



OCI

WE ARE
SOFTWARE
ENGINEERS.

I've seen Grails code you wouldn't believe...

Iván López @ilopmar

© 2018, Object Computing, Inc. (OCI). All rights reserved. No part of these notes may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior, written permission of Object Computing, Inc. (OCI)

objectcomputing.com



OCI

WE ARE
SOFTWARE
ENGINEERS.

I've seen Grails code you wouldn't believe...

...Or how to properly write
maintainable Grails applications

Iván López @ilopmar

Hello!

I am Iván López (@ilopmar)



OCI | WE ARE
SOFTWARE
ENGINEERS.



Groovy Users Group



Greach
the Groovy spanish conf



THIS IS A TRUE STORY.

The events depicted in this talk took place in “some projects” in 2017.

At the request of the survivors, the names have been changed.

Out of respect for the dead, the rest has been told exactly as it occurred.

Great examples

(Please don't do this)

The larger a class, the better

```
68 class Controller {
69     ...
70     ...
71     ...
72     ...
73     ...
74     ...
75     ...
76     ...
77     ...
78     ...
79     ...
80     ...
81     ...
82     ...
83     ...
84     ...
85     ...
86     ...
87     ...
88     ...
...
3848 ...
3849 ...
3850 ...
3851 ...
3852 ...
3853 ...
3854 ...
3855 ...
3856 ...
3857 ...
3858 ...
3859 ...
3860 ...
3861 ...
3862 ...
3863 ...
3864 ...
3865 ...
3866 ...
3867 ...
3868 }
```

Controller

15 beans injected

3868 lines!

The larger a method, the better

238 lines

```
43 def index(Integer max) {  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
...  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280 }
```

- Some stats...
- 41 if / 16 else
 - Up to 5 nested ifs
 - 19 service calls
 - 12 dynamic finders
 - 87 access to "params"
 - 0 tests

The larger a *setup* method in a test, the better



All same data and mocks
shared among 129 tests



How to **detect** it?

- You can't test it (or too complicated)
- IDE is slow
- Big setup methods because of lots of dependencies

How do we **fix** it?

- Small classes (it's free write more classes)
- Small methods
- Small (or non-existent) shared test code

Why does it **matter**?

- To write clean and maintainable code
- Be able to test the code

Because **copy-paste** is always the fastest way...

```
static constraints = {  
    code unique: true, nullable: false  
    name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    email email: false, nullable: true, size: 1..255  
    address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }  
    city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    zipCode nullable: true  
    phone nullable: true  
    bU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }  
    vStyle nullable: true, maxSize: 128  
    origin nullable: true, maxSize: 128  
    defaultgl nullable: true  
    currencyCode nullable: true  
}
```

Because **copy-paste** is always the fastest way...

```
static constraints = {  
    code unique: true, nullable: false  
    name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    email email: false, nullable: true, size: 1..255  
    address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }  
    city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    zipCode nullable: true  
    phone nullable: true  
    bU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }  
    vStyle nullable: true, maxSize: 128  
    origin nullable: true, maxSize: 128  
    defaultgl nullable: true  
    currencyCode nullable: true  
}
```

Because **copy-paste** is always the fastest way...

```
static constraints = {  
    code unique: true, nullable: false  
    name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    email email: false, nullable: true, size: 1..255  
    address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }  
    city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    zipCode nullable: true  
    phone nullable: true  
    bU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }  
    vStyle nullable: true, maxSize: 128  
    origin nullable: true, maxSize: 128  
    defaultgl nullable: true  
    currencyCode nullable: true  
}
```

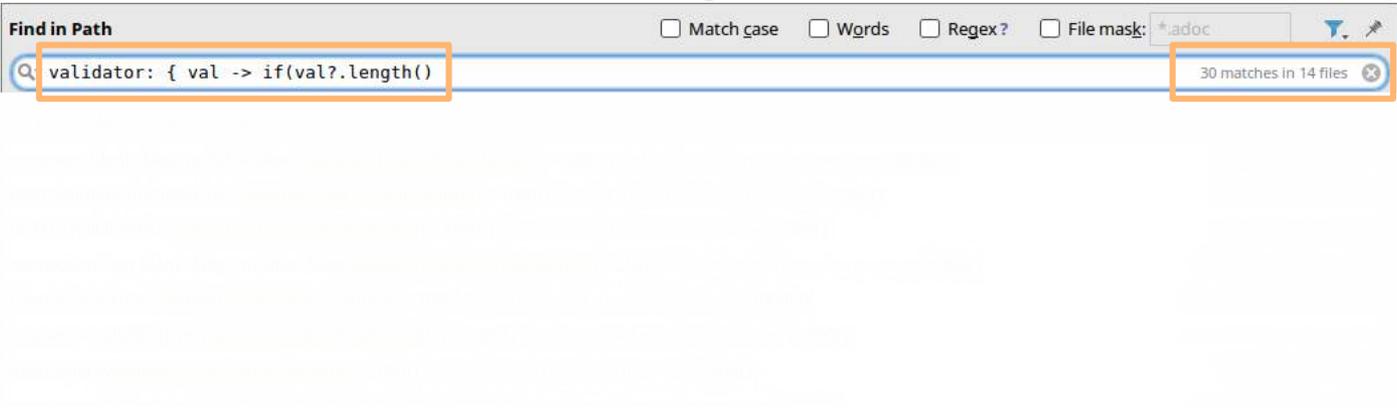
Because **copy-paste** is always the fastest way...

```
static constraints = {  
    code unique: true, nullable: false  
    name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    email email: false, nullable: true, size: 1..255  
    address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }  
    city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }  
    zipCode nullable: true  
    phone nullable: true  
    bU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }  
    vStyle nullable: true, maxSize: 128  
    origin nullable: true, maxSize: 128  
    defaultgl nullable: true  
    currencyCode nullable: true  
}
```

