モドキ

株式会社SRA ソフトウェア工学研究所
新田稔

プログラモイドとは

一般的なプロセス・プログラムを考えると、それはプログラムと言うよりは、プログラモイドである。プログラモイドとは、プログラム(program)+オイド(-oid)の合成語で、プログラムに似たようなもの、と言う意味である。プログラムが、実行前にすべての記述が完成していて、実行中に変更されることがないのに対して、プログラモイドは、実行前に記述が完成している分けではなく、また実行中に変更されることがあるという性質を持っている。逆にいえば、プログラムはプログラモイドの特殊なものである。

「プログラムでも自分自身を書き換えるものがある」と言われるかもしれないが、そのようなプログラムで書き換えているのは、プログラムの実行インスタンスであり、プログラムの記述が書き変わっている分けではない。プログラモイドは、実行中にその記述自体が書き変わり、実行終了後もそれは残る。ソフトウェア開発に限らず、世の中の如何なるプロセスのプログラムもプログラモイドである。

なぜプロセス・プログラムはプログラモイドであるか

あるソフトウェアについて、それを開発することの難しさと、開発プロセスのプログラムを作ることの難しさを比べると、後者の方がかなり難しい作業であろう。難しいという言葉があいまいなら、より多くのパワーがいる、長い期間がかかる、コストが高い、あるいは記述量が膨れ上がる、と言いかえても良い。

したがって、プロセス・プログラムは、たとえそれに則ったソフトウェア開発のコストがゼロであったとしても、再利用できるものでなければ価値がない。我々は寸分違わぬソフトウェアを二度作ることはないから、プロセス・プログラムには対面しているソフトウェア開発に合わせて書き換えられる部分を残しておく必要がある。

「与えられた問題を入力とすることで、同じプロセス・プログラムが異なったソフトウェアを生み出すことができる。プロセス・プログラム自体を書き換える必要はない」という主張があるかもしれない。しかし、与えられた問題をプロセス・プログラムの入力とするには、まずその問題をプログラムに入力できる形に分析する必要がある。

そして問題の分析こそが、ソフトウェア開発のもっとも重要な部分である。したがって、ソフトウェア開発に入る前に、そのプロセス・プログラムへの入力を完成させようという試みは、全くの二度手間になってしまう。これを避けるのは、プロセスを進めながらその記述も進めるという方式であり、そのプロセス開始前の記述はプログラモイドとなる。

実行主体

プログラムの実行主体は計算機であり、プログラモイドの実行主体は計算機と人間の複合体である。しかし、プログラムの実行においても、たとえばオペレータが磁気テープをセットするなど、人間が仕事の一部を行なうものもあるから、プログラムとプロセス・プログラムの実行主体の違いはは、質的に違っているというよりは、計算機と人間の仕事の割合の程度差であると考えた方がより正確だろう。

計算機と人間の違いは、計算機は記述されていることだけを実行するのに対して、人間は記述されて

いないことも実行することがある、という点である。この人間の「記述されていないことでも実行する」という性質が、プログラモイドの実行を可能にしていると言える。

しかし、人間を何を実行するかわからないものとして放っておいたのでは、ソフトウェア開発の工業化にはつながらない。ソフトウェア開発を工業的に行なうには、プログラモイドであるプロセス・プログラムを、なるべく部分的・段階的にプログラムに変換しながら記述されていることに従って仕事を進めていくというアプローチが必要である。

プロセス・プログラムの工業的実行

プログラモイドであるプロセス・プログラムを部分的・段階的にプログラムに変換しながら実行していく手順を一言でいえば「ある時点でわかっている情報をもとに少し先までのプログラムを作り、それを実行してその先のプログラムを作るのに必要な情報を得る」ということになる。

この手順で実行するプロセス・プログラムには、自分自身をどう記述していくかという、内省プログラムが含まれていなければならない。この内省プログラムは、たとえば、記述の自由度を低く押えるなら、プロセス記述のパラメータ的な部分を空白にしておき、それをどのように埋めれば良いかを書いたものである。また、自由度を大きくとるなら、記述言語の定義を与えるものになるだろう。

プロセス・プログラムを工業的に実行する具体的な手順は、まず現在わかっている情報を入力とし内省プログラムを実行し、未確定部分の記述を少し進めることから始まる。以下、次のような手順で実行する。

1.内省プログラムを実行し、現在入手可能な情報より記述できるところまで未確定部分を記述する。未確定部分の記述はプログラモイドになっている

2.記述された未確定部分の中で、すでにプログラムとなっている小部分をみつける。一般的には、プログラムとなっている小部分は複数あり、またあるプログラムが他のプログラムの一部分になっていたり、あるいは複数のプログラムが共通部分を持つこともある。

3.みつけたプログラムのうち、未確定部分の記述を進行させるために必要と思われる情報を開拓できそうなものを実行する。しかし、現実的にはプログラムとなっている小部分がないかもしれない。そのときには、部分的なプログラモイドを、人間の「記述されていないことでも実行する能力」によって実行することになる。このときには、実際に実行した過程をもとの記述にフィードバックしておく。

4.

(a)新しく開拓された情報によって未確定部分の記述が進むなら1〜3を繰り返し、未確定部分の記述が進まないなら、2でみつけた他のプログラムを試す。

(b)新しく開拓された情報により、未確定部分のまだ実行されていない部分の変更が要求されるなら、その部分を変更し2〜3を繰り返す。また、未確定部分のすでに実行されている 記述の変更が要求されるなら、記述を変更して2へ戻る。このときには、未確定部分の記述内、およびプロダクトとの関係において何らかの矛盾か起きる可能性がある。

(c) 2でみつけたすべてのプログラムを実行しても未確定部分の記述を進めるような新しい情報が得られないとき、このソフトウェア開発は失敗である。

5.未確定部分の記述が完成、すなわちプログラムになったなら、それ全体を実行してソフトウェア開発作業を終了する。

人間の身勝手さの削減

さて、当面の間プロセス・プログラムの実行には人間の「記述されていないことを実行する能力」が必要とされる分けであるが、人間に無制限の自由度を許すと、ソフトウェア開発プロセスはどこに行き着くかわからない。しかし一方、自由度が制限された中では、創造の楽しみも半減し、働く意欲も失われがちになるだろう。

人間に束縛感を与えないで、しかもプロセス・プログラムを確実にある方向に向かって実行させるには、やはりプロダクトに注目した記述を行なうのが最も良いと思われる。つまり、人間は何をしてもよく、目的ソフトウェアは完成されたがっている、ソースコードはコンパイルされたがっている、設計書は書かれたがっているなど、プロダクトがその進むべき方向を知っているという方針で、記述を行なうのである。

ただし、このようにプロダクトを中心としたプロセス・プログラムを記述するには、現存するオブジェクト指向的言語ではまだ不十分である。たとえば、Smalltalkでは、インスタンスを作るにはそのクラスを一意に指定しなければならないし、またインスタンスはメッセージにしたがって受身的に動くだけであり、インスタンス(変数)の履歴を取ることも難しい。

しかし我々がソフトウェア開発で扱うオブジェクトは、その境界が曖昧であったり、変化したり、進むべき方向があったり、集団で働いたり、過去に遡ったりする性質を持っている。すなわち、次のようなオブジェクトが扱える言語が望まれるのである。

1.概念だけがあって、まだ実物が存在しないオブジェクト(たとえば目的ソフトウェアは、プロセスの初期には存在しないが、我々はプロセスの終りにはそれが存在するであろうことを知っているし、またその性質のいくつかも知っている)

2.時間とともに発生・成長・分割・統合・消滅などの変化を伴うオブジェクト(たとえば、まずモジュール分割を決めてそれぞれのモジュールの設計を行なっていく過程で、あるモジュールとあるモジュールは統合した方がよいとか、あるモジュールはいくつかに分割した方がよいとかがわかってくることがある)

3.はじめは何であるのかわからないが、次第にその姿が明らかになっていくようなオブジェクト(たとえば、単なるメモ書きが後に設計書の一部となる)

4.進むべき方向をそれ自身が知っているオブジェクト

5.集団である意味を持つオブジェクト

6.履歴を持つオブジェクト

これからの課題

ソフトウェア・プロセス、プロセス・モデル、プロセス・プログラムなどと言葉の響きは良いが、まだ研究の基盤がしっかりしていないような感じを受ける。これからは、プロセス記述用の言語、アルゴリズミックな実行手順の研究・開発が不可欠だろう。

また、すでにソフトウェア工場などで採用されている標準作業工程・標準プロダクトなどに対しても、ノウ・ハウにとどまらず、なぜそういうことになったのか、他の分野に応用するにはどうすればよいか、もっと生産効率をあげるにはどこをオプティマイズすればよいか、人間と計算機の役割分担はこれでよいのか、人間の創造力をもっと活かすにはどうすればよいのかなどの突き詰めた分析が必要である。

