RMoD
Analyses and Language Constructs for Modular Object-Oriented Applications

http://rmod.lille.inria.fr

Stéphane Ducasse
Roadmap

Context
Maps
Evolution in the large
Others
Current work
Long term research vision

How to build and evolve ever *running* software systems?
Objectives in synergy

1: How to maintain/evolve large software systems?
Moose: a platform for software and data analysis
Synectique.eu

2: Infrastructure for ever-running systems

3: Ecosystem around Pharo
Platform&dynamic language used to create wealth and innovation
Objectives in synergy

Software Evolution

Bugs Rules Transformation

Dynamic Languages

Slots Reflectivity Tailoring Ownership

rmod research

moose

Pharo

external world

Teachers Research groups Companies
Software is Complex
Laws of software evolution

Continuing change
• A program that is used in a real-world environment must change, or become progressively less useful in that environment.

Increasing complexity
• As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure.
Software is a living entity...

• Early decisions were certainly good at that time
• But the context changes
• Customers change
• Technology changes
• People change
We only maintain useful successful software
Maintenance is continuous Development

Between 70% and 90% of global effort is spent on “maintenance”!

18% Adaptive (new platforms or OS)
18% Corrective (fixing reported errors)
60% Perfective (new functionality)

“Maintenance”
50% of development time is lost trying to understand code!

Between 50% and 80% of the overall cost is spent in the evolution

We lose a lot of time with inappropriate and ineffective practices
We design tools and analyses to tame software
Expertise
code analysis, metamodeling, software metrics, program understanding, *program visualization*, evolution analysis, refactorings, quality, changes analysis, commit, dependencies, rule and bug assessment

**semi-automatic migration**
example-based transformations
test selection, rearchitecturing blockchains, *ui-migration*

Collaborations
Pleiad (Chile), UFMG (Brazil), SCG (Switzerland), Soft-VUB (Belgium)
You would not see this dentist!

Why doing it for your software?
Building dedicated tools

Data → Model

Develop Analysis

Analysis

Take decision

an analysis should lead to a decision
Example: Who is behind package X?
Step 1 - Model Creation/Import

- (1) Extraction
- (2) Modèle
- (3) Analyses
- (4) Visualisation

Definition of a model to represent entities
Data Extraction (CVS...)

Inria
Step 2 - Analyses

Who wrote how many lines of code?
Step : 3 - Creating the Map
All JBoss at a glance

Interactive tool

Data in perspective

(1) Extraction

(2) Modèle

(3) Analyses

(4) Visualisation
McCabe = 21
LOC = 753,000

classes select: #isGod
Some software maps
Understanding large systems [PhD Lanza]
Understanding systems [PhD M. Lanza]
Understanding classes [PhD M. Lanza]

<table>
<thead>
<tr>
<th>Initialize</th>
<th>Interface</th>
<th>Internal</th>
<th>Accessor</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Initialize" /></td>
<td><img src="image2" alt="Interface" /></td>
<td><img src="image3" alt="Internal" /></td>
<td><img src="image4" alt="Accessor" /></td>
<td><img src="image5" alt="Attribute" /></td>
</tr>
</tbody>
</table>
Understanding classes [PhD M. Lanza]
How a property spread on a system?
How to support remodularisation? [PhD Hany Abdeen]
Software Evolution in the Large
Evolution in the small
Evolution in the small

Modification (refactoring)
- On 1 entity
- Small number of parameters
- Well defined behavior
- “Preserve code semantic”
- Generic (constrained only by the programming language)
Evolution in the LARGE

- Eclipse v2.1 → v3.0

<table>
<thead>
<tr>
<th>v2.1 (Extensible IDE)</th>
<th>v3.0 (Rich Client Platform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help</td>
<td>Update</td>
</tr>
<tr>
<td>UI</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>IDE Views</td>
<td></td>
</tr>
<tr>
<td>Workbench (with IDE personality)</td>
<td></td>
</tr>
<tr>
<td>JFace</td>
<td></td>
</tr>
<tr>
<td>SWT</td>
<td></td>
</tr>
<tr>
<td>Runtime</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Help (optional)</th>
<th>Update (optional)</th>
<th>Text (optional)</th>
<th>IDE Text</th>
<th>Compare</th>
<th>Debug</th>
<th>Search</th>
<th>Team/CVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UI (Generic Workbench)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JFace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime (OSGi)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources (optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evolution in the LARGE

