事務システムにおける形式仕様適用例

佐原伸
日本フィッツ(株)トレードワン事業部
Shin.Sahara@jfits.co.jp

事務処理ソフトウェア開発では，曖昧な仕様に起因するトラブルが非常に多い．仕様段階で，自然言語により曖昧かつ手続的に仕様を記述し，それを手続的にプログラミングしていくことにより，重大な欠陥を含み，保守性が無く，再利用性のないソフトウェアが大量に生産されている．

本稿では，オブジェクト指向形式仕様記述言語 VDM++ を，Java と C++ を使った，オブジェクト指向による開発を行っている事務システム作成に使用し，より欠陥の少ないシステムを作成した経験を報告する．

1.0 はじめに

事務処理ソフトウェア開発の仕様記述工程において，仕様は自然言語により手続的に記述されることが多い．最近，UML[1] などを使った，オブジェクト指向による仕様記述も行われるようになったが，詳細な仕様記述に関しては相変わらず自然言語による手続的記述が行われている．このような仕様記述法は，(1) 自然言語の曖昧さに起因する設計の欠陥，(2) 手続的な記述から「意味」を把握する困難，(3) 仕様記述段階で不完全な手続的設計が混入することによる後続工程の困難 ... などが発生し，トラブルの原因となりやすい．

また，実用的な大きさのシステムにおける仕様記述では，自然言語による記述といっても，一種の疑似コード的記述が必要となり，その疑似コードの構文をその場で「発明あるいは決定」する時間も馬鹿にならない．

このような状況を変え，高品質の事務処理ソフトウェア・パッケージを開発するため，形式仕様記述言語を使うに至った．

今回は，仕様記述言語としてデンマーク IFAD 社の VDM++[2] とツール VDMTools を使用した．この VDM++ は，集合をベースとしたモデル指向の ISO 標準仕様記述言語 VDM-SL[3][4] を，オブジェクト指向システム用に拡張したもので，集合の操作が基本である事務処理ソフトウェア開発で，かつ C++ や Java による実装を行うシステムの記述に適していると判断したからである．

以下，第 2 節では本稿の対象とした「マル優管理システム」の概要と，適用工程などプロジェクトの概要を示す．第 3 節では，VDM++ 適用について記述する．

2.0 マル優管理システム

2.1 システム概要

日本の高齢者または障害者は，預金ならびに信託資産が生み出す利息への課税の免除をある限度まで受けることができ，この制度を通称「マル優」と呼ぶ．

弊社では，証券会社のバックオフィス機能を提供するソフトウェア・パッケージ「トレードワン」を開発しているが，開発対象のマル優管理システムは，「トレードワン」システムのサブシステムであり，この税金免除に関わる申告額と顧客の証券残高を管理する．

主とする管理項目は，顧客毎のマル優申告額と残高，および顧客の累積投資コース毎のマル優申告額(通称「マル限限度額」)と残高である．

システムのアーキテクチャは，3 層モデルである．クライアント側は WebLogic Server 上に Java で開発し，サーバー側は WebLogic Enterprise 上に C++ で開発し，データベースはオラクルを使用している．アーキテクチャは，「トレードワン」システム標準のものを使用し，マル優管理システムでは特に変更を加えていないため，本稿ではこれ以上言及しない．

2.2 プロジェクトの概要

マル優管理システムは，2000 年 9 月中旬から 11 月末まで「VDM チーム」が設計・開発を行い，11 月中旬から 2001 年 1 月まで連結テスト・総合テストを，テストチームが行っている．「VDM チーム」も，後述する「仕様修正」のため，2001 年 1 月 9 日まで修正作業を行った．

「VDM チーム」は，全員，ソフトウェア開発歴 20 年以上で，フルタイムが 3 人，0.4 人月 / 月が 2 人，0.2 人月 / 月が 1 人の計 6 人で，投入工数としては 4 人月 / 月である．一人を除いて全員証券業務作成の経験はなく，VDM++[1] の使用も初めてであった．プログラム作成者は，C++，Java を初めて本格的に使用した[2]．ただし，形式手法・形式仕様に関しては，4 人が知識を持っていて，ある程度の経験もあった．