新田　稔

ポジション・ペーパー

日本電子計算（株）　野村　敏次

Q1．あなたは、ソフトウエア・プロセスの形式的モデル化や、そのモデルの現実への適用にどのような関心をお持ちですか？

今まで様々な（と言っても数は少ないが）プロセスモデルが出て来ているが、現実に適用されており、プロジェクト管理を行い易いモデルは、ウォーターフォールモデルであるという事実は否定出来ない。私自身、今までウォーターフォールモデルの不備について指摘はしているものの、少なくとも汎用大型機を利用したアプリケーション開発では、ウォーターフォールモデルに、プロトタイピングや再利用の概念をうまく取り入れることができれば、充分現実に適応できると考える。

Q2．あなたは、既存のいろいろなプロセス・モデルのうち、どれが最も現実的だ（あるいは興味深い）と思われますか？それはなぜですか？

問題は、ワークステーションを利用した分散開発環境における開発プロセスである。これは、開発環境そのものが、一義に規定できないところに起因していると思う。確かに端末やワークステーションの普及は目ざましく、2.5 人に 1 台利用するところまでいっているが、それは必ずしも分散開発環境の普及を意味していないのではないか？

Q3．あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは、どんな風にモデル化できそうですか？また、そのプロセスにおいて、改善を要する主な問題点は何ですか？

ある特定の開発環境を想定し、その環境上では、このようなプロセスで開発を進めれば、非常に生産性の向上が図れるというプロセスモデルを考えるか、逆に何等かのモデルを設定し、そのモデルを実現するためには、どのような開発環境（ハード／ソフトを含め）を構築しなければならないといった限定された範囲の話が必要である。一般論でプロセスのモデル化を考えていくと、モデルはモデル、現実は現実といった状態が続いていくものと思われる。

ソフトウェア開発法に対する記述モデルと属性文法記述

馮　安
（大阪大学　基礎工学部　情報工学科）

1．まえがき

ソフトウェアの生産性，品質を高める目的で種々の開発方法論が提案されている[1][6]．こうした開発方法論に基づく開発環境を構築するには，方法論を形式的に記述することが重要である[2][3][6]．

本稿では，属性文法を用いた開発法の形式的記述を試みる．開発法の核となる部分を抽出するために，プロセスを中心にした記述モデルを提案する．このモデルを用いて，開発法におけるプロセス，及びその実行手順と入出力を記述する．次に，プロダクトの構文と意味定義，実行手順の制御条件を加えて，属性文法による記述を作成する．ここでは，代表的な設計法の１つである構造化設計(Structured Design，以下 SD と略す)法[6]を例として取り上げる．

2．記述モデル MSDM

各種のソフトウェア開発法は，品質のよいプロダクトを開発するという共通の目的の元で，異なる開発手順を提唱している．これらの開発手順こそは，開発法を特徴付けるもの，即ち，核となる部分である．

開発手順は，その静的な部分（プロセスの構成）と動的な部分（実行手順の制御）で定めることができる．ここでは，開発法におけるプロセス及び実行手順を抽出するモデル MSDM を提案する．

記述モデル MSDM は次の３項組で構成される．

MSDM ＝ (PSM, CTM, PDM)

(1)プロセス記述 PSM=(PS,SR)．

PS,SR はそれぞれプロセス，分解規則の集合である．

(2)制御記述 CTM=(ER, ES, SS)．

ES, ER は，それぞれ，実行手順の制御規則の集合，制御規則とプロセスの関係である．SS∈ER は開始規則である．

(3)プロダクト記述 PDM=(PD,DS)．

PD,DS は，それぞれ，プロダクトの集合，プロダクトとプロセスの対応規則の集合である．

モデル MSDM の直観的な説明図を図１に示す．Piはプロセス，PDjはプロダクトである．太い有向枝，太い無向枝，あみかけ枝，細い有向枝はそれぞれ ER,SR,ES,DS に対応する．太い有向枝(ER)の中で，共通の元のあるもの(例えば，P2からP3，P4）は並列プロセスの fork，共通の先のあるもの(例えば，P3，P4からP2）は並列プロセスの join を表す．細い有向枝(DS)の方向は，入力プロダクトか出力プロダクトかを識別する．

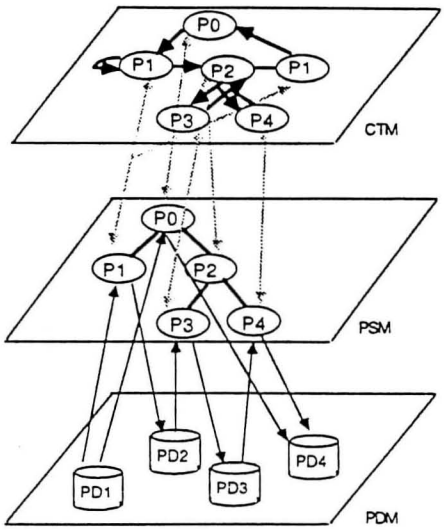


図１　記述モデルの概念図

```
SD = GEN DES
DES = SEL TFD TAD HRD
TFD = IAE FFA FBR DEP
FBR = IFR OFR TFR
TAD = IST STO ITA NPS TAM ACM DMD
HRD = LSD FOD SED
```

図２　ＰＳＭ記述

```
r = SD:r1                          /* 開始規則 SS */
r1 = GEN:r11 DES:r11
r11 = (SEL (TFD:r111 | TAD:r112)+ (HRD:r113)*)+
r111 = (IAE)+ FFA (FBR:r1111)+ (DEP)*
r1111 = IFR:r11111 @ OFR:r11112 @ TFR:r11113
r112 = IST (STO ITA (NPS (TAM @ ACM @ DMD))+)+
r113 = LSD | FOD | SED
```

図３　ＣＴＭ記述

3．モデルによる SD 法の記述

SD法は，ソフトウェア設計のために提案された方法論である[6]．SD法ではモジュール構成の良し悪しを判断する明確な基準を与え，その基準の下で良いモジュール構成を求めるための設計手順を示している．

SD法の設計過程の概要について説明する．先ず，データフローDF図(dfd)で入力から出力までのデータ変換の流れを記述する(GEN)．次に，DF図からシステムのモジュール構成を表す構造図(Structure Charts，sch)を求める

(DES)．構造図を求めるには，変換中心手法(TFD)，トランザクション中心手法(TAD)，発見的手法(HRD)などを適用する．

SD法の設計過程に対する PSM 記述を図２に示す．プロセス P のサブプロセスが P1,...,Pn であることを "P-P1 ... Pn" で表す．

図３は SD 法における１つの設計の流れを示している．プロセス ps と対応する制御規則 r を "ps : r" で表す．逐次制御と並列制御を，それぞれ，"r1 r2" と "r1 ∥ r2" のように示す．

SD法におけるプロダクトに対する PDM 記述を図４に示す．プロダクト pd がプロセス ps の入力，出力であることをそれぞれ "IN(pd)-{ps}"，"OUT(pd)-{ps}" で示す．

4．属性文法による記述

4.1　属性文法

属性文法は４項組 G-(Ge,A,R,C) で定義される．ここで，Ge-(N,T,P,S) は文脈自由文法で，A,R,C は属性，意味規則，文脈条件の集合である[4]．属性文法に対しては構文解析と属性評価が行われる．先ず，文法 Ge を用いた構文解析を行い，構文木を作成する．属性評価では，意味規則 R に基づいて，構文木上の各節点で文脈条件 C の下で属性 A の値を評価する．

4.2　属性文法記述

開発法に対する属性文法記述の概要を表１に示している．図５にSD法の属性文法記述を示す．

```
PD - { spe, sch, dfd, dgm, sid, sod }  /*プロダクトの集合*/
                    /*プロダクトとプロセスの対応規則*/
IN(spe) - { SD,GEN }
IN(sch) - { TFD,TAD,HRD,FFA,FBR,DEP }
OUT(sch) -{DES,TFD,FFA,FBR,DEP,TAD,IBR,OBR,TBR,
          HRD,LSD,FOD,SED }
IN(dfd) - { SLB,IIO,DES,SEL }
OUT(dfd) - { GEN }
IN(dgm)-{TFD,IAE,FFA,FBR,DEP,TAD,IST,
         STO,ITA,DMD }
OUT(dgm) - { SEL }
IN(sid) - { FFA }
OUT(sid) - { IAE }
IN(sod) - { FFA }
OUT(sod) - { IAE }
```

