Mem. Fac. Sci. Eng. Shimane Univ.
Series A
37, pp. 65-73（2003）

プログラム構造と仕様追従性

伊藤貴子 1)，門田洋太郎 1)，佐藤匡正 2)

1)島根大学大学院総合理工学研究科　数理･情報システム学専攻
2)島根大学総合理工学部　数理･情報システム学科

Program Structure and Traceability Analysis

Takako ITOU, Yotaro KADOTA and Tadamasa SATOU
1)*Department of Electric and Control System Engineering,*
Interdisciplinary Faculty of Science and Engineering, Graduate School of Shimane University
2)*Department of Electric and Control System Engineering,*
Interdisciplinary Faculty of Science and Engineering, Shimane University

Abstract

It is important to grasp program design method characteristics such as similarity and dissimilarity among various ones, focused on a viewpoint of traceability which is one of the software design quality, giving the technological judgment for the choice among them in software development. It attempts to analyze program design method characteristics focused on traceability by an approach that it includes program implementations and analysis of them. In implementation, it is required to select a proper problem considered to keep the results to be general and correct. In analysis it is applied the program structure formalization method that enables to identify the differences among isomer programs in structures that are implemented by applying target program design methods to the same specification. It concludes that the structure can be an analysis measure of traceability, as differences of program structures are due to the differences in the program design methods' paradigms.

1. 序　論

ソフトウェアの一つの品質尺度の一つとして仕様追従性がある．追従性とは，仕様とプログラムの対応関係を保つことで，これが良好であれば保守や改版に効果をもたらす設計と言える．一般的に事物基軸型設計方法(Object-Oriented Design Method，以後 OOM)は，仕様追従性が高いとされている．そこで，仕様追従性について，事物基軸型をプログラム設計方法の手順型(Procedure-Oriented Design Method，以後 POM)と対応付けて，方法としての同等性や差異が把握できれば望ましい．ところが，プログラム設計方法は，それぞれ考えの枠組み(paradigm)が異なるため，その評価は，定性的に長所，短所項目の列挙によって行われている．これは，実践的な観点からみると心もとない．設計方法の特性が仕様追従性の観点から直接比較できることが望まれる．

この分析方法の一つとして，同一例題を用いる手法がある．一つの問題について，様々な設計方法を適用してプログラムを作成し，これらを仕様追従性に着目し分析することによって，設計方法の特徴を把握する考えである．一つの問題に対して複数の設計方法を適用すると，一般にその設計方法の特徴を反映した複数のプログラム，つまりプログラム変異体が得られる．これらのプログラム間における類似性や相違点を仕様に基づいて，比較，分析することによってこの相違を生み出している設計方法の特性を把握する．本分析法を「プログラム変異体分析法」と呼ぶことにする．この場合，少数例から得られる結果を一般化してしまう惧れがある．プログラムの差異の生ずる要因は設計方法以外にも，適用問題領域の性質，設計者の習熟度や観点などの個人差，あるいは処理形態の差などが考えられる．これらの要因を押さえるために，客観性をもつ例題の選定が大切となる．

Bergland は設計方法の特性を本手法によって解説した[4][5]．またこのプログラムに対して，その構成上の違いを正規表現で形式化して示すプログラム構造形式化手法が提案され[1][2]，これに基づいての差異の分析がなされた[3][7]．

そこで本稿では，プログラム構造形式化手法を適用して，従来からの手順型に基軸を置く設計方法と基軸を事物に変化させた方法の二つのプログラム設計方法の特性を仕様追従性に着目し分析を行う．分析にあたっては比較基盤の共通化が肝要である．とくに，設計基軸の違いのままでは比較できない．この共通基盤として，ここでは処理手順を採用する．

2. 分　析　方　法

2.1 分　析　手　順

POM を適用して実装したプログラムの構造はいずれの方法においても直接処理手順に基づいて構築されているので，その差異は，手順を基準として比較できる．この結果処理手順に基づいて POM と OOM 適用のプログラムを同一基盤で比較分析できることになる．一方，OOM の実装は，POM 言語を適用するので，類(class)，事物(object)，および手段(method)などの枠を取り除いて手順として展開して分析する．本分析は，図 1 に示す 6 段階によって行う．

