ソフトウェアプロセスの基本制御構造†

何 克清†† 宮本衛市†††

本論文では，ソフトウェアプロセスを記述するうえでの基本制御構造を提案する．その制御構造の特徴は，ソフトウェアプロセスのバックトラック構造を制御構造の基本的な枠組とするものであり，その構造は従属直積バックトラック構造，独立直積バックトラック構造および直和バックトラック構造の3つからなる．これらを再帰的に用いることにより，バックトラックを基本とする任意のソフトウェアプロセスを構造的に記述することができる．それを実証するため，いろいろなソフトウェアプロセスの実例の記述を行った．また，本論文で論じた基本制御構造を用いて，ソフトウェアプロセスの作成手順，ソフトウェアプロセスの複雑度の評価方法と簡略化ルールについても提案し，実例を用いて説明した．

1. はじめに

ソフトウェアの品質向上と開発コスト減少のため，SPM (Software Process Model) に関する研究が盛んに行われ，様々なモデルが提案されてきた．そのようなモデルとして，たとえば，Royce の Waterfall Model[1]；Basili らの Iterative Enhancement Model[2]；Agresti の Prototyping Model[3]；Boehm の Spiral Model[4]；Humphrey らの SP 成熟度モデル[5]；Potts の Generic Model[6]などがある．しかし，これら SPM の記述が形式的には行われず，自然言語を利用しており，それがこれら SPM の普及を妨げてきた原因の1つともなっている．

一方，Osterweil がソフトウェアプロセスもまたソフトウェアである[7]と提唱して以来，SP (Software Process) 記述に関する研究が注目されている．Osterweil らは，ソフトウェア開発過程における変更管理に重点を置いたソフトウェアプロセスを記述するために，プロセス記述言語 APPL/A を開発した[8]．APPL/A は ADA を基本とし，変更管理に必要な永続データの関係定義や制御構文などを追加した．Williams はソフトウェアの開発活動を記述するために，各活動を前置条件，行動，後置条件およびメッセージの4つ組で表し，正規表現に並列活動の記述を追加した拡張正規表現で，活動間の関係を記述することを提案し，各種の SPM の記述を行った[9]．Katayama は属性文法に基づく階層的関数的ソフトウェアプロセスモデル HFSP を提案し，それに基づく言語 AG を開発した[10]．そこでは，SP の構造は関数を階層的に展開することによって構築される．Inoue らは関数型言語 PDL を開発し，ソフトウェアプロセスを段階的に詳細化する方法を提案した[11]．そこでは，SP の構造は各プロセスに設定される前置および後置関数で構築される．

これらの提案では，ソフトウェアプロセスにとって不可避なバックトラックを，基本的には繰返し構文を用いて，プロセス記述のプログラムに意味的に組み込もうとしている．大木，落水は Prolog ルールにより，ソフトウェア設計プロセスにおけるバックトラック制御構造を記述しようとした[12]．しかし，SP の記述は，本論文で述べるように，通常のプログラムの記述とは本質的に異なるというべきであり，SP における基本的な制御構造の解明はまだ行われていない．われわれは Williams の拡張正規表現の考え方を継承しているが，SP の基本活動から SP の正規関係を定義し，これにバックトラックを統合して，バックトラックを基本的な枠組としてもつ SP の制御構造を提案する．

SP 自身もソフトウェアとして見たとき，それは通常のソフトウェアとは様相の異なる制御構造とデータ構造を持っている．なぜなら，SP は人間が主体となって行う創造的な活動であり，その活動の失敗と再試行は避けることができない．このようなバックトラックはプログラムにおける繰返しとは異なり，活動の事後に生じる手戻しであり，SP にとって基本的な活動形態である．したがって，バックトラック構造を SP 制御構造の基本的な枠組として捉える必要がある．このような観点にたち，本論文では，SP の制御構造の基本的な枠組を提案し，これらを利用して SP を構造的に記述できることを明らかにする．

† Fundamental Control Structure of Software Process by KEQING HE (State Key Laboratory of Computer Software Engineering, Wuhan University) and EIICHI MIYAMOTO (Department of Information Engineering, Faculty of Engineering, Hokkaido University).