図４　ＰＤＭ記述

表１　属性文法記述の概要

属性文法	開発法
文脈自由文法	プロダクトの構文 実行手順
属性	プロダクトの性質 プロセスの入出力 プロセスの実行状態
意味規則	プロダクトの性質を定義する規則 入出力関係 実行状態の遷移関数
文脈条件	構文規則の適用条件 実行手順の適用条件

```
N - {SD,GEN,...,DFD,DGMs,...}        /*非終端記号*/
S - SD                               /*開始記号*/
T - {null,...}                       /*終端記号*/
A - {GEN.spe : SPE, SD.sch : SCH, ...  /* 属性 */
      DFD.dgm_no : int, DGMs.no : int, ... }

SD - GEN DES
 { GEN.spe - SD.spe;
   DES.dfd - GEN.dfd;
   SD.sch - DES.sch
 }
DES - DESb DFD DESbs                          ┐
 { DESb.in_dfd - DES.dfd;                     │
   DFD.i_str - DESb.out_dfd                   │
   DESbs.dgm_no - DFD.dgm_no;                 ① 
   DESbs.in_dfd - DESb.out_dfd                │
   DES.sch - DESb.sch + DESbs.sch             │
 }                                            ┘
DESbs - DESb DFD DESbs
  where (DESbs.dgm_no != 0) {...}
DESbs - null
  where (DESbs.dgm_no == 0)
 { DESbs.sch - null }
DESb - SEL DGM TFD_TADo SCH
       TFD_TADos SCH HRDs     { ... }
TFD_TADo - TFD                                ┐
  where (TFD_TADo.dgm_type == IS_TFD)         ②
 { TFD.dgm - TFD_TADo.dgm }                   │
TFD_TADo - TAD                                ┘
  where (TFD_TADo.dgm_type == IS_TAD) {...}
TFD_TADos - ...
TFD - IAE SID SOD IAEs FFA FBR
      DGM SCH FBRs SCH DEPs
 { ... }
IAEs - ...
FBRs - FBR DGM SCH FBRs {...}
FBRs - null {...}
FBR - IFR OFR TFR                             ┐
 { IFR.dgm - FBR.dgm;                         │
   IFR.in_sch - FBR.in_sch;                   │
   OFR.dgm - FBR.dgm;                         ③
   OFR.in_sch - FBR.in_sch;                   │
   TFR.dgm - FBR.dgm;                         │
   TFR.in_sch-FBR.in_sch;                     │
   FBR.out_sch-                               │
       IFR.out_sch+OFR.out_sch+TFR.out_sch    │
 }                                            ┘
   ...

DFD - DGMs                                    ┐
 { DFD.dgm_no - DGMs.no; ... }                │
DGMs - DGM DGMs                               ④
 { DGMs[1].no - DGMs[2].no + 1; ... }         │
DGMs - null                                   │
 { DGMs.no - 0;  ... }                        │
DGM - ...                                     │
SCH - ...                                     ┘
```

図５　属性文法による記述

実行手順を表す文脈自由文法（図５の下線付き部分）は，MSDM の SR（図２），ER（図３）から変換される．プロセスの入出力を表す属性（GEN.spe,SD.schなど）は，MSDM のDS（図４）から求められる．

開発法を厳密に定義するため，プロダクトの構文と性質，プロセスの実行状態などを記述する必要がある．プロダクトの構文は文脈自由文法として記述する（図５のあみかけ部分）．プロセスの性質，プロダクトの実行状態は属性(DFD.dgm_noなど）として表す．これらの属性の値を定義するため，一連の意味規則を導入する（図５の④）．２つ以上の実行手順をもつプロセスに対しては，実行手順の適用条件を文脈条件（whereではじまる述語）として記述する（図５の②）．

4.3 属性文法の評価

ソフトウェア開発においては事前にプロセス系列（属性文法の文字列に当たる）を確定できない．そこで，構文解析と属性評価を交互に行うことによって，構文木と属性を定める．図５の属性文法記述に対する１つの評価例を図６に示している．

構文木上の節点はプロセスに対応するもの（SD,GENなど）と，プロダクトに対応するもの（DFD,SCHなど）の２種類がある．属性文法の評価は，次の３つの基本操作からなる：

①プロセスに対応する節点の展開（図６の太い有向枝）
②プロダクトに対応する節点の展開（図６のあみかけ有向枝）
③属性の評価（図６の細い有向枝）

これらの基本操作はそれぞれ，ソフトウェア開発過程におけるプロセスの実行，プロダクトの評価，入出力及び実行状態の設定と対応する．又，X の相続属性の値が求まった時，かつその時に限り，X からの構文木の展開が行える．従って，同時に展開できる２つの節点（IFR,OFR,TFR,図５の③参照）は並列プロセスを表す．そうではない節点（DESb,DESbs，図５の①参照）は逐次プロセスを表す．

5．むすび

本稿では，ソフトウェア開発法に対する記述モデルを提案すると共に，属性文法による形式的記述について述べた．これまでに構造化分析法[2]と構造化設計法[3]の記述に適用してみた．今後は形式的記述に基づく検証について検討し，開発法に対する開発支援環境の構築を目指す予定である．

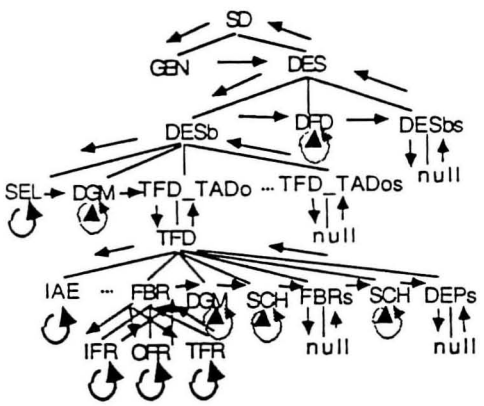


図６　属性文法の評価

参考文献

[1]T.DeMarco: "Structured Analysis and System Specifica- tion", Yourdon Press(1979).
[2]馮，大野，井上，菊野，鳥居：“属性文法による構造化分析 法の形式的記述”，信学技報，COMP88-3，pp.21-30(1988).
[3]馮，菊野，鳥居：“属性文法による構造設計法の形式的記述”，信学情報基礎論セミナー,pp.86-87(1989).
[4]D.E.Knuth:"Semantics of context-free languages", Math.Syst.Theory，2，2，pp.127-145 (1968).
[5]野村，井上，鳥居：”システム開発法 JSD の定義付の試み”，情報処理学会ソフトウェア工学研究会資料，58-1(1988).
[6]E.Yourdon and L.L.Constantine: "Structured Design", Yourdon Press(1978).

第1回ソフトウェア・プロセス・ワークショップに参加して

二木 厚吉

1　はじめに

ワークショップに参加する前にソフトウェア・プロセスのモデル化とその表現法に対する自分の素朴な気持ちポジションペーパに書いた。「素朴な」というのは深く考えもせず感覚的にという意味である。

ワークショップに参加している間は参加者の話しを聞きながら、ソフトウェアプロセスのことをそれなりに集中して考えた。議論はソフトウェア・プロセスの管理、またはソフトウェア・プロセスにおける人的要因の話題に偏り勝ちではあったが、ソフトウェア・プロセスに対する色々な立場からの興味の抱き方が分かって面白かった。モデル化とその表現方法に対する議論は、現状では概念的なものにならざるをえず、関心をもつ人も少数派という印象であった。

ワークショップに参加した後の今でも、参加する前に書いたポジションペーパの内容に対する気持は変わらない。むしろ一層その内容に対して(これもまたやはり)「素朴な」確信を深めている。以下では、そのポジションペーパの内容を少し整理して述べることで「参加報告」としたい。

2　ソフトウェア・プロセスに期待するもの

ソフトウェア・プロセスの話を最初に聞いたのは1987年にモンテレイで開かれた第9回ソフトウェア工学国際会議でのオスターワイル教授のプロセスプログラミングの講演であった。ソフトウェア自体の記述よりむしろソフトウェアを作る過程(プロセス)の記述が重要であるという議論はそれまでにもたびたび耳にしていたし、私自身もその重要性を認識していたので、彼が狙っているものはなるほどと納得できた。

確かに、ソフトウェア・プロセス(ソフトウェアの開発プロセス)のモデルを作り、その形式的で厳密な記述を与えることが出来れば次のようなことが期待できる。

- ソフトウェア開発過程をより厳密に客観的に理解できるようになる。
- ソフトウェア開発の種々のノウハウの蓄積とその再利用が可能になる。

- ソフトウェア開発の客観的で厳密なマニュアルが作成できるようになる。

これらのことが可能になればソフトウェア工学において最も困難であるとされていた部分が大きく進歩することになる。この意味でソフトウェア・プロセスの重要性やプロセス・プログラミングの効用に対する疑問はほとんどない。しかし、オスターワイルが考えているプロセスの記述が開発の手順を詳細に記述するというものであるらしいという点に関しては疑問が残った。

3　開発手順の有効な記述は可能か?

