「VDM++仕様記述と分析」の定石と手筋

佐原　伸
法政大学情報科学研究科
ss@shinsahara.jp

要旨

形式仕様記述言語VDM++で仕様を記述し，それを分析できる産業界の技術者を育てることは簡単ではない．

しかし，長年にわたって囲碁や将棋を学習し，多くの技術者や学生に，VDM++による仕様記述方法を教えた経験から，「仕様記述と分析」を定石や手筋を使って教育することが，囲碁や将棋の教育と同様に有効だと考えた．特に，仕様記述に重要な不変条件(invariant) ・事後条件(post-condition)・事前条件(pre-condition)の記述を容易にできるよう，定石と手筋を集めて検索可能にすれば比較的容易に「仕様記述と分析」行うことができる．

このような「定石集」を試作し，評価した結果，3節で述べるように，かなり役立ちそうである事が分かった．また，これらの定石をコンピュータ化すれば，検索や「仕様の一部」の自動生成に活用できる可能性もある．

0. はじめに

産業界の技術者は要求仕様や設計仕様のVDM++による仕様記述に困難を感じる人が多い．特に，不変条件，事後条件，事前条件の記述ができない．手続き的記述には慣れているが，状態を宣言的記述で表現することに慣れていないため，記述することが難しいようである．

例えば，顧客のデータベースをインスタンス変数「i客DB」で表し，そこに新しい客「a新しい客」を追加する操作(operation)では，事後条件を

post i客DB = i客DB~ union {a新しい客}

と記述すればよいのであるが，これがなかなか書けない．ここで，~は，インスタンス変数の旧値（操作が実行される前の値）であり，集合の合併演算子unionを使って，引数で与えられた「a新しい客」を要素に持つ集合「{a新しい客}」と合併した集合が，「i客DB」（操作が終わった後のインスタンス変数「i客DB」）と等しいと書けばよい．

このような「手筋」（仕様中のいくつかの場所で役立つ方法）を「集合に要素を追加する手筋」として整理し，仕様記述例と説明を与えておけば，仕様記述の上達に役立つ．囲碁や将棋では，このような手筋や定石が整理され蓄積されていて，囲碁の場合，アマチュア初段程度の実力でも2万個ほどの定石と手筋を使いこなさなければならないとされている．逆に言えば，定石や手筋を使いこなせれば，アマチュア初段程度の実力を発揮できるということであり，VDM++仕様記述の定石と手筋を整理して蓄積すれば，VDM++アマチュア初段レベルの技術者を養成できることになる．

しかし，囲碁や将棋のように仕様記述と仕様検査のための定石と手筋を集めようとして，文献1, 2, 3, 4など，関連しそうな文献を調べたが，いずれも証明が中心[1]であり，産業界向けの例題としては適当なものがなかった．そこで，筆者の作成した実用的なモデルをセミナー用に縮小したモデル群と，文献5を中心に定石と手筋を収集して，これを整理し，評価を行った．

[1] 筆者たちは，証明を使わずに仕様をテストする手法を中心に産業界へのVDM普及を行っている．現存するVDMのツールも，すべて仕様をテスト実行する仕様アニメーション機能しか提供していない．

評価は，「定石と手筋集」作成では参考にしなかった既存文献（文献5と6）上の例題を使って，どの程度定石と手筋が使われているかを調べた．その結果．すべての不変条件，事後条件，事前条件の記述に，この「定石と手筋集」の定石と手筋が使われていた事が分かった． 3節で．収集した定石・手筋がどの程度有効であるかの実験結果を示した．

収集した定石・手筋の構成は以下のとおりである:

1. 仕様記述
2. 分析
3. ツール
4. マニュアル

分析を追加したのは，記述した仕様を分析するための重要な技術である検証と妥当性検査について，開発現場の技術者に 知識があまり無いからである．
ツールとマニュアルの章を追加したのは，これらの知識がないと実際の普及の上で大きな障害になることが分かったからである．ただし，本論文では紙幅の関係で4.マニュアルについては省略した．

1. アイデア

1.1 仕様記述についてのアイデア

「1. 仕様記述」では，次の観点で定石・手筋を整理した．すなわち，VDM仕様の特定の場所でよく使う記述方法を定石と呼び，VDM仕様のいろいろな場所でよく使う記述方法を手筋と呼ぶ．