Because **copy-paste** is always the fastest way...

```
static constraints = {
```

```
code unique: true, nullable: false
name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
email email
address nullable
city nullable
state nullable
zipCode nullable
phone nullable
bu nullable
vStyle nullable
origin nullable
default nullable
currency nullable
}
```



Because **copy-paste** is always the fastest way...

```
class CustomValidations {  
    protected static final String INVALID_MAX_SIZE_MESSAGE = "default.invalid.max.size.message"  
  
    static def validateLength(Integer maxLength) {  
        return { val, obj ->  
            if (val?.length() > maxLength) {  
                return [INVALID_MAX_SIZE_MESSAGE, maxLength]  
            } else {  
                true  
            }  
        }  
    }  
  
    static constraints = {  
        code unique: true, nullable: false  
        name nullable: false, validator: CustomValidations.validateLength(255)  
        email email: false, nullable: true, size: 1..255  
        address nullable: true, validator: CustomValidations.validateLength(500)  
        city nullable: true, validator: CustomValidations.validateLength(255)  
        state nullable: true, validator: CustomValidations.validateLength(255)  
        zipCode nullable: true  
        phone nullable: true  
        bU nullable: true, validator: CustomValidations.validateLength(1000)  
        vStyle nullable: true, maxSize: 128  
        origin nullable: true, maxSize: 128  
        defaultgl nullable: true  
        currencyCode nullable: true  
    }  
}
```

Because **copy-paste** is always the fastest way...

```
class CustomValidations {  
    protected static final String INVALID_MAX_SIZE_MESSAGE = "default.invalid.max.size.message"  
  
    static def validateLength(Integer maxLength) {  
        return { val, obj ->  
            if (val?.length() > maxLength) {  
                return [INVALID_MAX_SIZE_MESSAGE, maxLength]  
            } else {  
                true  
            }  
        }  
    }  
  
    static constraints = {  
        code unique: true, nullable: false  
        name nullable: false, validator: CustomValidations.validateLength(255)  
        email email: false, nullable: true, size: 1..255  
        address nullable: true, validator: CustomValidations.validateLength(500)  
        city nullable: true, validator: CustomValidations.validateLength(255)  
        state nullable: true, validator: CustomValidations.validateLength(255)  
        zipCode nullable: true  
        phone nullable: true  
        bU nullable: true, validator: CustomValidations.validateLength(1000)  
        vStyle nullable: true, maxSize: 128  
        origin nullable: true, maxSize: 128  
        defaultgl nullable: true  
        currencyCode nullable: true  
    }  
}
```

Because **copy-paste** is always the fastest way...

```
class CustomValidations {  
    protected static final String INVALID_MAX_SIZE_MESSAGE = "default.invalid.max.size.message"  
  
    static def validateLength(Integer maxLength) {  
        return { val, obj ->  
            if (val?.length() > maxLength) {  
                return [INVALID_MAX_SIZE_MESSAGE, maxLength]  
            } else {  
                true  
            }  
        }  
    }  
  
    static constraints = {  
        code unique: true, nullable: false  
        name nullable: false, validator: CustomValidations.validateLength(255)  
        email email: false, nullable: true, size: 1..255  
        address nullable: true, validator: CustomValidations.validateLength(500)  
        city nullable: true, validator: CustomValidations.validateLength(255)  
        state nullable: true, validator: CustomValidations.validateLength(255)  
        zipCode nullable: true  
        phone nullable: true  
        bU nullable: true, validator: CustomValidations.validateLength(1000)  
        vStyle nullable: true, maxSize: 128  
        origin nullable: true, maxSize: 128  
        defaultgl nullable: true  
        currencyCode nullable: true  
    }  
}
```



How to **detect** it?

- Lot of duplicated code

How do we **fix** it?

- Don't repeat yourself (DRY)
- If you copy-paste +2 times → extract, refactor & reuse

Why does it **matter**?

