Model-Driven Co-Evolution of Contracts, Unit-Tests and Source-Code

Proposal for Master’s Thesis – Advisors: Max Kramer, Michael Langhammer, Erik Burger
Supervisors: Prof. Dr. Ralf Reussner, Jun.-Prof. Dr.-Ing. Anne Koziolek

Stephan Seifermann | May 12, 2014
public void fund(int amount) {
    ...
    balance += amount - getFundFee();
    ...
}
Motivation

/*@
@ requires amount >= getFundFee();
@ ensures getBalance() == old(getBalance()) +
    amount - getFundFee();
@*/

public void fund(int amount) {
    ...
    balance += amount - getFundFee();
    ...
}
Motivation

```java
/*@ 
 @ requires amount >= getFundFee();
 @ ensures getBalance() == \old(getBalance()) +
  amount - getFundFee();

 @*/

public void fund(int amount) {
    ...
    balance += amount - getFundFee();
    ...
}
```

```java
@Test
public void testFund() {
    testInstance.fund(10);
    assertEquals(10 - BankAccount.getFundFee(),
                 testInstance.getBalance());
}
```
Motivation

/*@ 
@ requires amount >= getFundFee();
@ ensures getBalance() == \old(getBalance()) + 
  amount - getFundFee();
@*/

public void fund(int amount) {
  ...  
  balance += amount - getFundFee() + getFundBonus();
  ...  
}

@Test
public void testFund() {
  testInstance.fund(10);
  assertEquals(10 - BankAccount.getFundFee(),
              testInstance.getBalance());
}
Motivation

/*@ 
@ requires amount >= getFundFee();
@ ensures getBalance() == \old(getBalance()) +
    amount - getFundFee() + getFundBonus();
@*/

public void fund(int amount) {
    ...
    balance += amount - getFundFee() + getFundBonus();
    ...
}

@Test
public void testFund() {
    testInstance.fund(10);
    assertEquals(10 - BankAccount.getFundFee(),
                 testInstance.getBalance());
}
Motivation

```java
/*@ 
@ requires amount >= getFundFee();
@ ensures getBalance() == \old(getBalance()) + 
    amount - getFundFee() + getFundBonus();
@*/

public void fund(int amount) {
    ...
    balance += amount - getFundFee() + getFundBonus();
    ...
}

@Test
public void testFund() {
    testInstance.fund(10);
    assertEquals(10 - BankAccount.getFundFee() +
        BankAccount.getFundBonus(), testInstance.getBalance());
}
```
PIBA

Problem
- Code, contracts and tests have semantic overlap
- Overlap has to be kept consistent manually

Idea
- Keep part of overlap consistent using model transformations

Benefit
- Less effort for developer
- Support for roundtrip engineering

Action
- Specify overlap and relations
- Combine existing approaches and bridge gaps
- Evaluate using open source projects
Foundations

Model-Driven Engineering

Models are primary artefacts of development process

Co-Evolution

Changes of implementation of code, tests and contracts together

Contracts

Obligations of user and provider of functionality (preconditions, postconditions, invariants)
Foundations

Model-Driven Engineering
Models are primary artefacts of development process

Co-Evolution
Changes of implementation of code, tests and contracts together

Contracts
Obligations of user and provider of functionality (preconditions, postconditions, invariants)
Foundations

Model-Driven Engineering
Models are primary artefacts of development process

Co-Evolution
Changes of implementation of code, tests and contracts together

Contracts
Obligations of user and provider of functionality (preconditions, postconditions, invariants)
Related Work

Relation between Contracts and Code

- Goldstein et al. 2006: Automated adjustment of contracts after refactorings
- Feldman et al. 2006: Inferring contracts from code
- Hull 2010: Effects on code after Push Down / Pull Up refactorings on contracts

