Making Refactoring Safe
in the
Presence of Meta-Data

Carlos Noguera

Benevol 2012
Suppose we want to rename the cellphone property to telephone...
Suppose we want to rename the cellphone property to telephone...

```java
String cellphone;

public String getCellphone() {
    return cellphone;
}

public void setCellphone(String cellphone) {
    this.cellphone = cellphone;
}
```

Refactor cellphone to Telephone
Suppose we want to rename the cellphone property to telephone...

```java
String cellphone;

public String getCellphone() {
    return cellphone;
}

public void setCellphone(String cellphone) {
    this.cellphone = cellphone;
}
```

Enter new name, press **Enter** to refactor

Run Tests

JUnit
Finished after 1.087 seconds

Runs: 1/1  Errors: 0  Failures: 0

testRetrieveExistingPersons (1.076 s)
Suppose we want to rename the cellphone property to telephone...

```java
String cellphone;
public String getCellphone() {
    return cellphone;
}

public void setCellphone(String cellphone) {
    this.cellphone = cellphone;
}
```

Enter new name, press Enter to refactor

Refactor cellphone to Telephone

Run Tests
Suppose we want to rename the cellphone property to telephone...

```
public String getCellphone() {
    return cellphone;
}

public void setCellphone(String cellphone) {
    this.cellphone = cellphone;
}
```

Refactor cellphone to Telephone

Run Tests
Suppose we want to rename the cellphone property to telephone...

Refactor cellphone to Telephone

Run Tests
Suppose we want to rename the cellphone property to telephone...

```java
String telephone;

public String getCellphone() {
    return cellphone;
}

public void setCellphone(String cellphone) {
    this.telephone = cellphone;
}
```

Refactor cellphone to Telephone

JUnit result:

```
Finished after 1.085 seconds

Runs: 1/1  Errors: 0  Failures: 1
```

```
Failure Trace

testRetreiveExistingPersons (1.080 s)

junit.framework.ComparisonFailure: expected: <555-1234> but was: <null>

at example.test.TestPerson.testRetrieveExistingPersons(TestPerson.java)
```
Suppose we want to rename the cellphone property to telephone...

```java
public String setCellphone(String cellphone) {
    this.cellphone = cellphone;
    return cellphone;
}
```

```
String telephone;
```
Why did the refactoring Fail?

@Entity
public class Person {

    //...

    String cellphone;

    public String getCellphone() {
        return cellphone;
    }

    public void setCellphone(String cellphone) {
        this.cellphone = cellphone;
    }

}
Why did the refactoring Fail?

- Class is a JPA Entity

```java
@Entity
public class Person {
    //...
    String cellphone;

    public String getCellphone() {
        return cellphone;
    }

    public void setCellphone(String cellphone) {
        this.cellphone = cellphone;
    }
}
```
Why did the refactoring Fail?

- Class is a JPA Entity
- Entity annotation induces implicit annotations

```java
@Table
@Entity
public class Person {

//...
@Column
String cellphone;

public String getCellphone() {
    return cellphone;
}

public void setCellphone(String cellphone) {
    this.cellphone = cellphone;
}
}
```
Why did the refactoring Fail?

- Class is a JPA Entity
- Entity annotation induces implicit annotations
- Annotations have code-dependent default values
Why did the refactoring Fail?

- Class is a JPA Entity

```java
@Entity(name = "Person")
@Table(name = "Person")
public class Person {
    private String cellphone;

    public String getCellphone() {
        return cellphone;
    }

    public void setCellphone(String cellphone) {
        this.cellphone = cellphone;
    }
}
```

- Entity annotation induces implicit annotations
- Annotations have code-dependent default values
Why did the refactoring Fail?

- Class is a JPA Entity
- Entity annotation induces implicit annotations
- Annotations have code-dependent default values

```java
@Table(name="Person")
@Entity(name="Person")
public class Person {

    //...
    @Column(name="telephone")
    String telephone;

    public String getTelephone() {
        return telephone;
    }

    public void setTelephone(String cellphone) {
        this.telephone = cellphone;
    }
}
```

The ER mapping changed because of the refactoring.
Refactoring in the presence of Meta-Data

- Meta-data impacts application behavior
- (Implicit) annotations
- XML configuration descriptors
- (functional) code conventions

Automated refactoring engines are oblivious to meta-data
Annotation Dependency Preservation
Annotation Dependency Preservation
Annotation Dependency Preservation
Annotation Dependency Preservation

Refactoring
Annotation Dependency Preservation
Annotation Dependency Preservation

Find dependencies, perform refactoring, check dependencies.
XML Dependency Preservation

Find dependencies, perform refactoring, check dependencies
Checking Invariant Preservation

Invariants and implicit annotations as source code querying problems
Checking Invariant Preservation

Invariants and implicit annotations as source code querying problems

"Column names are invariant"

if a field is annotated with @Column, then the name property should not change

Dependency is a solution to a query
Checking Invariant Preservation

Invariants and implicit annotations as source code querying problems

"Column names are invariant"

if a field is annotated with @Column, then the name property should not change

"Classes with @Entity carry an implicit @Table"

a class has a @Table if the class has a @Entity

Dependency is a solution to a query

Implicit annotation is an alternative answer to a query
Post-condition invariant checking in Eclipse
Post-condition invariant checking in Eclipse

- Dependencies obtained by running queries on the application's source code with the Ekeko Eclipse plugin
- Refactoring simulation based on refactoring-preview
Case Study: Spring

- Enterprise Framework for Java
- Annotation-based
- XML configuration

```java
@Entity
@Table
public class User extends AbstractNamedDomain {

    private String email;
    private String password;

    public String getEmail() {
        return email;
    }
}
```

```xml
<bean name="userService"
    class="ssel.banking.service.UserService"
    autowire="autodetect"/>
```
Spring Specification
Ingredients

Spring

(type-is-bean-with-id ?t ?b ?id)

XML Library

(element-has-attribute ?e ?a)

EKEKO

(typeHasAnnotationNamed ?t ?ann ?name)

Invariant predicate

(defpred invariant
   [?invariant ?context ?message]
   ;...
The name from the type is the default name of the @Table

The name of the @Table must be invariant

The name from the type
is the default name of the
@Table

The name of the
@Table must be
invariant
Bean class definitions must be invariant
All together....

```java
public class UserService implements IUserService {
    private String newManagerName;

    public void setUserManager(IUserManager userManager) {
        this.userManager = userManager;
    }
}
```
All together....
All together....
All together....
All together....

public class UserService implements IUserService {

private

public void method() {
    // Review the information provided in the list below. Click 'Continue' to proceed.
    // Found problems

    // Table mapping will change -- OLD User
    // Table mapping will change --- NEW Users

}
Making Refactoring Safe in the Presence of Meta-Data

Carlos Noguera
cnoguera@vub.ac.be

Refactoring in the Presence of Annotations.
Carlos Noguera, Coen De Roover, Andy Kellens, Viviane Jonckers:
ICSM 2012