



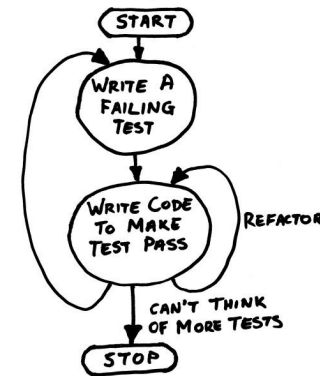
## Test-Driven Development of Model Transformations

Timo Kehrer, Sven Wenzel

Software Engineering Group  
University of Siegen, Germany  
{kehrer,wenzel}@informatik.uni-siegen.de

NW-MoDE 2009, Tampere, 28.8.2009

## Test-driven development



### Key concepts [1]

- Short development iterations (often only a few minutes)
- Pre-written test cases
- Each iteration produces code necessary to pass the test cases

### TDD requires little overhead for

- the specification of test data,
- implementing test stubs,
- test initiation,
- executing test cases,
- the interpretation of test results,
- etc.

Sven Wenzel

Test-Driven Development of Model Transformations

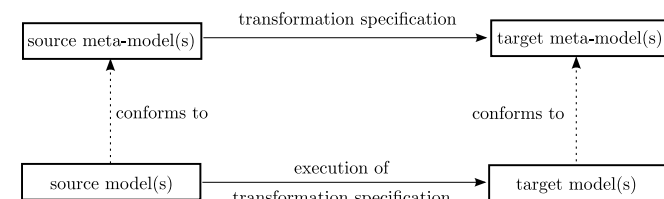
## TDD in code-centric development

- Well-established development technique in code-centric development
- Became popular with agile process models such as *extreme programming* [2]
- Typical subjects to unit testing are modules or classes
- Their behavior is tested by calling the declared functions or methods
- Test cases are typically target/actual-comparisons
- Functions or methods most often can be tested almost independently from each other with little effort

⇒ Feasibility of **short iteration cycles** and **incremental development**

⇒ **TDD works!**

## Model transformations



### A model transformation

- takes input models conforming to the source meta model(s),
- produces output models conforming to the target meta model(s),
- is specified through a set of different transformation rules [3].

## Testing model transformations

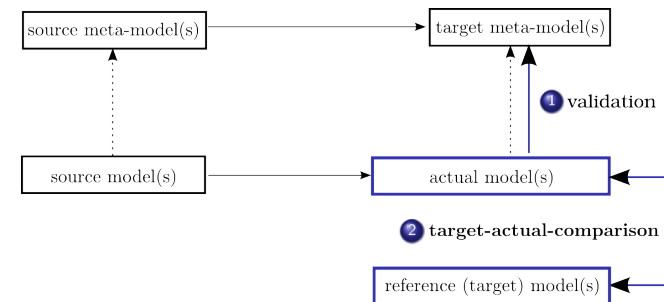
When testing transformations, we have to check:

- 1 **Syntactical Correctness:** Does the transformation produce models which are syntactically correct (i.e. they conform to the target meta model)?
- 2 **Semantical Correctness:** Is the transformation semantically correct in the sense that the produced result satisfies the developer's intention?

Possible solution ideas:

- 1 **Validation** against target meta-model (including constraints, i.e. static semantics)
- 2 Writing test cases for transformation rules in a **target/actual-comparison** style (according to conventional unit testing)

## Testing model transformations



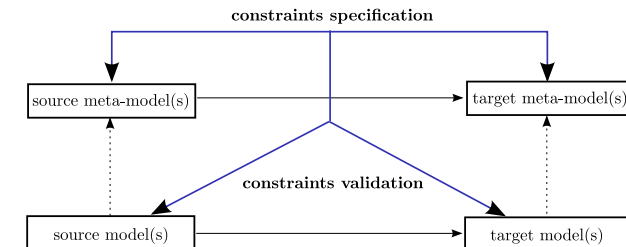
## Problems with the conventional unit testing approach

- 1 Collaboration of transformation rules
  - Testing transformation rules independently of each other requires a lot of effort for the specification of test data and the implementation of test stubs.
- 2 Graph characteristics of models
  - The manual specification of models or model fragments as input data and reference data most often is very tedious.
- 3 Difficulties with model comparison
  - target/actual-comparison of reference and output models is not a trivial problem [4]

Consequences

- Conventional unit-testing of transformation rules comes with enormous effort
- Test-driven development becomes challenging, as it requires short iteration cycles

## Key idea of our approach

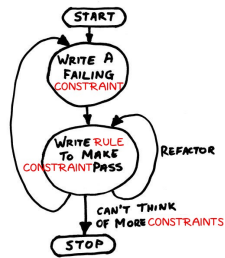


Abstain from the classical target/actual comparison

- 1 Use input models from real scenarios
- 2 Execute the transformation
- 3 Check a set of predefined constraints on the relationships of elements from the source to the target model.

Usage of declarative languages such as the OCL [5] to specify constraints.