私の感じた疑問は、開発手順を詳細に記述するのは上で述べたような目標を達成するための有効な方法なのか、ということである。そのように感じた理由は次のようなものである。

- 詳細に記述された手順は融通性が乏しく再利用しにくい。
- 抽象度が高く再利用がしやすい記述のためには最小限の手順を規定するのが望ましい。
- 手順をベースにすると並行に動作する複数プロセスの記述などが繁雑になりやすい。

すなわちソフトウェア開発プロセスの知識化を狙うのに手順の記述を中心に据えたいわゆる「手続き的な方法」には問題が多いということである。実はこうした問題点はソフトウェア・プロセスの研究者の間でも認識されており、上に述べた批判は「オスターワイルのプロセス・プログラミング」に対してなされるべきであるということは後から知った。

いずれにしろ、ソフトウェア・プロセスの理想的な記述のためには、一般的な知識表現やソフトウェアの形式的仕様記述に共通に含まれる多くの困難な問題を解決しなければならない。これらの問題の完全な解決には長い時間を必要とすると思われる。重要なことは、今の時点で何を狙いどこに焦点を絞っていくかであろう。

4　我々のアプローチ

上で述べたような認識のもとで、我々が試みようとしているアプローチは次のようなものである。

> ソフトウェア・プロセスの記述をソフトウェア開発環境のオブジェクト指向的な形式仕様としてとらえ、現在我々の手元にある代数的仕様記述言語で記述してみる。

このようなアプローチを取ろうとする、基本的な意図は次のようなものである。

- ソフトウェア開発プロセスの「手順」を詳細に記述するのではなく、開発者が最低限従わなければならない制約を宣言的に記述する。
- ソフトウェアの開発環境を利用可能な資源とツールの集合体とし捉え、その厳密な仕様をなるべく整った構造を持たせて記述する。

このようなアプローチを取ることで得られるものは開発手順そのものの詳細な記述というよりは、開発の途中でのチェック・ポイントだけを示した「ゆるい記述」とでもいうべきものである。そうした「ゆるい記述」を形式的に整理して提供する方がソフトウェア・プロセスの知識化と再利用を計る上で有効であろうというのが現時点での我々の「素朴な」思い込みである。

このようなアプローチを取ることにしても、ソフトウェア・プロセス記述における最も困難な次のような問題はなんらかの方法で解決したければならない。

- 空間的にも時間的にも分散して動作する各々のエージェント(システム/人間)がある目的のために行なう協調動作をどのように記述するか。

この問題はソフトウェア・プロセスのモデル化と記述において最後まで残る困難なものであると思われ、ある統一的な方法ですべて解決するという性質のものではないであろう。しかしその解決のためには、少なくとも次のようなより基本的な問題を解決する必要があろう。

- 開発環境を構成するシステム、利用者、管理者などの間のインターフェイスをいかに記述するか。(機械・機械、人間・機械、人間・人間インターフェースの問題)
- ソフトウェアの開発活動を目的に向かって進めるということをいかに記述するか。(開発プロセスの評価の問題)

5　おわりに

ソフトウェア開発プロセスのモデルを作りその厳密な記述を与えることは大変挑戦的なテーマでありソフトウェア工学に対する大きな貢献が期待される。それはまたソフトウェア工学の側からの次の世代の情報システムの総合的なアーキテクチャの提示という大きな流れにつながる可能性を持っている。しかし、実際の開発現場である程度の成果が得られるまでには多くの着実な努力を積み重ねなければならないであろう。

ポジション・ペーパー

(株) SRA 松尾 正敏

(1) プロセス・モデルの分類

ソフトウエアのプロセスモデルが語られる時には、必ず現れるのが、Water Fall Model または Spiral Model であり、これらのモデルに勝るモデルはいまだ出現していないように思える。作業の繰り返しや、保守作業のとらえ方の問題が指摘されてはいるものの、大規模なソフトウエア開発作業をモデル化したものは、他に提唱されているとは思えない。しかしながら、Water Fall Model は、その生い立ちから考えると『管理指向』のプロセス・モデルといえるのではないだろうか。日本でも、ある種のプロジェクトでは、一回作業をすると、その作業は 2 度と行なってはならないとする考えがあり、現実の開発作業とのギャップを埋めるために、表面上の作業の名称と実作業とが異なっているという状況も存在している。プロジェクトを外側から管理する立場からは、Water Fall Model ほど、単純でわかりやすいものはないであろう。その一方で、モデルにしたがって作業する側は、表面をつくろうのに苦労しているわけである。ここでの根本的な問題は、ある作業またはプロセスの終了を規定できないことによると思われる。つまり、現実の作業を抽象化し、モデル化はしたが、ここのプロセスを厳密に規定できなかったといえるだろう。この問題を多少改善したモデルとして、 Spiral Model を考えることができるが本質的な問題はかわっていないようだが、いくつかのメトリックスと組み合わされた場合には、より有効になったことは確実なようである。

プロセス・モデルというよりは、現実のプログラムのモデルとして忘れてはならないものとして、 Lehman の E-Program モデルがある。このモデルは、Water Fall Model に比較して、抽象度が高いと思われ、直接的に、ソフトウエア・プロジェクトをガイドしたりするには難しいと思える。しかし、現実のソフトウエアとそれがおかれる外界との関係を示したものとして貴重なものであると思える。しかし、このモデルは日本ではあまり話題となっていなかったが、 Water Fall Model と保守の問題を考えると、忘れてはならないものではなかろうか。ただし、このモデルが、ソフトウエア開発作業を直接的にガイドしたり、コントロールことに有効というレベルのものではなく、むしろ、『概念的』な枠を示していると考えた方が良いように思える。

最近のホットな話題は、Process Programming であるが、現実的には、開発支援環境を考える際に、どのようにツールで支援していけば良いか、というところから考えられたモデルであるといえよう。したがって、実際のプロセス・モデルの組み立て方は、全くボトム・アップのアプローチであると思える。

(2) モデルの世界と現実の世界

現実の世界において、大規模な開発作業では、『管理』は非常に重要なことであることは確かなことである。しかし、これが当てはまらないプロジェクトがいくつも存在するところから、さまざまな亀裂が発生している。特に、ソフトウエア環境を考える事態に直面すると、そのとたん、Water Fall でも Spiral Model でも何の役にもたたなくなってしまう。しいて、役立たせようとすると、プログラミング以降の作業に関して、作られるプロダクトをもとにして、支援ツールとその統合環境というものがかろうじて考えられる程度である。このことは、『管理』用のモデルと、支援環境を考える際のモデルが異なったものでなければ有効でないことを示している。もちろん、モデルの相違のみでなく、プロセスやプロダクトに関する理解度の相違による面も多大にありそうだが、本質的に、目的のことなるモデルを利用しようとするところに問題があるのであろう。

こうしてみると、現実には、ソフトウエア・プロセスの解明がほとんどなされていないのが現状のようである。また、ソフトウエア・プロセスとモデルとの関係が、管理的な側面からしか考えられていないともいえよう。実際のところ、ソフトウエアの設計作業を支援しようとしても、現実には、設計作業がどのようなものであるかがきれいにモデル化することも困難な状況であることがわかった。

現在は、ある意味で、抽象度の高い Water Fall や Spiral モデルから、より現実に近い、もう少し抽象度の低いモデルを作り出さなければ、有効な支援環境を作成することができないような気がする。

品質管理機械化を志向したＣＡＳＥにおけるプロセス・コントロール

松本　正雄
（日本電気㈱）

はじめに

プロセス・コントロールの在りようを見ることは，重要であろう。ここでは，単に作業プロセスのコントロールのみならず，品質等の管理機能の機械支援のためも含めたプロセス・コントロールの１つの枠組を提案している。

要求事項

今後のＣＡＳＥに求められている要件として，少なくとも次があろう：

①直接作業のプロセスのガイドまたはコントロール‥‥工程分割による開発つまりフェーズ付け開発（Phased Development）アプローチのもとで作業階層構造（work breakdown structure）上にツールを対応づけ，手順書に則して開発者は，自らの判断に基き，ＣＡＳＥを使用し作業を進める形態から，手順自体をＣＡＳＥに覚えこませ（もちろん，プロジェクトの不測の事態に対処すべく変更可能性を保持），作業展開（process）をガイド（強制力をもたせたい場合は，コントロール）する形態へ。

②管理のプロセスの機械化‥‥管理の展開については，管理マニュアルやプロジェクト計画書（スケジュール，品質，原価などに関するもの）を念頭に，直接作業の状況を物語るオペレーショナル・レベルの管理データを提供してもらい，管理機能をオフライン的（および，しばしば遅延時間的）に発揮する形態から，管理データを作業現場から直結的（オンライン的かつ実時間的）に把握し，より正確で迅速な管理（オペレーショナル・レベル）を行う形態へ。

これら機能は，企業ベースの多くの開発の場に有効であろう。

要旨

主な点は以下のとうり：

①　直接作業プロセスと管理プロセスの干渉

ソフトウェア開発・進化プロジェクト（以下開発と略す）を離散事象よりなる一連のプロセスとみなす。開発は，さらに直接作業プロセスと品質・進捗・原価などの視点より直接作業を管理するプロセスの二層構造で成り立つものとする。直接作業プロセスのあるアクティビティの後置条件（Post-condition）が満されるか否かのチェックポイントで，管理プロセスの対応アクティビティが起動される。満されていると判定された場合は，次のアクティビティへ進んでも良いとする（トークン移動）。（そのアクティビティが，開始可能かは発火状態であるか否かによる。）満されていると判定されない場合は，上級管理者の判断により，後戻り，続行，中断待機のいずれかとなる。中断待機の場合は，管理者は直接作業プロセス変更，資源割付見直しなどのアクションをとるものとする。

②　プロセス・コントロール

開発プロジェクトにおけるプロセス・コントロールの記述方法として，オフィス業務フロー，petri ネット，Ｒネット，実体・関係（E-R）モデルなどが考えられるが，記述性の満足度の比較に基き，Petri ネットを候補とした。管理プロセスへの適用の妥当性の面ならびに二層プロセス構造間の干渉の取扱いの面で，Petri ネットの変形もしくは拡張が必要である。