VDM++ の適用工程は，詳細設計工程である．本来，仕様記述言語は要求仕様工程から使うべきであるが，「VDM チーム」が開発を引き継いだ時点で，すでに要求仕様と分析は終了し，サブシステム間のインタフェースも決定されていた．また，後述する「業務詳細仕様書」,「データベース仕様書」で，本システム自体の設計の枠組みもほぼ決定されていた．

時間的余裕があれば，分析工程に遡って VDM++ を適用することは可能であるが，予想ステップ数 3 万行のシステムを，開発着手から約 2ヶ月で一部機能を連結テストチームに引き渡さなければならず，2ヶ月半で全機能を連結テストチームに引き渡すため，適用効果が限定されることを覚悟して，詳細設計工程から取り組んだ．

本システム開発用の文書として,「業務詳細仕様書」,「データベース仕様書」「画面仕様書」,「チェック仕様書」などが存在した．

業務詳細仕様書は「システム要求」を記述した文書であるが，機能中心に分割され，手続的に記述されていて，実際にはいわゆる「詳細設計書」に近い[3]．データベース仕様書は，リレーションの項目一覧に，キー項目などの注釈があるだけのものである．画面仕様書は，画面遷移と画面イメージと画面の項目説明からなる．チェック仕様書は入力項目のチェック方法について，自然言語と表で記述してある．

3.0 VDM++ の適用

3.1 適用機能

VDMTools の構文チェック・型チェック機能を主として使用し，詳細設計を行った．クラス図のドキュメント化は，Rational 社 Rose ツールとの相互接続機能を使って行い,「トレードワン」システムと独立の VDM++ 共通ライブラリーのテストには，VDMTools の仕様実行インタープリタとコードカバレージ機能を使用した．

3.2 適用内容

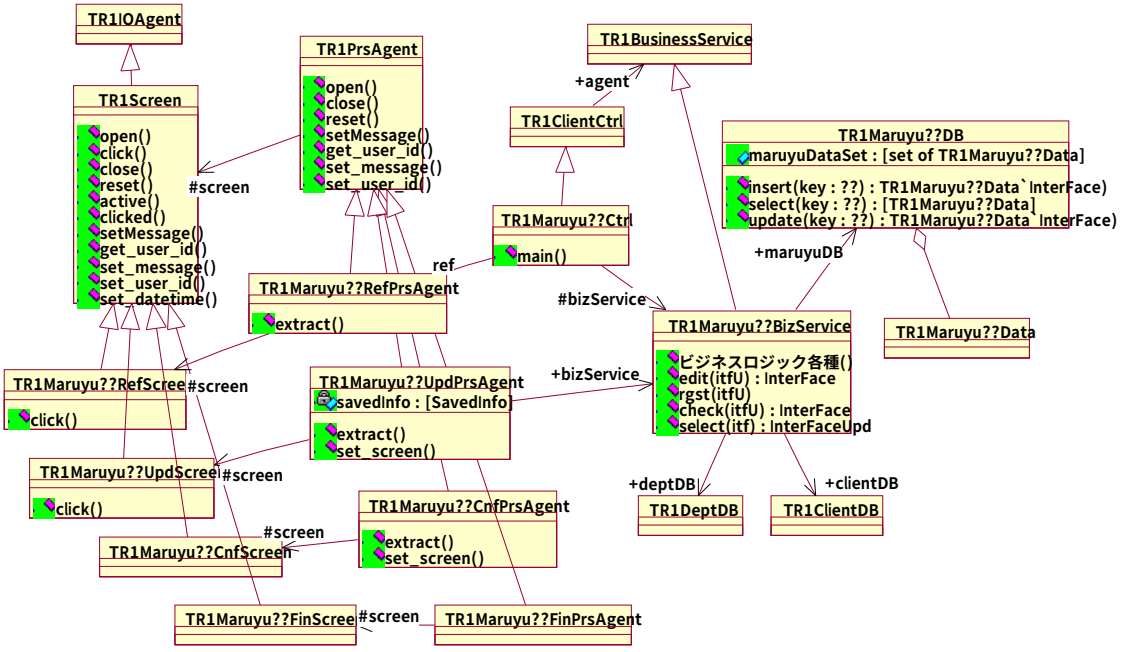
3.2.1 クラス構成

本システムは，典型的な 3 層システムで，GUI 層・ビジネスロジック層・データ層を持つ．そこで，VDM++ 仕様も 3 層とし，各機能毎のクラス群を下図のような標準クラス構成で記述することにした．また，各クラスの共通メソッド・インタフェースを規定し，標準化による教育効果と，生産性および保守性の向上を図った．

