Improving the quality of EMF models using metrics, smells, and refactorings

Gabriele Taentzer, Thorsten Arendt
Tutorial at ECMFA, July 2\textsuperscript{nd} 2012, Lyngby, Denmark
Outline

• Introduction
• Running Example
• Demo: Application of Model Quality Assurance Techniques
• Demo: Specification of Model Quality Assurance Techniques
• Conclusion
Motivation: Model quality assurance
Motivation: Model quality assurance

Problems

Risks

Model Refactorings

Model Smells

Model Metrics

Model Quality Assurance Process

Improving the quality of EMF models
A model quality assurance process

Diagram:

- Quality Aspect
- Model Smells
  - Metric
  - Anti-Pattern
- Refactoring
- Check Model Smells
- Interpretation
- Model Modifications
  - Refactoring
  - Manual Change
A model quality assurance process

Specification of a project-specific model quality assurance process

- Quality Aspect
- Question
- Model Smells
  - Metric
  - Anti-Pattern
- Refactoring

- Check Model Smells
- Interpretation
- Model Modifications
  - Refactoring
  - Manual Change
A model quality assurance process: 6C Goals

- Correctness
- Completeness
- Changeability
- Comprehensibility
- Confinement
- Consistency

Specification of a project-specific model quality assurance process

A model quality assurance process: Smells

- Example: *Are there equally named classes in different packages?*
  - Smell: **Multiple definitions of classes with equal names**

---

**Specification** of a project-specific model quality assurance process

---

Improving the quality of EMF models
A model quality assurance process: Smells

**Specification** of a project-specific model quality assurance process

- Example: *Are there abstract classes without any concrete subclasses?*
  - Smell: **No concrete subclass**
A model quality assurance process: Refactoring

**Specification** of a project-specific model quality assurance process

- Smell: *Multiple definitions of classes with equal names*
  - Refactoring: **Rename Class**
A model quality assurance process: Refactoring

Specification of a project-specific model quality assurance process

- Smell: *Multiple definitions of classes with equal names*
  - Refactoring: **Rename Class**

- Smell: *No concrete subclass*
  - Refactoring: **Insert Concrete Subclass**
A model quality assurance process

Application of the specified process to concrete software models
Tool support in Eclipse

EMF Metrics

Quality Aspect

Question

Anti-Pattern

Model Smells

Metric

Anti-Pattern

Refactoring

Check Model Smells

Interpretation

Model Modifications

Refactoring

Manual Change

Improving the quality of EMF models
Tool support in Eclipse

Improving the quality of EMF models
Tool support in Eclipse

- Quality Aspect
- Model Smells
  - Metric
  - Anti-Pattern
- Refactoring
- Check Model Smells
  - Interpretation
- Model Modifications
  - Refactoring
  - Manual Change

EMF Refactor

Improving the quality of EMF models
Topic of Tutorial

Improving the quality of EMF models
Outline

• Introduction

• **Running Example**

• Demo: Application of Model Quality Assurance Techniques

• Demo: Specification of Model Quality Assurance Techniques

• Conclusion
Example: Vehicle Rental Company

- **Project**
  - Development of a web application for renting vehicles

- **Semantic domain**
  - Company with
    - Vehicles
    - Customers
    - Services

- **Technical domain**
  - Web application

- **Modeling purpose**
  - Modeling of the semantic domain

Improving the quality of EMF models
Running example: UML class model
Outline

• Introduction
• Running Example
• Demo: Application of Model Quality Assurance Techniques
• Coffee Break
• Demo: Specification of Model Quality Assurance Techniques
• Conclusion
Improving the quality of EMF models
EMF Metrics: result view
EMF Metrics: export dialog
EMF Metrics: HTML export (bar diagram)
EMF Metrics: PPT export (pie diagram)

Smell “Doer and Knower“ represents a lack of encapsulation.
EMF Smell: configuration dialog
EMF Metrics: result view
EMF Smell: highlighting
EMF Smell: PDF export (list)

```
<table>
<thead>
<tr>
<th>Smell</th>
<th>Concrete Superclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The model contains an abstract class with a concrete superclass.</td>
</tr>
<tr>
<td>Occurrences</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Smell</th>
<th>Equal Attributes in Sibling Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Each sibling class of the owning class of an attribute contains an equal attribute.</td>
</tr>
<tr>
<td>Occurrences</td>
<td>6</td>
</tr>
</tbody>
</table>
```

---

**Smell Detection Analysis Report**

- **Analysis No.**: 1
- **Analysed File**: VRC.emx
- **Date of Analysis**: Thu Jun 28 15:28:44 CEST 2012

---

**Lesezeichen**

- 1. Concrete Superclass
  - Equal Attributes in Sibling Classes
  - Large Class
  - Long Parameter List
  - Speculative Generality (Abstract Class)
  - Speculative Generality (Interface)
Relation: smell erasing refactorings
Relation: smell causing refactoring
Dialog: suggested refactorings

- Invocation on Smell **Equal Attributes in Sibling Classes**
  - Occurrence `{Motorbike, power}`
Dialog: applicable refactoring

- Invocation on Smell **Equal Attributes in Sibling Classes**
  - Occurrence `{Motorbike, power}`
EMF Refactor: parameter dialog

- Refactoring **Pull Up Attribute**
  - Context `VehicleRentalCompany::Motorbike::power`
EMF Refactor: smell analysis

The following model smell occurrences are found before applying refactoring ‘Pull Up Attribute’:
- Equal Attributes in Sibling Classes: 6
- Long Parameter List: 3
- Large Class: 2
- Speculative Generality (Abstract Class): 1
- Speculative Generality (Interface): 1
- Concrete Superclass: 1