†† 武漢大学ソフトウェア工学国家重点研究実験室
††† 北海道大学工学部情報工学科

以下，第2章では SP の2種類の基本的な制御構造，すなわち，直積バックトラック構造と直和バックトラック構造を提案する．第3章では様々なレベルのソフトウェアの開発活動を例として取り上げ，その形式的な記述を行う．第4章では McCabe 指標[13]に基づいて，SP の制御構造の複雑度を定義し，SP の簡略化について議論する．最後に，第5章では本論文の提案した SP の制御構造の問題点を議論し，今後の研究課題を挙げる．

2. SP における基本制御構造

本章では SP の構造を分析し，SP の基本制御構造を提案する．そして，それに基づいた SP 作成法についても議論する．

2.1 SP 活動

個々の SP 活動を前件処理，活動本体，後件処理からなるものとする．また，1つの活動はいくつかのサブ活動に展開され，さらにこれが再帰的に適用されることにより，SP は階層的な木構造で構成されるものとする．そこで，活動の単位を図1のように設定する．

（a） 前件処理：前提条件を満足させて，活動本体を起動する．バックトラック時には代替となる前提条件を設定する．代替案をたてられないときには，制御構造に基づいて，さらに前方にバックトラックする．

（b） 活動本体：与えられた条件のもとで活動本体を実行する．必要に応じて，さらに階層的にサブ活動に展開される．

（c） 後件処理：活動本体の実行結果を保存し，評価を行い，実行の成功・失敗を決定する．失敗の場合には制御構造に基づいてバックトラック先を判断する．

2.2 SP の基本制御構造

SP の制御構造は活動間の制御関係を記述するものである．したがって，SP の基本制御構造を解明するためには，その活動間の基本的な関係を分析する必要がある．いま，一連の活動を考えた時，後続の活動が前出の活動に従属する場合と従属しない場合とがある．ここで後続の活動の機能が前出の活動の結果に従属している場合を従属関係という．前後の活動間に従属関係がない場合を独立関係と言う．このような観点から，SP 活動の関係を分析して，SP に関する正規関係を設定する．

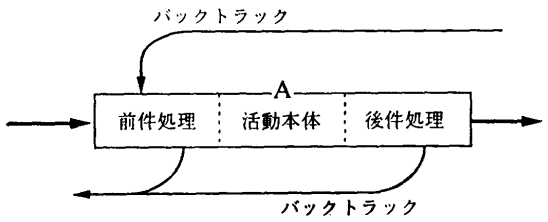

図 1 SP の活動
Fig. 1 Activity of software process.

2.2.1 正規関係

SP の正規関係は直積関係と直和関係からなっているものとする．

（1） 直積関係

一連の活動がすべて成功したときのみ全体の成功となり，1つの活動でも失敗すれば全体の失敗となるような関係を直積関係という．直積関係は従属直積関係と独立直積関係からなる．

a．従属直積関係

これは相異なる性質をもつ活動 $A_1, A_2, \cdots, A_n$ に従属関係があり，全体が成功するためにはこれらすべてが成功する必要があるような関係である．これを $A_1 * A_2 * \cdots * A_n$ のように表す．例えば，品質機能展開活動を5つの活動の従属直積関係で表すと次のようになる．

品質機能展開活動＝
　要求品質展開＊品質特性展開＊
　設計品質の設定＊工程設計の実施＊
　工程管理方法の決定

この場合，品質機能展開活動が成功するためには，右辺のすべての活動が成功しなければならず，1つでも失敗すれば全体の失敗となる．

b．独立直積関係

これは活動 $A_1, A_2, \cdots, A_n$ に独立関係があり，全体が成功するためにはこれらすべてが成功する必要があるような関係である．これを $A_1 \| A_2 \| \cdots \| A_n$ のように表す．もし，$A_1, A_2, \cdots, A_n$ が同じ活動であれば，$A\|^n$ のように表す．独立な関係にある各活動は並列に実行することも可能である．例えば，プログラミングをいくつかのサブシステムに分けて行う場合，次のように表すことができる．

プログラミング活動＝
　サブシステムプログラミング1‖
　サブシステムプログラミング2‖…‖
　サブシステムプログラミング n
＝サブシステムプログラミング$\|^n$

（2） 直和関係

これは活動 $A_1, A_2, \cdots, A_n$ の間に選択可能な関係があり，少なくとも1つの活動が成功すれば全体の成功と見なすような関係である．これを $A_1+A_2+\cdots+A_n$ と表す．例えば，アルゴリズム設計に，いくつかの代替案があるとき，次のように表すことができる．

アルゴリズム設計＝アルゴリズム設計(案1)
＋アルゴリズム設計(案2)

この場合，少なくとも1つの活動が成功すれば全体の成功につながり，右辺全体がすべて失敗したときのみ全体の失敗と考える．各活動は独立に実行することもできる．

2.2.2 基本制御構造

表1に示す関係活動を実行したとき，成功の場合には，SP の制御は後に続く活動に渡すが，失敗の場合には，SP にバックトラックを起こさせなければならない．それゆえ，SP に本質的に附随するバックトラックを正規関係に統合して構造化し，SP の基本制御構造とする．そこで，前項の正規関係に基づいて，SP に関する次の2つの基本制御構造を設定する．