③　品質管理機能の機械化

管理チェック・ポイント（トランジッション）でのチェックを機械化するため，プロセス・コントロール可能なＣＡＳＥに，次を付加する：

ﾌﾟﾛｾｽﾌﾟﾛｸﾞﾗﾐﾝｸﾞ国内ﾜｰｸｼｮｯﾌﾟposition paper（1989年2月）

プロセスプログラミングに関するある経験と研究

松本　吉弘（京都大学情報工学科）

１．まえがき　　プロセスプログラミングという語のもとに，ソフトウェアの製作過程を形式的に表現し，ソフトウェア品質，生産性，保守などの改善に役立たせようとする試みがなされてきた．筆者は，ソフトウェア生産過程を第２項に示すような考えでとらえ，それを工程管理，生産性管理の観点から，プロセスプログラミングの指向する方向へ向かって形式化する試みを行い，実務に役立たせてきた[MAT86][MAT87A]．しかし，さらにプロセスプログラミングを効果的に推し進めるためには，管理の目的から踏みでて，製作者個人とって日常の仕事を積極的に助けるような性質をもたせるように発展させるべきであろうとの実感を抱いている．そこで，第３項では，筆者がこれから進めようとしている研究の概要を述べ，その期待している効果について述べる．

２．ソフトウェア生産管理へのある利用　　ソフトウェア生産過程は，一般にウオータフォール型（逐次型）とよばれるモデルに従って行なわれているが，詳細には，単位仕事過程（unit workload:以下ＵＷと略称する）を網状に連接したモデルで表わすことができる．ＵＷとは，プロジェクトのなかのある区分，たとえば要求定義，設計，プログラミング，試験などにおいて，ひとりの担当者に与えられる仕事の単位のことである．ひとつのプロジェクトをとってみると，ＵＷは他のＵＷと連接され，ＵＷの集合は網の形状をなす．プロジェクトの進行に合わせて，網は並行して実行される．ＵＷネットワークは，古くからPERT(program evaluation and review technique)チャートまたは，PERTネットワークとして親しまれてきたものとほぼ同一のトポロジイに従う．ＵＷネットワークでは，網のノードに当たる部分にＵＷが割り当てられる．矢に相当するものは，ＵＷ間で交換される情報の流れを表わし，矢の向きの方向に情報が流れる．このＵＷネットワークには，従来のPERTネットワークに対してつぎのような新しい意味づけを与えた．

a.PERTでは，網ノードの属性は開始時刻，終了時刻，実行に必要な時間，開始必要条件，終了必要条件などに限られていたが，ＵＷネットワークの各ノードはこれに加えて対応するＵＷに関する目的生産物，目的管理指標（再利用度など），ＵＷ完成度，ＵＷ実行生産性などをデータ属性としてもたせる．

b.ＵＷネットワークは，計画フェースにおいて作成され，監視評価フェースにおいて実行される．前者においては，担当者の生産性に関する能力，再利用度とその生産性に対する影響を勘案して，ＵＷネットワークの各ノードのデータ属性が決定される．

c.監視実行フェースでは，ＵＷネットワークがプロジェクトの進行に合わせて計

算機上で仮想的に実行される．作業の進捗，消費資源（時間，計算機利用時間，使用材料など），実績完成度は，担当者に決められた時刻毎に入力させ，これをＵＷネットワークの計算機上の解釈系が分析して，ＵＷネットワークと時間軸との関連を追跡する．すなわち，計算機は，各ＵＷに関して，対応するＵＷネットワークノードのデータ属性に関してこの入力を分析し，ＵＷネットワークを仮想的に進める．
d.次のステップでは，第３項に述べるように，ＵＷネットワークの各ノードには，上記のデータ属性に加えて，手続き属性を与えたいと考えている．この手続き属性は，監視実行フェースで，管理者，ＵＷ担当者に対して，対話モードで必要なガイドを与えるためのものである．

以上に述べたＵＷネットワークによる生産管理システムは，d.に述べた機能を除いて，筆者が所属していたある大規模なソフトウェア工場で実施され，実績を挙げている[KAT88].

３．オブジェクト指向ＵＷネットワーク

上記の実績を基にして更にプロセスプログラミングへ向けてシステムを更新していくために，オブジェクト指向ＵＷネットワーク（object based unit workload network:以下ＯＢＵＷと略称する）の構想をもっている．これは，上記のＵＷネットワークノードをオブジェクトインスタンスに対応づけ，上記のデータ属性および手続き属性をインスタンスのもつ内部状態変数およびメソド（method）によってそれぞれ表わす．オブジェクトを指向する理由は，ＵＷというものの属性を，多くのクラス（たとえば，”仕様書作り”と”システム分析技術者の仕事”というふたつのクラスからの継承）の性質の継承によって表す必要があるからである．ＯＢＵＷの開発に対する要求の主な点を次に列挙する．
a.継承にassociation, classificationなどのいくつかの型を設けてインスタンス生成を行う．さらに，多重継承を可能とする．
b.ＯＢＵＷは，上記で述べた方法によって計画フェースにおいて，各プロジェクト毎に生成される．以下，計画フェースにおけるＯＢＵＷに対する要求について述べる．
＊　図1に，ＯＢＵＷの生成方式案を示す．ＯＢＵＷは，製作しようとするソフトウェアのもつ性質にもっとも適する特徴を一般にもっている．例えば，火力発電システム用ＯＢＵＷと電力系統システム用ＯＭＵＷでは，それぞれ異なった特徴あるパターンを示す．このパターンの特徴をドメイン記述言語によって定義する．ドメイン記述言語の文法をメタコンパイラに与え，構文規則，意味定義を生成し，これをドメイン記述言語コンパイラに与える．
＊　コンパイラには，与えられた構文規則，意味定義およびプロセスモデル定義に従って，ドメイン記述を分析し，プロセスプログラムジェネレータに必要なデータを与える．
＊　ここで対象とするプロセスモデルは，ＵＷネットワークである．プロセスモ

デル定義では，ＵＷネットワークを構成するために基底となる定義を与える．
＊　プロセスプログラムジェネレータには，各ＵＷに相当するインスタンスのもつ定数や変数の初期値を入力する．

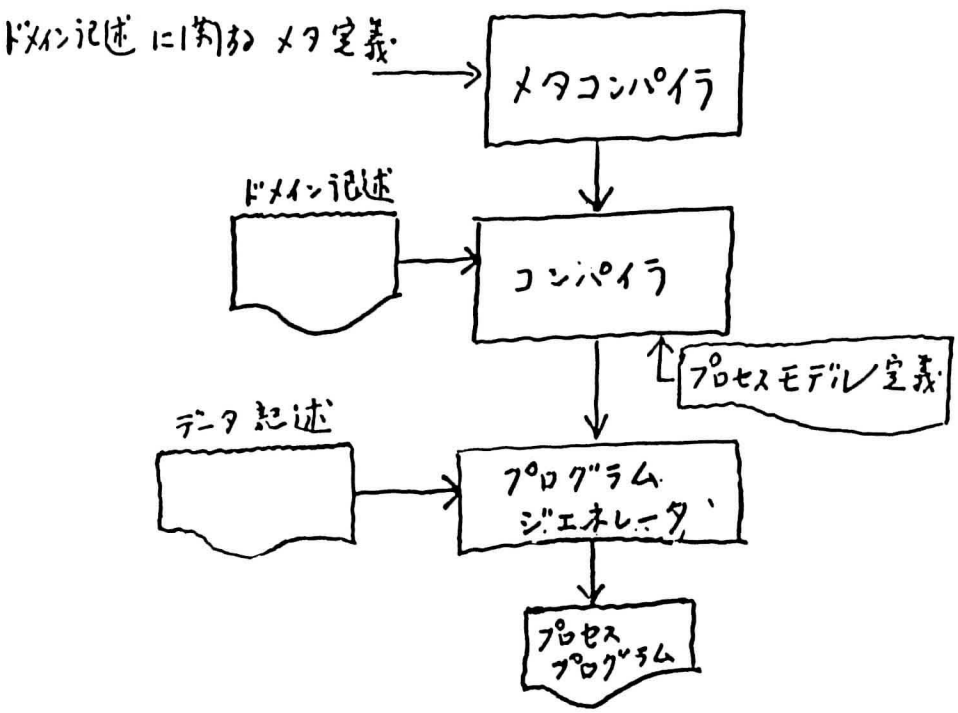
c.ＯＢＵＷは，監視実行フェースにおいて，実時間で実行される．以下，実行フェースにおけるＯＢＵＷに対する要求について述べる．[MAT87B]
＊　ＯＢＵＷの各インスタンスは，特別な実行環境（unit workload environment:以下ＵＷＥと略称する）において並行して実行される．
＊　ひとつのプロジェクトに対するＯＢＵＷの実行開始から終了まで各インスタンスの状態は継続して保存され，この継続をトランザクションと名付ける．トランザクションはプロジェクトの大きさに応じて長期間に亙る．
＊　インスタンスの活動は，初期条件が満たされているとき人によって開始され，完了条件が満たされているとき人によって終了される．
＊　ロールバック機能：インスタンスの実行軌跡上の中間状態において，その前の状態に戻ってやり直すことができる．このとき，並行して進められた関連する他のインスタンスも必要な前状態まで戻す．これをロールバックと称する．
＊　各インスタンスのメソドは，状態変数の書き換え，他のインスタンスとの交信，同期の他，使用者との対話も行う．また，各メソドは，ＵＷＥのもつツールの制御，ツール間のデータ受渡しの制御も行う．