- Restructure architecture
- Large refactor
- Break a big class
- Introduce a design pattern (e.g. MVC, Hibernate)
- Migrate to a new library version
- …
Evolution in the LARGE

• Can/Should occur regularly (≠often) in the life of a system
• Evolution
  – On several entities
  – Complex behavior
  – Specific to the domain, system, and task
  – May break code semantic temporarily
Evolution in the large: Automated Rules to support migration
Mining API Change Rules

• In Eclipse, 42% of the methods in version 1 were not in v.2

• In Pharo, a simple API change affected thousands of clients

• Clients should not but *do* call internal API
How to help migrate from version to version?
Mining API Change Rules

1. Extracting deltas

```
Rev 1  ➔  Rev 2  ➔  Rev 3  ➔  ...  ➔  Rev n

revisions

deltas
```
1. Extracting deltas

2. Discovering rules

- Extracting deltas
  - Rev 1
  - Rev 2
  - Rev 3
  - ... (until Rev n)

- Discovering rules
  - Select deltas
  - Create rules (data mining)
  - Rules
Mining API Change Rules

Format of the changes

deltas

deleted-invoc(context-id, signature)
added-invoc(context-id, signature)

Diff of method foo() between version 1 and 2

- self.add(MooseModel.root().add(model));
+ self.add(model.install());

Formatted changes

deleted-call("foo()-rev2", "MooseModel.root()")
deleted-call("foo()-rev2", "MooseModel.add(MooseModel)")
added-call("foo()-rev2", "MooseModel.install()")
Mining API Change Rules

Request: foo()
Mining API Change Rules

Request: foo()
1st step: selecting deltas
Mining API Change Rules

Rule:  

1\textsuperscript{st} step: selecting deltas

2\textsuperscript{nd} step: discovering rules

$$\text{foo()} \rightarrow \text{bar()} \quad \text{Confidence} = 75\%$$

deltas

$$\text{d() \quad e() \quad f() \quad g() \quad h() \quad i() \quad j() \quad k()}$$

$$\text{a() \quad e() \quad f() \quad g() \quad h() \quad i() \quad j() \quad k()}$$

$$\text{foo() \quad bar() \quad c()}$$

$$\text{foo() \quad bar() \quad c()}$$
Mining API Change Rules

isNil().ifTrue(*) → ifNil(*)

keys().do(*) → keysDo(*)

intersect(*) → intersectIfNone(*,*)

Scanner.new().scanTokens(*) → parseAsLiteralToken()

RegisterAsApplication(*) →
   WAAdmin.registerAsApplicationAt(*,*)

Character.cr() → ROPlatform.current().newLine()
Tool Support

1. Input pane
   - Min support
   - Request

2. Association rule pane
   - Association Rules
   - All Deltas
   - Confidence and support of the rule

3. Delta pane
   - Default displaying: list of deltas

4. Example helper pane
   - Deleted call
   - Added call

Request
- ClassOrganizer default
Evolution in the large: Supporting specific refactorings
System Specific Refactorings

JHotDraw 7.4.1 → 7.5.1

class HSLColorSpace extends ColorSpace {

    public HSLColorSpace() { /* … */ }

    // ...
}

class HSLColorSpace extends ColorSpace implements NamedColorSpace {

    private static HSLColorSpace instance;
    public static HSLColorSpace getInstance() {
        /* … */
    }

    private HSLColorSpace() { /* … */ }

    public String getName() {
        return "HSL";
    }

    // ...
}
System Specific Refactorings

JHotDraw 7.4.1 → 7.5.1

class HSVColorSpace extends ColorSpace
{

    public HSVColorSpace() { /* … */ }

    // …
}

class HSVColorSpace extends ColorSpace implements NamedColorSpace {

    private static HSVColorSpace instance;
    public static HSVColorSpace getInstance() {
        /* … */
    }