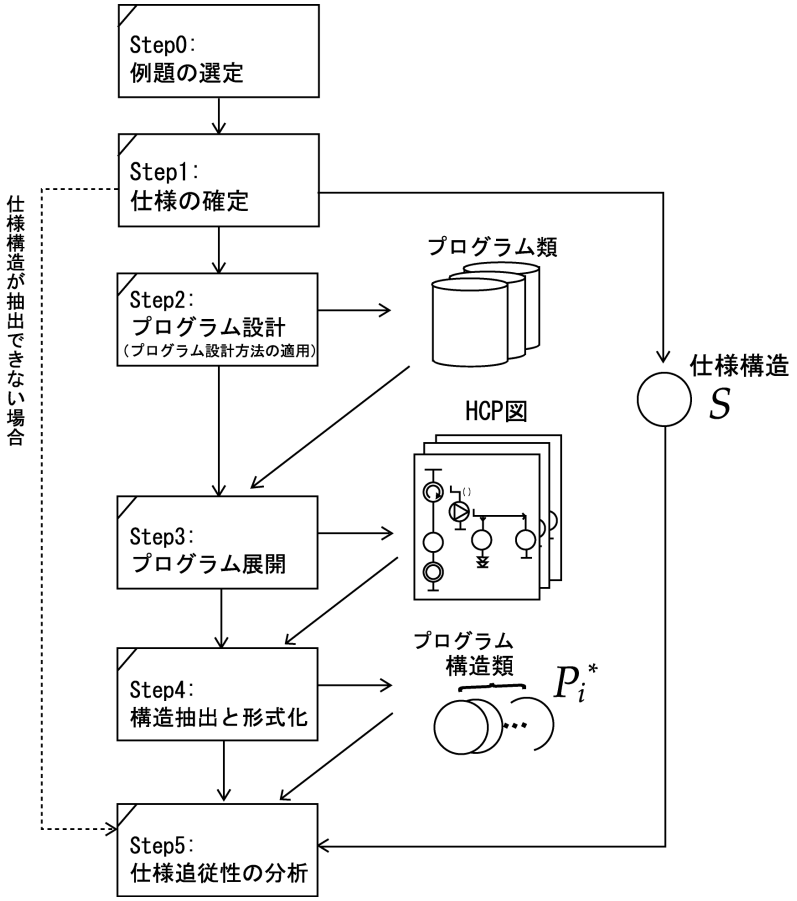

図 1　分析手順

Step0:例題の選定

例題は，異なる設計方法を適用することにより，プログラム変異体分析の道具立てとして共通とする基軸を選択する．

Step1：仕様の確定

例題を仕様化する．プログラム変異体との対応が分析できるように定義する．可能であれば何らかの形式化を行う．

Step2：プログラム設計

上の仕様に基づいて，それぞれの設計方法によってプログラムの設計し，実装する．

Step3：プログラム展開

同一の共通基準で比較するために，全ての関数をインライン展開した HCP 図(JIS X0128(ISO 8631))を作成する．これによって関数化の範囲や引数の授受方法の違いや冗長処理を階層的に吸収できる．

Step4：構造抽出と形式化

HCP 図においてプログラム構造を抽出する．構造を後述する形式化手法によって構造を正規表現で表わす．

Step2～Step4 を設計方法を変えて実施する．

Step5：仕様追従性の分析

正規表現化されたプログラム変異体の構造について，仕様項目の構成法に対する差異すなわち追従性を分析し，その結果から設計方法の特性を把握する．

2.2 例　題

本手法では,例題を分析での共通する基準とするため，表 1 に示す要件を考え，次の 2 題を選定する．

例 1：気象測候所システム

例 2：倉庫管理システム

表 1　要因と排除するための措置

影響要因	例題採用要件
恣意的な例か	公刊書の例題
設計方法を代表しているか	教科書的な例題
個人差によらぬか	設計済みの例題
処理形態の違いの影響	リアルタイム/バッチの両例題

2.2.1　気象測候所システム

本例題は OOM の解説用として掲載されている.リアルタイム処理形態の事物基軸型設計例である[6]．この仕様のあらましを次に示す.