４．むすび　　ＯＢＵＷは，ＵＷネットワークが目指した管理の合理化，計算機支援化だけではなく，個人が仕事の能率を挙げるために喜んで使うようなものにしたい．さらに，ソフトウェア自動生成へ向かって一歩でも近付くためのものにしたいと考える．従来の自動生成手順を，知的に連接するような，いってみればインテリジェントプログラム生成システム(intelligent program-generation:これは筆者の造語）ともよべるようなものを狙いとしている．

文献

[MAT86]Matsumoto,Y., Self-management of software production by every project member, Proc. of National Computer Conf., American Federation of Information Processing, pp.85-86(June 1986)
[MAT87A]Matsumoto,Y., Approaching productivity and quality in software production-- How to manage a software factory, Proc. of International Conf. on Information Technology Payoff, pp.103-115, Diebold Research Program(May 1987)
[MAT87B]Matsumoto,Y., Software reuse based on process programming system, Proc. of Fall Joint Computer Conf., American Federation of INformation Processing(Oct. 1987)
[KAT88]片山，他，ソフトウェア開発におけるプロジェクト管理，標準化と品質管理，Vol.41, pp.84-93(April, 1988)

図1　　プロセスプログラム生成



プロセスプログラミング言語の実現に関する考察

宮本 衛市

北海道大学工学部情報工学科

1　はじめに

ソフトウェアプロセスを記述するための一種のプログラミング言語は、通常のプログラミング言語にくらべて、きわめて異質な問題を内包している。通常の言語は、一般に、記述対象あるいは資源が計算機のなかで閉じているのに対し、ソフトウェアプロセスを記述するためには、人間の役割分担も含めないと問題が閉じないからである。すなわち、ソフトウェアプロセスの記述プログラムは未確定部分を有し、また処理系としても計算機側だけでは処理できず、人間側の処理に委ねなければならない領域が存在する。さらに、プログラムの進行に伴って、プログラム自体の変更もありうることから、プログラムはいわゆるリフレクティブな性質をもっていなければならない。このような条件を満たす言語はソフトウェアプロセスを記述するためばかりではなく、人間を含んだ系のスクリプトを記述することができるはずであり、言語としてもきわめて興味深いものがある。

2　ソフトウェアプロセス記述からみた条件

ソフトウェアプロセスを記述する上での特殊性として、記述の不完全性、人間含みの実行系、プログラムの動的な修正、任意の箇所からのバックトラック、人間側からの意味的な割込、最適な実行案の探索などがあげられる。ただし、これらは直交した事項ではなく、お互いに深く関連しあっている。

1. 記述の不完全性 ：プログラムとして事前に記述することができない場合や、いくつかの代替案があって確定できない場合などが考えられ、プロセスの一部として存在することは認められるが、事前には確定できない、ないしは詳細を記述できない場合がある。このような不確定な部分は実行にともなって確定してくることもあろうし、試行錯誤を繰り返す必要があるかもしれない。

2. 人間含みの実行系 ：ソフトウェアプロセスは、特にその前段では分析･設計などのアルゴリズム化の難しい部分を多く含み、その処理は人間に委ねなければならない。しかし、それはある処理の内部記述が人間の脳内にあることを除けば、一つのタスクとして取り扱うことができなければならない。

3. プログラムの動的な修正 ：プロセスの進行のやり直しにはいくつかの場合があろう。一つはプロダクトに欠陥があってやり直す場合で、同じプロセスをやり直す場合でも、どの箇所からやり直すかが問題となる。場合によっては、一部のプロダクトを再利用できるため、あるいはプロセス自体の欠陥が見つかり、それまでのプロセスを変更してやり直すこともある。一方、プロセス自体には欠陥がなくても、外部要因でプロセスを変更せざるをえない場合も起こる。したがって、プロセスプログラムはきわめて動的な性格をもっている。

4. 任意の箇所からのバックトラック ：前項でも述べたように、プロセスのやり直しがありうるが、やり直しの出発点は必ずしも最も手近なリスタート点とは限らない。一般には、人間側の高度な判断によることであり、指示された任意の箇所から与えられた条件で再実行できなければならない。

5. 人間側からの意味的な割込 ：プロセスの実行は人間の判断を常に受けており、その途中結果によっては、いわば意味的な割込とでもいうべき、プロセスとしては予測不能な割込を受け付けなければならない。上述の各処理は割込処理として受けることになろう。

6. 最適な実行案の探索 ：プログラムの実行からはいくつかの実行案があったとき、必ずしも深さ優先による探索が最善ではない。むしろ、全数探索的な試行錯誤を繰り返すよりは、最適と思われるプロセスを実行するのが最善であるのはいうまでもない。プロセスプログラミングの有用性は、究極的にはプロセスの最適化にあると見ることもできる。

3 プロセスプログラミング言語の実現の方途

前述の条件から、プロセスプログラムの実行環境は、通常のプログラミング言語の実行環境と、対話型処理における実行環境の中間をいくものと解することもできる。もちろん、その間のどこに位置するかによって、さまざまな言語または処理形態がありうるであろう。しかし、プロセスプログラムがプロセスのスクリプトをもっていることは必須の条件である。このようなプログラムの記述は、コンパイル型の手続き型言語では基本的に無理であり、インタプリーティブな言語にならざるをえないが、制御機構を陽に表現できなければならない。言語のクラスとしては、論理型、関数型、オブジェクト指向型などが考えられるが、プロセスは並行して働く資源を抱えていること、および動的に生成消滅されるプロダクトを管理しなければならないことから、並列オブジェクト指向のパラダイムに基本的に合致していると考えることができる。しかし、それはパラダイムのレベルでの考察であり、さらにモデル、言語、処理環境へとつめていかなければならない。

プロセスがいろいろな段階、あるいは活動範囲をもっていることから、現在、場を考慮した放送をメッセージの基本的な伝送形態とした並列オブジェクト指向モデル [1] でプロセス記述を把握できるかどうか検討している。このモデルはレベルが低いので、実際にはこのモデルに沿って、前述の条件を満たすような言語を設計しなければならない。プロセスの実現モデルとしては黒板モデルなどが考えられるが、プロセスモデル自体の動的な変更すらありうるのかどうかなど、モデルの根幹にかかわる検討事項が多くあり、今後の検討課題である。

4　おわりに

プロセスプログラムの記述という観点から、そのための記述言語として備えていなければならない条件を考察してみた。そのための言語は、従来の言語と対話型処理の間をいくものであり、言語自体としても大変に興味深い。基本的には、人間の活動のスクリプトを計算機に理解させることにつながり、新しいパラダイムを開拓する必要があり、その実現には今後解決していかなければならない問題点が多い。

参考文献

[1] 渡辺、原田、三谷、宮本: 場とイベントによる並列計算モデル－－ Kamui88 、コンピュータソフトウェア、VOL.6,NO.1(1989),PP.41-55.

付録

付録 1：

第一回ソフトウエア・プロセス・ワークショップ
プログラムと運営方針

付録 2：

第一回ソフトウエア・プロセス・ワークショップ
募集・申込書

付録 3：

第二回ソフトウエア・プロセス・ワークショップ
募集・申込書

付録 4：

6th INTERNATIONAL SOFTWARE PROCESS WORKSHOP
Call For Paper

第１回ソフトウェア・プロセス・ワークショップ　　1989/2/2-4@伊豆研修所

プログラムと運営方針

プログラム		
2/2（木）	13:00 - 13:30	受付
	13:30 - 17:00	オープニング・セッション 参加者全員のショート・プレゼンテーション（各５分） Chairman: 片山
	17:00 - 19:00	夕食休憩
	19:00 - 22:00	自由討論（１） Chairman: 久保
2-3（金）	08:00 - 09:00	朝食
	09:00 - 12:00	テーマ別セッション（１） －Suitability of Existing Models－ Chairman: 岸田，Reporter:松尾
	12:00 - 14:00	昼食休憩
	14:00 - 17:00	テーマ別セッション（２） －Instantiation of Process Models－ Chairman: 落水，Reporter:大篭
	17:00 - 19:00	夕食休憩
	19:00 - 22:00	自由討論（２） Chairman: 二木
2-4（土）	08:00 - 09:00	朝食
	09:00 - 12:00	テーマ別セッション（３） －Dynamics of Process Models and Model Instances－ Chairman: 片山，Reporter:中川