---

1. 2 人は，2000 年 7 月に，VDM++ を使ったプロトタイプ作成実験に約 2 週間携わった．また，4 人は 2000 年 7 月初旬に 5 日間行った VDM++ 入門セミナーに参加している．
2. 1 人は，Java 言語製品のサポート業務は経験している．
3.「トレードワン」システム第 2 版の業務詳細仕様書は，形式仕様を意識して，まだまだ不十分とはいえ，かなり改良されたものになっている．マル優管理システムは「第 1 版」のため，手続的記述が色濃く残っていた．

図 1. 標準クラス構成



クラス群は，大きく GUI 層を記述する IOAgent，PrsAgent，Ctrl，ビジネスロジック層を記述する BusinessService，データ層を記述する DB の 5 つに分かれる．

IOAgent は画面を表す抽象クラスであり，そのサブクラスは各画面の振る舞いを記述する．例えば，TR1MaruyuUpdScreen クラスは，顧客毎のマル優申告額と残高を管理する機能の「変更画面」を表す．ここでは，例えば，入力項目の単体チェック仕様などを記述する．

PrsAgent(Presentation Agent) は入出力媒体に依存しない入出力を表す抽象クラスであり，そのサブクラスには入出力の振る舞いを記述する．ここでは，例えば，入出力項目の外部表現と内部表現の変換などを記述する．

Ctrl(Control) はクライアント側の状態を表す抽象クラスであり，そのサブクラスにはクライアント側の状態遷移を記述する．ここには，例えば，状態遷移や例外処理などを記述する．

BusinessService はサーバー側のビジネスロジックを表す抽象クラスであり，そのサブクラスにはアプリケーションの中心となるビジネスロジックを記述する．ここでは，例えば，データの制約や編集方法，あるいはデータの参照・登録・更新などを記述する．

DB は一つのリレーションを表す抽象クラスであり，そのサブクラスには一つのリレーションのアクセス方法を記述する．このクラスは，スタブ・モジュールであり，実データベースにアクセスする訳ではない．ここでは，例えば，リレーションの参照・登録・更新などを記述する．なお，リレーション中の 1 レコードを表すクラスは，Data で終わる名前を付けた．

また，すべてのクラスは「トレードワン」システムのオブジェクト共通の振る舞いを表す TR1Object のサブクラスであり，TR1Object はオブジェクト共通の振る舞いを表す Object クラスのサブクラスである．

この他にも，共通クラスとして日付・集合・シーケンス・文字列に関するクラスを作成し，「トレードワン」システムの共通メソッドに対応するスタブ・モジュールとして共通クラス 20 個を作成した．

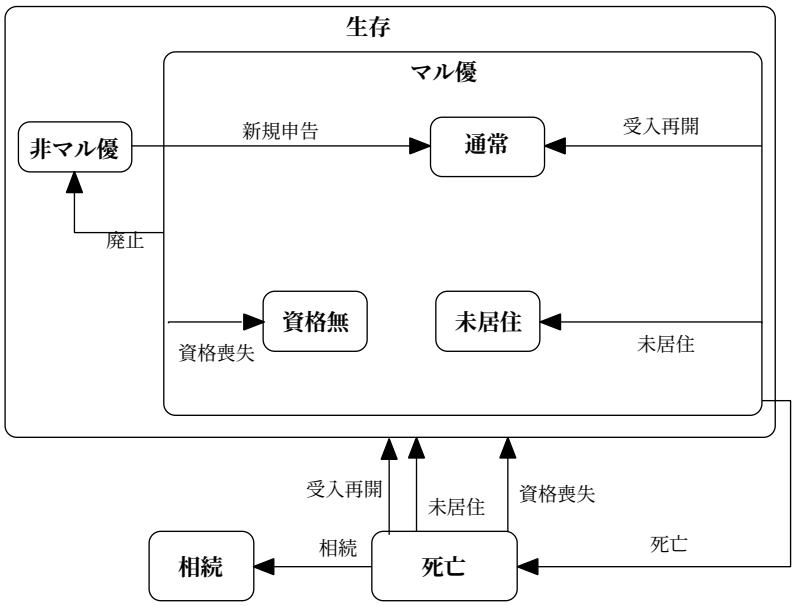
3.2.2 制約の記述

システム開発の最終段階で，サブシステム間インタフェースや，業務詳細仕様書に起因する欠陥が数多く発見されたため，業務詳細仕様書を見て修正することが困難になり，マル優管理システムの制約を記述し整理しなければならなくなった．これは，本来，設計に先立って行うべきことであり，前述の時間的制約のため省略した作業であるが，やはり必要となった．