    private HSVColorSpace() { /* … */ }

    public String getName() {
        return "HSV";
    }

    // …
}
# System Specific Refactorings

**JHotDraw 7.4.1 → 7.5.1**

```java
class HSVColorSpace extends ColorSpace {
    // ...

    public HSVColorSpace() { /* ... */ }

    // ...
}
```

```java
public String getName() {
    return "HSV";
}
// ...
```

```java
class HSVColorSpace extends ColorSpace implements NamedColorSpace {

    private static HSVColorSpace instance;

    public static HSVColorSpace getInstance() {
        /* ... */
    }

    private HSVColorSpace() { /* ... */ }

    public String getName() {
        return "HSV";
    }

    // ...
}
```
System Specific Refactorings

PackageManager 0.58 → 0.59

```plaintext
class GreasePharo30CoreSpec
    platform
    package
        addPlatformRequirement: #'pharo'.
    package
        addProvision: #'Grease-Core-Platform'

class GreasePharo30CoreSpec
    platformRequirements
        ^ #( #'pharo' )
    provisions
        ^ #( #'Grease-Core-Platform' )

class SeasideCanvas20CoreSpec
    platform
    package
        addPlatformRequirement: #'pharo2.x'.
    package
        addProvision: #'Seaside-Canvas-Platform'

class SeasideCanvas20CoreSpec
    platformRequirements
        ^ #( #'pharo2.x' )
    provisions
        ^ #( #'Seaside-Canvas-Platform' )
```
System Specific Refactorings

PackageManager 0.58 → 0.59

class GreasePharo30CoreSpec
  platform
  package
    addPlatformRequirement: #'pharo'.
  package
    addProvision: #'Grease-Core-Platform'

class GreasePharo30CoreSpec
  platformRequirements
    ^ #( #'pharo' )
  provisions
    ^ #( #'Grease-Core-Platform' )

class SeasideCanvas20CoreSpec
  platform
  package
    addPlatformRequirement: #'pharo2.x'.
  package
    addProvision: #'Seaside-Canvas-Platform'

class SeasideCanvas20CoreSpec
  platformRequirements
    ^ #( #'pharo2.x' )
  provisions
    ^ #( #'Seaside-Canvas-Platform' )
System Specific Refactorings

- Found (manually) in various systems histories

<table>
<thead>
<tr>
<th>Java systems</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse I</td>
<td>26</td>
</tr>
<tr>
<td>Eclipse II</td>
<td>72</td>
</tr>
<tr>
<td>JHotDraw</td>
<td>9</td>
</tr>
<tr>
<td>MyWebMarket</td>
<td>7</td>
</tr>
<tr>
<td>VerveineJ</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pharo systems</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>PetitDelphi</td>
<td>21</td>
</tr>
<tr>
<td>PetitSQL I</td>
<td>6</td>
</tr>
<tr>
<td>PetitSQL II</td>
<td>98</td>
</tr>
<tr>
<td>PackageManager I</td>
<td>50</td>
</tr>
<tr>
<td>PackageManager II</td>
<td>19</td>
</tr>
<tr>
<td>PackageManager III</td>
<td>64</td>
</tr>
<tr>
<td>PackageManager IV</td>
<td>7</td>
</tr>
<tr>
<td>Jenkins</td>
<td>7</td>
</tr>
<tr>
<td>MooseQuery I</td>
<td>16</td>
</tr>
<tr>
<td>MooseQuery II</td>
<td>8</td>
</tr>
</tbody>
</table>
System Specific Refactorings

- System/task specific
- “Incomplete” (may break semantics)
- Complex
- Tedious
- Error prone  

\{ Automate ? \}
System Specific Refactorings

1. Code edition
2. Transformation events
3. Changed code entity
4. Automatic configuration
5. Strategies
6. New code location
7. New changed location
8. Applying transformations
9. ADD
10. ADD
11. REM
System Specific Refactorings

• Record

Recorded actions (operators)

Operators’ parameters (to adapt)
System Specific Refactorings

- Replay

IDE (list of methods)

