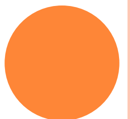
テスト駆動開発を利用した形式仕様記述

2014年02月28日

NECソフト株式会社　伊藤 淳

目次

- VDMの活用において
  経験した失敗と解決方法
- 提案手法の説明
- 結果と考察
- まとめと今後の課題



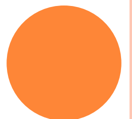
VDMの活用において経験した失敗と解決方法

- 誤りに気が付かず，仕様を書き続けてしまった

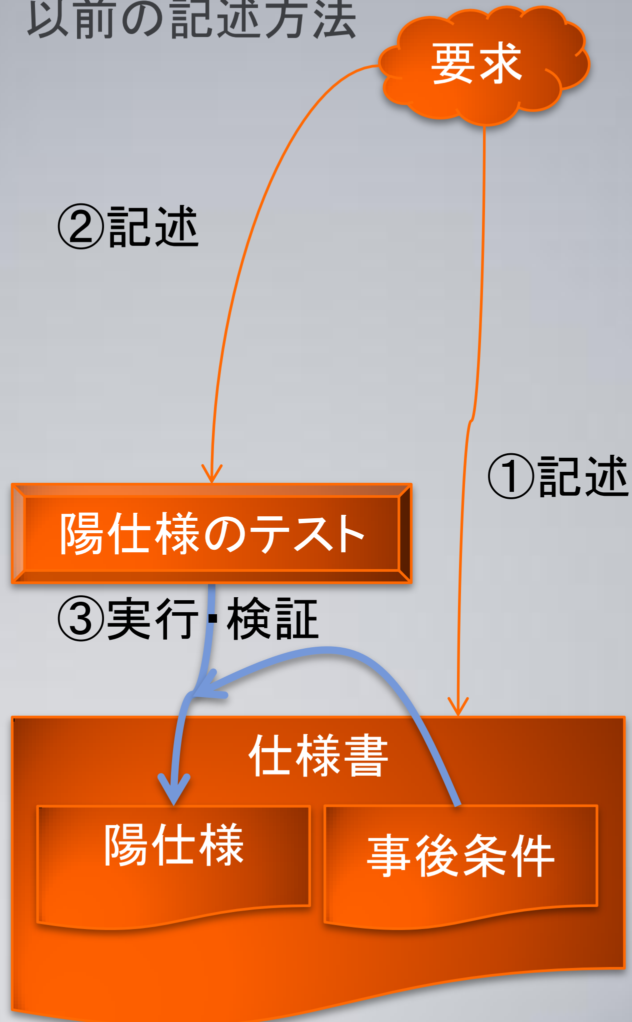
仕様を小さい機能単位で記述・検証し，
現状の範囲内で誤りを検出しよう！

- 事後条件の記述が十分でなく，誤検出や見落としがあった

事後条件の十分性をテストで確認しよう！



テスト駆動開発を利用した形式仕様記述（概念図）



提案手法

テスト駆動開発を利用した形式仕様記述（概念図）

以前の記述方法

要求

②記述

①記述

陽仕様のテスト

③実行・検証

仕様書

陽仕様

事後条件

やるべきことを書きだす

提案手法

要求

①作成

テストリスト

テスト駆動開発を利用した形式仕様記述（概念図）

以前の記述方法

要求

②記述

①記述

陽仕様のテスト

③実行・検証

仕様書

陽仕様

事後条件

提案手法

やるべきことをはきだす