ワークショップの運営方針

（１）　今回のワークショップでの討論の話題を来る１０月に予定されている 5th ISPW での３つのテーマ：

- Suitability of Existing Models
- Instantiation of Process Models
- Dynamics of Process Models and Model Instances

に合わせることにします．それぞれのテーマの詳細は，別添の Call for Paper（４ページ）を参照してください．

***** 次ページに続く *****

（2）　冒頭のオープニング・セッションでは，参加者全員が，そろぞれの持つ問題意識を上記３つのテーマの**いずれか１つ**と関連させて，１人５分間ずつのプレゼンテーションを行います（OHP 1-2枚）

（3）　第２～３日の昼間に予定されている３つのテーマ別セッションでは，それぞれのチェアマンが，参加者のポジション・ペーパおよびオープニング・セッションでの発表の中から，取り上げるべき問題点を選択し，セッションの最初に簡単な（１５分程度の）イントロダクションを行って，討論に入ります．発言者の指名や発言時間の割り当てを含めて，セッションの運営に関する絶対的な権限はチェアマンにあるものとします．したがって，オープニング時の各人の発表は，自分の問題意識をチェアマンにアッピールし，それを該当セッションで取り上げてもらうための一種の Sales Talk だとお考えください．

（4）　第２～３日夜のセッションでは，昼間論じ足りなかったことがらや，取り上げられなかった問題点等について，若干のアルコールを入れながら，オフレコで自由に討論したいと思います．

（5）　［事前準備］　全参加者は，オープニング・プレゼンテーション用のＯＨＰを１～２枚用意してください（配布用のコピーも２６部作って来てください）．テーマ別セッションで自分の意見または問題提起を発表したい方は，そのためのＯＨＰも御用意ください．ただし，それが取り上げられるか否かは，すべて，チェアマンの胸三寸にかかっています．

（6）　［レポート］　ワークショップ終了後，参加者全員にポジション・ペーパをRewriteして，御提出いただきます（カメラ・レディ，Ａ４版で数ページ）．締切りは２月２０日（月）厳守とさせていただきます．テーマ別セッションでの討論の記録は，担当のチェアマンおよびレポータが整理し取り纏めます（こちらの締切りは２月末日）．

（7）　［ISPW-5への応募］　４ページの Call for Paper にあるように，第５回 ISPW へのポジション・ペーパの締切りは３月1日です．今回の討論の成果を踏まえて，それぞれが英語の Position Paper を書いて応募することを心がけましょう．

第1回

ソフトウェア・プロセス・ワークショップ

参 加 者 募 集

主　催：日本ソフトウェア科学会 ソフトウェア・プロセス研究会
情報処理学会　ソフトウェア工学研究会（予定）
ソフトウェア技術者協会

期　日： １９８９年2月2日（木）～4日（土）　**会　場：** 機械振興協会伊豆研修所（静岡県伊東市）
定　員： ３０名　**参加費**（予定）：　3万円（一般）　8千円（学生）
実行委員長： 片山卓也（東京工業大学）
実行委員　： 阿草清滋（京都大学）　落水浩一郎（静岡大学）　岸田孝一（ＳＲＡ）
熊谷　章（ＰＦＵ）　久保宏志（富士通）　紫合　治（日本電気）
野村敏次（日本電子計算）　二木厚吉（電総研）　松本吉弘（東芝）

ソフトウェア開発の生産性や品質を考えるさいには，まず，いったいわれわれがどんな工程（プロセス）でどんな製品（プロダクト）を作っているのかを，はっきり認識することが必要です．

国際的なソフトウェア・エンジニアリング・コミュニティでは，ここ数年，とくにソフトウェア・プロセスを形式的にモデル化し，その特性を解明することに多くの関心が寄せられています．第8～9回の ICSE では，連続して「プロセス」が会議の基調テーマとして取り上げられ，それと関連して，ソフトウェア・プロセスに関する国際ワークショップ（ISPW）が 1984年以降すでに4回も開催されてきました．

このワークショップから生まれた主な成果としては，たとえば，開発にともなうさまざまなリスクの管理を含んだ渦巻型のプロセス・モデル（B. W. Boehm）や，木構造の契約型モデルにもとづくプロジェクト管理支援環境 ISTAR（M. M. Lehman），あるいは，厳密なアルゴリズムによるプロセス記述を目指したプロセス・プログラミングのアイデア（L. Osterweil）などがあげられます．

次回の ISPW は来年秋に米国 Maine州 Kennebunkport で開かれます．また，その次の第6回は 1990年のやはり秋に，今度は**日本での開催**が予定されています．討論テーマの候補としては，既存の各種モデルと現実のプロセスとの対比，特定の問題やプロジェクトに対するモデルの適応のメカニズム，モデルの進化とその支援メカニズム，モデルの自動的な解釈／実行およびそれルをベースとするプロジェクト管理，プロジェクト運営の途中での動的なモデルの変更，などがあげられています．いずれにせよ，モデルの現実への適用に関する実際の経験を参加者各自が持ち寄って，地に足のついた討論を展開しようというのが，当面の ISPW の基本的運営方針です．

そういう意味では，さまざまな工業分野において，すでに世界に冠たる製造技術・品質管理技術を持ち，そのソフトウェア領域への応用を推進しつつある（？）日本からの技術的な貢献が，かなり期待されているといえましょう．そこで，今回，第6回 ISPW の日本開催のための準備活動の出発点として，また，第5回 ISPW への参加申込みを Encourage する（Position Paper の締切りは来年3月1日）という2重の目的を持って，日本で初めてのプロセス・ワークショップを開催したいと思います．

ソフトウェア・プロセスに関心を持ち，何らかの実践あるいは研究を心掛けておられる方々の積極的な参加をお待ちします．

参加希望者は裏面の申込用紙（Identification + 簡単なアンケート形式の Position statement）に御記入の上，FAX または郵便で，実行委員の岸田孝一氏（SRA）宛お送りください．**締切りは１２月１０日（必着）**です．ただし，申込み多数の場合には，実行委員会でPosition Statement の内容を審査の上，参加者を選考させていただきますので，あらかじめ御了承ください（選考結果は，１２月２５日までに全申込者にお知らせします）．

********** 裏面に申込用紙があります **********

送付先：　　締切：12月 10日

〒102　千代田区平河町 1 - 1 - 1 （株）SRA　専務取締役　岸田孝一
TEL: 03 - 234 - 2610　Fax: 03 - 237 - 7006

第1回　ソフトウェア・プロセス・ワークショップ
参加申込用紙

氏　名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿（ふりがな）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

年　齢：＿＿＿＿＿＿（歳）　性　別：□ 男 □ 女

会社（大学）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

部門（学部）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿　役　職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住　所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

TEL：[　－　－　　内線　　]　FAX：[　－　－　　]

Position Statement:

次の３つの問いのいずれかに，簡単にお答えください．紙面が足りない場合は，別紙を追加していただいてかまいません．

Q1. あなたは，ソフトウェア・プロセスの形式的モデル化や，そのモデルの現実への適用にどのような関心をお持ちですか？

Q2. あなたは，既存のいろいろなプロセス・モデルのうち，どれが最も現実的だ（あるいは興味深い）と思われますか？　それはなぜですか？

Q3. あなた自身の周囲にある現実のプロジェクトにおける開発プロセスは，どんな風にモデル化できそうですか？　また，そのプロセスにおいて，改善を要する主な問題点は何ですか？

Call for Position Papers

第 2 回
ソフトウェア・プロセス・ワークショップ
参 加 者 募 集

主　催：日本ソフトウェア科学会－ソフトウェアプロセス研究会
情報処理学会－ソフトウェア工学研究会
ソフトウェア技術者協会（ＳＥＡ）

期　日： １９９０年２月１日（木）～３日（土）
会　場： 五景館（静岡県伊東市）
定　員： ３０名
参加費： ３万円（主催団体会員） ４万円（一般）
期間中の宿泊費・食費および資料代をふくみます．
実行委員長： 片山卓也（東京工業大学）
実行委員　： 落水浩一郎（静岡大学）　岸田孝一（SRA）　久保宏志（富士通）　二木厚吉（電総研）

ソフトウェア開発の生産性向上や製品の品質改善を考えるさいには，まず，いったいわれわれがどんな工程（プロセス）でどんな製品（プロダクト）を作っているのかを，はっきり認識することが必要です．

国際的なソフトウェア・エンジニアリング・コミュニティでは，ここ数年，とくにソフトウェア・プロセスを形式的にモデル化し，その特性を解明することに多くの関心が寄せられています．第８～９回の ICSE では，連続して「プロセス」が会議の基調テーマとして取り上げられ，それと関連して，ソフトウェア・プロセスに関する国際ワークショップ（ISPW）が 1984年以降今年までの間に５回開かれました．そして，来年の第６回 ISPW は，１０月末に日本（北海道）での開催が決まっています．それと併設して，ソフトウェア・プロセスに関する第１回の国際シンポジウムが，日米欧の３ヶ所で連続開催される予定です．