このシステムは，自動的に集められた気象データを用いて天気図を作成するシステムの一部である．気象測候所は，データを地区計算機にその要求に応じて送信する．CommsController は，地区計算機を識別し，伝言を解釈して気象報告を送るように要請を受信し，WeatherStation に報告する．WeatherStation は，WeatherData にデータ源から集められたデータの要約を要求する．システム構成を下図で示す.

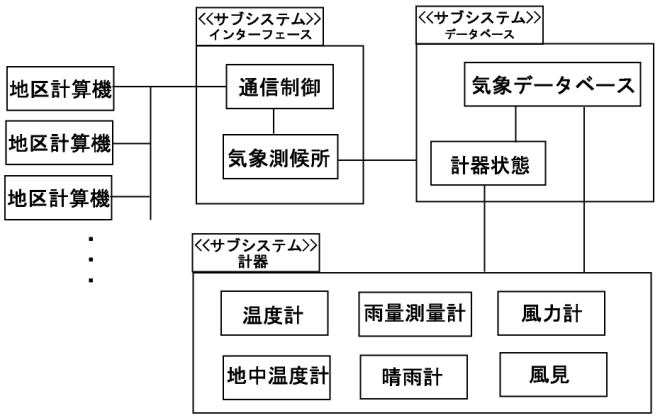

図 2　WeatherStation のシステム構成

2.2.2　倉庫管理システム

本例題は，POM の 4 種類の設計方法に共通するもので解説における比較基準として用いられている.バッチ式処理形態の設計例[4]である．処理構成上の特徴は,グループ分けにおける「先読み」である.この仕様のあらましを次に示し，システム構成を下図で示す.

[仕様のあらまし]

M 社の冷凍商品倉庫における商品ごとの在庫数の報告書を作成する.入力ファイルは商品コードで整列されたレコード群から成っていて，1 レコードは，商品ごとに入出庫が生じたときに記録される．このとき日別商品ごとの移動量の報告を作成する.

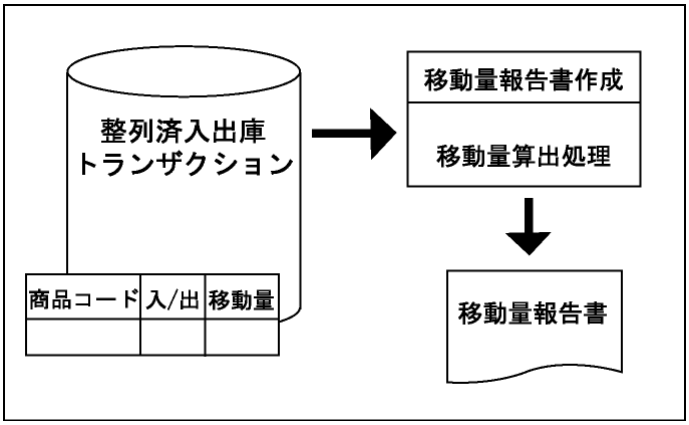

図 3　Bergland の例題のシステム構成図

2.3 形式化分析法

2.3.1　構造の意義

プログラムの記述には，外部仕様として規定される処理に相当する非決定性を与える枠組み部分と，この枠組みの条件成分に対して一意に条件付けて実行の流れに決定性を与える部分が混在している．この前者と後者を「構造」と「機構」として分ける[2]．この考えでは，構造は，選択や繰返しにおける命題で与えられる条件付けを無視することによって得られる．同時にこれらの命題に決定性を与える構成成分（要素処理）を取除いたものである.

2.3.2　構造の正規表現方法

構造は，いわゆる整制御構造(well structured)であれば正規表現として表現される．要素処理を正規表現の変数で表し，連接を・(積)，選択を+(和)，繰返しをクリーネの閉包+,* によってそれぞれ表現することができる．実際のプログラミングでは関数での戻りや繰返しでの打切りなどの強制制御が便宜的に使われている．これに配慮して強制制御を良構造に機械的に変換できるようにするために,展開の仕方を変換公式として確立させる[10]．この考えは，2 段階の扱いをする必要がある．まず，構造を特定の記号（#:戻り，」:打ち切り）を含む正規表現で表す．次に，これらの強制的な制御用の記号に対して，変換公式（付録参照）を適用して強制制御を含まない正規表現を得る.