（1） 直積バックトラック構造

直積バックトラック構造は従属直積バックトラック構造と独立直積バックトラック構造からなる．この2つの構造により，木構造で表すことのできる半順序関係にある活動を記述することができる．

a．従属直積バックトラック構造

従属直積関係にあるソフトウェア開発活動 $A_1, A_2, \cdots, A_n$ を順次に実行する時，$A_1 \sim A_n$ の活動がすべて成功すれば，SP の制御は後に続く活動に進む．もしある活動 A_i $(1 \leq i \leq n)$ が失敗に終われば，A_{i-1} にバックトラックし再試行する．なお，A_0 は $A_1 \sim A_n$ 全体の前件処理を表すものとする．これを従属直積バックトラック構造[14)~17)]と呼び，$[A_1 * A_2 * \cdots * A_n]$ と表すが，そのバックトラックの進み方は**図2**のように行われるものとする．ただし，"[" と "]" は構造の範囲を示すと同時に，$A_1 \sim A_n$ 全体の前件処理と後件処理も含んでいるものとする．全体の前件処理も失敗に終われば，この構造を含む外側の構造に制御を委ねる．例えば，ソフトウェア開発過程は次のように表すことができる．

ソフトウェア開発＝[ソフトウェア機能設計＊
ソフトウェア構造設計＊プログラミング＊
プログラムテスト]

大木らが提案している "繰返し活動構造"[12)]と名づけられた SP は，集合活動の中の要素活動 $A_1, A_2, \cdots, A_n$ が $A_1 * A_2 * \cdots * A_n$ の従属関係をもち，ある活動の試行が失敗すれば，起点活動 A_1 に戻って再試行するような構造である．これを，$A_2 \sim A_{n-1}$ の活動要素がバックトラックを受け付けない，従属直積バックトラック構造の特例と考えることができる．**図3**に，この構造におけるバックトラックを示す．

b．独立直積バックトラック構造

独立直積関係にある要素活動のうち，もしある活動 A_i $(1 \leq i \leq n)$ の試行が失敗すれば，$A_1 \sim A_n$ の実行を直ちに中断し，制御を全体の前件処理に委ねる．$A_1 \sim A_n$ の全部の活動が成功すれば，制御を後に続く活動に渡す．このような構造を独立直積バックトラック構造[14)~17)]と呼び，$[A_1 \| A_2 \| \cdots \| A_n]$ のように表す．この場合のバックトラックを図示すると，**図4**のようになる．

大木らが提案している "多角度からの検討"[12)]と名づけられた SP は，1つのオブジェクトを様々な観点から吟味し検討する活動である．独立直積バックトラック構造はこのような SP に対応した構造を与えている．例えば，システムレビューは次のように表すことができる．

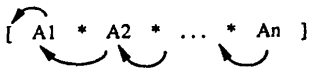

図2 従属直積バックトラック構造
Fig. 2 Dependent product backtrack structure.

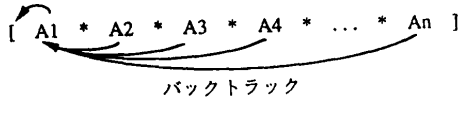

図3 "繰り返し活動構造" におけるバックトラック
Fig. 3 Backtracking in "repeated activity structure."

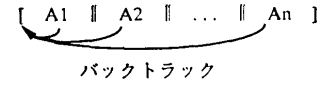

図4 独立直積バックトラック構造
Fig. 4 Independent product backtrack structure.

表1 基本的な活動関係
Table 1 Fundamental activity relations.

関　　係	成功の条件
従属直積関係 $A=A_1*A_2*\cdots*A_n$	$A_1 \wedge A_2 \wedge \cdots \wedge A_n$
独立直積関係 $A=A_1 \| A_2 \| \cdots \| A_n$	$A_1 \wedge A_2 \wedge \cdots \wedge A_n$
直和関係 $A=A_1+A_2+\cdots+A_n$	$A_1 \vee A_2 \vee \cdots \vee A_n$

システムレビュー＝
［システム機能ビューからのレビュー‖
システム性能ビューからのレビュー］

ここで，1つのビューからの検討が失敗すれば，制御は全体を見る前件処理にバックトラックするが，すべてのビューからの検討が成功すれば，SP の制御を後に続く活動に渡す．

（2） 直和バックトラック構造

直和関係にある幾つかの選択可能な SP 活動の中で，少なくとも1つの活動が成功すれば，この集合活動全体の成功と考えるような構造を直和構造[14)-17)]と呼び，$[A_1+A_2+\cdots+A_n]$ のように表す．ここで，少なくとも1つの要素活動が成功しても全体の成功と見なすが，すべての要素活動が失敗したときのみ，全体の失敗と見なしてバックトラックする．この場合のバックトラックを図示すると**図5**のようになる．

例えば，モジュール分割の試行を直和バックトラック構造で表すと次のようになる．

モジュール分割＝［モジュール分割(案1)
＋モジュール分割(案2)
＋モジュール分割(案3)］

2.2.3 SP の階層構造

1つのバックトラック構造を1つの SP として，他のバックトラック構造の要素活動に埋めこむことにより，SP の階層構造を構築することができる．すなわち，トップダウンに考えると，バックトラック構造中の要素活動を再びバックトラック構造に展開することもできる．バックトラック構造の範囲を指定する“［”と“］”は，そこで全体の前件処理および後件処理が行われることも示しているが，そのような処理の明示が必要でないとき，バックトラック構造の間に次の優先順位，