- Only need to change code in one place
- Reuse the code

Use **def** everywhere because #yolo

```
def createUserWithActionPermissions(final itemInstance, final currentParams, final currentUser) {  
  def actionList = null  
  def actionPermissionError = null  
  def actionPermissionList = null  
  def userInstance = null  
  def errorInstance = null  
  def exceptionInstance = null  
  def statusList = null  
  final customParams = this.buildUserParams((BookType) itemInstance, currentParams, currentUser)  
  ...  
  ...  
}
```

Use **def** everywhere because **#yolo**

What about the returned type?

Everything final but without type

```
def createUserWithActionPermissions(final itemInstance, final currentParams, final currentUser) {  
  def actionList = null  
  def actionPermissionError = null  
  def actionPermissionList = null  
  def userInstance = null  
  def errorInstance = null  
  def exceptionInstance = null  
  def statusList = null  
  final customParams = this.buildUserParams((BookType) itemInstance, currentParams, currentUser)  
  ...  
  ...  
}
```

A bunch of variables
without type



How to **detect** it?

- Code-reviews

How do we **fix** it?

- Always write your methods signatures as in Java
- If the type is not obvious, write it

Why does it **matter**?

- Be able to read the code
- Help IDE to help us: auto-completion, refactors, searches,...
- Help future you (and your team) to read the code

Use **String** everywhere

Targets

Occurrences of 'String' in Project

Found Occurrences 65 occurrences

Usage in string constants 65 occurrences

- ▶ In `String` constants: 65 occurrences
- ▶ In `String` constants: 3 occurrences
- ▶ In `String` constants: 4 occurrences
- ▶ In `String` constants: 1 occurrence
- ▶ In `String` constants: 2 occurrences
- ▶ In `String` constants: 19 occurrences
- ▶ In `String` constants: 12 occurrences
- ▶ In `String` constants: 1 occurrence
- ▶ In `String` constants: 2 occurrences
- ▶ In `String` constants: 8 occurrences

Targets

Occurrences of 'String' in Project

Found Occurrences 55 occurrences

Usage in string constants 55 occurrences

- ▶ In `String` constants: 55 occurrences
- ▶ In `String` constants: 3 occurrences
- ▶ In `String` constants: 5 occurrences
- ▶ In `String` constants: 1 occurrence
- ▶ In `String` constants: 3 occurrences
- ▶ In `String` constants: 2 occurrences
- ▶ In `String` constants: 5 occurrences
- ▶ In `String` constants: 19 occurrences
- ▶ In `String` constants: 4 occurrences
- ▶ In `String` constants: 7 occurrences

Use **String** everywhere

▼ Targets

Occurrences of 'String' in Project

▼ Found Occurrences 65 occurrences

▼ Usage in string constants 65 occurrences

- ▶ [blurred]
- ▶ [blurred]
- ▶ [blurred]
- ▶ [blurred] 3 occurrences
- ▶ [blurred] 4 occurrences
- ▶ [blurred] 1 occurrence
- ▶ [blurred] 2 occurrences
- ▶ [blurred] 19 occurrences
- ▶ [blurred] 12 occurrences
- ▶ [blurred] 1 occurrence
- ▶ [blurred] 2 occurrences
- ▶ [blurred] 8 occurrences

▼ Targets

Occurrences of 'String' in Project

▼ Found Occurrences 55 occurrences

▼ Usage in string constants 55 occurrences

- ▶ [blurred]
- ▶ [blurred] 1 occurrence
- ▶ [blurred] 3 occurrences
- ▶ [blurred] 2 occurrences
- ▶ [blurred] 5 occurrences
- ▶ [blurred] 19 occurrences
- ▶ [blurred] 4 occurrences
- ▶ [blurred] 7 occurrences



How to **detect** it?

- Code-reviews
- Codenarc

How do we **fix** it?

- Use Enum/Constants
- Hide behavior inside enum (static methods)

Why does it **matter**?

- Find usages of the constants
- Refactor the code

Use GORM everywhere...

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*Service.groovy'

▼ Found Occurrences 382 occurrences

▶ Unclassified occurrence 382 occurrences

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*.gsp'

▼ Found Occurrences 59 occurrences

▶ Unclassified occurrence 59 occurrences

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*Controller.groovy'

▼ Found Occurrences 277 occurrences

▶ Unclassified occurrence 277 occurrences

Use GORM everywhere...

- ▼ **Targets**
 - 🔍 Occurrences of 'find*By' in Project with mask '*Service.groovy'
- ▼ **Found Occurrences** 382 occurrences
 - ▶ **Unclassified occurrence** 382 occurrences

- ▼ **Targets**
 - 🔍 Occurrences of 'find*By' in Project with mask '*.gsp'
- ▼ **Found Occurrences** 59 occurrences
 - ▶ **Unclassified occurrence** 59 occurrences

- ▼ **Targets**
 - 🔍 Occurrences of 'find*By' in Project with mask '*Controller.groovy'
- ▼ **Found Occurrences** 277 occurrences
 - ▶ **Unclassified occurrence** 277 occurrences

Use GORM everywhere...