要求

①作成

テストリスト

②記述

陽仕様のテスト

「**誤検出しないか？**」をテストするテストケースをVDMで記述する

テスト駆動開発を利用した形式仕様記述（概念図）

以前の記述方法

要求

②記述

「誤検出しないか？」をテストするテストケースをVDMで記述する

①記述

陽仕様のテスト

③実行・検証

仕様書

陽仕様

事後条件

提案手法

やるべきことを書きだす

要求

①作成

テストリスト

②記述

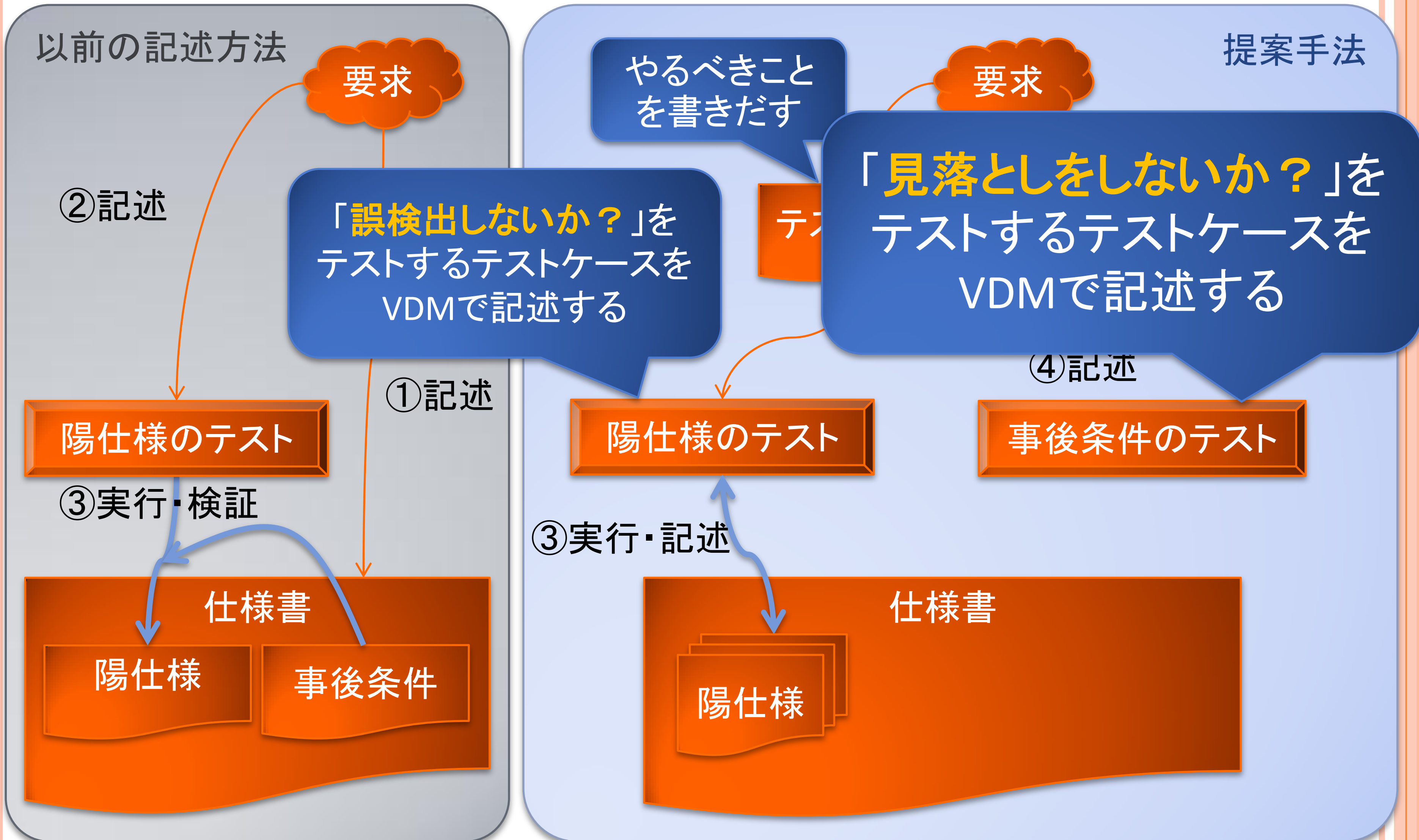
陽仕様のテスト

③実行・記述

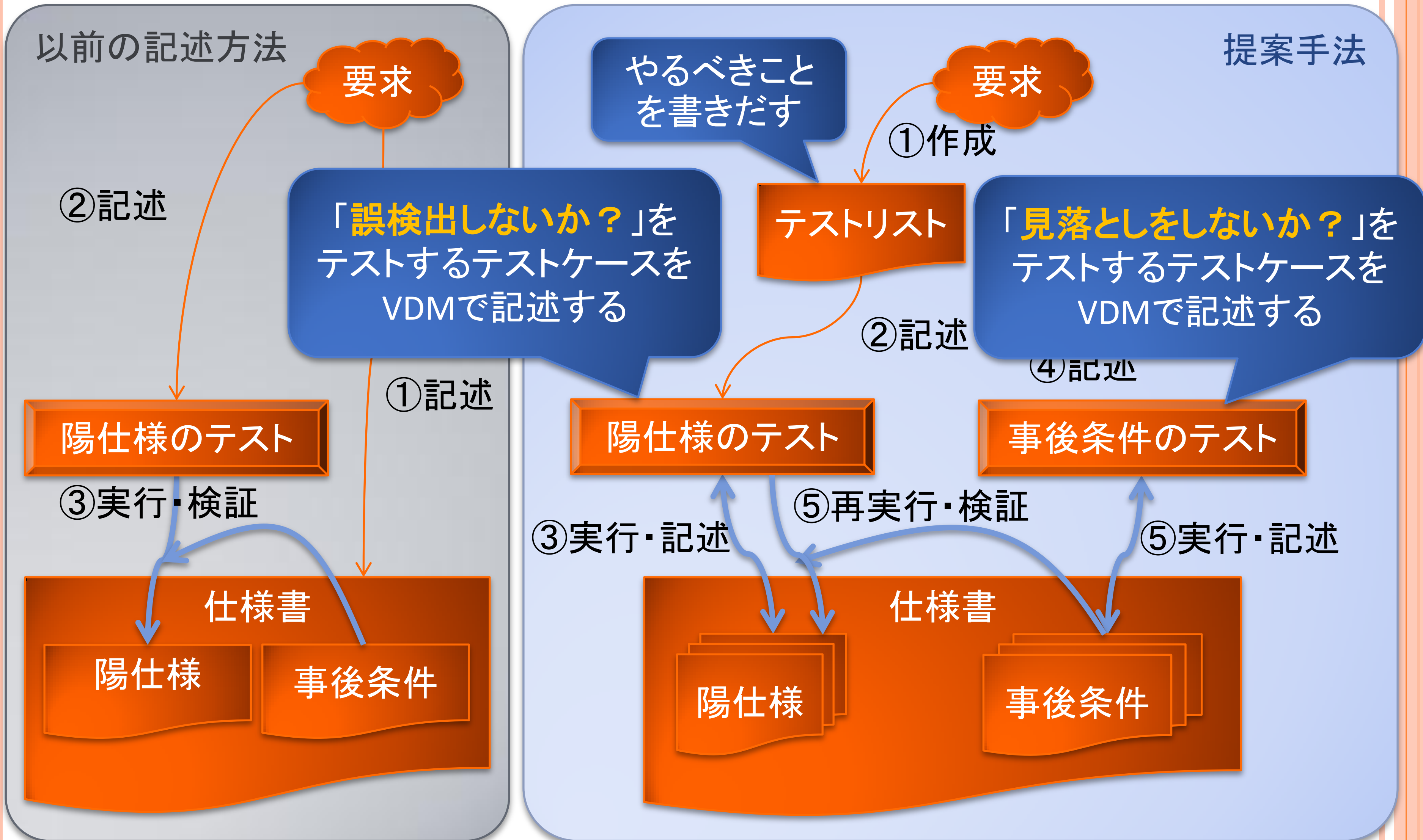
仕様書

陽仕様

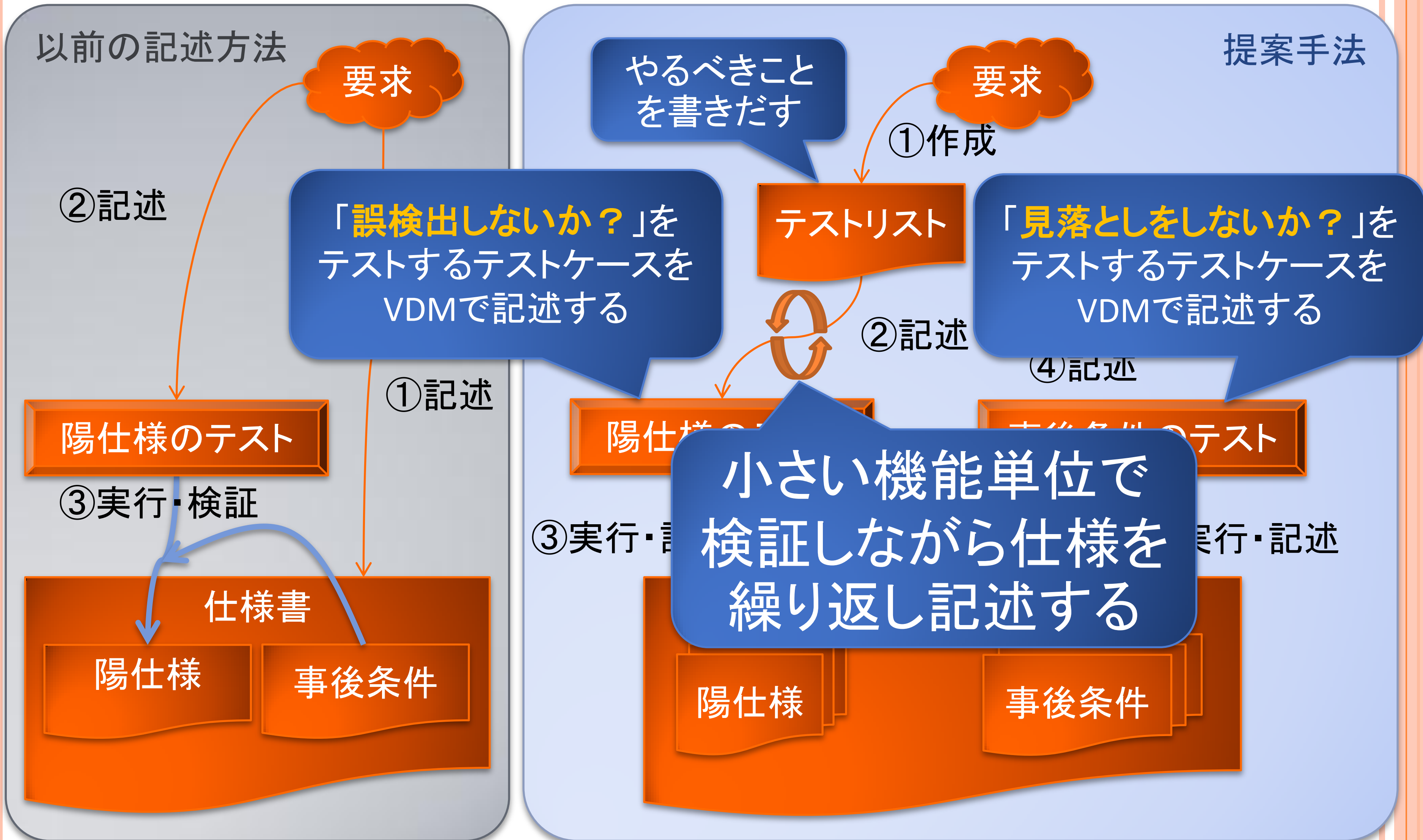
テスト駆動開発を利用した形式仕様記述（概念図）



テスト駆動開発を利用した形式仕様記述（概念図）

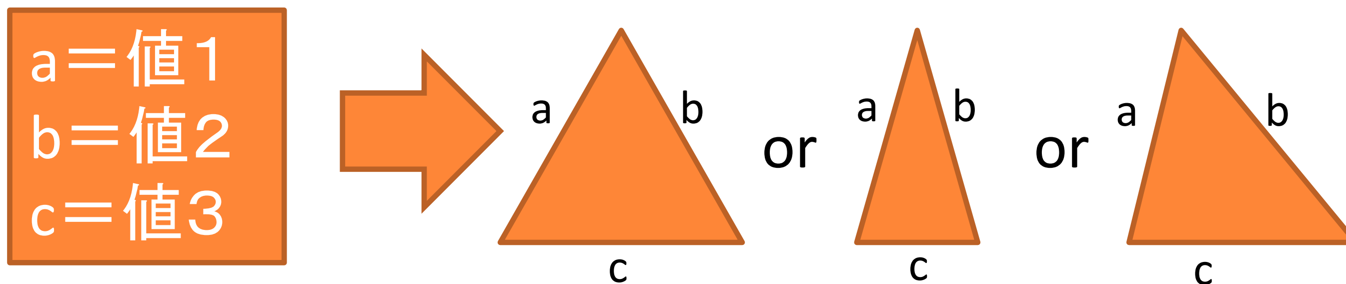


テスト駆動開発を利用した形式仕様記述（概念図）



例題を用いた提案手法の説明

- マイヤーズの三角形問題
  - 3つの値が与えられたとき，それが正三角形，二等辺三角形，不等辺三角形のうちどれであるかを判断する



出典：http://www.kindaikagaku.co.jp/information/kd0329.htm