$* > \| > +$

を設定することにより，“［”と“］”の対を省略することができる．たとえば，

$[A_1+[A_2\|[A_3*A_4]]] \longrightarrow [A_1+A_2\|A_3*A_4]$

と略記することができる．

図6 に SP の階層的な構成図の一例を示す．同図で，Aの活動本体は A_1，A_2，A_3 に展開しているが，Aの前件処理と後件処理は A_1，A_2，A_3 全体の前件処理と後件処理とする．同じように，A_2 の活動本体は

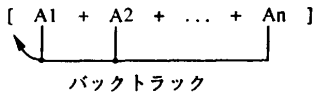

図5 直和バックトラック構造
Fig. 5 Union backtrack structure.

A_{21}, A_{22}, A_{23} に展開しており，A_2 の前件処理と後件処理は A_{21}，A_{22}，A_{23} 全体の前件処理と後件処理とする．バックトラック活動は階層的に行われるので，もし活動 A_3 の実行が失敗すれば A_2 にバックトラックするが，A_2 の前件処理を行って，A_2 の再試行に必要な情報を回復し，A_2 の再実行，すなわち，A_{21} 以下の活動が再試行される．そこで，もし A_{23} の実行が失敗すれば A_{22} にバックトラックし，A_{22} の実行が失敗すれば A_{21} にバックトラックする．さらに，A_{21} の実行が失敗すれば A_2 の前件処理に戻る．そこで，A_2 の前件処理で再試行の可能性が検討され，再試行が可能であれば再び A_{21} 以下の試行に制御が移るが，再試行が不可能であれば A_1 へバックトラックする．

2.3 SP の作成

本節では，前節で提案した SP 制御構造を基にして SP を作成する手順について述べる．

手順1：対象とする SP をいくつかの要素活動に分割する．

手順2：活動間の機能を分析して基本的な関係を設定し，SP の制御構造を決定する．

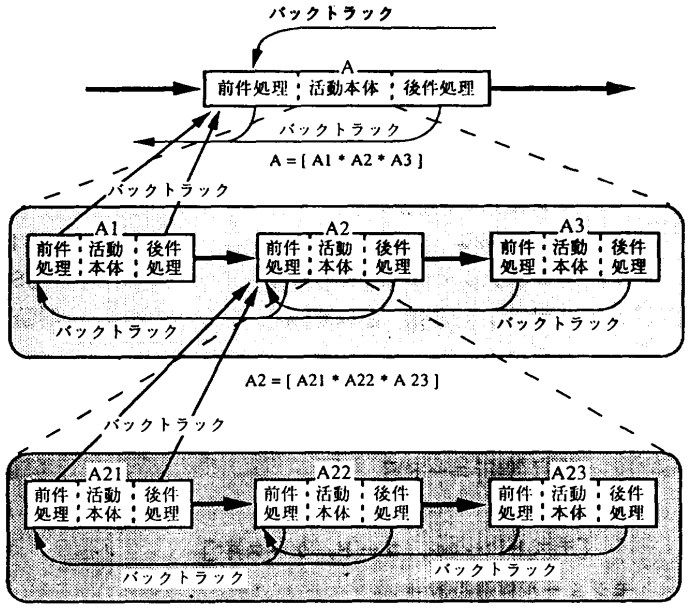

図6 SP の階層構成例
Fig. 6 An example of hierarchical construction of software process.

手順3：SPの制御構造とデータ構造に基づき，前件処理，活動本体および後件処理を設定する．

各活動の実行本体が実行可能なSPに詳細化されるまで，手順1から手順3までを再帰的に適用する．SPの制御構造で本作成手順自身を記述すると，SPの作成活動は次のようになる．

SP作成活動＝
[[手順1＊手順2]＊手順3]

次に，プログラマ活動を例にとってSP作成手順を説明する．

手順1：プログラマ活動をモジュール設計とプログラミングとテストに分割する．

手順2：プログラマの活動機能の意味を考え，各活動の間の関係を設定する．上で3つに分割した活動間には従属直積関係があることから，従属直積バックトラック構造を適用する．

