Case Studies on Combination of VDM and Test-Driven Approaches: Application, Model Finding and Refinement

Fuyuki Ishikawa
National Institute of Informatics, Japan

First of All

Thanks to the Overture community!

Today in my role as an educator for the industry:

- Top SE program at NII since 2007

  (reported in the 7th WS, and today in FMSEET'15)

  - Every year 20-35 persons (among 40) take the VDM class (as the starting point of FM)
  - A few of them choose VDM for their 3 or 6 month in-depth studies (e.g., a case study with Crescendo)
- SQiP program since 2011
  - Every year several engineers use VDM for 9 month studies and discussions on specification

Today's Topic

- Connection between VDM and Test-Driven Development (TDD)
  - Well, TDD and its extensions (such as BDD, ATDD) are somewhat popular for a certain community of engineers (the agile camp)
  - In the discussions with engineers/researchers, some of them tried to investigate the connections
- This talk reports three case studies
  - Application of TDD to VDM
  - Model finding for TDD and VDM
  - Refinement by Example

TOC

- Introduction to TDD
- Application of TDD to VDM
- Model Finding for VDM and TDD (or Testing)
- Refinement by Example

How TDD Works

Sample: a classical problem to judge a triangle type, equilateral, isosceles, scalene and non-triangle (Myers)

(Initial) Check List (TODOs)

- Can judge equilateral
- Can judge isosceles
- Can judge scalene
- …

How TDD Works

Sample: a classical problem to judge a triangle type, equilateral, isosceles, scalene and non-triangle (Myers)

Check List (TODOs)

- Can judge equilateral
- Can judge isosceles
- Can judge scalene
- …

Choose a simple one and write a test case

```
TestEquilateral : () ==> ()
TestEquilateral() ==
assertTrue(judge(5,5,5)=<EQUI>);
```

Note: usually target program code, here explained using the VDM syntax

How TDD Works

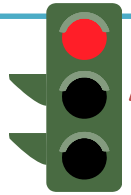
Sample: a classical problem to judge a triangle type, equilateral, isosceles, scalene and non-triangle (Myers)

Check List (TODOs)

- Can judge equilateral
- Can judge isosceles
- Can judge scalene
- …

Choose a simple one and write a test case

```
TestEquilateral : () ==> ()
TestEquilateral() ==
assertTrue(judge(5,5,5)=<EQUI>);
```



Fail

How TDD Works

Sample: a classical problem to judge a triangle type, equilateral, isosceles, scalene and non-triangle (Myers)

Check List (TODOs)

- Can judge equilateral
- Can judge isosceles
- Can judge scalene
- …

Choose a simple one and write a test case

```
TestEquilateral : () ==> ()
TestEquilateral() ==
assertTrue(judge(5,5,5)=<EQUI>);
```

Write code to pass

```
types
TType = <EQUI>;
```

```
functions
judgeTriangle : int*int*int -> TType
judgeTriangle(a, b, c) ==
  return <EQUI>;
```

How TDD Works

Sample: a classical problem to judge a triangle type, equilateral, isosceles, scalene and non-triangle (Myers)

Check List (TODOs)

- ~~Can judge equilateral~~
- Can judge isosceles
- Can judge scalene
- …

Choose a simple one and write a test case

```
TestEquilateral : () ==> ()
TestEquilateral() ==
assertTrue(judge(5,5,5)=<EQUI>);
```

Pass

Write code to pass

```
types
TType = <EQUI>;
```

```
functions
judgeTriangle : int*int*int -> TType
judgeTriangle(a, b, c) ==
  return <EQUI>;
```

How TDD Works

Repeat the small step

Sample: a classical problem to judge a triangle type, equilateral, isosceles, scalene and non-triangle (Myers)

Check List (TODOs)

- ~~Can judge equilateral~~
- Can judge isosceles (case a=b)
- Can judge scalene
- …

May evolve

Choose a simple one and write a test case

```
TestIsosceles1 : () ==> ()
TestIsosceles1 ==
assertTrue(judge(4,4,2)=<ISO>);
```

Pass

Write code to pass

```
types
TType =
<EQUI> | <ISO>;
```

```
judgeTriangle(a, b, c) ==
  if a = b and b <> c then
    return <ISO>
  else return <EQUI>;
```

Generalize by triangulation

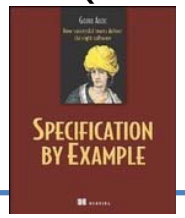
(Part of) Why TDD?