定石は，事前条件，事後条件，不変条件でよく使う仕様記述方法であるので，下記のように分類した．

- 1.1.1 定石
  - 1.1.1 事前条件
  - 1.1.2 事後条件
  - 1.1.3 不変条件

1.1.1, 1.1.2, 1.1.3 は「規則」と「定石」を記述している．ここで，「規則」とはVDM++の文法規則のことである．例えば，関数の場合の事前条件式は，式中で局所名以外の名前はパラメータ名しか使えない．これも一種の「定石」ではあるので「規則」として記述した．
「1.1.3 不変条件」定石は，下記のように分類した．

- 1.1.3.1 型の不変条件
- 1.1.3.2 インスタンス変数の不変条件
  - 1.1.3.2.1 インスタンス変数の正当性
  - 1.1.3.2.2 インスタンス変数同士の制約

「1.1.3.1 型の不変条件」中には，例えば「1.1.3.1.1.1 データが意味的に正当か確認する」定石(IDは**JIT_wf_1**)があり，例えば以下のように，レコード型である予約明細型は予約明細を表しているの，属性**f座席集合**は空集合ではないという制約を記述している．この定石は，3節の「定石・手筋の評価」の結果，かなり使われていることが分かった．

```
types
public 予約明細 ::
  f顧客名 : 顧客名
  f列車番号 : 列車番号
  f乗車駅 : 駅番号
  f降車駅 : 駅番号
  f座席集合  : set of 座席番号
inv
```

```
w予約明細 == w予約明細.f座席集合 <> {};
```

「1.1.3.2.2 インスタンス変数同士の制約」中には，例えば「1.1.3.2.2.2 写像の定義域・値域[2]同士の部分集合関係を記述する」定石(IDはJII_ibV_subset2)があり，例えば以下のように，インスタンス変数同士の制約条件を記述する．

```
types
public 管制官 = token;
public 空域 = token;
public 航空機 = token;
public 当直集合 = set of 管制官;
public 管制 = inmap 空域 to 管制官;
public s位置 : 位置 := {|->};

instance variables
  public s当直集合 : 当直集合 := {};
  public s管制 : 管制 := {|->};
  public s許容数 : 許容数 := {|->};
  public s位置 : 位置 := {|->};

inv
   rng s管制 subset s当直集合 and
   dom s管制 subset dom s許容数 and
   rng s位置 subset dom s管制
```

ここでは，3つのインスタンス変数同士の部分集合関係を記述しているが，例えば，この内の3番目は，インスタンス変数「s位置」の値域「rng s位置」は，インスタンス変数「s管制」の定義域「dom s管制」の部分集合であるという制約を記述している．

もちろん，インスタンス変数同士の制約は「対象問題分野」の知識がなければ記述できないのであるが，定石に基づいて「写像型を使ったインスタンス変数の定義域や値域に部分集合関係はないか?」という疑問を持ち，調べることによって制約を発見して記述するきっかけになる．この定石も，3節の「定石・手筋の評価」の結果，使われていることが分かった．

手筋は，下記のように分類した．

1.2手筋
- 1.2.1 成功手筋
- 1.2.2 失敗手筋

1.2には手筋として成功手筋と，失敗手筋を記述した．本稿では，成功することが多い手筋を成功手筋と呼び，失敗することが確実，または失敗することが多い手筋を失敗手筋と呼ぶ．

この中には，定石と呼んでよいものもかなりあるが，仕様中のほとんどの個所で使えるパターンが多く，手筋とした．

以下の定石・手筋の説明では，紙幅の都合上ごく一部しか紹介していないが，全部で定石・手筋は129個ある．このため，定石や手筋を一意に識別するため，VDMの証明に関する教科書(文献9)の証明IDの命名法を参考にして，定石と手筋のIDを付けることにした．この命名法で付けられたID名は，人間に分かりやすい名前を短縮したものなので，人間が読むにはやや分かり難くなるが，証明IDや囲碁の定石名などと同様に，最終的に3千から2万個ほどになる可能性があるため，唯一性と検索容易性を優先した．