手順3：各プログラマ活動の前件処理と後件処理と活動本体を設定する．

上記の3つの活動をさらに展開するため，各活動ごとに手順1に戻ってSPの記述の詳細化を行う．最終的なプログラマ活動は次のようになる．

プログラマ活動＝[モジュール設計＊
プログラミング＊テスト]
モジュール設計＝
[モジュール機能詳細化(構造仕様)＊
モジュール構造設計]
プログラミング＝
[コーディング(ドキュメント)＊
エディット(仕様，コード)＊
コンパイル(ソースコード，
オブジェクトコード)＊
デバッグ(コード，エラー表)＊
実行(実行コード)]
テスト＝
[テスト(レベル，コード，ケース)$\|^n$]
モジュール構造設計＝
[モジュール構造仕様設計＊
ドキュメンテーション(仕様)＊
レビュー(ドキュメント，ファイル)]
モジュール構造仕様設計＝
[モジュール構造仕様設定
(モジュール・ファイル)＊
インタフェース設計(仕様)]

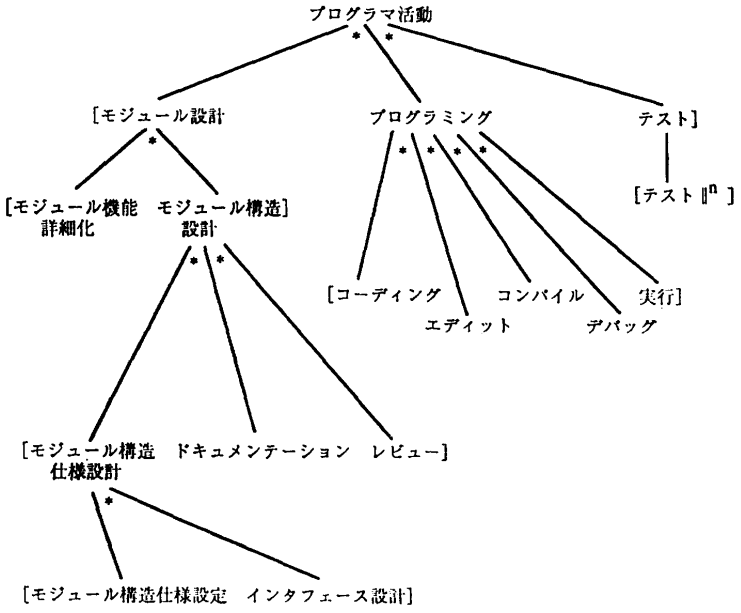

図7 プログラマ活動作成例
Fig. 7 An example of the construction of programmer activity.

ここで，活動の名前の後に続くカッコ内は，この活動に対する入出力パラメータを表すものとする．図7は作成したプログラマ活動を木構造で表したものである．

3. 記述例

本章ではSPの制御構造に基づいて各種のソフトウェア開発活動を記述する．

(1) JSDにおけるハイレベル活動

JSD (Jackson System Development) のハイレベル活動[9),15)]には以下のような活動がある．

ent＊act : entity and action step
struct : entity structure step
model : initial model step
inter : interactive function step
info : information function step
time : system timing step
impl : implementation step

これらの活動をもとに，JSDに基づくSPを記述すると次のようになる．

JSD=[[[ent*act]*struct*model]*
[inter‖info]*time*impl]

ここで，initial model step まで多重の従属直積バックトラック構造となる．この範囲内でいずれかの活動が失敗しても，制御は entity and action step に戻る．このような活動を繰り返して initial model を確定する．

（2） システム設計者活動

システム設計者活動[14]を次のように記述する．

システム設計者活動=
[設計活動*プログラミング管理活動]
設計活動=
[設計関連表の作成活動*
方法設定(関連情報，レビュー結果)*
概要設計(ユーザ仕様，概要仕様)]
設計関連表の作成活動=
[表作成(ユーザ要求仕様，イベント)*
関連分析(イベント，関連情報)*
レビュー(関連情報，分析結果)]
プログラミング管理活動=
[仕事配分 ($P_1, P_2, \cdots, P_n$)*
[プログラミング (P_1)‖
プログラミング (P_2)‖…‖
プログラミング (P_n)]]

システム設計者活動を設計活動とプログラミング管理活動からなる従属直積バックトラック構造として記述し，設計活動を設計関連表の作成活動，方法設定および概要設計からなる従属直積バックトラック構造として記述している．そして，プログラミング管理活動では，1つのプログラミング活動でも失敗すれば，仕事配分に戻して再試行するものとしており，これを独立直積バックトラック構造で表している．

（3） システムアナリスト活動

システムアナリスト活動[14]を次のように記述する．

システムアナリスト活動=
[交渉活動*工程管理活動*工程報告]
交渉活動=
[[ユーザとの交渉(ユーザ要求仕様)*
システム設計者との交渉(ユーザ要求仕様，概要仕様)]*
レビュー(ユーザ要求仕様，概要仕様)*
部品決定(概要仕様，部品仕様)]
工程管理活動=
[工程計画(部品，配分条件)*
進捗管理(スケジュール，状態，データベース)]

システムアナリスト活動は交渉活動，工程管理活動および工程報告からなる従属直積バックトラック構造として表している．そして，交渉活動は再び従属直積バックトラック構造で記述している．すなわち，レビューあるいは部品決定が失敗すれば，ユーザやシステム設計者との交渉活動に戻ることを表している．

