36. Story Driven Modeling –
A Practical Guide to Model Driven Software Development

Courtesy to Prof. Albert Zündorf, University of Kassel, Germany, Given in Dresden in 2005
Content

1. Overview
2. The running example: Ludo
3. Use case description
4. Object oriented analysis with story boards
5. Test derivation
6. Derivation of design and implementation
7. Validation
Fujaba Graph Rewriting Tool

- http://www.fujaba.de/
- http://www.fujaba.de/no_cache/publications.html
Overview

Story Driven Modeling:

Steps:
- Textual use case description
- Story Boarding (OOA) (Test specification)
- Class diagram derivation (OOD)
- Behavior derivation (Coding)
- Code generation
- Validation (Testing)

Features:
- Use Case Driven
- Model Driven
- Iterative
- Test Driven Development
36.2. The running example: Ludo
36.3. Use case diagrams (Rpt.)

Requirements elicitation as usual:

- Use case diagrams for overview
Use case description (cont.)

Textual scenario descriptions:

- focus on scenarios
- several scenarios per use case
- focus on one example situation at a time
- use concrete names

Use case __________, __________:

Start situation: __________________
______________________________
______________________________

Invocation: __________________

Step 1: __________________
______________________________

Step 2: __________________
______________________________

Result situation: __________
______________________________
Story-Driven Modeling (SDM)

SDM approach:
- analyse the text scenarios
- nouns become *objects*
- verbs become *method invocations or links*
- ...

Start Situation: __________
----------------------------------
____________
____________
____________
____________
____________
____________
____________
____________
____________
____________
36.4 Object oriented analysis with story boards

- use case execution is modeled by one method invocation
- drawn as collaboration message
- multiple scenarios for one use case call the same method (but in different situations)
- this method implements the use case
- use case ↔ method mapping enables tracebility
- step descriptions may become implementation comments

Actor step 1: _ _ _ _ _ _ _ _ _ _ _

1: _ _ _ _ _ _ _ _ _ _ _
Object oriented analysis with story boards

use case ↔ method mapping

- uc1 <<uses>> uc2 → method uc1() may call method uc2()
- uc1 <<includes>> uc2 → uc1() always calls uc2()
- uc2 <<extends>> uc1 → uc1() provides extension points / call backs. uc2() may subscribe for such a call back
Object oriented analysis with story boards

- <<create>> and <<destroy>> markers
- := attribute assignments
- recurring objects without class name
  first time on stage with class name (change of perspective)
- collaboration messages
- alternatively sequence diagrams
Object oriented analysis with story boards

Result situation:
- models resulting object structure
- used for testing

Result Situation: _______________________________________________

```
| theDie |
v == 0  |
```

```
| tom    |
die     |
counters|
counter1|
at      |
| f4     |
```
Derivation of Class Diagrams

Collect the types from the story boards:

- Classes
- Associations
- Attribute declarations
- Method declarations
Derivation of Class Diagrams (cont.)

- Class diagram derivation is straight forward
- Semi-automatic tool support by Fujaba
- Intermediate story board step results in much better domain level class diagrams
- code generation for class diagrams
- *story boards are appropriate for the analysis and discussion of behavior*
- story boards also useful during refinement and coding
- story boards may serve as test specifications
- story boards may drive the implementation
36.5. Test Derivation

- Scenarios \(\rightarrow\) JUnit Tests
- start situation \(\rightarrow\) setup code
- invocation \(\rightarrow\) invocation
- result situation \(\rightarrow\) code that checks object structure equivalence
Test Derivation (cont.)

- Scenarios → JUnit Tests, start situation → setup code and fixture

Start Situation: Tom rolled a 3 and selects counter 1 for moving

```
tom : Player
die

theDie : Die
v == 3

: TestMoveUsual
```

```
class TestMoveUsual implements TestCase {
    private Player tom;
    private Die theDie;
    private Counter counter1;
...
    void setUp () {
        tom = new Player ();
        theDie = new Die ();
        theDie.setV (3)
        tom.setDie (theDie);
        counter1 = new Counter ();
        tom.addToCounters (counter1);
    ...  
}
```
Test Derivation (cont.2)

- Scenarios → JUnit Tests, start situation → setup code

Invocation: counter 1 is moved

```
  class TestMoveUsual implements TestCase {
    ...
    void testMoveUsual () {
      this.counter1.move();
      ...
  }
```

1: move ()
Test Derivation (cont.3)

- Scenarios → JUnit Tests, start situation → setup code

Result Situation: the die is counted down to zero and counter 1 reached field 4

```
class TestMoveUsual implements TestCase
{
    void testMoveUsual ()
    {
        this.counter1.move();
        assertTrue (tom.getDie() == theDie);
        assertTrue (theDie.getV() == 0);
        assertTrue (counter1.getPlayer () == tom);
        assertTrue (counter.getAt () == f4);
    }
}
```
Test Derivation (cont.4)

- more complex result situations work, too (see later)
- start situation, invocation, result situation $\rightarrow$ JUnit tests
- steps may be exploited, too, cf. [SCESM05]
- analysis scenarios $\leftrightarrow$ tests
- test driven software development
Derivation of the Implementation

- Dobs + BeanShell + Coding

Tool Demo

or

- combine story boards to rule diagrams [SCESM04]
- assign execution semantics
- code generation
Derivation of the Implementation (cont.)