Custom refactoring name

[Diagram showing a list of methods with an option for custom refactoring]
System Specific Refactorings

• Case studies

<table>
<thead>
<tr>
<th>Pharo systems</th>
<th>#Repetition</th>
<th>#Operators</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PetitDelphi</td>
<td>21</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>PetitSQL I</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>PetitSQL II</td>
<td>98</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>PackageManager I</td>
<td>50</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>PackageManager II</td>
<td>19</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>PackageManager III</td>
<td>64</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>PackageManager IV</td>
<td>7</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Jenkins</td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MooseQuery I</td>
<td>16</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MooseQuery II</td>
<td>8</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
System Specific Refactorings

- “Can the tool configure parameters?”

<table>
<thead>
<tr>
<th>Pharo systems</th>
<th>#repetitions</th>
<th>#Operators</th>
<th>#Parameters</th>
<th>%Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>PetitDelphi</td>
<td>21</td>
<td>2</td>
<td>3</td>
<td>99%</td>
</tr>
<tr>
<td>PetitSQL I</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>67%</td>
</tr>
<tr>
<td>PetitSQL II</td>
<td>98</td>
<td>3</td>
<td>6</td>
<td>83%</td>
</tr>
<tr>
<td>PackageManager I</td>
<td>50</td>
<td>5</td>
<td>4</td>
<td>29%</td>
</tr>
<tr>
<td>PackageManager II</td>
<td>19</td>
<td>3</td>
<td>5</td>
<td>54%</td>
</tr>
<tr>
<td>PackageManager III</td>
<td>64</td>
<td>2</td>
<td>4</td>
<td>72%</td>
</tr>
<tr>
<td>PackageManager IV</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>79%</td>
</tr>
<tr>
<td>Jenkins</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>49%</td>
</tr>
<tr>
<td>MooseQuery I</td>
<td>16</td>
<td>1</td>
<td>3</td>
<td>36%</td>
</tr>
<tr>
<td>MooseQuery II</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>28%</td>
</tr>
</tbody>
</table>
System Specific Refactorings

- Efficiency depends on type of operator parameter

<table>
<thead>
<tr>
<th>Param. type</th>
<th>Correct</th>
<th>Incorrect</th>
<th>%Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>21</td>
<td>3</td>
<td>77%</td>
</tr>
<tr>
<td>Method</td>
<td>6</td>
<td>5</td>
<td>98%</td>
</tr>
<tr>
<td>Pragma (annotation)</td>
<td>98</td>
<td>6</td>
<td>13%</td>
</tr>
<tr>
<td>Prototype</td>
<td>50</td>
<td>4</td>
<td>99%</td>
</tr>
<tr>
<td>Source code</td>
<td>19</td>
<td>5</td>
<td>9%</td>
</tr>
<tr>
<td>variable</td>
<td>64</td>
<td>4</td>
<td>100%</td>
</tr>
</tbody>
</table>
Evolution in the large: GWT to angular

PhD CIFRE of Benoit Verhaeghe for Berger-Levrault
GWT Typical Application

- 26,000 Classes
- 450 web pages
- 974,000 invocations
- 1,063,163 LOC (UI)
GWT big kitchen (bac a sable)

bigKitchen ~ 8979 éléments
Approach
Approach
Metamodel
Results

Export

- Navigation par phase
  - Ouvrir un onglet
  - Ouvrir une boîte de dialogue modale
  - Ouvrir une boîte de dialogue non modale
  - Ouvrir une boîte de dialogue unique non modale

- Boîtes de dialogue (hors phase)
  - Boîte de dialogue modale
  - Boîte de dialogue non modale
  - Message d’information
  - Message d’avertissement
  - Message d’erreur (métier)
  - Message de confirmation
  - Message d’erreur (exception)
Results

Export

<table>
<thead>
<tr>
<th>Etiquettes formatées (pour les listes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Montant : 10 000 000,00 €</td>
</tr>
<tr>
<td>Pourcentage : 1,50%</td>
</tr>
<tr>
<td>Booléen (1) : Oui</td>
</tr>
<tr>
<td>Booléen (0) : Non</td>
</tr>
<tr>
<td>Date : 26/06/2018</td>
</tr>
<tr>
<td>Durée : 3 ans, 2 mois et 1 jour</td>
</tr>
<tr>
<td>Enumération : jour(s)</td>
</tr>
<tr>
<td>Entier : 999999999</td>
</tr>
<tr>
<td>Entier long : 99999999999999999997</td>
</tr>
<tr>
<td>Entier short : 999</td>
</tr>
</tbody>
</table>

35
Future work

Behavior code model
Specific representation of layout
Extract complex structure (such table data structure)
How to evaluate GUI migration?
How can we help merging?