（4） ソフトウェア品質機能展開活動

3つのソフトウェア品質展開活動[18),19)]の制御構造を記述すると，それぞれ次のようになる．

ソフトウェア品質機能展開活動=
[[適用システムの決定*
[適用市場の情報把握*
対象ユーザの情報把握]*
要求品質展開表の作成*
要求品質重要度の設定]*
[代用特性展開表の作成*
代用特性項目の定義表の作成*
代用特性の他社との比較分析*
代用特性の重要度の決定]*
[品質表作成*品質表の改良*
クレーム分析]*
[ソフトウェア製品の品質の設定と評価*
サブシステム展開表の作成と評価]*
[開発計画の審査*
[ソフトウェア開発工程展開表の作成*
工程展開・代用特性展開対応表の作成]*
[[管理項目・点検項目の展開*
品質標準・作業標準・検査項目の設定]*
管理項目・点検項目の審査]*
品質管理工程表の作成]]
要求品質展開活動=
[[システム化すべき対象業務と対象となるユーザ市場の把握*
原始情報の収集]*
原始情報から言語情報への変換*
言語情報のグルーピング(種類)*
[要求品質展開表の作成*
要求品質の重要度の設定(重み)]]
原始情報の収集活動=
[事前準備*

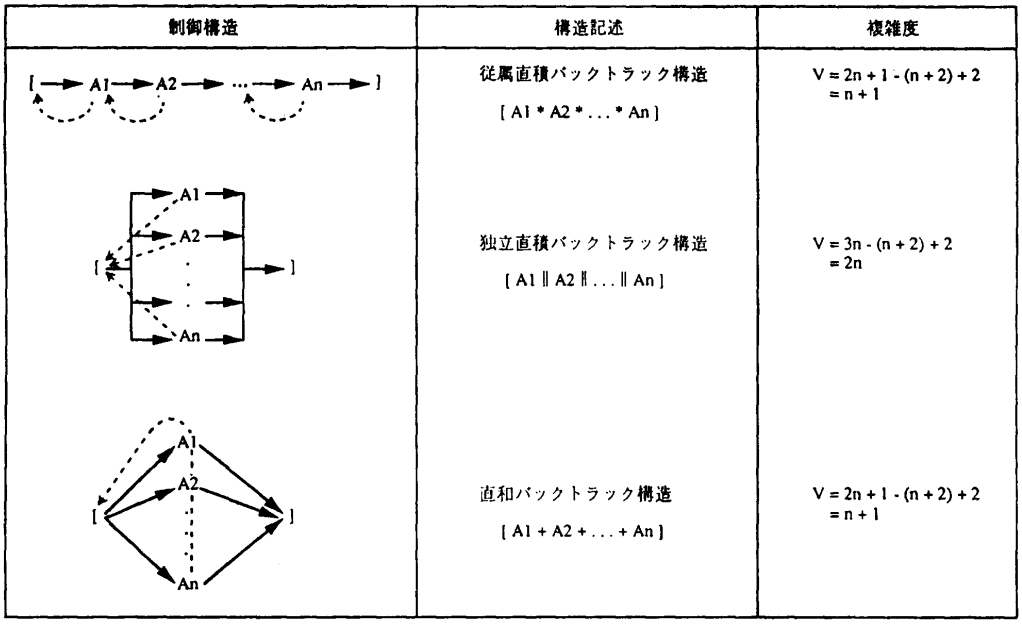

図 8 SP 制御構造の複雑度の算出
Fig. 8 Calculation of complexity of software process control structure.

[トータル指向の対話＋
問題発掘型の対話＋
革新型の対話]]

4. SP の複雑度

本章では，記述された SP に McCabe 指標[13),19)]を適用し，SP の制御構造の複雑度を評価する．さらに，SP の制御構造の改良に関する方策について述べる．

4.1 SP の制御構造の複雑度の評価

McCabe はプログラムの制御構造の複雑さを，プログラムを有向グラフで表したときの基本パス数で与えた．そこで，SP の制御構造の複雑度に対しても，McCabe 指標を適用する．そのため，SP の活動をノード，成功時の活動間の制御の流れと失敗時のバックトラックの流れをアークとし，SP を有向グラフで表したとき，その基本パス数で複雑度と定義することにする．ここで基本パス数とは，SP の有向グラフ G に含まれる独立な閉路数 $V(G)$ であり，グラフ理論により次式で求めることができる．

$$V(G)=e-n+2p$$

ここで，e は SP グラフの辺の数，n は節点の数，p は分離独立したグラフの個数である．入り口が1つ，出口が1つで，すべての節点に対して入り口から到達でき，またすべての節点から出口に到達できるような SP に対しては $p=1$ となる．図8は基本制御構造に対する複雑度の算出例である．

