ソフトウェア工学 77－8
（１９９１．２．７）

検証指向のプログラム設計技法

VOP（Verification Oriented Programming）

松尾谷 徹
日本電気（株）情報処理・システム開発本部
E-mail:matsu@sysd.mt.nec.co.jp

ソフトウェアのテストはライフサイクルを通して考えると、信頼性と生産性を支配する重要な技術である。本稿は現在のテスト技法でテスト可能な範囲内でプログラムを設計するための制約条件を持ったプログラム設計技法：VOP（Verification Oriented Programming）について紹介する．VOPはプログラム設計とテストデータ設計を一体化して行う技法であり、従来のテスト技法では難しい状態遷移や複雑な条件の組み合わせに対して、効率的で完全なテストを可能にするものである。

Verification Oriented Programming:VOP

Tohru MATSUODANI

Technology Development Department
EDP Systems Development Divison
NEC Corporation
1-35,shiba 5-Chome,Minatoku,TOKYO 108,JAPAN.
E-mail:matsu@sysd.mt.nec.co.jp

This paper discusses a programming methodolgy that was named Verification Oriented Programming(VOP):it is posible that generate exhaustive test-datas. This methodolgy incorporated some design constraint that enable to test state-transition, unnecesary access,and astronomical combinations of input circumstances;it make a simle program for testing.

1．　まえがき

ソフトウェア開発の需要は増加の一途をたどり、高信頼性ソフトウェアを短納期で効率的に開発する技術が強く求められている。　実際のソフトウェア開発過程における検証過程の占める役合は大きく、ソフトウェアの信頼性を決定すると共に開発プロセスの費用と期間を支配する大きな要因となっている。この傾向はソフトウェアの規模が大きくなるほど顕著であり、検証過程の割合が50%を超えるケースも稀ではない。　このような問題を解決する方法として、設計上流工程を重視し、ＣＡＳＥなどの自動化を行うアプローチが盛んに行われていが、まだ検証過程が減少するにはいたっていない。

我々はこのような背景のもとでテストを中心とした検証技術の改善を行ってきた。[1]-[6]　これらの研究の結果、テストの効率化を行う一つの手段として、現在のテスト技法で検証出来る範囲に入るように、対象となるプログラムに制約条件を付加し開発することが有効であると言う結論に達した。

本報告ではこの考え方に従って実際のプログラムを設計する方法と、その時同時にテストデータを設計する方法について提案を行う。　我々はこのテスト可能性（容易性）を重視したﾌﾟﾛｸﾞﾗﾐﾝｸﾞ設計技法をＶＯＰ：（Verification Oriented Programming）と呼んでいる。

2．　テスト可能なプログラムの要件

まずテストの定義を行いテスト可能なプログラムの要件を整理する。

2．1テストの定義

プログラムのテストについて定義を行う。

(1)ブラックボックス・テスト

プログラムのテストについて色々な方法が存在するが、ここでは図-1に示すブラックボックス・テスト（機能テスト）により検証を行うものとする。

この時、有限な入力の組み合わせを与え、その出力からプログラムの機能の誤りと漏れを確認出来ることをテスト可能と呼ぶ。

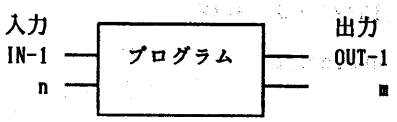


図-1　ブラックボックス・テスト

(2)テストの単位と範囲

テストの単位は単体テストと呼ばれるプログラムの構成要素について行われるものから、複数の構成要素を組み立てた状態でテストする結合テストまでを含むものとする。　ここではテストの範囲にプログラムの機能自身が有効であるかどうかの検証は含めないとする。何らかの定義された仕様（プログラム仕様）をプログラムが正しく満たすかどうかを確認する手段を対象とする。

(3)テストデータとテスト項目数

テストは外部からテストデータを与え結果を確認する。テストデータとは、入力データの組とその入力が与えられたときプログラムが出力すべき結果（正解値）からなる。　テストにより確認すべき出力（正解値）の数をテスト項目数と定義する。