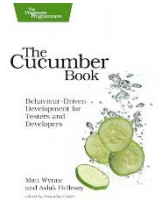
- "Clean code that works" is often too difficult
  (in the triangle problem I pretended to think so)
  - Let's start with "code that works"
  - "Design for clean code" usually does not work with (mysteriously) some of the test cases
- Test what you wrote quickly
  - Writing long code without test accumulates faults, leading to hard debug and rollback
- Test cases give confidence as examples
  - It is difficult to have confidence on validity only with declarative, general descriptions

cf. BDD and ATDD

- BDD (Behavior-Driven Development)
  ATDD (Acceptance Test-Driven Development)
  - Start with test cases for high-level TODOs (outside-in, not starting with the unit testing level)
  - Link human-readable text/graphical description to executable test cases

```
Scenario: A woman over 15 years old can marry in Japan
  Given I have entered false into the system (asked is male?)
  And I have entered 16 into the system
  When I press “can marry?”
  Then the result should be true
```





➧ *Tests as Documents & Specification by Example*

TOC

- Introduction to TDD
- Application of TDD to VDM
- Model Finding for VDM and TDD (or Testing)
- Refinement by Example

Application of TDD to VDM

- No reason to deny similar application of TDD for (executable specification of) VDM, if you want
  - Except for your feeling of doubts, and the difficulty to prove real effectiveness (e.g., [Erdogmus, TSE05])
- A case study with an engineer
  - He had a feeling that it is good for the first step of VDM learning (anyway run, check asap)
  - A similar way of application worked
  - One additional point is: we should test on pre- and post-condition functions

Notable Experiences

- Value of testing and testing in small steps

```
post
  a=b and b=c <=> ¥RESULT=<EQUI>
  a<>b and b<>c <=> ¥RESULT=<SCA>
  a=b and b<>c or b=c and c<>a or
    c=a and a<>b <=> ¥RESULT=<ISO>
  ...
```

```
assertTrue(
  judge(5,3,5)=<ISO>
)
```

Post-cond fails

- Test cases to check “post-conditions deny wrong result”, or find false-positive check by weak post-conditions

```
post
  ¥result=<SCA> => a<>b and b<>c
```

```
assertFalse(
  post_judge
  (5,3,5,<SCA>)
)
```

Return <SCA> from the explicit part for
the input (5,3,5) is accepted by this post-condition SILENTLY

TOC

- Introduction to TDD
- Application of TDD to VDM
- **Model Finding for VDM and TDD (or Testing)**
- Refinement by Example

Model Finding

- TDD (or just testing) relies on “good” test cases
  - Test design, or specification of test cases, matters
    e.g., at least one case for (a=b and b<>c)
- What we want to do is validation among specification (properties), test-design and test cases (examples)
- Prototyped a “Spec-Test-Go-Round” tool:
  - Language to mix implicit specification, test design and test case descriptions (on VDM or Java)
  - Tool to generate test cases by a constraint solver (current impl. Is the Java version)

Simple Example

- First give precondition

```
pre a > 0 and b > 0 and c > 0
```

Cases with valid inputs

a	b	c
4	2	1 [LowerB]
6	6	6
3	12	2
1 [LowerB]	1 [LowerB]	7

Cases with invalid inputs

a	b	c
-5 [Under]	2	1 [LowerB]
5	-1 [Under]	0 [UnderB]
0 [UnderB]	0 [UnderB]	2
12	12	-30 [Under]

Simple Example

- Add weak/partial postcondition

```
post a = b and b = c => ¥result = <EQUI>
```