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*Service.groovy'

▼ Found Occurrences 382 occurrences

▶ Unclassified occurrence 382 occurrences

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*.gsp'

▼ Found Occurrences 59 occurrences

▶ Unclassified occurrence 59 occurrences

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*Controller.groovy'

▼ Found Occurrences 277 occurrences

▶ Unclassified occurrence 277 occurrences

Use GORM everywhere...

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*Service.groovy'

▼ Found Occurrences 382 occurrences

▶ Unclassified occurrence 382 occurrences

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*.gsp'

▼ Found Occurrences 59 occurrences

▶ Unclassified occurrence 59 occurrences

▼ Targets

🔍 Occurrences of 'find*By' in Project with mask '*Controller.groovy'

▼ Found Occurrences 277 occurrences

▶ Unclassified occurrence 277 occurrences



How to **detect** it?

- Code-reviews

How do we **fix** it?

- Create new layer RepositoryService, GormService,...
- Only access db from this layer

Why does it **matter**?

- Decouple from the persistence
- Integration tests vs Unit tests

Life is too short for **NullPointerException**...

```
boolean unlockBook(String bookId, ...) {  
    def finalWS = ['Pending', 'End', 'Error']  
    Book updatedBook = Book.findById(bookId)  
    updatedBook?.workflowStatus;  
    updatedBook?.statusType;  
    updatedBook?.id;  
    updatedBook.refresh()  
    updatedBook?.w  
    updatedStatus?.s  
    cachedBookStatus?.get  
    ck?.ca  
    unt?.cc  
  
    LoanLevel?.FULL  
    s?.id  
  
    us?.st  
    us?.id  
    ck?.wo  
    finalWS?.any  
}
```



How to **detect** it?

- Code-reviews
- Codenarc (UnnecessarySafeNavigationOperator Rule)

How do we **fix** it?

- Only use '?' when values **really** can be null
- Think twice before using it

Why does it **matter**?

- Easy to read/maintain code

Pass **params** to Services, mutate and return

▼ Targets

🔍 Occurrences of 'params' in Project with mask '*Service.groovy'

▼ Found Occurrences **219 occurrences**

▼ Unclassified occurrence 219 occurrences

- ▶ [blurred] 12 occurrences
- ▶ [blurred] 6 occurrences
- ▶ [blurred] 6 occurrences
- ▶ [blurred] 25 occurrences
- ▶ [blurred] 55 occurrences
- ▶ [blurred] 10 occurrences
- ▶ [blurred] 14 occurrences
- ▶ [blurred] 38 occurrences
- ▶ [blurred] 43 occurrences
- ▶ [blurred] 8 occurrences
- ▶ [blurred] 1 occurrence
- ▶ [blurred] 1 occurrence



How to **detect** it?

- Code-reviews

How do we **fix** it?

- Just don't do it
- Use types, static inner classes, @Builder

Why does it **matter**?

- Params belongs to Controllers layer
- Decouple Controllers from Services

Who needs Command Objects? **params FTW!**

▼ Targets

🔍 Occurrences of 'params' in Project with mask '*Controller.groovy'

▼ Found Occurrences 2491 occurrences

▼ Unclassified occurrence 2491 occurrences

- ▶ `com.vaadin.flow.spring.security` 26 occurrences
- ▶ `com.vaadin.flow.spring` 11 occurrences
- ▶ `com.vaadin.flow.spring.data` 114 occurrences
- ▶ `com.vaadin.flow.spring.security` 3 occurrences
- ▶ `com.vaadin.flow.spring` 16 occurrences
- ▶ `com.vaadin.flow.spring.data` 331 occurrences
- ▶ `com.vaadin.flow.spring.data.jpa` 127 occurrences
- ▶ `com.vaadin.flow.spring.data.jpa` 61 occurrences
- ▶ `com.vaadin.flow.spring.data.jpa` 46 occurrences
- ▶ `com.vaadin.flow.spring` 20 occurrences
- ▶ `com.vaadin.flow.spring` 7 occurrences
- ▶ `com.vaadin.flow.spring.data` 291 occurrences
- ▶ `com.vaadin.flow.spring` 143 occurrences
- ▶ `com.vaadin.flow.spring.data` 9 occurrences
- ▶ `com.vaadin.flow.spring` 3 occurrences
- ▶ `com.vaadin.flow.spring` 747 occurrences
- ▶ `com.vaadin.flow.spring` 328 occurrences
- ▶ `com.vaadin.flow.spring` 33 occurrences
- ▶ `com.vaadin.flow.spring` 92 occurrences
- ▶ `com.vaadin.flow.spring` 78 occurrences

Who needs Command Objects? **params FTW!**

▼ Targets

🔍 Occurrences of 'params' in Project with mask '*Controller.groovy'