成功手筋は，さらに以下のように分類し，整理した．すなわち，

[2]写像の定義域・値域は，キーとデータの対応を持つ表だとイメージすればよい．キーの集合が定義域であり，データの集合が値域である．

1.2.1.1　物の集まりの型｛集合，写像，列｝を使う手筋
1.2.1.1.1集合を使う手筋
1.2.1.1.2写像を使う手筋
1.2.1.1.3列を使う手筋

さらに，集合，写像，列を使う手筋は，内包式[3]を使う手筋と，限量式[4]を使う手筋と，その他の手筋に分けた．

「1.2.1.1.2写像を使う手筋」の中には，「1.2.1.1.2.1.2 写像に要素を追加する」手筋(IDTSCMCR_Append_++)があり，写像の上書演算子++と写像の旧値を使用し，写像の旧値と追加する写（maplet）を写像化したもの(写を'｛'と'｝'で囲うと写像になる)を上書(定義域のキーが重複した場合，演算子++の右辺により上書き)した写像が，追加後の写像と等しいことを示している． この手筋に従えば，事後条件の記述と，操作本体の記述が容易になる．また，この定石は，3節の「定石・手筋の評価」の結果，かなり使われていることが分かった．

```
instance variables
public s許容数 : 許容数 := {|->};

public 委託する: 空域 * nat ==> 許容数
委託する(a空域, n) == (
    s許容数 := s許容数 ++ {a空域 |-> n};
    return s許容数
)
pre
   a空域 not in set dom s許容数
post
    s許容数 = s許容数~ ++ {a空域 |-> n};
```

失敗手筋は，以下のように分類した．

1.2.2.1 ブール演算で注意すべき失敗手筋
1.2.2.2 集合で注意すべき失敗手筋

「1.2.2.1 ブール演算で注意すべき失敗手筋」の中の「1.2.2.1.2 含意演算子と全称限量子 forall の組合せで注意すべき失敗手筋．」は「forall e in set S & false となる式で，S が空集合の場合は，forall 式は常に true になる．」(手筋 ID TFB_tautology)というものだが,VDM が使用している一階述語論理の日本語テキスト中で，筆者の知る限り,この定理に言及したものは無いため,VDM++仕様記述の初心者は必ず間違えるため手筋とした．

1.2 分析についてのアイデア

分析は，仕様を分析し欠陥を修正する．VDM++の分析で利用できるツールの機能は，内部矛盾がないことを検証する検証機能として組合せテストと証明課題生成があり，仕様が顧客の要求を満たしているかを検査するため回帰テスト機能がある．このため，次のように分類した．

2 分析
2.1 検証
2.1.1 組合せテスト
2.1.2 証明課題生成
2.2 妥当性検査

[3] 内包式はデータの範囲と条件を指定して物の集まりを作り出す式である．
[4] 限量式は，物の集まりのすべての要素が条件を満たすか，ある要素が条件を満たすかを判定する式である．

2.2.1 回帰テスト

例えば,「2.1.2 証明課題生成」の定石(ID Veri_PO)は, 教科書には記述されていないツールの証明課題生成結果をどう使うかを, VDMTools 設計者の Peter Gorm Larsen 教授にインタビューして得た結果を記述している.
すなわち,**「ツールで生成された証明課題は, VDM 本来の手法としては証明するのであるが, 産業界の技術者向けに作られた VDM のツール群は証明機能を支援していない. そのため, 生成された証明課題をレビューすることにより, 未記述の事前条件や不変条件を見つけ, 仕様の品質を高める. 」**

1.3 ツールについてのアイデア

VDM++を支援するツールは, Overture ToolsとVDMToolsとvdmjの3つあり, 各々の長所を組合せて使用すると効果的である. そこで, 以下のように分類した.

3. ツール
  3.1 Overture Tools と VDMTools と vdmj の協働
    3.1.1 ツールによるエラーの修正
      3.1.1.1 エラー特定の手筋
      3.1.1.2 テスト実行の手筋
      3.1.1.3 テスト結果の比較の手筋
    3.1.2 ツールによる回帰テスト
    3.1.3 ツールによる組合せテスト