主要な制約は，マル優顧客の状態と，管理項目の制約式であった．また，帳票を作成するためのデータの時系列制約は，業務上は重要ではないが，このデータをバッチ処理時におけるオンライン時との乖離の修正に使っていたため，設計仕様および効率上重要な制約となった．

マル優顧客の状態遷移は，以下のようになった．この仕様は，もともと業務詳細仕様では一つの区分の値として，状態と遷移が混在する形で記述されていたため，仕様に様々な漏れが生じていた．

図 2. 顧客の状態遷移



管理項目の主要な制約式としては，全部で 22 個洗い出され，例えば，以下のようなものがある．

「マル優口座が存在し，かつ，その口座のマル優取扱区分が，通常か未居住か資格無であり，かつ，以下の式が成り立つこと．ここで，Σ は「顧客毎の合計」を意味する．」

マル優申告額 = マル優余裕額 ＋ Σ 一般投信マル優残高 ＋ Σ マル限限度額 **(式 1)**

マル優残高 = Σ 累投証券残 ＋ Σ 一般投信マル優残高 **(式 2)**

累投証券残 ≦ マル限限度額 **(式 3)**

この制約式は，「ある特定の条件下で，マル優限度額を越えた買付がいくらでもできてしまう」という業務詳細仕様書に隠れていた重大な意味的欠陥を発見することに役立った．

3.2.3 適用結果

作成した仕様は，全体で 92 クラス 1 万 1856 行[1] になった．このうち，VDM++ で事務システムを開発するのに必要な共通クラスが 5 クラス，774 行あり，「トレードワン」システム開発のための共通クラスが 20 クラス 1342 行である．また，本来要求仕様あるいは分析仕様に含まれるべき，マル優システム全体の制約をまとめた

1. 注釈行は除外している．

ものが 1539 行になった．従って，マル優管理システムのソースコードに対応した詳細設計レベルの VDM 仕様は 8102 行ということになる．

対応する Java のソースコードは，20 クラス 9403 行で，C++ のソースコードは 23 クラス 9028 行で，合計 1 万 8431 行あった．

表 1. 仕様とソースコード

分類	VDM++ 仕様	Java	C++	プログラム合計
事務処理共通クラス	774 行			
トレードワン共通クラス	1342 行			
制約仕様	1539 行			
詳細設計	8102 行	9403 行	9028 行	1 万 8431 行
合計	1 万 1856 行	9403 行	9028 行	1 万 8431 行

これとは別に，IDL 生成機能[1] とデータベース・インタフェース生成機能とを使って生成したソースコードなどがあるが，これらは上記計算に入れていない．

幸い，システムスタート後の欠陥・修正は，まだ報告されていない．

連結テスト以後のテストで発見された欠陥・修正は以下のようになった．

表 2. 連結テスト以後に発生した欠陥・修正

欠陥・修正原因	件数
サブシステム間インタフェースの了解違いによるもの	9
ユーザーレビューによる機能追加	1
業務詳細仕様書の考慮漏れ	1
用語集の未整備による項目定義の誤解	1
合計	12

単体テスト以前に，レビューで発見された欠陥・修正は以下の通りである．

表 3. 単体テスト以前に発見された欠陥・修正

欠陥・修正原因	件数
業務詳細仕様書の考慮漏れ	35
チェック仕様書の考慮洩れ	13
エラーメッセージの申請忘れなど管理上のミス	3
ユーザー要求による修正	2
画面仕様書の間違い	2
テーブル仕様書の間違い	2
合計	57

1. マル優管理システムの開発の一環として作成されたが，記録が残っていない．

開発期間は約 3ヶ月で，工数は以下の通りである．

表 4. 工数

作業内容	工数 / 人月
仕様作成	5
プログラム作成	6
管理	1
開発工数合計	12
仕様修正	0.8
プログラム修正	0.5
修正管理	0.5
修正工数合計	1.5
工数総合計	13.3