▼ Found Occurrences: 2491 occurrences

▼ Unclassified occurrence 2491 occurrences

- ▶ [blurred] 26 occurrences
- ▶ [blurred] 11 occurrences
- ▶ [blurred] 114 occurrences
- ▶ [blurred] 3 occurrences
- ▶ [blurred] 16 occurrences
- ▶ [blurred] 331 occurrences
- ▶ [blurred] 127 occurrences
- ▶ [blurred] 61 occurrences
- ▶ [blurred] 46 occurrences
- ▶ [blurred] 20 occurrences
- ▶ [blurred] 7 occurrences
- ▶ [blurred] 291 occurrences
- ▶ [blurred] 143 occurrences
- ▶ [blurred] 9 occurrences
- ▶ [blurred] 3 occurrences
- ▶ [blurred] 747 occurrences
- ▶ [blurred] 328 occurrences
- ▶ [blurred] 33 occurrences
- ▶ [blurred] 92 occurrences
- ▶ [blurred] 78 occurrences

Who needs Command Objects? **params FTW!**

```
def updateVehicle() {
```

Modifying params

```
params.dateFormat = currentUser.dateFormat.format  
params.wheels = params.wheelsRadio?.equals("user") ? params.wheels?.equals("null") ? null : params.wheels : null  
params.wheelsGroup = params.wheelsRadio?.equals("group") ? params.wheelsGroup?.equals("null") ? null : params.wheelsGroup : null  
params.buildStatus = BuildStatus.findById(uiStatus)?.workflowStatus
```

```
VehicleCommand vehicleCommand = new VehicleCommand()  
bindData(vehicleCommand, params)  
vehicleCommand.description = params.description  
vehicleCommand.category = params.category  
vehicleCommand.messageSource = messageSource  
  
VehicleDetailCommand vehicleDetailCommand = new VehicleDetailCommand()  
bindData(vehicleDetailCommand, params)  
vehicleDetailCommand.messageSource = messageSource
```

Use COs the wrong way

```
vehicleService.updateVehicle(vehicleInstance, params, vehicleCommand, vehicleDetailCommand)
```

Pass params and
COs to service

```
}
```



How to **detect** it?

- Code-reviews

How do we **fix** it?

- Always use Command Objects
- Never use/modify "params"

Why does it **matter**?

- Know exactly the expected params
- Test data binding and params transformations

Who needs Services if we have @Transactional Controllers?

Targets

Occurrences of '@Transactional' in Project with mask '*Controller.groovy'

Found Occurrences 208 occurrences

Unclassified occurrence 208 occurrences

- 1 occurrence
- 4 occurrences
- 4 occurrences
- 4 occurrences
- 6 occurrences
- 9 occurrences
- 4 occurrences
- 41 occurrences
- 12 occurrences
- 8 occurrences
- 54 occurrences
- 22 occurrences
- 5 occurrences
- 18 occurrences
- 16 occurrences

Who needs Services if we have @Transactional Controllers?

▼ Targets

Occurrences of '@Transactional' in Project with mask '*Controller.groovy'

▼ Found Occurrences 208 occurrences

▼ Unclassified occurrence 208 occurrences

- ▶ [blurred] 1 occurrence
- ▶ [blurred] 4 occurrences
- ▶ [blurred] 4 occurrences
- ▶ [blurred] 4 occurrences
- ▶ [blurred] 6 occurrences
- ▶ [blurred] 9 occurrences
- ▶ [blurred] 4 occurrences
- ▶ [blurred] 41 occurrences
- ▶ [blurred] 12 occurrences
- ▶ [blurred] 8 occurrences
- ▶ [blurred] 54 occurrences
- ▶ [blurred] 22 occurrences
- ▶ [blurred] 5 occurrences
- ▶ [blurred] 18 occurrences
- ▶ [blurred] 16 occurrences

208 actions with business logic that can't be reused



How to **detect** it?

- Code-reviews
- Codenarc

How do we **fix** it?

- Put business logic on services
- If you need to add @Transactional in controller → Move to service

Why does it **matter**?

- Reuse business logic if it's in Services
- HTTP Layer ≠ Business Logic

Tests are for people who write bugs...

```
/* This is not used yet, should be fine but no tests */  
def doStuff(final BookType bookTypeInstance, final Map incomingParams) {  
  def results = [:]
```

Tests are for people who write bugs...

```
/* This is not used yet, should be fine but no tests */  
def doStuff(final BookType bookTypeInstance, final Map incomingParams) {  
  |... def results = [:]
```



How to **detect** it?

- Code-reviews
- Coverage reports

How do we **fix** it?

- TDD
- Just don't push code without tests

Why does it **matter**?

- Modifications, refactors, new features
- Sleep well at night

Should I write tests?

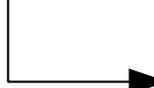
Yes

What if I just need to finish something fast?

Yes

What if I just...

OMG YES!!!



Default **UrlMappings** because I'm too lazy...

```
class UrlMappings {  
  
    static mappings = {  
        "/$controller/$action?/$id?(.$format)?"{  
            constraints {  
                // apply constraints here  
            }  
        }  
  
        "/"(view: "/index")  
        "500"(view: '/error')  
        "404"(view: '/notFound')  
    }  
}
```

Default **UrlMappings** because I'm too lazy...

```
class UrlMappings {  
  
    static mappings = {  
        "$controller/$action?/$id?(.$format)?" {  
            constraints {  
                // apply constraints here  
            }  
        }  
  
        "/"(view: "/index")  
        "500"(view: '/error')  
        "404"(view: '/notFound')  
    }  
}
```



How to **detect** it?