『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
);
```

3つの辺が全て「1」のとき，正三角形と判定されることを期待する

TDD

Red → Green

凡例

テストケース

陰仕様・陽仕様

『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
);
```

「種類を判定する」操作が存在しないためエラー・・・つまり

TDD

Red → Green

凡例

テストケース

陰仕様・陽仕様

『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
);
```

存在しない操作を記述すればよい

```
public 種類を判定する : int * int * int ==> 種類
種類を判定する(a辺1,a辺2,a辺3) == (
  is not yet specified
);
```

操作の中身(陽仕様)はまだ考えず，
定義のみを記述する

TDD

Red ⇨ Green

凡例

テストケース

陰仕様・陽仕様

『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
);
```

```
public 種類を判定する : int * int * int ==> 種類
```

```
public 種類を判定する : int * int * int ==> 種類
種類を判定する(a辺1,a辺2,a辺3)== (
  if (a辺1 = 1 and a辺2 = 1 and a辺3 = 1) then (
    return <正三角形>;
  )
  return <三角形でない>;
);
```

最低限動作するように「具体例」で陽仕様を記述する

TDD

Red → Green

凡例

- テストケース
- 陰仕様・陽仕様

『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
  assertTrue(i三角形.種類を判定する(2,2,2) = <正三角形>);
);
```

“3つの辺が全て「2」”という同じ意味のテストを追加し，テストケースが失敗することを確認する

```
p
種
)
                                                    種類
種類を判定する(a辺1,a辺2,a辺3)== (
  if (a辺1 = 1 and a辺2 = 1 and a辺3 = 1) then (
    return <正三角形>;
  )
  return <三角形でない>;
);
```

TDD

Red ➡ Green

凡例

テストケース

陰仕様・陽仕様

『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
  assertTrue(i三角形.種類を判定する(2,2,2) = <正三角形>);
);
```

```
public 種類を判定する : int * int * int ==> 種類
種類を判定する(a辺1,a辺2,a辺3)== (
   if (a辺1 = a辺2 and a辺2 = a辺3) then (
     return <正三角形>;
   )
```

どちらのテストも動作するように「具体例」を変数に置き換えて一般化する

TDD

Clean

Red Green

凡例

テストケース

陰仕様・陽仕様

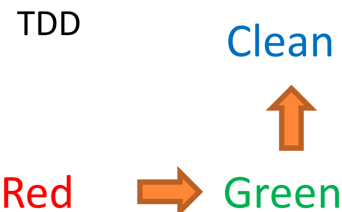
『“三辺の長さが等しいとき正三角形である”ことを判定する仕様』をどう書くか？

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
```

```
public 三辺の長さが等しいとき正三角形である : () ==> ()
三辺の長さが等しいとき正三角形である() == (
  assertTrue(i三角形.種類を判定する(1,1,1) = <正三角形>);
```

一時的に追加したテストや汎用的でない部分を整理してクリーンにする