次に，入れ子構造を含む SP の複雑度は次式で評価する．

$$V(G)=V_0(G)+\sum_i(V(g_i)-1)$$

ここで，$V_0(G)$ は G の各入れ子 i を単一の活動とみなしたときの複雑度で，$V(g_i)$ は入れ子 i ごとの局所的なグラフ g_i に対する複雑度であり，入れ子が多重になっているときには，上式を再帰的に用いて評価する．$V(g_i)$ から1を引いているのは，$V_0(G)$ で1つのパスは折り込みずみだからである．階層的な入れ子構造になっているため，複雑度は挿入された入れ子構造の複雑度の単純な総和となっている．例えば，SP＝[[A*B]+[C*D]] の複雑度 $V(G)$ は次のように評価する．

$V_0(G)=2+1=3$,
$V(g_1)=2+1-1=2$,
$V(g_2)=2+1-1=2$,
$V(G)=3+2+2=7$.

次に，プロセス記述の例をあげ，複雑度にどのように反映されるかを考察してみる．いま，ソフトウェア要求分析・定義を RA，システム設計を SD，詳細設計を DD とすると，ウォータフォールモデルによるプロセスは

[RA*SD*DD]

と記述でき，その複雑度は4となる．一方，要求分析定義にプロトタイピングを適用したプロセスは，PDをプロトタイピング設計，PTをプロトタイピングテストとすると，

[[RA*[PD*PT]]*SD*DD]

と記述でき，その複雑度は8となる．それゆえ，後者の複雑度は前者の2倍になるが，それは必ずしもプロセスの実行に要する作業量が2倍になることを意味しない．むしろ，後者は前者よりも基本パス数が4つ多い分だけ，より複雑なシステムであるという目安を与えている．

4.2 SP 制御構造の改良

本節では，SP の制御構造の複雑度に基づいて，SP の制御構造を改良するルールと方法について述べる．

(1) SP 構造の簡略化ルール

SP 構造の簡略化はその改良の重要な手段の1つである．ここで簡略化とは，意味的に等価な SP 構造へ変換する簡略化ルールに基づき，SP の複雑度を低減させることである．表2に簡略化のルールをまとめて示す．

(2) SP の特殊化による改良

一般的に設定された SP から特定のソフトウェア開発過程に特殊化することにより，SP を改良することができる．3章(2)では，一般的なシステム設計者活動に関する SP 記述が与えられているが，例えば事務処理ソフトウェアのように，開発方法が設定されているような場合には，方法設定活動や概要設計活動を設計活動から外すことができ，次のように特殊化することができる．

システム設計者活動＝
[設計関連表の作成活動*方法設定*
概要設計*プログラミング管理活動]
設計関連表の作成活動＝
[表作成(ユーザ要求仕様，イベント)*
関連分析(イベント，関連情報)*
レビュー(関連情報，分析結果)]
プログラミング管理活動＝
[仕事配分 ($P_1, P_2, \cdots, P_n$)*
[プログラミング (P_1)‖
プログラミング (P_2)‖…‖
プログラミング (P_n)]]

ここで，一般的なシステム設計者活動の複雑度は $10+2n$ であり，特殊化したシステム設計者活動の複雑度は $9+2n$ である．

表 2 SP 構造の簡略化
Table 2 Simplification of software process structure.

簡略化	複雑度	
	改良前	改良後
[A*[B*C]]———→[A*B*C]	5	4
[[A*B]+[A*C]]→[A*[B+C]]	7	5
[[A*B]‖[A*C]]→[A*[B‖C]]	8	6
[[A+B]+C]———→[A+B+C]	5	4
[A+[B+C]]———→[A+B+C]	5	4
[[A+B]*[A+C]]→[A+[B*C]]	7	5
[[A+B]‖[A+C]]→[A+[B‖C]]	8	6
[[A‖B]‖C]———→[A‖B‖C]	7	6
[A‖[B‖C]]———→[A‖B‖C]	7	6
[[A‖B]*[A‖C]]→[A‖[B*C]]	9	6
[[A‖B]+[A‖C]]→[A‖[B+C]]	9	6

5. おわりに

本論文では，SP を構造的に記述するために基本となる制御構造をバックトラックに基づいて明らかにした．この基本構造を再帰的に適用することにより，SP の階層的記述を行うことができる．そのため，バックトラック先は構造上一義的に決定される．しかし，ある活動が失敗に終わったとき，実際には失敗の状況に応じて最適なバックトラック先があるはずであるが，本論文の基本構造ではそのような記述ができない．これはバックトラックを構造化したための制約であって，入れ子になっているバックトラック構造の内側から適当なバックトラック先を辿っていかなければならない．

SP の制御構造についての研究はソフトウェア開発過程に対する基本的な研究であるが，今後研究されなければならない課題を以下に挙げる．

- SP のための記述言語，
- SP のデータ構造，
- SP の評価とそのツールの開発，
- SP の部品化，再利用，
- バックトラック構造と並列構造との関係など．