Cases with valid inputs and correct outputs

a	b	c	¥result	Case Prop.
8	8	8	EQUI	
1 [LowerB]	3	3	NON	
6	6	1	SCA	
3	5	7	EQUI	

Cases with valid inputs and incorrect outputs

a	b	c	¥result	Case Prop.
5	5	5	NON	
3	3	3	SCA	
1 [LowerB]	1 [LowerB]	1 [LowerB]	SCA	
10	10	10	ISO	

Simple Example

- Add a specific test case (or example)

```
case ¥"ex-equi“
    a=5 and b=5 and c=5 and ¥result=EQUI
```

Cases with valid inputs and correct outputs

a	b	c	¥result	Case Prop.
5	5	5	EQUI	[ex-equi]
1 [LowerB]	3	3	NON	
6	6	1	SCA	
12	12	12	EQUI	
3	5	7	EQUI	

(detected if inconsistent with the specification)

Simple Example

- Add test designs

missing c <> a

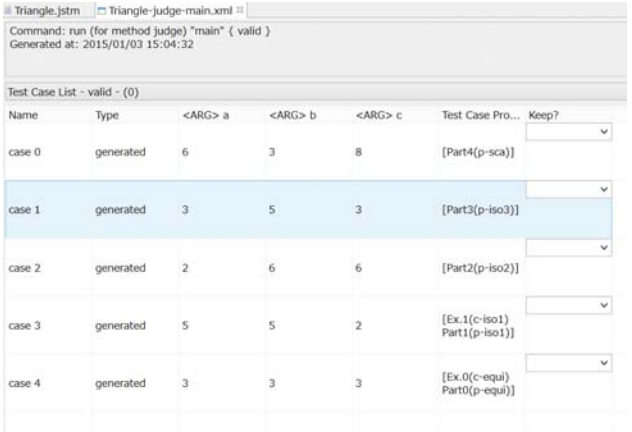
```
partition{
    ¥"p-equi" a=b && b=c, ¥"p-sca" a<>b && b<>c,
    ¥"p-iso1" a=b && b<>c, ¥"p-iso2" b=c && c<>a,
    ¥"p-iso3" c=a && a<>b
}
```

Cases with valid inputs and correct outputs

a	b	c	¥result	Case Prop.
5	5	5	EQUI	[ex-equi, p-equi]
5	3	4	SCA	[ex-sca, p-sca]
4	4	2	ISO	[p-iso1]
1	10	10	EQUI	[p-iso2]
5	2	5	NON	[p-iso3,p-sca]
3	6	6	NON	[p-iso2]

Prototype and Seminar

- Other functions/usages (details omitted)
  - Validation of test cases with test designs
  - Automated insertions of test designs :
    e.g., make A true in (A => B),
    use all of Boolean or enum values, etc.
- The first prototype simply based on Alloy Analyzer (with symbolic encoding heuristics)
- Experiment through one-day seminars with 60 people in total (most from the industry)


Triangle.jstm Triangle-judge-main.xml

Command: run (for method judge) "main" { valid }
Generated at: 2015/01/03 15:04:32

Test Case List - valid - (0)

Name	Type	<ARG> a	<ARG> b	<ARG> c	Test Case Pro...	Keep?
case 0	generated	6	3	8	[Part4(p-sca)]	
case 1	generated	3	5	3	[Part3(p-iso3)]	
case 2	generated	2	6	6	[Part2(p-iso2)]	
case 3	generated	5	5	2	[Ex.1(c-iso1) Part1(p-iso1)]	
case 4	generated	3	3	3	[Ex.0(c-equi) Part0(p-equi)]	

Questionnaire Results (1)

- Advantages (multiple choices allowed)

Enable to incorporate principles from TDD, formal methods, test design	22
Enable to run the tool as soon as some description is given	22
Make easier use of formal methods and solvers	17
Provide opportunities to enlarge and learn viewpoints	16
Is a general-purpose tool to support various tasks to some extent	14
Enable lightweight usages such as partially automated test design	14

Questionnaire Results (2)

- Effective Usages (multiple choices allowed)

Education and enhancement of understanding and awareness to enlarge insights of mid-level engineers	15
Management and discussion on test cases in TDD	14
Introduction and education of foundations for beginners	12
Clarification and validation of constraints in domain analysis or specification construction	11
Assistance of test design for quality assurance	10
Clarification and validation of logic regardless of the task	9
Machine-readable standard comment formats on program code	7
Formalization and validation before using existing formal methods	7
Easy use of solvers on complex problems	4