3. 分　析

3.1 気象測候所の例題

3.1.1　仕様の形式化

プログラムは状態の変化と捉えることができるので，仕様のあらましから入力状態の軌跡を考える．本システムの入力状態は，報告書の要請に収束するから，報告書の要請｛RequestID｝の入力集合と定め，このシステムはアルファベットΣ=｛RequestID｝とする順序機械と捉えることができる．この状態機械を図４に示す．Σは，図に示す記号A,B,C,D,Eの５要素からなる．入力集合｛RequestID｝は，Σ上の言語となり，状態機械によって受理される．この組合せは，限定されているので状態遷移図として表わすことができる．

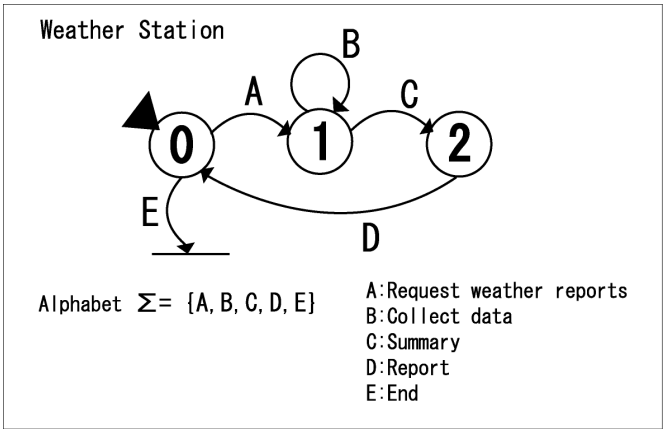

図４　WeatherStationの仕様

この状態遷移図から，プログラムの外部仕様は形式化される．またΣの各要素に対して正規表現することができ，次式が得られる．

$$Sw = (AB^*CD)^*E \qquad (1)$$

3.1.2　プログラム構造

プログラム *Po*, および *Pp* において，機構を除いて構造を取り出す．両者は一致し，*Pw* が得られる．プログラム構造において e_i は，プログラム記述における要素処理である．
このとき,」は繰返し構造の打ち切りを示す.

$$\begin{aligned} Pw &= (e_7(e_8e_9+e_{10})(\varepsilon \text{」} + \varepsilon)((e_{11}+e_{12}+e_{13}+e_{14}+e_{15}+e_{16})(e_{17}e_{18})^*e_{20}^*e_{21}e_{22}))^{\infty} \\ &= (e_7(e_8e_9+e_{10})((e_{11}+e_{12}+e_{13}+e_{14}+e_{15}+e_{16})(e_{17}e_{18})^*e_{20}^*e_{21}e_{22}))^*e_7(e_8e_9+e_{10}) \end{aligned} \qquad (2)$$

3.1.3　分析と考察

外部仕様から得られた *Sw* とプログラムから得られた *Pw* の対応を以下の表２に示す.

表２　*Sw* と *Pw* の対応表

Sw	*Pw*	処理内容
A	$e_7(e_8e_9+e_{10})$	気象報告の要請
*B**	$(e_{17}e_{18})^*$	データを収集
C	$e_{20}{}^*e_{21}$	データを要約
D	e_{22}	報告

表 2 から対応しているプログラム構造の要素処理を *Sw* の記号で表わすと次式が得られる．

$$Pw = (A((e_{11}+e_{12}+e_{13}+e_{14}+e_{15}+e_{16})\ B\ ^*CD)\ )^*A$$

ここで，要素処理 e_{11}～e_{16} は，プログラムとしての処理を実現するためのものである．これらは仕様の規定外なので空要素処理 ε として考えることができ，次式が得られる．

$$Pw = (AB^*CD)^*A$$

このとき式 *Pw* の最後の *A* は，気象報告の無い場合に要請を終了するように実現されており，*E*：要請終了とみなすことができ，*Sw*＝*Pw* となる．つまり，仕様の構造が直接プログラム構造に反映されており，プログラム構造からみた仕様追従性は良好と言える．

また,気象測候所が受動的で処理は地区計算機に由来しており,OOM と POM を適用したプログラム構造 *Po* と *Pp* は一致する．これは，入力集合{RequestID}によって決定される処理が定形的であり，間合わせが一通りである．*Pw* は，プログラム全体が気象報告要請を入力とする繰り返し構造である．