(4)テスト工程の位置づけ

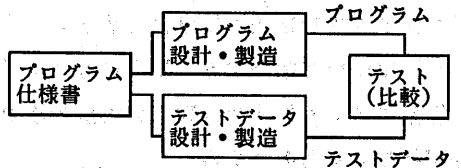


図-2　テスト工程

テストデータはプログラムの仕様から作成され、プログラムの設計・製造とは独立である。テストはこの２つの異なった設計過程の二重化であり、その結果を比較するのものである。プログラム設計が誤ることもまたテスト設計が誤ることも起こり得る。

2．2テストの特性

(1)誤りの種類

与えられた仕様に対しプログラムが起こし得る誤りには次の３つのタイプに分類される。

(a)定義された処理が誤って作られた。（処理誤り）
(b)定義された処理が漏れてしまった。（処理漏れ）
(c)定義されていな処理が混入した。　（誤配線　）

これらの誤りを検出する能力のことをテストの網羅性と呼ぶことにする。　テストの効率は網羅性とテスト規模（テスト項目数）によって評価するものとする。

(2)網羅性

テストの網羅性を次の二つに分類する。

(a)第１種テスト網羅性

プログラムの誤りのうち処理誤りと処理漏れを検出する網羅性であり、仕様書に定義されていることを完全にテストすることによって実現できる。

(b)第２種テスト網羅性

定義されていない処理を付加していない（誤配線）ことを検出する網羅性。具体的には入力から見て定義されていない出力へ悪影響を及ぼさないことを確認することに相当する。従来のテストでは膨大な量になる。

(3)テスト量

第１種テスト網羅性を実現するために必要なテスト量を増加させる要因には次のものがある。

(a)入力の場合の数

一つの入力に着目しても、離散的な値を用いると多くの組み合わせが起こり得る。例えば「８文字以下の英数字」と言う入力条件でも３６の８乗となり無限に近いテスト項目となる。

(b)組み合わせ範囲

出力からみて、関係する入力項目と、その論理関係の問題、最悪の場合にはそれぞれの入力の場合の数の積となる。

(c)状態遷移

状態遷移を含む場合、テストデータは時系列的な順序が求められる、その結果やはり膨大な量になる。

３．ＶＯＰの概要

３．１考え方

従来のテスト技法はプログラムの設計と独立に行われていた、そしていかなる設計を行っても誤りや漏れを検出することが求められている。　しかし、この暗黙のテストアプローチには限界があり、先に説明した「第２種テスト網羅性」や「テスト量」などどの項目をとっても完全なテストを行うことは不可能である。

完全なテスト（網羅性のあるテスト）とはテストの特性でもあり、またプログラムの特性でもある。

ＶＯＰの考え方は、現在のテスト技法でテスト可能な範囲内の制約条件でプログラムを設計しようとするものである。　この考え方の例を図-3に示す、図-3aに示すような２つの入力がそれぞれ２つの同値を持つとする、テストケースは積とし４個とするのが一般的である。　これは図-3cのようなプログラムが作られた場合に備えているのである。ところが図-3bのようなプログラムの場合には積とする必要は無いのである。

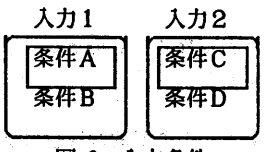


図-3a 入力条件

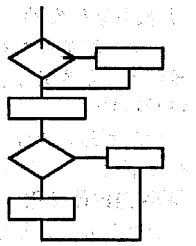


図-3bプログラムＸ

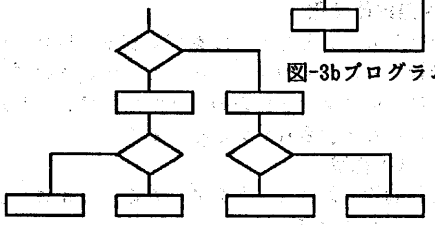


図-3c プログラムＹ

プログラム設計と関連するテスト可能性のための制約条件には以下のもが考えられる。

(1)第２種の漏れ

定義された処理以外のものが作られないようにする。データの隠ぺい、参照制約、カプセル化などの方法で行う。　制御の誤配線については制御フローなどの定義により制約する。