# Incremental, test-driven development

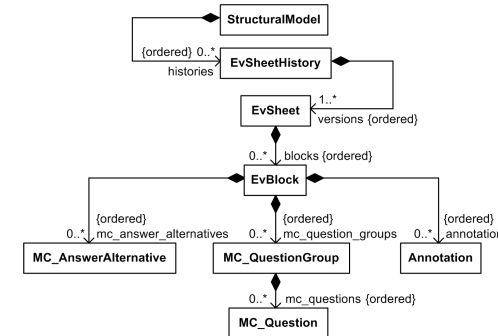


- ### TDD of model transformations
- Define constraints
    - on the target model
    - on the relationships of elements from the source to the target model.
  - Execute the transformation with input models from real scenarios.
  - Implement the parts of the transformation, which are necessary to fulfill the pre-defined constraints
  - Incrementally increase the set of constraints.

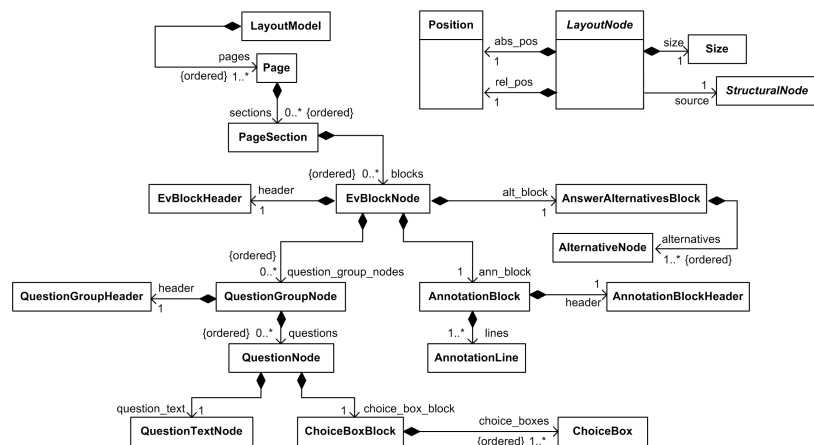
- Thereby, follow the composition relationships of the meta-model, i.e. proceed in a top-down manner starting from the spanning tree's root  
E.g. in case of class models, we start with packages, process classes and interfaces afterwards, before we continue with associations, attributes and operations, and so on.

# Example Application: ScanScore

- System to support the evaluation of study courses [6]
- Users can define an evaluation survey by creating the structure in an EMF model [7]
- The system automates the graphical rendering of evaluation sheets (e.g. PDF, HTML)



# Target: Layout meta-model



# Sample test case (1)

- The root node of the source model of type *StructuralModel* must have a matching root node in the target model of type *LayoutModel*.

### Test case specification:

```

context StructuralModel inv:
  LayoutModel.allInstances()->one(LayoutNode n|n.source==self);
  
```

- Subsequently, a transformation rule realizing the according mapping can be implemented to satisfy the test case.
- Similarly, we can continue with all children in the spanning tree.

## Sample test case (2)

- The number of alternatives rendered in the layout has to be equal to the number of answer alternatives defined in the conceptual structure (i.e. the evaluation block representing the source of the transformation).

### Test case specification:

```
context EvBlockNode inv:  
  self.alt_block.alternatives->size() ==  
  ((EvBlock)source).mc_answer.alternatives->size();
```

## Conclusions

- TDD is well-established in code-centric development
- Application to model transformations so far hampered by
  - tedious specification of reference models
  - complex target/actual-comparison

### Our approach

- Abstain from target/actual-comparisons
- Use models from real scenarios
- Define constraints that need to be fulfilled by transformations

Questions?

Thank you for your attention!      Kiitos!

## Sample test case (3)

- The absolute position of any layout node has to be equal to the sum of its parent's absolute position and its own relative position.

### Test case specification:

```
context LayoutNode inv:  
  abs_pos.x ==((LayoutNode)eContainer).abs_pos.x + rel_pos.x &&  
  abs_pos.y ==((LayoutNode)eContainer).abs_pos.y + rel_pos.y;
```

## References

- [1] Beck, K.  
Test-Driven Development by Example  
Addison Wesley, 2003
- [2] Beck, K., Andres, C.  
Extreme Programming Explained: Embrace Change (XP)  
Addison Wesley, 2004
- [3] Czarnecki K., Helsen S.  
Feature-model-based survey of model transformation approaches  
In IBM Systems Journal archive 45(3), 2006
- [4] C. Treude, S. Berlik, S. Wenzel, and U. Kelter  
Difference computation of large models  
In Proc. ESEC/FSE 2007, p. 295–304, Dubrovnic, Croatia, 2007
- [5] Object Management Group  
Object Constraint Language (OCL) Specification (formal 2006-05-01)  
<http://www.omg.org/spec/OCL/2.0/>, 2006
- [6] ScanScore Project  
<http://www.scanscore.de/> (2008)
- [7] The Eclipse Foundation: Eclipse Modeling Framework,  
<http://www.eclipse.org/emf/> (2008)