- Code-reviews

How do we **fix** it?

- Delete the default mappings
- Create manually a mapping per controller action

Why does it **matter**?

- Have in one place all urls in application

The master piece

```
def getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    Sql sql = Sql.newInstance(dataSource) as Sql
    def insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);

    String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String companies=companyList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String departments=departmentList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String regions=regionList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String divisions=divisionList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String stmt=""select count(*) as counter from insurance_policy_condition where""
    if(insurancePolicyList?.size(>0){
        stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""
    }
    if(companyList?.size(>0){
        stmt=stmt+"" and company_id in (${companies})""
    }
    if(departmentList?.size(>0){
        stmt=stmt+"" and department_id in (${departments})""
    }
    if(subdivisionTypesList?.size(>0){
        stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
    }
    if(regionList?.size(>0){
        stmt=stmt+"" and region_id in (${regions})""
    }
    if(divisionList?.size(>0){
        stmt=stmt+"" and division_id in (${divisions})""
    }
    stmt=stmt.replaceFirst("and","")
    def row

    if(stmt!=""){
        row = sql.firstRow(stmt)
        return row.counter
    }else{
        return 0
    }
}
```

The master piece

```
def getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    Sql sql = Sql.newInstance(dataSource) as Sql
    def insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);

    String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String companies=companyList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String departments=departmentList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String regions=regionList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String divisions=divisionList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String stmt=""select count(*) as counter from insurance_policy_condition where""
    if(insurancePolicyList?.size()>0){
        stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""
    }
    if(companyList?.size()>0){
        stmt=stmt+"" and company_id in (${companies})""
    }
    if(departmentList?.size()>0){
        stmt=stmt+"" and department_id in (${departments})""
    }
    if(subdivisionTypesList?.size()>0){
        stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
    }
    if(regionList?.size()>0){
        stmt=stmt+"" and region_id in (${regions})""
    }
    if(divisionList?.size()>0){
        stmt=stmt+"" and division_id in (${divisions})""
    }
    stmt=stmt.replaceFirst("and",",")
    def row

    if(stmt!=""){
        row = sql.firstRow(stmt)
        return row.counter
    }else{
        return 0
    }
}
```

The master piece

```
def getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    Sql sql = Sql.newInstance(dataSource) as Sql
    def insurancePolicyList = InsurancePolicy.findAll<BookType>(bookType);

    String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String companies=companyList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String departments=departmentList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String regions=regionList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String divisions=divisionList?.id?.toString()?.replace("[",",")?.replace("]",",")?.replace(" ",",")?.trim()
    String stmt=""select count(*) as counter from insurance_policy_condition where""
    if(insurancePolicyList?.size(>0){
        stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""
    }
    if(companyList?.size(>0){
        stmt=stmt+"" and company_id in (${companies})""
    }
    if(departmentList?.size(>0){
        stmt=stmt+"" and department_id in (${departments})""
    }
    if(subdivisionTypesList?.size(>0){
        stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
    }
    if(regionList?.size(>0){
        stmt=stmt+"" and region_id in (${regions})""
    }
    if(divisionList?.size(>0){
        stmt=stmt+"" and division_id in (${divisions})""
    }
    stmt=stmt.replaceFirst("and","")
    def row

    if(stmt!=""){
        row = sql.firstRow(stmt)
        return row.counter
    }else{
        return 0
    }
}
```

Let's pretend we didn't see this signature...

The master piece

```
def getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    Sql sql = Sql.newInstance(dataSource) as Sql
    def insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);

    String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
    String companies=companyList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
    String departments=departmentList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
    String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
    String regions=regionList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
    String divisions=divisionList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
    String stmt=""select count(*) as counter from insurance_policy_condition where""
    if(insurancePolicyList?.size()>0){
        stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""
    }
    if(companyList?.size()>0){
        stmt=stmt+"" and company_id in (${companies})""
    }
    if(departmentList?.size()>0){
        stmt=stmt+"" and department_id in (${departments})""
    }
    if(subdivisionTypesList?.size()>0){
        stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
    }
    if(regionList?.size()>0){
        stmt=stmt+"" and region_id in (${regions})""
    }
    if(divisionList?.size()>0){
        stmt=stmt+"" and division_id in (${divisions})""
    }
    stmt=stmt.replaceFirst("and", "")
    def row

    if(stmt!=""){
        row = sql.firstRow(stmt)
        return row.counter
    }else{
        return 0
    }
}
```

The **master** piece

String companies =

```
companyList?.  
id?.  
toString()?.  
replace("[", "").?  
replace("]", "").?  
replace(" ", "").?  
trim()
```