設計方法に関係なく処理が定形的であるのは，リアルタイム処理形態の特性を反映しているからである．リアルタイムでは，入力集合{RequestID}に応じてそのデータを収集し要約を報告する処理であり，入力側からの処理要求発生に依存している．これはまた，地区計算機からの要請を仮想的な入力列とした連続する処理の連なりで，バッチ処理形態を構成していると言える．つまりプログラム全体は，バッチ処理形態をとり，リアルタイム処理形態はバッチ処理形態に包含されている．また，プログラム設計方法の差異は同一構造として吸収されている．

3.2 倉庫管理システムの例題

3.2.1　仕様の形式化

3.1.1 と同様に,仕様のあらましから，本システムは，ファイル･レコード{FileRecord}と移動量{Change} の 2 つの入力集合と定める．このシステムはアルファベットΣ＝{FileRecord, Change}の順序機械である．Σは，可能な組合せ集合の 3 要素とする．またこの組合せは，限定されているので状態遷移図として表すことができる．この様子を図 5 に示す.

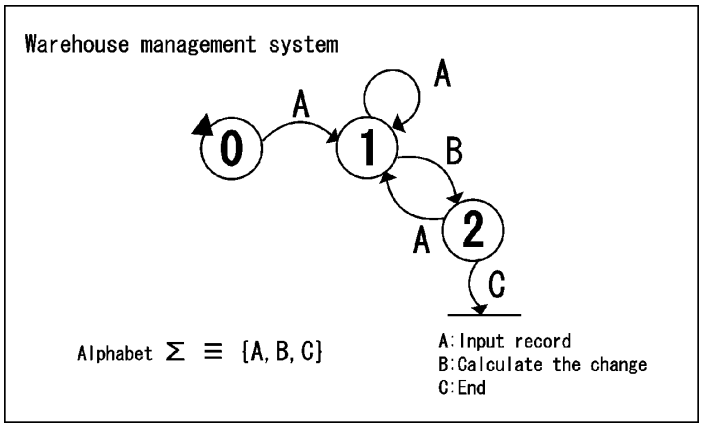

図 5　Bergland 例題の仕様

この状態遷移図から，外部仕様が形式化され，次式が得られる．

$$S_b=A(A^*B)^*C \qquad (3)$$

3.2.2 プログラム構造

Po，および *Ppi* において，機構に配慮して構造を取り出すと次が得られる．e_i は，プログラム記述における要素処理である．

OOM(Object-Oriented Design Method):

$$P_o = e_1((((e_{20}+\varepsilon)e_{21}*(e_{22}+\varepsilon)(e_8+\varepsilon))+e_{10}*e_8)((e_{11}+e_{12})e_{13}(e_{20}+\varepsilon)e_{21}*$$
$$= (e_{22}+\varepsilon)((e_{11}+e_{12})e_{13}\rfloor+\varepsilon))*e_{14}*e_{17}e_{18}e_{19}*e_{16}e_3)*e_4*e_5e_6*e_7 \quad (4)$$

DFDM(Data flow Design Method):

$$Pp_1 = e_{20}\, e_1\, e_5\, e_{21}*((\, e_8\, (\, \varepsilon\, \rfloor+\varepsilon\,)\, (\, e_{13}*\rfloor+\varepsilon\,)\, e_{13}*e_{21}*)*e_{17}\, (e_{18}\, e_{19})*\, e_{16}\, e_6\, e_4\, (\, \varepsilon\, \rfloor+\varepsilon\,))*\, e_7\, e_{22} \quad (5)$$

DSDM(Data Structure Design Method):

$$Pp_2 = e_{20}\, e_1\, e_5\, e_{21}*(\, e_8\, e_{17}\, (e_{19}\, e_{16}\, (\, \varepsilon\, \rfloor+\varepsilon\,)\, e_{21}*)*\, e_4\, e_6)*(\, e_8\, e_{17}\, e_{19}e_{16}+\varepsilon\,)\, e_7\, e_{22} \quad (6)$$