Main Memory Objects

- **ludo:Game**
  - phase == "move"

- **theDie:Die**
  - value == 2

- **tom:Player**
  - players current
  - players

- **albert:Player**
  - players current

- **c1:Counter**
  - 1: move()
  - next
  - at

- **pos3:Position**
  - at
  - next

- **pos4:Position**
  - next

- **pos5:Position**
  - next

- **pos6:Position**

Rule Diagram / Program

- **myPlayer**
  - counters
  - next
  - at

- **theDie**
  - value > 0

- **otherPlayer:Player**
  - counters

- **otherCounter:Counter**
  - counters

- **pos:Position**
  - at "destroy"
Derivation of the Implementation (cont.2)

Main Memory Objects

- **ludo:Game**
  - `phase == "move"`
- **theDie:Die**
  - `value == 1`
- **tom:Player**
- **albert:Player**
- **c1:Counter**
  - `move()`
- **c2:Counter**
- **pos3:Position**
  - `at`
- **pos5:Position**
  - `next`
- **pos6:Position**
  - `at`
- **pos4:Position**
  - `next`

Rule Diagram / Program

- **Counter:move 0: Void**
- **myPlayer**
- **theDie**
  - `value > 0`
  - `value := value + 1`

Fujaba Tutorial  
Story Driven Modeling  
© 2005 Albert Zündorf, University of Kassel
Derivation of the Implementation (cont.3)

Main Memory Objects

- ludo : Game
  - phase == "move"
  - die
  - players current
  - players

- theDie : Die
  - value == 0
  - die

- tom : Player
  - players current
  - players

- albert : Player
  - players current

- c1 : Counter
  - 1 : move()
  - counters

- pos3 : Position
  - at
  - next

- pos6 : Position
  - at
  - next

- pos4 : Position
  - next

- pos5 : Position
  - next

Rule Diagram / Program

- this
  - counters
  - die

- myPlayer
  - counters
  - die

- theDie
  - counters

- pos
  - at
  - next
  - destroy

- pos : Position
  - at
  - destroy

- otherPlayer : Player
  - counters

- otherCounter : Counter
  - counters

© 2005 Albert Zündorf, University of Kassel
Derivation of the Implementation (cont.4)

Main Memory Objects

- **ludo:** Game
  - phase == "move"

- **theDie:** Die
  - value == 0

- **tom:** Player

- **albert:** Player

- **c1:** Counter
  - 1: move()

- **pos3:** Position
  - next

- **pos6:** Position
  - next

- **pos4:** Position

Rule Diagram / Program

- **this**
  - counters
  - at

- **pos**
  - next

- **myPlayer:** Player

- **otherPlayer:** Player

- **otherCounter:** Counter

- **theDie:** Die
  - value > 0
  - value := value - 1

- **Counter:** move 0: Void
  - counters
  - at
Derivation of the Implementation (cont.5)

Main Memory Objects

Rule Diagram / Program

Main Memory Objects

Rule Diagram / Program
class Counter {
    public void move () { Position pos; . . .
        while (sdmSuccess) {
            try {
                sdmSuccess = false;
                pos = this.getAt ();
                JavaSDM.ensure (pos != null);
                next = pos.getNext ();
                JavaSDM.ensure (next != null);
                myPlayer = this.getOwner ();
                JavaSDM.ensure (myPlayer != null);
                theDie = myPlayer.getDie ();
                JavaSDM.ensure (theDie != null);
                JavaSDM.ensure (theDie.getV() > 0);
                sdmSuccess = true;
                this.setAt (null);
                this.setAt (next);
                theDie.setV(theDie.getV() - 1); }
            catch (SDMException e) {} // while
    } // public void move ()
Derivation of the Implementation (cont. 7)

Game::collectThrownCounters () : Void

```
1: collectThrownCounters()

this

ludo::

players

homes

albert::Player

c1::Counter

noPos

p1::Position

looser

lostCounter

noPos

p3::Position
```

© 2005 Albert Zündorf, University of Kassel
class Game {
public void collectThrownCounters () { . . .
Iterator looserIter = this.iteratorOfPlayers();
while (!sdmSuccess && looserIter.hasNext()) {
    try {
        sdmSuccess = false;
        looser = looserIter.next ();
lhome = looser.getHome ();
JavaSDM.ensure (lhome != null);
countersIter = looser.iteratorOfCounters ();
while (!sdmSuccess && countersIter.hasNext()) {
        try {
            lostCounter = countersIter.next ();
JavaSDM.ensure (lostCounter.getAt() == null);
        sdmSuccess = true;
lostCounter.setAt (lhome);
} catch (SDMException e) {} 
} // while
} catch (SDMException e) {}
} // while
Derivation of the Implementation (cont.9)

- manual derivation of rule diagrams from stories
- brain required
- systematic guide lines provided e.g. in [SCESM04]
- automatic code generation [GraGra]
Summary

Story Driven Modeling

- model level analysis with story boards
- model level tests
- model level implementation with rule diagrams
- code generation
- model level testing / debugging

www.fujaba.de  zuendorf@uni-kassel.de
Addendum A: Story Pattern Elements:

**Variables:**

\[ v : \text{Class} \]
\[ v : \text{Class} \]
\[ v : \text{Class} \]
\[ v : \text{Class} \]
\[ v : \text{Class} \]
\[ v : \text{Class} \]

**Attributes:**

<table>
<thead>
<tr>
<th>attr (&lt;\text{cmpOp}&gt;) value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr := value</td>
</tr>
</tbody>
</table>

**Constraints:**

\[ \{ \text{boolExpr} \} \]

**Links:**

- assoc
- assoc[<key>]
- ref
- 1: m()
- 2 [x>0]: m()
- 2.1 [while x>0]: m()
References


- [SCESM05] Leif Geiger, Albert Zündorf: *Story Driven Testing*; in proc. 4th International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'05) ICSE 2005 Workshop