TOC

- Introduction to TDD
- Application of TDD to VDM
- Model Finding for VDM and TDD (or Testing)
- **Refinement by Example**

Simplistic? View on Refinement

I think we should make a more systematic way for validation of the relationship between an abstract model and its next model

Abstract Model

Concrete Model

Simplistic? View on Refinement

I think we should make a more systematic way for validation of the relationship between an abstract model and its next model

Abstract Model

Concrete Model

We have refinement!

Simplistic? View on Refinement

For example …

Scenario Increment

Behavior only for success cases

Behavior including fault handling

Abstract Model

Concrete Model

Simplistic? View on Refinement

For example …

Scenario Increment

Behavior only for success cases → *Behavior including fault handling*

Abstract Model	Concrete Model

Nondeterministic Success/Failure → *Conditional Branch*

Removal of Non-Deteminism

This is the rule of Event-B!

Simplistic? View on Refinement

Abstract Model

Concrete Model

$I = P(x_{in\_a\_1}, \cdots, x_{in\_a\_m}, x_{in\_c\_1}, \cdots, x_{in\_c\_n})$

You connect the models by link/gluing invariants

Simplistic? View on Refinement

Are the internal representation in the models ("mock-up" or "ghost" variables) really central and should we make effort on them?

Abstract Model

Concrete Model

$$I = P(x_{in\_a\_1}, \cdots, x_{in\_a\_m}, x_{in\_c\_1}, \cdots, x_{in\_c\_n})$$

You connect the models by link/gluing invariants

Simplistic? View on Refinement

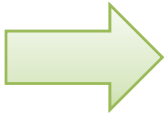
- User-defined rules?
  - At the cost of losing correctness-by-construction (which actually most developers do not have now)
- Definition by test cases on observable behavior?
  - Do not constraint the internal variables
- Refinement by Example?
  - Use test cases that give confidence on inheritance of the essences, (as in Specification by Example)
  - Allow custom mapping rules between test cases for the two models
- A case study (not formalized into a framework)

Simple Example

```
types
authenticator = token;

operations
public login :
  token ==> ()
login(authenticator)
  == ...;
```

Conceptual System Model
(Use Case Level Interface)



```
types
Authenticator ::
  username : seq of char
  password : seq of char;

operations
public login :
 Authenticator ==> ()
login(authenticator)
  == ...;
```

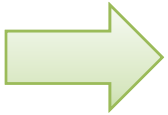
Design Model
(Design Level Interface)

Simple Example

```
types
authenticator = token;

operations
public login :
  token ==> ()
login(authenticator)
  == ...;
```

Conceptual System Model
(Use Case Level Interface)



```
types
Authenticator ::
  username : seq of char
  password : seq of char;

operations
public login :
 Authenticator ==> ()
login(authenticator)
  == ...;
```

Design Model
(Design Level Interface)

```
test1A()==
  ...
  login(mk_token("tom"));
  ...
```

```
test1A()==
  ...
  login(mk_Authenticator
          ("tom", "tompwd"));
```

Define matching rule for the test cases
(e.g., assertion over arguments in each line pair of invocation of the corresponding methods)

Summary

- 3 studies to discuss TDD and VDM together
  - Run by curiosity and ideas of engineers
  - More focus on what practitioners are discussing, often in (superficially) different wording

Can help in application of VDM,
as well as active community discussions

- Test (for V&V or as examples) is one of the keys factors in VDM
- People enjoy and have more confidence if they can link the discussions to their camps, trend words, etc.

Overture Community

In our FM foundation lecture using VDM,

- After 1 year: I can give demos or tooling ideas about of essential topics in FM with Overture-based tools
  - Connecting with stakeholders, model checking, etc.
  - Currently with other tools and abstract guidelines
- After 5 year: I can let students try such features only with Overture-based tools
- After 10 year: I can let students try some valuable functions I cannot imagine now

Always: I can continue including a "emerging topics" session every year