または、動的なテスト以前に静的なテストで検証する方法もある。

(2)同値分割とその連続性

入力、出力とも取り得る場合の数を明確にするため同値と呼ばれる部分集合として排他的に定義する。このとき異常同値も漏れなく定義し、同値内では処理の連続性が保証される設計を行う。

(3)出力同値と入力同値の関係（組み合わせ）

各出力状態（同値）に対して入力の同値がどの様に関係するかを論理的（ブール関数）に定義する。これは積の関係になるものを制約する。

(4)状態遷移の定義

状態遷移について取り得る状態とその遷移を定義し、仕様で定義された状態以外の内部状態を禁止する。定義された状態遷移はテストのために順序回路をスキャニングする機能を付加して設計する。

３．２ＶＯＰの構成

ＶＯＰは図-4に示すような構成である。主な特徴として以下のものがある。

(1)設計制約条件

テスト容易なプログラムを作るためのプログラム設計上の制約条件となるもの。

(2)静的テスト

設計制約が守られていることを静的な検証により行う手段。

(3)段階的仕様化

テストの入力となる仕様をプログラム設計において詳細化されたものを用いる。これは実際のプログラムにおいてエラー処理のようなものはこの段階にならないと定義されないからである。

(4)テスト基準

設計制約と対になるもので、どのレベルまでテストを行うかを規定する。

(5)単体テストと結合テスト

テストにより検証する範囲を分割して行う、状態遷移などは単体テストでしか検証できない。

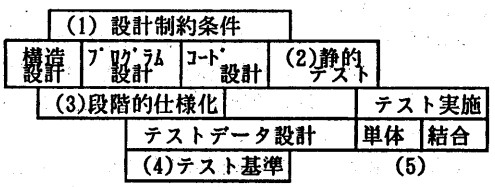


図-4　ＶＯＰの構成

４．ＶＯＰによるプログラム設計

４．１プログラム設計工程

ＶＯＰでは図-5に示すような工程で設計を行う。

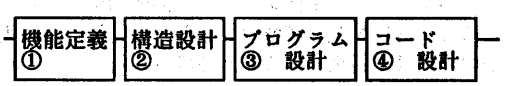


図-5プログラム設計工程

テストは入力データをもとに出力で検証を行うので、機能を細分化する設計方法よりもデータを中心に設計を行う必要がある。　この考え方は機能情報関連図やＤＦＤ(data flow diagram)を用い、データを考慮した設計技法と同じ考え方である。

ＶＯＰではさらにその入出力データの取り得る範囲（同値と呼ぶ）を正常系だけでなく異常値をも含め設計する点にある。

４．２機能定義

機能定義において以下の項目を定義する。

(1)入力と出力の定義

プログラムの入力と出力データについてその取り得る値の範囲を含め定義されていること。

(2)機能と入出力の関係

機能の詳細と入出力の関係がＤＦＤのような方法で定義されていること。

４．３構造設計

プログラムの内部モジュール構成を設計するが、特徴は内部モジュールを３つの種類に区分けして設計を行う点である。図-6に示す手順で行う。

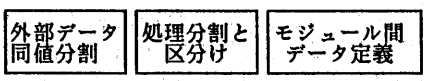
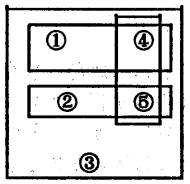


図-6構造設計の手順

(1)外部データの同値分割

機能仕様で与えられた入力と出力について同値分割を行い、値の取り得る範囲を部分集合として定義する。このとき入力される可能性のある、誤ったデータも異常同値として定義する必要がある。

同値分割とはデータの取り得る値の部分集合に分割したもので、図-7に示すようなベン図を応用した同値分割図として表す。



① ８文字以内の英数字
② 省略（デフォルト）
④⑤ 該当する名前が存在する
③ ①②以外の指定

図-7同値分割図の例

(2)処理分割と区分け

論理ＤＦＤから物理ＤＦＤを作る過程であるが、分割する処理単位を次の３つの種類のモジュールと２つのデータ（内部変数）に分割する。　ここでの入出力はモジュール間の内部データを含む。