なお，業務詳細仕様書は修正不能な状態なので，設計仕様書という形で，業務詳細仕様書と VDM 設計仕様の両方を含んだドキュメントを作成した．

4.0 考察

4.1 品質について

連結テスト以後のテストで発見された欠陥・修正は，すべて VDM++ 適用以前の工程で発生したものであり，いずれも，今回 VDM++ を適用した工程以後では発見が不可能なものであった．典型的な欠陥例は，他のサブシステムから渡されてくるデータに，必要な情報が欠落しているというものである．例えば，締め後の約定データは，翌日の約定になるのだが，本システム側では，当日の約定データか翌日の約定データなのかが区別できないという欠陥がある．

それでも，VDM++ 適用以前の工程で発生した欠陥のうち，発見可能だった業務詳細仕様書の欠陥は，単体テスト以前にすべて発見し修正している．

従って，VDM チームは，品質の高い開発を行ったと言える．

4.2 オブジェクト指向形式仕様記述言語 VDM++ について

VDM++ およびそのツール VDMTools については，開発開始時点でチーム全員がほぼ理解していたとはいえ，あくまでも入門セミナーを終えたレベルでの理解であり，実際の仕様記述ではかなりの試行錯誤が必要であった．筆者の場合には，途中まで書いた仕様を 2 回全面的に書き直した．しかも，参考になり得る実用レベルでの事務システム開発例あるいは事務システム用ライブラリが無い条件下で，高品質なソフトウェアをかなり厳しい期限内で完成し得たことは，VDM++ とツールの威力が大きかったことを示している．

前件と後件を記述することで，手続中心の業務詳細仕様書と対比して，問題点を洗い出すことができた．特に，VDMTools の型チェック機能を使用することにより，無数の小さなミスを簡単に修正でき，結果として意味的欠陥に集中することができたことは確かである．また，型チェックシステムは，仕様の修正余波をかなり適格に検出し，通常のソフトウェア開発で頻発する「修正に伴う欠陥の発生」をかなり防ぐことができた．

筆者の定性的観察として「複合設計や構造化分析／設計やオブジェクト指向分析／設計などの従来手法より，仕様記述言語の曖昧さが無く，言語レベルが高いので，仕様を書きやすい」ことも確認できた．特に，集合の内包（Set Comprehension）やパターンによる束縛あるいは構成子といった構文が役立った．また，一般の形式仕様では，時間に関する制約の記述が難しいのだが，事務処理システムの場合，バッチ処理に使用するファイルなどの時系列データの制約をファイル内の前後関係として記述することで，時間制約が比較的簡単に記述できることも分かった．

高品質の要因は，開発要員の優秀さによるところも大きく，どこまでが VDM++ 言語とツールの効果かを厳密に分離できてはいないが，生産性は開発要員を最優秀と仮定した COCOMO[6] による見積の約 3 倍（工数・期間とも COCOMO 見積の３分の１）であった．また，開発したプログラム行数は，当初見積の 60% で済んだ．

一点だけ困ったことは，後件の記述中で使用する前件での値（old value）が，型のインスタンスでは正しい値を示すが，オブジェクトの場合はオブジェクト参照値であり，前件と後件で同じ値を返すので，オブジェクトの値を参照する後件の記述が困難だったことである．

4.3 アーキテクチャについて

図１の標準クラス構造は，３層構造システムの仕様記述について，仕様記述者が，どのクラスにどの責任を負わせ，どのようなメソッドが必要かを迷う必要がないので，VDM++ による記述に慣れていない仕様記述者による，大規模な開発に有効であることが確認された．

ただし，本システムでの BusinessService クラスは，ビジネスオブジェクトとそれらが活躍すべき「場」という，本来分離されているべき２種類のオブジェクトが一体になってしまっているため，事務処理システムで頻繁に起こる，ビジネスオブジェクトに関する修正の変更余波が大きくなっていることが確認できた．

4.4 改善点について

4.4.1 オブジェクト指向的設計

本システムの開発では，大域情報をオブジェクト内に局所化するという，本来のオブジェクト指向的設計が十分にはできなかった．特に，売買区分や証券種類といった，各種の区分の値を「トレードワン」システム全体の大域情報として持っているため，局所化が時間的に困難であった．このため，複数のクラス内で，例えば売買区分に関してそれぞれ判定するというコードの重複が生じている．