FDDM(Functional Decomposition Design Method):

$$Pp_3 = e_{20}\, e_1\, e_5\, e_{21}*(\, e_{19}\, e_{16}\, e_{21}*+\, e_4\, e_6\, e_8\, e_{17}+\, e_8\, e_{17})*\, e_{19}e_{16}\, e_4\, e_6\, e_7\, e_{22} \quad (7)$$

3.2.3 分析と考察

外部仕様から得られた *Sw* とプログラムから得られた *P* の対応を表 3 に示す．

表 3　*Sb* と *P* の対応表

Sb	*P*	処理内容
A	e_{21}	レコード入力
B	e_{19}	移動量算出
C	e_4	総移動量算出

表 3 から対応するプログラム構造の要素処理をを S_b の記号で表わす．ここで，e_1～e_{17} は，プログラム中の処理を実現するための付随処理である．これらについては，仕様とは対応していないから空要素処理と考えることができ次式が得られる．外部仕様から得られた *Sw* と比較して，仕様追従性に着目し，分類を行うと図 6 が得られる．

OOM: $Po = (A*B*)*C$
DFDM: $Pp_1 = A*(B*C)*$
DSDM: $Pp_2 = A*(B\,A*)*(B\,C+\varepsilon)$
FDDM: $Pp_3 = A*(BA*+C)*B\,C$

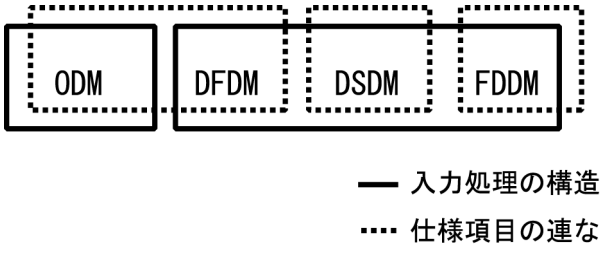

図 6　プログラム構造による設計方法の分類

図 6 の実線の枠組みは，入力処理構造の分類を示す．入力処理の構造による分類は，*A* のレコード入力による．*Po* は，*A* が繰り返し構造にあるのに対して，*Ppi* は *A* が繰り返し構造から独立している．この違いは，「入力を収集する」と見なすか「入力を授受する」と見るかの違いである．この違いは設計の枠組み(paradigm)による．OOM において，入力は事物であり，事物が主体性をもつプログラム構造を構成する．一方，POM は，入力に同期させる構造となる．また，プログラム構造は，規定された処理の並びであるから本来同等であるはずだが，*Ppi* と *Po* に差異がある．この差異は，プログラム設計方法に由来する処理の区切り方に違いがある．なお，構成上からは変換公式を用いて同等となることが示せる．

図 6 の破線の枠組みは，仕様項目の連なりによるプログラム構造の分類を示す．OOM と DFDM は，仕様項目の並びが同等であり，*Sw* に対して仕様追従性が良好であると言える．OOM は手段でのやり取りを行っているが，引数や戻り値などによって，データのやり取りを行う設計であり，バッチ処理形態の特性からやり取りが一方向であることに由来している．

DFDM は，データの流れに着目した設計であり，データの区切りが処理の区切りである．これもバッチ処理形態の特性から一方向にデータが流れることに由来している．データの並びについて，入力の処理カードをグループごとに区切り，「グループごとの在庫数を求め，入出庫を小計する」に着目して考える．DFDM の特徴は「処理カード」,「グループ」の処理の間に処理の区切りがある．これは，単一のデータごとに処理を分割しているためである．一方 OOM の特徴は．DFDM と同様に，大きな処理の区切りが「カード」と「グループ」の処理の間にあることである．これは「入力カードを得る」,「グループを得る」の単一の振るまいごとに手段が存在しているためである．両者とも続いて，グループの終りのカードを鍵として同一の繰り返し構造で，グループごとの移動量算出を行う．

また他の二つの手順型のプログラム構造については，DSDM の最後の小計部分である$(B\,C+\varepsilon)$ は，その前の繰り返し構造が前判定であるため一番最後のグループについて繰り返しの後に小計する構成をとっている．OOM や DFDM と比較して，同一のアルファベットを 2 つもつプログラム構造となり，仕様追従性が良いとは言えない．また FDDM は，繰返し構造の中に選択構造をもつ構造となる．他の 3 つと比較して最も仕様追従性の不良な構造をとる．この背景には，プログラム設計方式の着眼点が異なることに由来している．