—————▶ The result will be a String, at
least is not def

The **master** piece

```
String companies =  
companyList?.
```

```
id?.
```

```
toString()?.
```

```
replace("[", "").?
```

```
replace("]", "").?
```

```
replace(" ", "").?
```

```
trim()
```



List<Company> companyList
according method signature

The master piece

```
String companies =  
    companyList?.  
    id?.  
    toString()?.  
    replace("[", "")?.  
    replace("]", "")?.  
    replace(" ", "")?.  
    trim()
```

Collect and extract the id of the domain objects

[2, 5, 7]

The master piece

```
String companies =  
    companyList?.  
    id?.  
    toString()?.  
    replace("[", "").?  
    replace("]", "").?  
    replace(" ", "").?  
    trim()
```

String representation of the list

"[2, 5, 7]"

The master piece

```
String companies =  
    companyList?.  
    id?.  
    toString()?.  
    replace("[", ""). → Remove [  
    replace("]", "").  
    replace(" ", "").  
    trim()
```

"2, 5, 7]"

The master piece

```
String companies =  
    companyList?.  
    id?.  
    toString()?.  
    replace("[", "").?  
    replace("]", "").? → Remove ]  
    replace(" ", "").?  
    trim()
```

"2, 5, 7"

The **master** piece

```
String companies =  
    companyList?.  
    id?.  
    toString()?.  
    replace("[", "").?  
    replace("]", "").?  
    replace(" ", "").? → Remove "space"  
    trim()
```

"2,5,7"

The master piece

```
String companies =  
    companyList?.  
    id?.  
    toString()?.  
    replace("[", "")?.  
    replace("]", "")?.  
    replace(" ", "")?.  
    trim()
```

→ Trim any space left

"2,5,7"

```
String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()  
String companies=companyList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()  
String departments=departmentList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()  
String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()  
String regions=regionList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()  
String divisions=divisionList?.id?.toString()?.replace("[", "")?.replace("]", "")?.replace(" ", "")?.trim()
```

The master piece

```
String stmt=""select count(*) as counter from insurance_policy_condition where""
if(insurancePolicyList?.size()>0){
    stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""
}
if(companyList?.size()>0){
    stmt=stmt+"" and company_id in (${companies})""
}
if(departmentList?.size()>0){
    stmt=stmt+"" and department_id in (${departments})""
}
if(subdivisionTypesList?.size()>0){
    stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
}
if(regionList?.size()>0){
    stmt=stmt+"" and region_id in (${regions})""
}
if(divisionList?.size()>0){
    stmt=stmt+"" and division_id in (${divisions})""
}
}
```

Multi-line string
because...

The master piece

```
String stmt=""select count(*) as counter from insurance_policy_condition where""
if(insurancePolicyList?.size(>0){
    stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""
}
if(companyList?.size(>0){
    stmt=stmt+"" and company_id in (${companies})""
}
if(departmentList?.size(>0){
    stmt=stmt+"" and department_id in (${departments})""
}
if(subdivisionTypesList?.size(>0){
    stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
}
if(regionList?.size(>0){
    stmt=stmt+"" and region_id in (${regions})""
}
if(divisionList?.size(>0){
    stmt=stmt+"" and division_id in (${divisions})""
}
```

The master piece

```
String stmt=""select count(*)
if(insurancePolicyList?.size()>0){
    stmt=stmt+"" and insurance_policy_condition where""
}
if(companyList?.size()>0){
    stmt=stmt+"" and company_id in (${companies})""
}
if(departmentList?.size()>0){
    stmt=stmt+"" and department_id in (${departments})""
}
if(subdivisionTypesList?.size()>0){
    stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""
}
if(regionList?.size()>0){
    stmt=stmt+"" and region id in (${regions})""
}
i    Build the sql query
}    concatenating strings
}    id in (${divisions})""
```

Variable on
method signature

The string with the
ids: "2,5,7"

The master piece

```
String stmt=""select count(*) as counter from insurance_policy_condition where""  
if(insurancePolicyList?.size(>0){  
    stmt=stmt+"" and insurance_policy_id in (${insurancePolicies})""  
}  
if(companyList?.size(>0){  
    stmt=stmt+"" and company_id in (${companies})""  
}  
if(departmentList?.size(>0){  
    stmt=stmt+"" and department_id in (${departments})""  
}  
if(subdivisionTypesList?.size(>0){  
    stmt=stmt+"" and subdivision_type_id in (${subdivisionTypes})""  
}  
if(regionList?.size(>0){  
    stmt=stmt+"" and region_id in (${region  
}  
if(divisionList?.size(>0){  
    stmt=stmt+"" and division_id in (${divisions})""  
}  
stmt=stmt.replaceFirst("and", "")
```

Remove the first "and"



The master piece

```
String stmt=""select count(*) as counter from insurance_policy_condition where""  
...  
def row  
if(stmt!=""){  
    row = sql.firstRow(stmt)  
    return row.counter  
}else{  
    return 0  
}
```

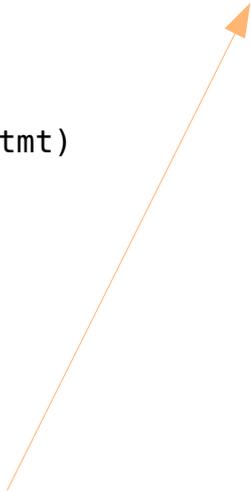
“stmt” can’t never be empty



The **master** piece

```
String stmt=""select count(*) as counter from insurance_policy_condition where""
```

```
...  
def row  
  
if(stmt!=""){  
    row = sql.firstRow(stmt)  
    return row.counter  
}else{  
    return 0  
}  
}
```



This is the query executed
and it fails!