次期システムの開発では，マル優申告情報といったビジネスオブジェクト以外に，顧客コードや銘柄コードといった各項目をオブジェクトとすることを検討した．

例えば，項目のチェックであればその項目オブジェクトのみがチェック方法を保持し，他のクラスはその項目オブジェクトに「あなたは正しいか？」というメッセージを送るようにする．項目の外部表現[1] と内部表現[2] の変換も，他のオブジェクトからは「外部表現に変換せよ」といったメッセージを送ればよい．

4.4.2 ビジネスオブジェクトと「場」の分離

今回の開発では，前述したように，ビジネスオブジェクトとそれらが活躍するべき「場」とを一体とした BusinessService クラスを作成したが，チェックし，編集し，データの登録・変更を行う，といった「場」は，事務システムの場合，業種に関わらず，かなり汎用的に構築し得る．一方，ビジネスオブジェクトはビジネス固有の情報を保持しているのだから，業種や会社を越えた汎用化は行いにくい．

そこで，この両者を分離し，汎用の「場」の上を，固有のビジネスオブジェクトが動き回る分析モデルあるいは設計モデル[3] が構築可能であることを検討した．また，情報代数の利用が可能ではないかとの指摘もあった[5]．

5.0 まとめ

本稿では，証券会社のバックオフィスシステムの１サブシステムである，マル優管理システム開発の詳細設計工程に，オブジェクト指向形式仕様記述言語 VDM++ を適用した．

---

1. 例えば，文字列で 8 桁というような，画面上の表現を表す．
2. 例えば，10 進オブジェクトというような，ビジネスオブジェクト内での表現を表す．
3. 事務員の机の上のトレイ上の書類が次々と処理されていくこととの比喩で，トレイモデルと呼んでいる．トレイとそれに付随するロジックが「場」オブジェクトで，書類がビジネスオブジェクトということになる．

結果として，該当工程での欠陥の発生は無く，前工程の欠陥もかなり発見・修正することができた．また，Java と C++ によるプログラミングの生産性も高かった．

しかし，要求分析工程および分析工程さらには概要設計工程での欠陥を全て防ぐことは，当然のことながらできなかった．

今後は，今回の試みで実現できなかった，

■ 要求分析工程からの VDM++ 適用

■ より大きな開発での VDM++ 適用

■よりオブジェクト指向的な仕様作成

■ VDM++ インタープリタによる仕様の実行と確認

を，2001 年 5 月から行う予定である．

■ VDM++ コード生成機能による，C++ あるいは Java の生成機能の適用

も，今後の課題としたい．

6.0 謝辞

VDM++ の採用は森威文さん (日本フィッツ) の決断である．VDM++ の標準クラス構成は谷津弘一さん (日本ユニシス) と酒匂寛さん (デザイナーズデン) の案によるところが大きい．同僚の佐藤圭さん (日本フィッツ) と西川典子さん (日本フィッツ) は，初めての証券業務にも関わらず，VDM++ による業務仕様全体を把握し，プログラムの作成も行った．Paul Mukherjee さん (IFAD) は，VDMTools のサポート・機能拡張・修正を迅速に行って下さった．荒木啓二郎さん (九州大学) と峰尾欽二さん (フリー) および西川典子さん，谷津弘一さんから論文に対する意見を頂いた．

また，山崎利治さん（フリー）は，10 年前に，VDM が事務処理システムの仕様記述にも向いていることを指摘してくださった．岸田孝一さん (SRA) は，7 年前にマカオの国連大学高等ソフトウェア研究所における，形式手法セミナーへの参加を勧めて下さり，今回のシステム開発のきっかけを作って下さった．

ここに記して感謝する．

7.0 参考文献

[1] OMG. Unified Modeling Language Specification. Object Management Group, Inc., Version 1.3, June 1999

[2] VDMTools:The IFAD VDM++ Language - Revised for V6.6. IFAD, 2000

[3] VDMTools:The IFAD VDM-SL Language - Revised for V3.3. IFAD, 2000

[4] John Fitzgerald, Peter Gorm Larsen. Modelling Systems: Practical Tools and Techniques in Software Development. Cambridge University Press, 1998

[5] 山崎利治 : 情報代数 , SEA sigfm ホームページ (http://shinsahara.com/www/sigfm/material/infAlgebra.pdf), 2001

[6] Barry W.Boehm: Software Engineering Economics, Prentice-Hall,Inc., 1981