4. 結　論

プログラム構成上の違いを正規表現で形式化して示すプログラム構造形式化手法に基づき，異なる設計方法を適用したプログラム構造を仕様追従性に着目し分析した．この分析は，リアルタイムとバッチの２つの処理形態の例題を用いた．リアルタイム処理形態として「気象測候所システム」について，手順型と事物基軸型のプログラム構造を比較した．その特性から構造上からみた OOM と POM の差異はなく，両者とも仕様追従性が高いと言える．またこのプログラム構造は，気象測候所からの電文(気象データ)を仮想的な入力列と見なしたバッチ処理形態を構成していることが分った．

バッチ処理形態の特性比較では，Bergland の「倉庫管理システム」について手順型と事物基軸型のプログラム構造を比較した．OOM と POM では，入力を処理する構成法における違いが観察された．また，DFDM と OOM は同等のプログラム構造をとり，仕様追従性が高いと言える．

プログラム構造の差異は，プログラム設計方法の考えの枠組みの差異に由来するので，仕様追従性の分析尺度となり得ると言える．

付　録

(1) 戻り(RETURN)[11]

戻りの要素処理を記号# で表す．# を含む正規表現f は#lは#を含まない次式に変換できる．

$$\Delta_{\#}\,(f_E\ +\ f_R\ \#\ f_0) = f_E + \ f_R$$

ここで,f_E,f_R は#を含まず，f_0 は#を0 個以上含む．$\Delta_{\#}$ は，# のみに働く変換子である．

(2) 繰返しの打切り(BREAK)[12]

打切りは戻りと同様に考えることができる．打切りを記号」で表す．繰返しの一般式は # を」と考えればよいので，繰返しの型に応じて次の展開公式が得られる．

1）前判定: $\Delta_{」}\ (f_E + f_B{}_{」} f_0)^* = (f_E)^* + (f_E)^* f_B$

2）後判定: $\Delta_{」}\ (f_E + f_B{}_{」} f_0)^+ = (f_E)^+ + (f_E)^* f_B$

3）継続　: $\Delta_{」}\ (f_E + f_B{}_{」} f_0)^\infty = (f_E)^* f_B$

$\Delta_{」}$ は $_{」}$ の変換子である.

文　献

[1] 佐藤匡正:プログラム構造の仕様逆追従性，平成 14 年 5 月電子情報通信学会信技法，SS2002-3 pp.13-17

[2] 佐藤匡正:HCP 図法で記述されたプログラム解法の S 代数による定式化，情報処理論文誌，Vol.27，NO.6(1986)

[3] 佐藤匡正:プログラム構造からみた変異の分類法，　情報処理学会論文誌 Vol.32 NO.2(1991)pp.237-245

[4] Bergland,G.D.: A Guided Tour of Program Design Methodologies, Computer, Vol.14,No.10(Oct.1981)

[5] 佐藤匡正:プログラム処理要素の共用化度合いから見た解法の相違,情報処理学会論文誌,Vol.26,No.4(Jul.1985)

[6] Ian Sommerville: Software Engineering (6th Edition),pp267-280,Addison-Wesley Pub Co, 2000.

[7] 門田，伊藤，松原，佐藤：Bergland 例題における事物基軸設計方法の位置付け，平成１４年度電気･情報関連学会中国支部連合大会講演論文集，pp519

[8] Satou Tadamasa, Kishimoto Yorinori, Md. Atiqul Islam : Program　Structure Formalization and Traceability Analysis with Examples, International Conference on Computer and Information Technology (2002) (to be published)

[9] Satou　Tadamasa:　Universal　Form　for　Program Iteration Structure,ICCIT2000pp.240-243(2001)

[10] 佐藤匡正，岸本頼紀：プログラム構造形式化による追従性分析法，第 9 回ソフトウェア工学の基礎ワークショップ pp.175-178(2002)

[11] 佐藤匡正：強制戻りをもつプログラム構造の形式化，電子情報通信学会総合大会，pp.346-347(2000)

[12] 佐藤匡正：打切りのある繰返しをもつプログラム構造の形式化，電学技法，SS2000-8 pp.17-24(2000)