What happens if all lists in
method signature are
empty?

The **master** piece: The solution

```
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
```

```
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
```

```
    return InsurancePolicyCondition.createCriteria().count {
```

```
        if (insurancePolicyList) {  
            'in' 'insurancePolicy', insurancePolicyList
```

```
        }
```

```
        if (companyList) {  
            'in' 'company', companyList
```

```
        }
```

```
        if (departmentList) {  
            'in' 'deparment', departmentList
```

```
        }
```

```
        if (subdivisionTypesList) {  
            'in' 'subdivision', subdivisionTypesList
```

```
        }
```

```
        if (regionList) {  
            'in' 'region', regionList
```

```
        }
```

```
        if (divisionList) {  
            'in' 'division', divisionList
```

```
        }
```

```
    }
```

```
}
```

The **master** piece: The solution

```
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
```

```
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
```

```
    return InsurancePolicyCondition.createCriteria().count {
```

```
        if (insurancePolicyList) {  
            'in' 'insurancePolicy', insurancePolicyList  
        }  
        if (companyList) {  
            'in' 'company', companyList  
        }  
        if (departmentList) {  
            'in' 'deparment', departmentList  
        }  
        if (subdivisionTypesList) {  
            'in' 'subdivision', subdivisionTypesList  
        }  
        if (regionList) {  
            'in' 'region', regionList  
        }  
        if (divisionList) {  
            'in' 'division', divisionList  
        }  
    }
```

```
}
```

```
}
```

The **master** piece: The solution

```
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
```

```
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
```

```
    return InsurancePolicyCondition.createCriteria().count {
```

```
        if (insurancePolicyList) {  
            'in' 'insurancePolicy', insurancePolicyList  
        }  
        if (companyList) {  
            'in' 'company', companyList  
        }  
        if (departmentList) {  
            'in' 'deparment', departmentList  
        }  
        if (subdivisionTypesList) {  
            'in' 'subdivision', subdivisionTypesList  
        }  
        if (regionList) {  
            'in' 'region', regionList  
        }  
        if (divisionList) {  
            'in' 'division', divisionList  
        }  
    }
```

```
}
```

The **master** piece: The solution

```
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList,
List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
```

```
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
```

```
    return InsurancePolicyCondition.createCriteria().count {
        if (insurancePolicyList) {
            'in' 'insurancePolicy', insurancePolicyList
        }
        if (companyList) {
            'in' 'company', companyList
        }
        if (departmentList) {
            'in' 'deparment', departmentList
        }
        if (subdivisionTypesList) {
            'in' 'subdivision', subdivisionTypesList
        }
        if (regionList) {
            'in' 'region', regionList
        }
        if (divisionList) {
            'in' 'division', divisionList
        }
    }
```

```
}
```

The **master** piece: The solution

```
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
```

```
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
```

```
    return InsurancePolicyCondition.createCriteria().count {
```

```
        if (insurancePolicyList) {  
            'in' 'insurancePolicy', insurancePolicyList
```

```
        }
```

```
        if (companyList) {  
            'in' 'company', companyList
```

```
        }
```

```
        if (departmentList) {  
            'in' 'deparment', departmentList
```

```
        }
```

```
        if (subdivisionTypesList) {  
            'in' 'subdivision', subdivisionTypesList
```

```
        }
```

```
        if (regionList) {  
            'in' 'region', regionList
```

```
        }
```

```
        if (divisionList) {  
            'in' 'division', divisionList
```

```
        }
```

```
    }
```

```
}
```



Just using GORM the right way!



How to **detect** it?

- Code-reviews

How do we **fix** it?

- Teach developers their tools, languages,...
- Mentoring
- Buy books, send them to conferences,...

Why does it **matter**?

- Write clean and maintainable code
- Help your team grow

“

*With great power comes
great responsibility*



Do you want to learn more?

- <http://guides.grails.org/>
- <http://grails.org/community.html>
- <https://grails.org/documentation.html>

UPCOMING EVENTS AND TRAINING

Events:

- objectcomputing.com/events

Training:

- objectcomputing.com/training
- grailstraining.com
- micronauttraining.com

Or email info@ocitraining.com to schedule a custom training program for your team online, on site, or in our state-of-the-art, Midwest training lab



OCI | WE ARE SOFTWARE ENGINEERS.

CONNECT WITH US



1+ (314) 579-0066



@objectcomputing



info@ocitraining.com