(a)パッシブモジュール

入力が与えられると何らかの処理が行われ出力が生成される。処理自身の複雑性ではなく入力の組み合わせに影響しないモジュール。

(b)組み合わせモジュール

入力の組み合わせによって色々な出力が選択されるモジュールで、状態遷移を含まないもの。

(c)順序モジュール

(b)と同じであるが、状態遷移により出力が変化するもの。ただしこの場合には状態を表す変数をモジュールの中に定義せず、(e)の状態変数として明確に分離して定義する。　どのような状態遷移が必要かの仕様については仕様として与えられるものとする。

(b)(c)には極力処理を含めず、処理はパッシブモジュールとして分離する。

(d)内部データ（ファイル）

モジュール間のデータで蓄積されるものを定義する。テストを容易にするためレコード間の関係をＥ－Ｒモデル等を用いて定義する。

(e)状態変数

順序モジュール（複数も可）から参照、設定される状態遷移を表す変数を特別に定義する。

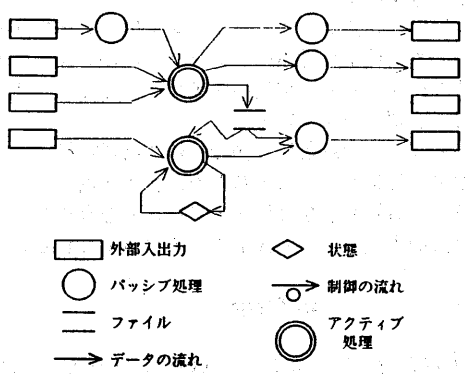


図-8モジュール分割の例

従来の物理ＤＦＤとの大きな相違点はモジュールの種類を区別する点と、その区別の中の順序回路を含むモジュールの状態変数をモジュール外に定義する点である。

(3)モジュール間データ定義

外部データと同様に、モジュール間でやりとりされるデータの同値分割を行う。このとき状態変数についても状態の数だけ分割を行う。

４．４プログラム設計

モジュール単位の入出力と処理機能をもとに、それを実現するアルゴリズムを設計する。　プログラム設計では次の項目を制約条件として設計を行う。

(1)規定された入出力以外へのアクセス

プログラム構造で定義されたデータ以外にアクセスをしないことを保証する設計を行う。

(2)規定されたモジュール以外への処理起動

同じくプログラム構造で定義されたモジュール以外へ制御を移さない設計を行う。この両者はソースコードを静的に解析するツールでさらに検証できる。

(3)モジュール内部状態の禁止

構造設計で定義されたモジュール外部変数としての状態値以外に内部状態を持たない。そのためにモジュール内の判断文において入力から導出される値と定数以外の使用を禁止する。（LOOPは別）

(4)同値の連続性保証

定義された入力同値の連続性を保証する必要がある。連続性とは与えられた同値の境界条件以外で判断文を作成しないことである。

構造設計で定義された入出力の同値をさらに細分化する必要がプログラムの詳細設計で生じたならば、必ず仕様を修正する。これはエラー処理などで実際に詳細設計を行わなければ分からない部分に対応する。

(5)入力と出力の関係を定義する。

デシジョンテーブルのような漏れの無い表記法をを用いて入力同値と出力同値の関係を記述する。

なをこれらの設計においてパターン化設計や部品化設計を用いることは言うまでもない。

４．５コード設計

コード設計はプログラム設計で付加された制約(1)から(4)までを守って作成する。制約が守られていることの検証はプログラム設計のところで示した静的なテストとしてツールにより行うことが可能である。

５．ＶＯＰによるテスト設計

ＶＯＰの特徴は先に述べたように、網羅性の高いテストデータを少ないテスト項目数で実現出来る点である。プログラム設計において詳細化された仕様を基にテストデータを抽出する。おおむねこの手順はＣＦＤ（Case Flow Diagram）と呼ぶ方法に従っている。[1]-[3]