例えば「3.1.1.2 テスト実行の手筋」(ID TU_EC_PE_VT)は,**「Overture Toolsは, エラーがあるとエラーの無い部分の実行もできない. VDMToolsは, 完成した部分からテスト実行できる. 構文エラーや型エラーがある時でも, エラーの無い実行可能な部分をテスト実行したい場合はVDMToolsを使う. 」**であり, アジャイル的に仕様を組み立てていく時はVDMToolsの方が有効であるが,「3.1.2 ツールによる回帰テスト」手筋の中の「3.1.2.1 ツールによる回帰テスト使い分けの手筋」(ID TU_RT_OT_VT)によると,**「Overture Toolsは比較的簡単に回帰テストケースを作成できるので, テストケースがさほど多くない回帰テストはOverture Toolsで行い, テストケースが非常に多い場合はVDMToolsを使う. Overture Toolsの回帰テストライブラリーは, VDM++の名前がtest(大文字も可)で始まる操作をテストケースとして自動的に探索し実行してくれるため, 回帰テストの行数が少なくて済む. ただし, Overture ToolsとVDMToolsの回帰テスト・ライブラリを切り替え, テストケース記述もVDMTools用に追加しなければならない. 一方, VDMToolsの方が, 大量の回帰テストケースを実行できた実績がある. 」**であり, アジャイル的な仕様作成が大体終わり, 回帰テストを使う最初の段階では**Overture Tools**を使う方が有効であり, 大量のテストケースによる回帰テストを行う時は**VDMTools**の方が安心であることが分かる.

1.4 マニュアルについてのアイデア

マニュアルの手筋の分類は以下のとおりである.

4.マニュアル
  4.1 Overture Tools のマニュアル
  4.2 言語仕様の要約版
  4.3 VDMTools のマニュアル
  4.4 vdmj のマニュアル

これらを要約すると,VDM++の初心者は「日本語マニュアルのため読みやすい」手筋「4.3 VDMTools のマニュアル」(ID MU_VT)を使い,実際に仕様を書く時は「4.2 言語仕様の要約版」(ID MU_OT_Quick)を参照すればよい.
VDM10 などの最新の情報を調べるには「4.1 Overture Tools のマニュアル」(ID MU_OT)が必要になり,「4.4 vdmj のマニュアル」は「GUI 版はなく,コマンド版のマニュアルであり,プロ向きの記述である」(ID MU_VJ)は,VDM++初心者には必要ないが,VDM++上級者には必要になることがある.

2. 主要な定石と手筋

1節で紹介した定石と手筋以外の、3節「定石・手筋の評価」で使われている主要な定石と手筋を,紙幅の許す範囲で紹介する.

1.2.1.1.2.1.3 写像から削除する手筋

種類による分類	式による分類	式に使える名前	**定石 | 手筋 | 規則 | 注意事項**}ID	{**定石 | 手筋 | 規則 | 注意事項**}ID**説明**
手筋	**<-: dom**	-	TSCMCR_Delete	

写像から要素を削除する手筋で，写像型の定義域削減演算子**<-:**と，削除する写像の定義域集合を得る演算子**dom**を使用して，写像の旧値から削除本の要素を削除したものが，削除後の写像と等しいことを示している。

例 *1.2.1.1.2.1.3.1:* 写像「*i*蔵書」の旧値から削除本の要素を削除したものが、削除後の写像*i*蔵書と等しいことを示している例。

```
types
public 蔵書 = map 蔵書ID to 本;
public 貸出 = inmap 利用者 to 蔵書;

instance variables
private i蔵書 : 蔵書 := {|->};
private i貸出 : 貸出 := {|->};
inv 蔵書に存在する(merge rng i貸出, i蔵書);

operations
public 蔵書を削除する: 蔵書 ==> ()
蔵書を削除する(a削除本) == is not yet specified
post i蔵書 = dom a削除本 <-: i蔵書~;
```

例 *1.2.1.1.2.1.3.2:* 写像「*s*許容数」の旧値から空域集合に対応する要素を削除したものが削除後の写像「*s*許容数」と等しいことを示している。

```
public 使用をやめる : 空域 ==> () --TSCMCR_Delete
使用をやめる(a空域) ==
   s許容数 := {a空域} <-: s許容数
pre a空域 in set (dom s許容数 ¥ dom s管制)
post s許容数 = {a空域} <-: s許容数~;
```

1.2.1.1.1.1 集合に要素を追加する手筋。

種類による分類	式による分類	式に使える名前	定石 | 手筋 | 規則 | 注意事項}ID	{定石 | 手筋 | 規則 | 注意事項}ID説明
手筋	union	旧値が使えるのは、事後条件式の中だけである。	TSCSCR_Append	

集合の合併演算子unionを使用してインスタンス変数に1要素を追加する手筋である

例 *1.2.1.1.1.1.1:* 集合の合併演算子unionを使用してインスタンス変数に*1*要素を追加する式の例

```
--操作本体で使う場合の例
i客DB := i客DB union {a新しい客}

–事後条件で使う場合の例
post i客DB = i客DB~ union {a新しい客}

public 出勤する : 管制官 ==> () --TSCSCR_Append
出勤する(a管制官) ==
    s当直集合 := s当直集合 union {a管制官}
pre a管制官 not in set s当直集合
post s当直集合 = s当直集合~ union {a管制官};
```