The following model smell occurrences are found after applying refactoring ‘Pull Up Attribute’:
- Long Parameter List: 3
- Equal Attributes in Sibling Classes: 3
- Large Class: 2
- Speculative Generality (Abstract Class): 1
- Speculative Generality (Interface): 1
- Concrete Superclass: 1

The following model smell occurrences will change when applying refactoring ‘Pull Up Attribute’:
- Equal Attributes in Sibling Classes: -3
EMF Refactor: result preview
EMF Refactor: refactored model
Outline

• Introduction
• Running Example
• Demo: Application of Model Quality Assurance Techniques
• Coffee Break
• Demo: Specification of Model Quality Assurance Techniques
• Conclusion
Used UML specification diagrams

Figure 7.14 - The Packages diagram of the Kernel package
Used UML specification diagrams

Figure 7.12 - Classes diagram of the Kernel package
Specification of new metric NACP (Total number of attributes in classes within the package): general data
Specification of new metric NACP (Total number of attributes in classes within the package): Java code

```java
package de.unimarburg.swt.emf.metric;

import java.util.List;

public final class NACP implements ICalculateClass {

    private List<EObject> context;

    @Override
    public void setContext(List<EObject> context) {
        this.context = context;
    }

    @Override
    public double calculate() {
        org.eclipse.uml2.uml.Package in =
        (org.eclipse.uml2.uml.Package) context.get(0);
        double ret = 0.0;
        // TODO fill 'ret'
        return ret;
    }
}
```

generated code

```java
package de.unimarburg.swt.emf.metric;

import java.util.List;

public final class NACP implements ICalculateClass {

    private List<EObject> context;

    @Override
    public void setContext(List<EObject> context) {
        this.context = context;
    }

    @Override
    public double calculate() {
        org.eclipse.uml2.uml.Package in =
        (org.eclipse.uml2.uml.Package) context.get(0);
        double ret = 0.0;
        // begin custom code
        for (PackageableElement pe : in.getPackagedElements()) {
            if (pe instanceof Class) {
                Class cl = (Class) pe;
                for (Property attr : cl.getAttribute()) {
                    if (attr.getAssociation() == null) ret++;
                }
            }
        }
        // end custom code
        return ret;
    }
}
```

completed code
Specification of new metric NOCP (Total number of operations in classes within the package): general data

![Image of the New Metric tool with details filled in: Project: ecmfa.tutorial.uml.metrics, Metric data: Name: NOCP, Description: Total number of operations in classes within the package, Metamodel: http://www.eclipse.org/uml2/3.0.0/UML, Context: org.eclipse.uml2.uml.Package, Source: OCL]
Specification of new metric NOCP (Total number of operations in classes within the package): OCL expr.

OCL Expression

Enter the OCL Expression.
(required fields are denoted by "*"

Metric data

OCL Expression *:
self.packagedElement -> select(oclIsTypeOf(Class))
-> collect(oclAsType(Class).getAllOperations()) -> size()
Specification of new metric KFP (Knowing Factor): general data
Specification of new metric KFP (Knowing Factor): metric data
Specification of new smell ‘Too Many Knowers in Package’: general data
Specification of new smell ‘Too Many Knowers in Package’: metric data
Used UML specification diagrams (cont.)
Used UML specification diagrams (cont.)
Used UML specification diagrams (cont.)
Specification of new smell ‘Unused Interface’: general data
Specification of new smell ‘Unused Interface’: Henshin pattern rule specification
Refactoring application workflow

Refactoring Invocation -> Initial Precondition Check

Parameter Input

Final Precondition Check -> Error Message

Model Change Preview

Model Change

Integration of Smells and Refactorings within EMF
Specification of new refactoring ‘Remove Unused Interface’: general data
Specification of new refactoring ‘Remove Unused Interface’: Henshin data
Specification of new refactoring ‘Remove Unused Interface’: Henshin’s initial check
Specification of new refactoring ‘Remove Unused Interface’: Henshin’s model change
Specification of new refactoring ‘Remove Unused Interfaces from Package’: CoMReL data
Specification of new refactoring ‘Remove Unused Interfaces from Package’: CoMReL specification
Outline

• Introduction
• Running Example
• Demo: Application of Model Quality Assurance Techniques
• Coffee Break
• Demo: Specification of Model Quality Assurance Techniques
• Conclusion
A model quality assurance process

**Specification** of a project-specific model quality assurance process

**Application** of the specified process to concrete software models
Tool environment for model quality assurance
Tool environment for model quality assurance

• Highly integrated
  – Specific tools use each other
    • Metric-based model smells
    • Refactoring invocation from within model smell results view
    • Model smell analysis during refactoring application
  – Each tool can be used directly on the model within a certain EMF-based modeling environment
    • EMF instance editor
    • Unified Modeling Language (UML)
      – Eclipse Plugin Papyrus
      – IBM Rational Software Architect
Tool environment for model quality assurance

- Flexible specification mechanisms
  - Usage of existing quality assurance techniques
    - Combination of existing model metrics
    - Metric-based model smells
    - Composite model refactorings
  - Usage of concrete specification languages
    - Java
    - OCL
    - EMF model transformation language Henshin
    - … further languages can be integrated by new adapters
Implemented model quality assurance techniques

<table>
<thead>
<tr>
<th></th>
<th>Ecore models</th>
<th>UML2EMF models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metrics</td>
<td>17</td>
<td>106</td>
</tr>
<tr>
<td>Smells</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Refactorings</td>
<td>22</td>
<td>30</td>
</tr>
</tbody>
</table>
Tool environment for model quality assurance

• Download
  – http://www.eclipse.org/modeling/emft/refactor/
    • Downloads
    • Specials