テストデータはプログラムの個々の構成モジュールに対して行う単体テストと、モジュールを組み立てて行う結合テストに分けて行う。

５．１単体テストデータの設計

単体テストは該当するモジュールの種類によってテストデータの設計方法が多少異なる。いずれの場合も「テストの第２種の漏れ」についてはプログラム設計上の制約で解決されているとする。

(1)パッシブモジュール

入力データにより一意に出力が決まるので、組み合わせを考慮する必要が無い。出力に着目し、その出力を得るために必要な入力をテストデータとして抽出する。

(2)組み合わせモジュール

入力の組み合わせによって出力が変化するモジュールであるが状態遷移を含まない。入力同値と出力同値との論理関係を求めテストケースを作る。

モジュールの入力と出力の同値図を並べ、原因流れ図（ＣＦＤ）により関係を求める。[1] 別の方法としては原因が結果グラフを用いても良い。[7]

テスト項目はいずれの方法を用い場合でもデシジョンテーブルにまとめる。デシジョンテーブルをＣＦＤあるいは原因結果グラフから求めるアルゴリズムはすでに存在する。[1][7]

(3)順序モジュール

内部に状態遷移のあるモジュールのテストは難しいが、ＶＯＰでは状態変数をモジュールの外に出して設計する制約を課している。これは単体テストの段階でこの状態変数を入力、あるいは出力として取り扱うためである。

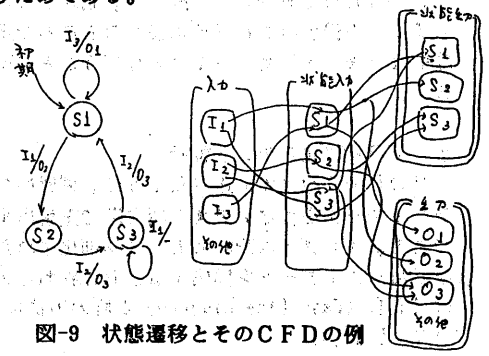


図-9　状態遷移とそのＣＦＤの例

テストデータの作成は組み合わせモジュールと同様で、状態変数の同値分割図を含めてＣＦＤを記述しテストケースを抽出する。

５．２結合テスト

結合テストは外部入力データと外部出力との間で行われる。ここでも「テストの第２種の漏れ」に対するテストは行わない。また原理的に単体テストで外部入力データの全ての同値に対するテストは終わっている、そこでここではモジュール間の接続を中心にテストデータを抽出する。

テストデータの抽出手順は外部出力に着目し、その出力から逆にＤＦＤを辿り入力データを決定する。出力の同値の内、正常同値のみにを対象としする。また全てのモジュールが網羅されることを結合テストの基準とする。

６．今後の課題

以上検証指向のプログラミング技法：ＶＯＤを説明した。検証の行為は、プログラムとテストの相対的な問題であり、相互の関係についてさらに深い検討が必要と考えている。

もう一つの課題は、機能分析や機能定義など上工程の検証に対してこの考えを拡張する点である。ＶＯＡ（分析）、ＶＯＤ（デザイン）を含めシステム設計の方法論として体系化する必要を感じている。

参考文献

[1]松尾谷、”テストケース抽出の一方式”、37回情処学会全国大会　1988年後期

[2]松尾谷、赤沢、”原因流れ図によるテストケース抽出技法”、38回情処学会全国大会　1989年前期

[3]松尾谷、”テストケース抽出の一方式「順序回路を含む場合」”,39回情処学会全国大会　1989年後期

[4]松尾谷、”ソフトウェアテストにおけるテストリソースの最適配分”、電子情報通信学会　信頼性研究会　R90-4 1990年5月

[5]藤野、松尾谷、”デバグ工学における制度配分問題”、平成元年度電気情報関連学会連合大会、1990年9月

[6]松尾谷、真野、他、”ソフトウェア検査モデル１～３”、29回情処学会全国大会　1984年後期

[7]石井康雄編、”ソフトウェアの検査と品質保証”日科技連ソフトウェア品質管理シリーズ第４巻