```
public 種類を判定する : int * int * int ==> 種類
種類を判定する(a辺1,a辺2,a辺3)== (
   dcl w種類 : 種類 := <三角形でない>;
   if (a辺1 = a辺2 and a辺2 = a辺3) then (
    w種類 := <正三角形>;
   )
   return w種類;
);
```



凡例

- テストケース
- 陰仕様･陽仕様

最終的に完成した形式仕様記述（の一部）

陽仕様

```
public 種類を判定する : int * int * int ==> 種類
  種類を判定する(a辺1,a辺2,a辺3)== (
    dcl 結果 : 種類 := <三角形でない>;
    if (a辺1 + a辺2 <= a辺3 or a辺1 + a辺3 <= a辺2 or a辺2 + a辺3 <= a辺1) then (
     結果 := <三角形でない>;
    ) else if (a辺1 = a辺2 and a辺2 = a辺3) then (
     結果 := <正三角形>;
    ) else if (a辺1 = a辺2 or a辺2 = a辺3 or a辺1 = a辺3) then (
     結果 := <二等辺三角形>;
    ) else (
     結果 := <不等辺三角形>;
    );
    return 結果;
  )
pre 三辺の長さはすべて0より大きいこと(a辺1,a辺2,a辺3)
post ある二辺の長さの和が他の一辺の長さ以下のとき三角形でないと判定される(a辺1,a辺2,a辺3,RESULT) and
     三辺の長さが等しいときのみ正三角形と判定される(a辺1,a辺2,a辺3,RESULT) and
    二辺の長さが等しいときのみ二等辺三角形と判定される(a辺1,a辺2,a辺3,RESULT) and
    三辺の長さが異なるときのみ不等辺三角形と判定される(a辺1,a辺2,a辺3,RESULT);
```

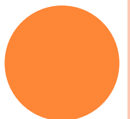
事前条件

事後条件

テストケースをヒントに仕様を記述することができ，誤りを即座に修正することができた

目次

- 形式仕様記述における課題
- 提案手法の説明
- **結果と考察**
- まとめと今後の課題



結果と考察(1/2)
テスト駆動開発の利用に対する効果

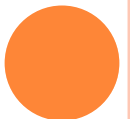
- テストファーストで記述することで，作成すべき仕様のヒントを得ながら記述できた
- 仕様を小さい機能単位で記述し，修正するたびに回帰テストをすることで，素早く仕様の誤りを発見できた
- 事後条件に対しても，テストファーストで記述することで，一般式への書き換えが容易にできた

品質を確保しながら，かつ，ヒントを得ながら仕様を記述できるようになった

結果と考察（2/2）
事後条件に対するテストを追加したことによる効果

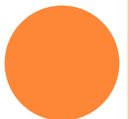
- 事後条件に対して，陽仕様のテストとは「逆の視点」でテストをすることによって，これまで見逃していた事後条件の誤りを発見できるようになった
- レビューだけでは自信が持てなかった事後条件の正しさを，テストによって確認できるようになった

事後条件の正しさを確認する手段を，手順として確立できた



目次

- 形式仕様記述における課題
- 提案手法の説明
- 結果と考察
- **まとめと今後の課題**



まとめと今後の課題

まとめ

- テスト駆動開発を利用することで，具体例をヒントにして記述ができ，さらに事後条件をテストすることで見逃しがちだった誤りを検出しながら仕様を記述できる手法を提案した
- なお，本手法の徐々に仕様を記述する特長は，形式仕様記述（VDM++）初学者に対して，理解を促す効果が期待でき，学習教材として利用できると考えている

今後の課題

- QCD（学習コスト，実装コスト，品質）に関して，本手法の有効性を検証すること
- 学習教材としての利用価値の検証をすること

以上，ご清聴ありがとうございました