1.2.1.1.1.2 集合から要素を削除する手筋

種類による分類	式による分類	式に使える名前	定石 | 手筋 | 規則 | 注意事項}ID	{定石 | 手筋 | 規則 | 注意事項}ID説明
手筋	¥(集合の差)	旧値が使えるのは、事後条件式の中だけである。	TSCSCR_Delete	

集合の差演算子¥と集合の旧値を使用し，旧値と削除する要素(を集合化したもの)の差が集合の値と等しいことを示した例である。

例 *1.2.1.1.1.1.2:* 集合の差演算子¥と集合の旧値を使用し、旧値と削除する要素(を集合化したもの)の差が集合の値と等しいことを示した例である。

```
--操作本体で使う場合の例
i客DB := i客DB ¥ {a削除する客}

–事後条件で使う場合の例
post i客DB = i客DB~ ¥ {a削除する客};

public 退勤する : 管制官 ==> () --TSCSCR_Delete
退勤する(a管制官) == s当直集合 := s当直集合 ¥ {a管制官}
pre a管制官 in set s当直集合 ¥ rng s管制
post s当直集合 = s当直集合~ ¥ {a管制官}
```

3. 定石・手筋集の評価

ここでは，本定石・手筋集がどの程度有効であるかを，定石と手筋の収集に使っていなかった参考文献5と6を使って評価した．

定石と手筋	文献5 化学プラント仕様	文献6 航空管制システム 最上位仕様
定石・手筋使用率	100%	100%
写像に上書演算子を使って追加 TSCMCR_Append_++		6
写像から定義域削減演算子<-: を使って削除 TSCMCR_Delete		3
集合の合併演算子unionを使って追加 TSCSCR_Append		1
集合の差演算子¥を使って削除 TSCSCR_Delete		1
データが意味的に正当か確認 JIT_wf_1	3	
写像のドメイン同士に部分集合関係がある JII_ibV_subset2	1	
集合内に存在することを確認 TSCSQ_exsists_set	3	

表6. 定石・手筋集の評価

上記2つのVDM++仕様については，すべての不変条件，事後条件，事前条件が，本定石・手筋集に記述されたものであった．また，全部で19個所ある定石と手筋の適用個所で使われている定石と手筋はわずか7種類であり，今後より多くのモデルのデータを集めることにより，主要な定石と手筋を特定できる可能性が強いことが分かった．

当然のことながら,集合型を中心とした仕様記述は集合に関する手筋を使い,写像型を中心とした仕様記述は写像と集合に関する手筋を多用するので,成功手筋の中で集合と写像と列[5]で手筋を分類したのは有効だった．

これらのVDM++仕様は定石と手筋の使用率が高いものを選んだわけではなく，たまたま最初に評価した2つのVDM++仕様だった．

現在のデータ数では，まだ「本定石・手筋集の有効性が高い」と言い切ることはできないが，一応定石と手筋集は役立ちそうだといえる．

4. 関連研究

文献 1. 2.など,関連ありそうな情報を探したが,今のところ囲碁や将棋やチェスの定石と手筋という観点から「定石と手筋集」を作成・整理しようという関連研究は無かった．

[5] 列型を使った仕様はまだ評価していないが,過去の経験から,確実に有効であろうと考えている．

文献 3. 4. 6.7.などの VDM の証明に関する教科書は,本研究のヒントになる情報は多いが,VDM 仕様の証明が主題であり，仕様記述の際に活用しようというものではない．文献 12.も,囲碁などの棋譜や音楽の楽譜からの発想で，証明譜を使い,それを証明に活用しようというものであり,やはり仕様記述の際のヒントにしようというものではない.

5. 結論と追加研究

産業界でのVDM++仕様記述で一番困難な，不変条件，事後条件，事前条件のパターンを定石・手筋集として試作した.

本定石・手筋集の有効性はある程度確認できたので，今後，実用的なモデルの定石・手筋使用率をさらに調べるていく予定である．調査対象モデルとしては，文献5の原子力追跡機モデル，筆者のカレンダー・ライブラリ， 文献9のNetwork Security Policy Modelなどを考えている.