Problems

- One way approaches
- No public tools except for Hull 2010
Related Work

Relation between Contracts and Code

- Goldstein et al. 2006: Automated adjustment of contracts after refactorings
- Feldman et al. 2006: Inferring contracts from code
- Hull 2010: Effects on code after Push Down / Pull Up refactorings on contracts

Problems

- One way approaches
- No public tools except for Hull 2010
Related Work

Relation between Contracts and Tests
- Cheon et al. 2002: Use contracts as test oracles
- Zimmerman et al. 2011: Constructing test data via tests

Problems
- One way approaches
- Unfeasible amount of test cases
Related Work

Relation between Contracts and Tests

- Cheon et al. 2002: Use contracts as test oracles
- Zimmerman et al. 2011: Constructing test data via tests

Problems

- One way approaches
- Unfeasible amount of test cases
private void stopListeningIfTwoPlayersConnected() {
    synchronized (playersMonitor) {
        if (moreThanOnePlayerConnected()) {
            LOGGER.info("Stopping listening.");
            stopListening();
        }
    }
}
private void stopListeningIfTwoPlayersConnected() {
    synchronized (playersMonitor) {
        if (moreThanOnePlayerConnected()) {
            LOGGER.info("Stopping listening.");
            stopListening();
        }
    }
}

JUnit Code

@Test
public void receiveRejectAfterJoin() throws IOException {
    for (Socket s : p) {
        BasicCommandSending.sendJoin(s);
        validatePendingReject(s);
    }
}

Java Code

@requires amount >= getFundFee();
@ensures getBalance() == old(getBalance()) + amount - getFundFee();
@public abstract void fund(int amount);

JML Annotations

Legend

←→ bidirectional synchronisation
private void stopListeningIfTwoPlayersConnected() {
    synchronized (playersMonitor) {
        if (moreThanOnePlayerConnected()) {
            LOGGER.info("Stopping listening.");
            stopListening();
        }
    }
}
Evaluation

- Case study using open source projects
- Possible candidates: JAVA CARD API and KOA\(^1\)
- Perform supported changes

Goals

- Syntax is still correct (checked by compiler of Java and JML)
- Modified models are equal to expected ones

Expected results:

- Goals are fulfilled
- Semi-automated changes might miss the goals

\(^1\)Kiezen op Afstand (means Remote-Voting)
## Realisation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.4</td>
<td>20.4</td>
<td>27.4</td>
<td>4.5</td>
<td>11.5</td>
<td>18.5</td>
<td>25.5</td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>8.6</td>
<td>15.6</td>
<td>22.6</td>
<td>29.6</td>
<td>6.7</td>
<td>13.7</td>
</tr>
<tr>
<td>3</td>
<td>20.7</td>
<td>27.7</td>
<td>3.8</td>
<td>10.8</td>
<td>17.8</td>
<td>24.8</td>
<td>31.8</td>
</tr>
<tr>
<td>4</td>
<td>7.9</td>
<td>14.9</td>
<td>21.9</td>
<td>28.9</td>
<td>5.10</td>
<td>12.10</td>
<td>19.10</td>
</tr>
<tr>
<td>5</td>
<td>26.10</td>
<td>2.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 1: Proposal
- 2: Preparation
- 3: Code->Tests
- 4: Code->Contracts
- 5: Code Review
- 6: Tests->Contracts
- 7: Contracts->Code
- 8: Case Study
- 9: Writing

---

Stephan Seifermann – Model-Driven Co-Evolution of Contracts, Unit-Tests and Source-Code