謝辞 本研究は日本学術振興会の論博事業に基づいて著者の一人が日本に滞在中に行われたものであり，その機会を与えてくれた武漢大学の学長を始め，同大学ソフトウェア工学国家重点研究実験室の諸氏，および貴重な討論をいただいた北海道大学工学部情報工学科言語情報工学講座の赤間清，渡辺慎哉，三谷和史の3氏に感謝申し上げます．

参考文献

1) Royce, W. W.: Managing the Development of Large Software Systems: Concepts and Techniques, *Proc. of IEEE WESCON*, Vol. 14, Session A/1, pp. 1-9 (Aug. 1970).
2) Basili, V. R. and Turner, A. J.: Iterative Enhancement: A Practical Technique for Software Development, *IEEE Trans. Softw. Eng.*, Vol. 1, No. 4, pp. 390-396 (1975).
3) Agresti, W. W.: What Are the New Paradigms?, *New Paradigms for Software Development*, pp. 6-10, IEEE Computer Society Press, Washington (1986).
4) Boehm, B. W.: A Spiral Model of Software Development and Enhancement, *ACM Soft. Eng. Notes*, Vol. 11, No. 4, pp. 14-24 (1986).
5) Humphrey, W. S. and Kellner, M. I.: Software Process Modeling: Principles of Entity Process Models, *Proc. of the 11th-ICSE*, pp. 331-342 (1989).
6) Potts, C.: A Generic Model for Representing Design Methods, *Proc. of the 11th-ICSE*, pp. 217-226 (1989).
7) Osterweil, L.: Software Process Are Software Too, *Proc. of the 9th-ICSE*, pp. 2-13 (1987).
8) Sutton, S. M., Jr., Heimbigner, D. and Osterweil, L. J.: Managing Change in Software Development through Process Programming, Technical Report CU-CS-531-91, University of Colorado (Jun. 1991).
9) Williams, L. G.: Software Process Modeling: A Behavioral Approach, *Proc. of the 10th-ICSE*, pp. 174-186 (1988).
10) Katayama, T.: A Hierarchical and Functional Software Process Description and Its Enaction, *Proc. of the 11th-ICSE*, pp. 343-352 (1989).
11) Inoue, K., Ogihara, T., Kikuno, T. and Torii, K.: A Formal Adaptation Method for Process Descriptions, *Proc. of the 11th-ICSE*, pp. 145-152 (1989).
12) 大木敦雄，落水浩一郎：ソフトウェア設計に関する作業プロセスの Prolog ルールによる記述について，情報処理学会ソフトウェア工学研究会資料，58-2, pp. 9-16 (1988).
13) McCabe, T. J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol. 2, No. 4, pp. 308-320 (1976).
14) 何　克清：ソフトウェア開発に於ける知識表示と利用の研究，中国自然科学進展，Vol. 1, No. 4, pp. 328-336 (1991).
15) 何　克清：ソフトウェア開発に於ける知識表示と利用モデル SOKM，中国計算機科学，Vol. 89, No. 6, pp. 27-39 (1989).
16) 何　克清：SOKM/L の JSD 知識表示に於ける利用，中国計算機科学，Vol. 90, No. 1, pp. 25-32 (1990).
17) 落水浩一郎：ソフトウェア設計プロセスの記述と実行に関する一考察，情報処理学会ソフトウェア工学研究会資料，62-5, pp. 1-8 (1988).
18) 菅野文友：ソフトウェア品質管理，日科技連 (1987).
19) 吉沢　正：品質機能展開による高品質ソフトウェアの開発方法，コンピュータエージ社 (1989).

(平成4年2月10日受付)
(平成4年9月10日採録)

何　克清

1947年生．1970年武漢大学数学科卒業．同年より武漢大学計算機科学科でハードウェアの研究・開発，特に計算機 GNB-1, 2, 3 の開発に従事．1975年よりソフトウェアの研究，特に OS の解析・評価に従事．1980年より岩手大学大学院工学研究科情報工学専攻および日立製作所ソフトウェア工場で研修．1982年武漢大学計算機科学科ソフトウェア工学研究室主任・講師．現在，同大学の中国国家ソフトウェア工学重点研究実験室主任・教授．オブジェクト指向に基づくソフトウェア開発プロセス・環境などの研究に従事．IEEE 会員，武漢ソフトウェア工学学会副理事長．

宮本　衛市（正会員）

1940年生．1962年北海道大学工学部電気工学科卒業．1964年同大学院修士課程修了．同年北海道大学工学部電気工学科講師，同助教授を経て，1984年より情報工学科教授．工学博士．分散システム，並列オブジェクト指向モデル，プログラミング環境などの研究に従事．著書に「PASCAL—プログラミングと翻訳技法」など．電子情報通信学会，ソフトウェア科学会，人工知能学会，IEEE 各会員．