VDMの証明についての教科書である文献7と文献9は，不変条件，事後条件，事前条件例を見つけ出すためにかなり役立つことが分かったので，重点的に調べていく予定である.

定石・手筋から仕様の骨格を生成でき,ツールに組み込んでいくことも可能である.例えば,Overture ToolsのEdit Template機能(文献 13.参照)で,下記のような「写像に要素を追加する．」手筋(ID TSCMCR_Append_++)用の”appendpp”という名前のテンプレートを記述しておくと,

```
types
public ${TypeName} = ${TypeName0};

instance variables
public ${instancevariable} : ${TypeName} := ${statement};

operations
public ${operationname}: ${argTypes} ==> ${resultType}
${operationname}(${arg1}, ${arg2}) == (
        ${instancevariable} := ${instancevariable} ++ {${arg1} |-> ${arg2}};
        return ${instancevariable}
)
pre
        ${arg1} not in set dom ${instancevariable}
post
        ${instancevariable} = ${instancevariable}~ ++ {${arg1} |-> ${arg2}};
post
        ${instancevariable} = ${instancevariable}~ ++ {${arg1} |-> ${arg2}};
```

エディター画面でappendppとタイプしてからコントロール・キーとスペース・キーを同時に押すことにより,テンプレートのテキストが挿入され,エディターで${instancevariable}部分を**s許容数**に変えると,すべての${instancevariable}が**s許容数**に変わり,下記のようになる.

```
types
public TypeName = TypeName0;

instance variables
public s許容数 : TypeName := statement;

operations
public operationname: argTypes ==> resultType
operationname(arg1, arg2) == (
        s許容数 := s許容数 ++ {arg1 |-> arg2};
```

```
        return s許容数
)
pre
        arg1 not in set dom s許容数
post
        s許容数 = s許容数~ ++ {arg1 |-> arg2};
```

作成した手筋テンプレートはまだ1つであるが,今後,徐々に追加していく予定である.

なお, 現在コンサルティング中のプロジェクトで本定石・手筋集の一部を教育し, 実プロジェクトで試行していく予定である.

参考文献

1. JÖRG ACKERMANN, FORMAL DESCRIPTION OF OCL SPECIFICATION PATTERNS FOR BEHAVIORAL SPECIFICATION OF SOFTWARE COMPONENTS, 2005
2. ANDREW IRELAND AND BILL J. ELLIS AND TOMMY INGULFSEN, INVARIANT PATTERNS FOR PROGRAM REASONING, 2004
3. CLIFF B JONES, CASE STUDIES IN SYSTEMATIC SOFTWARE DEVELOPMENT SECOND EDITION, 1990
4. CLIFF B JONES, SYSTEMATIC SOFTWARE DEVELOPMENT USING VDM, SECOND EDITION , 1990
5. ジョン・フィッツジェラルド他著, 酒匂寛 訳, VDM++によるオブジェクト指向システムの高品質設計と検証, 2010
6. JIM WOODCOCK, MARTIN LOOMES, SOFTWARE ENGINEERING MATHEMATICS: FORMAL METHOD DEMYSTIFIED, 2007
7. JUAN C. BICARREGUI, JOHN FITZGERALD), PETER A. LINDSAY, RICHARD MOORE, BRIAN RITCHIE, PROOF IN VDM: A PRACTITIONER'S GUIDE, 1995
8. 佐原伸, IPA, 対象を如何にモデル化するか?, 2013
9. JUAN C. BICARREGUI (ED.) PROOF IN VDM: CASE STUDIES, 1998
10. VDM — THE VIENNA DEVELOPMENT METHOD, ANDREAS MULLER, APRIL 20, 2009
11. DANIEL JACKSON. SOFTWARE ABSTRACTION : LOGIC, LANGUAGE, ANALYSIS . THE MIT PRESS, CAMBRIDGE, 2006
12. Kokichi FUTATSUGI1, Joseph A. GOGUEN2, and Kazuhiro OGATA: Verifying Design with Proof Scores, 2008
13. PETER GORM LARSEN, KENNETH LAUSDAHL, PETER TRAN-JØRGENSEN, JOEY COLEMAN, SUNE WOLFF, LU´IS DIOGO COUTO AND VICTOR BANDUR: OVERTURE VDM-10 TOOL SUPPORT: USER GUIDE, VERSION 2.4.6, 2017