Motivation | PIBA | Foundations | Approach | Evaluation | Realisation | Summary
---|---|---|---|---|---|---
| | | | | | May 12, 2014 | 9/10
## Realisation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.4</td>
<td>20.4</td>
<td>27.4</td>
<td>4.5</td>
<td>11.5</td>
<td>18.5</td>
<td>25.5</td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>8.6</td>
<td>15.6</td>
<td>22.6</td>
<td>29.6</td>
<td>6.7</td>
<td>13.7</td>
</tr>
<tr>
<td>3</td>
<td>20.7</td>
<td>27.7</td>
<td>3.8</td>
<td>10.8</td>
<td>17.8</td>
<td>24.8</td>
<td>31.8</td>
</tr>
<tr>
<td>4</td>
<td>7.9</td>
<td>14.9</td>
<td>21.9</td>
<td>28.9</td>
<td>5.10</td>
<td>12.10</td>
<td>19.10</td>
</tr>
<tr>
<td>5</td>
<td>26.10</td>
<td>2.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **Proposal**
2. **Preparation**
3. **Code->Tests**
4. **Code->Contracts**
5. **Code Review**
6. **Tests->Contracts**
7. **Contracts->Code**
8. **Case Study**
9. **Writing**

Motivation | PIBA | Foundations | Approach | Evaluation | Realisation | Summary
---|---|---|---|---|---|---
Stephan Seifermann – Model-Driven Co-Evolution of Contracts, Unit-Tests and Source-Code | May 12, 2014 | 9/10
## Realisation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13.4</td>
<td>20.4</td>
<td>27.4</td>
<td>4.5</td>
<td>11.5</td>
<td>18.5</td>
<td>25.5</td>
</tr>
<tr>
<td></td>
<td>1.6</td>
<td>8.6</td>
<td>15.6</td>
<td>22.6</td>
<td>29.6</td>
<td>6.7</td>
<td>13.7</td>
</tr>
<tr>
<td></td>
<td>20.7</td>
<td>27.7</td>
<td>3.8</td>
<td>10.8</td>
<td>17.8</td>
<td>24.8</td>
<td>31.8</td>
</tr>
<tr>
<td>1</td>
<td>7.9</td>
<td>14.9</td>
<td>21.9</td>
<td>28.9</td>
<td>5.10</td>
<td>12.10</td>
<td>19.10</td>
</tr>
<tr>
<td></td>
<td>26.10</td>
<td>2.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Proposal
2. Preparation
3. Code->Tests
4. Code->Contracts
5. Code Review
6. Tests->Contracts
7. Contracts->Code
8. Case Study
9. Writing
### Motivation

Stephan Seifermann – Model-Driven Co-Evolution of Contracts, Unit-Tests and Source-Code

### PIBA

- Foundations
- Approach
- Evaluation

### Realisation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.4</td>
<td>20.4</td>
<td>27.4</td>
<td>4.5</td>
<td>11.5</td>
<td>18.5</td>
<td>25.5</td>
</tr>
<tr>
<td>2</td>
<td>1.6</td>
<td>8.6</td>
<td>15.6</td>
<td>22.6</td>
<td>29.6</td>
<td>6.7</td>
<td>13.7</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Code-&gt;Tests</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Code-&gt;Contracts</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Code Review</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Tests-&gt;Contracts</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contracts-&gt;Code</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Case Study</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Writing</td>
<td></td>
</tr>
</tbody>
</table>

### Summary

May 12, 2014
Summary

Problem
- Code, contracts and tests have semantic overlap
- Overlap has to be kept consistent manually

Idea
- Keep part of overlap consistent using model transformations

Benefit
- Less effort for developer
- Support for roundtrip engineering

Action
- Specify overlap and relations
- Combine existing approaches and bridge gaps
- Evaluate using open source projects
What are the advantages of model transformations?

- existing techniques like JaMoPP can be used
- easier to use than working with text/code
- more modular (core stays the same, only model printers/parsers are changed)
Why do you use JML instead of XYZ?

- quite popular (more or less)
- projects for case study exist
- many tools and approaches based on it
- syntax easy to understand (for common constructs)
What about C4J?

- **Pros**
  - contracts are code
  - good supported by Eclipse and a plugin
  - automated refactoring support

- **Cons**
  - almost unknown
  - no projects for case study
  - no documentation / related work
  - automated refactoring support not working for final classes
References I