What is the impact of a change?

PhD of Veronica Uquillas (Paid VUB)
How to support merging branches?

- Manual tasks are needed
- Dependencies between changes
- Integrator is not the author of the changes
- No guarantee that the system will work
Assisted Integration

Stream of changes (chains of commits)

Single delta (commit)

Source Code Meta-Model (Ring)

Change & Dependency Meta-Model and Analyses (RingC)

History Meta-Model and Analyses

Single-delta Change Meta-Model and Analyses (RingS)

JET

Torch

Inria
Torch: Supporting Commit Understanding
Package Structure

Visualization: Changed Packages (details)
Omnipresent source code

```ruby
instance method Protocol: browse Author: AndyKellen

browseAllObjectReferencesTo: anObject except: objectsToExclude ifNone: aBlock
"Bring up a list inspector on the objects that point to anObject.
If there are none, then evaluate aBlock on anObject."

| aList shortName |
| aList := anObject pointersToExcept: objectsToExclude. |
| aList := PointerFinder pointersTo: anObject except: objectsToExclude. |
| aList size > 0 ifFalse: [^aBlock value: anObject]. |
| shortName := (anObject name ifNil: [anObject printString]) contractTo: 20. |
| aList inspectWithLabel: 'Objects pointing to ', shortName |
```
Torch: Which changes?
Where? Who? What?

A set of changes, involving:
5 packages,
9 classes,
~40 methods
Streams of Changes:
On what other changes does this change depend?
Other Software Evolution Challenges

Blockchain, test selection,
Which tests to rerun?  
[PhD CIFRE ATOS Vincent Blondeau]

Test Selection

Automatic Test Selection

3 hours  \rightarrow 5 minutes

Select only the tests related to the changes and run only those ones
SmartAnvil: Tools for SE-Blockchains

Dashboard

Gas analysis

Ukulele
Smart Inspect
Fog Live
Fog Comm

Call Graph
SmartMetrics
SmartGraph
SmaccSol

Fog ByteCode

Blockchain (Ethereum)

Code (Solidity)

Moose

SmartInspect
SmartMetrics
SmartGraph
SmaccSol
Fog ByteCode
Dashboard
Gas analysis
Ukulele
Fog Live
Fog Comm
Blockchain (Ethereum)
Code (Solidity)

Dashboard
Gas analysis
Ukulele
Fog Live
Fog Comm
Blockchain (Ethereum)
Code (Solidity)

Dashboard
Gas analysis
Ukulele
Fog Live
Fog Comm
Blockchain (Ethereum)
Code (Solidity)

Dashboard
Gas analysis
Ukulele
Fog Live
Fog Comm
Blockchain (Ethereum)
Code (Solidity)
Blockchain modelisation
[PhD - S. Bragagnolo - Berger-Levrault]

How to capture and model existing process?
How to model and simulate domain of trust and possible architecture?
What are the tools for business blockchain engineers?
And embedded procedures? [PhD Julien Delplanque]

- **Dependency Analysis** — *How to know which entities potentially need to be changed when an entity is changed?*

- **Recommendations** — *How to adapt an entity to a change on a entity it depends on?*

- **Embedded Behaviour Management** — *How to keep the behaviour embedded inside the database working after a change?*
Refactorings for Embedded procedures

SQL meta-model

Structural Entities

Behavioural Entities

References Entities
Creating analyses to tame software

Interested by your challenges
- migration help
- assessment
- rule extraction
- software map
- rearchitecture
- service/micro service identification
- blockchainisation :)

RMoD