すでに，さまざまな工業分野において，すぐれた製造技術・工程管理技術の存在を実証した日本に対して，ソフトウェア領域での技術的な貢献が，世界中からかなり期待されているといえましょう．そこで，このさい，今年の冬に引き続いて，第６回 ISPW へのウォーミング・アップを兼ねた国内プロセス・ワークショップを，上記の要領で開催したいと思います．

今回は，特に討論テーマを限定せず，ソフトウェア・プロセスの認識とモデル化，その形式的記述，プロセスの自動化と支援環境，モデルと現実の対比，プロダクト（アプリケーション・ドメイン）とプロセスとの相関など，参加される方々の意識に対応したフレキシブルなプログラム編成を試みたいと考えています．ソフトウェア・プロセスに関心を持ち，何らかの実践あるいは研究を心掛けておられる方々の積極的な参加をお待ちします．

参加希望者は裏面の申込用紙（Identification ＋ 簡単なアンケート）に御記入の上，Ａ４版３～５ページのポジション・ペーパを添えて，郵便または FAX または E-Mail で，実行委員長宛にお送りください．**締切りは１２月１５日（必着）**です．ただし，申込み多数の場合には，実行委員会でポジション・ペーパの内容を審査の上，参加者を選考させていただきますので，あらかじめ御了承ください（選考結果は，１２月２５日までに全申込者にお知らせします）

参加申込先
〒152　東京都目黒区大岡山２－１２－１
東京工業大学　工学部　情報工学科
片山卓也
TEL: 03 - 726 - 1111（内 3045）　FAX: 03 - 729 - 1399
E-Mail: katayama@cs.titech.ac.jp

送付先：　　締切：12月 15日

〒152　目黒区大岡山２－１２－１　東京工業大学　工学部情報工学科　片山卓也
TEL: 03 - 726 - 1111 (内3045)　Fax: 03 - 729 - 1399
E-Mail: katayama@cs.titech.ac.jp

第2回　ソフトウェア・プロセス・ワークショップ
参加申込用紙

氏　名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿（ふりがな）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

年　齢：＿＿＿＿＿（歳）　　性　別：□ 男　□ 女

会社（大学）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

部門（学部）：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

役　職：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

住　所：（〒　　）＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

TEL: [　　－　　－　　内線　　]　FAX: [　　－　　－　　]

E-Mail: ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

申込種別（いずれかにチェック）：

□ ソフトウェア科学会員　□ 情報処理学会員　□ SEA会員　□ 一般

アンケート:

御面倒ですが，以下の質問にお答えください：

[Q1] これまでに５回開催された ISPW についてどの程度御存じですか？

□ 参加したことがある　□ Proceedings を読んだ　□ 聞いたことがある　□ 知らない

[Q2] L.Osterweil が 1987年に提唱したプロセス・プログラミングの概念は，その後いろいろな波紋を巻き起こしましたが，あなたはこれについてどの程度御存じですか？

□ よく知っている　□ 関連論文を読んだ程度　□ 聞いたことがある　□ 知らない

[Q3] CMU/SEI では，ソフトウェア開発組織におけるプロセスの成熟度を次の５段階に分類しています：

1. Initial(Chaotic): まだよく定義されておらず，コントロールもされていない．
2. Repeatable(Intuitive): いちおう定常的に利用されているが，文書化はされておらず，その使い方は個人に依存するところが大きい．
3. Defined(Qualitative): よく定義され，理解され，組織的に運用されているが，まだ定性的である．
4. Managed(Quantitative): 管理対象のプロセスは，単に理解されているだけでなく，定量化され，計測され，制御されている．
5. Optimizing: プロセスをモニタリングして得られたデータを分析した結果にもとづいて，プロセス自体の改善が行われている．

あなたが所属する組織におけるプロセスの成熟度は，どのレベルですか？　下のスケール上に×印でマークしてください．

1　　2　　3　　4　　5

・－－－－－・－－－－－・－－－－－・－－－－－・

[Q4] 世の中では，Waterfall, Spiral, LST, Contract など，いろいろなプロセス・モデルが提案されていますが，あなたの周辺の典型的なプロジェクトにおけるソフトウェア開発は，どんなモデルにあてはまると思いますか？　簡単に説明してください．

＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

6th INTERNATIONAL SOFTWARE PROCESS WORKSHOP

Support for the Software Process

Hokkaido, Japan 29 - 31 October 1990

(Sponsored by the Rocky Mountain Institute of Software Engineering - cooperation requested from Japanese and US professional societies)

ORGANIZING COMMITTEE

Gail Kaiser Takuya Katayama
Kouichi Kishida Bill Riddle Jack Wileden

The 6th International Software Process Workshop will focus on the provision of support for the software process by software development environments. Of particular concern are environment architectures and mechanisms that directly support the recording, monitoring, guiding and enaction (that is, interpretation or execution) of process models. Among the more important issues are the following:

Process Oriented Architectures

Environments that support process model enaction will need specialized architectural features.

- What overall environment structures are needed for software process support?
- What environment components e.g. process model interpreters, are required?
- How can process support be built on existing bases such as PCTE?

Process-related Mechanisms

Process oriented environments will need appropriate mechanisms to create, customize and instantiate process models, as well as to record, monitor, guide and enact model instances.

- What mechanisms are needed to record, monitor, guide and enact process model instances?
- What mechanisms are needed to create, customize and instantiate process models?
- What mechanisms are needed for dynamic modification and refinement of process model instances?

Support for Development-In-the-large

Industrial scale software development requires process models and supporting mechanisms which address group cooperation and coordination, as well as support and/or enforce project-wide management policies.

- How is group coordination and cooperation supported?
- How are policies supported or enforced?
- How is support for management activities integrated with support for other project activities?

The three day workshop, which will be held in Hokkaido, Japan will consist of intensive discussion of these issues by at most 35 Participants. Prospective participants should submit a maximum 3 page position paper in English by 1 April 1990, explicitly addressing one of the workshop issues, and suitable for publication in the proceedings. A small number of participants will be requested to prepare short keynote presentations to initiate discussion. Papers (6 copies or electronic mail) should be sent to:

William E. Riddle
Rocky Mountain Institute of Software Engineering
PO Box 3521, Boulder, Colorado, USA
email: ispw6%sunset@uunet.uu.net

electronic mail submissions should include author's full address and telephone number

ソフトウェア技術者協会（SEA）は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985年 12月に設立されました．

その主な活動は，機関誌 SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約 200人にすぎなかった会員数もその後飛躍的に増加し，現在，北は北海道から南は沖縄まで，1650 名を越えるメンバーを擁するにいたりました．法人賛助会員も約 60 社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます． SEA は，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

代表幹事： 岸田孝一

常任幹事： 臼井義美　久保宏志　熊谷章　佐藤千明　藤野晃延　松原友夫　吉村鉄太郎`

幹事： 青島茂　天池学　飯沢恒　岩田康　岡田正志　落水浩一郎　片山禎昭　川北秀夫　岸勝義　杉田義明　武田知久　田中慎一郎　玉井哲雄　玉川滋　中来田秀樹　中園順三　中野秀男　野村敏次　野村行憲　針谷明　平尾一浩　深瀬弘恭　藤本司郎　北條正顕　細野広水　盛田政敏

会計監事： 辻淳二　吉村成弘

常任委員長： 臼井義美（技術研究）　久保宏志（企画総務）　藤野晃延（会誌編集）　杉田義明（セミナー・ワークショップ）

分科会世話人 環境分科会(SIGENV)：田中慎一郎　渡邊雄一

管理分科会(SIGMAN)：相沢圭一　川北秀夫　芝原雄二　野々下幸治

教育分科会(SIGEDU)：大浦洋一　杉田義明　中園順三

ネットワーク分科会(SIGNET)：青島茂　野中哲

法的保護分科会(SIGSPL)：能登末之

ＣＡＩ分科会(SIGCAI)：大木幹雄　中谷多哉子　中西昌武

ドキュメント分科会(SIGDOC)：田中慎一郎　野辺良一

支部世話人 関西支部：臼井義美　中野秀男　盛田政敏

横浜支部：熊谷章　林香　藤野晃延　松下和隆

長野支部：小林貞幸　佐藤千明　細野広水

名古屋支部：岩田康　鈴木智

九州支部：植村正伸　小田七生　藤本良子　平尾一浩　松本初美　中島泰彦　後藤芳美

ＳＥＡＭＡＩＬ編集グループ：岸田孝一　佐原伸　芝原雄二　関崎邦夫　田中慎一郎　中村昭雄　長井修治　成沢知子　野辺良一　藤野晃延　渡邊雄一

ＳＥＡＭＡＩＬ　Ｖｏｌ．４，　Ｎｏ．１０－１１　　平成元年１１月２４日発行

編集人　岸田孝一

発行人　ソフトウェア技術者協会（ＳＥＡ）

〒１０２　東京都千代田区隼町２－１２　藤和半蔵門コープビル５０５

印刷所　サンビルト印刷株式会社　　〒１６２　東京都新宿区築地町８番地

定価　１０００円　（禁転載）

ソフトウェア技術者協会
〒102 東京都千代田区隼町2-12 藤和半蔵門コープビル505
TEL.03-234-9455 FAX.03-234-9454