
# 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

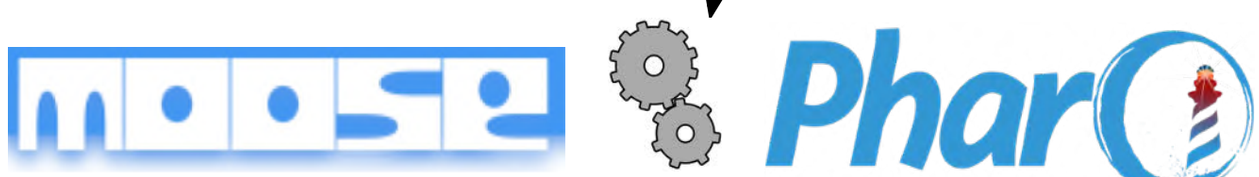
Software Evolution

Bugs Rules
Transformation

Dynamic Languages

Slots Reflectivity
Tailoring Ownership

*rmod research*




*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**

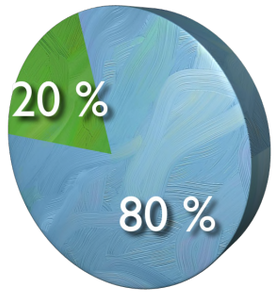
- As a program evolves, it becomes more complex, and extra resources are needed to preserve and simplify its structure.

# 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

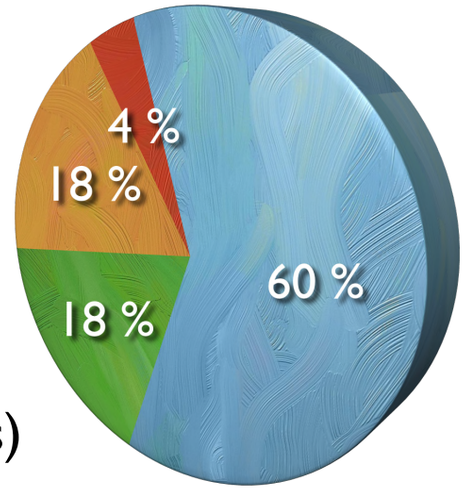
# Maintenance is *continuous* Development




Between **70**% and **90**% of ***global*** effort is spent on "maintenance" !

**18% Adaptive**
(new platforms or OS)

**4% Other**

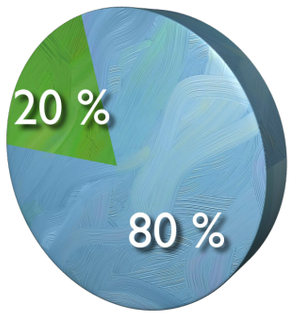


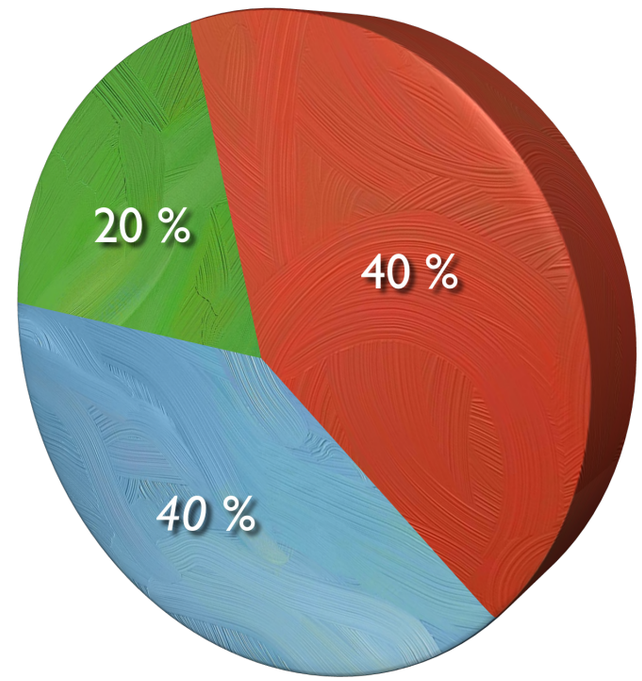

**18% Corrective**
(fixing reported errors)

**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**

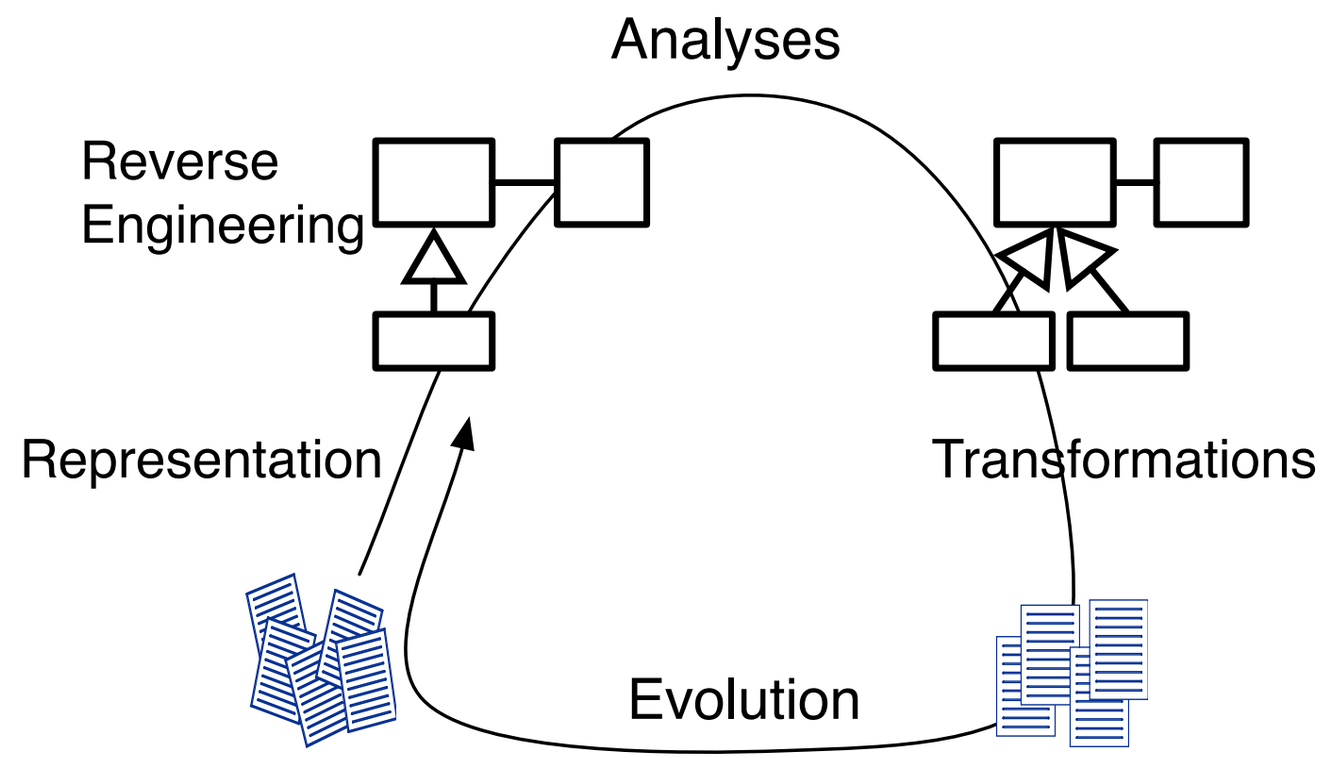
# We design tools and analyses to tame software

# 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***

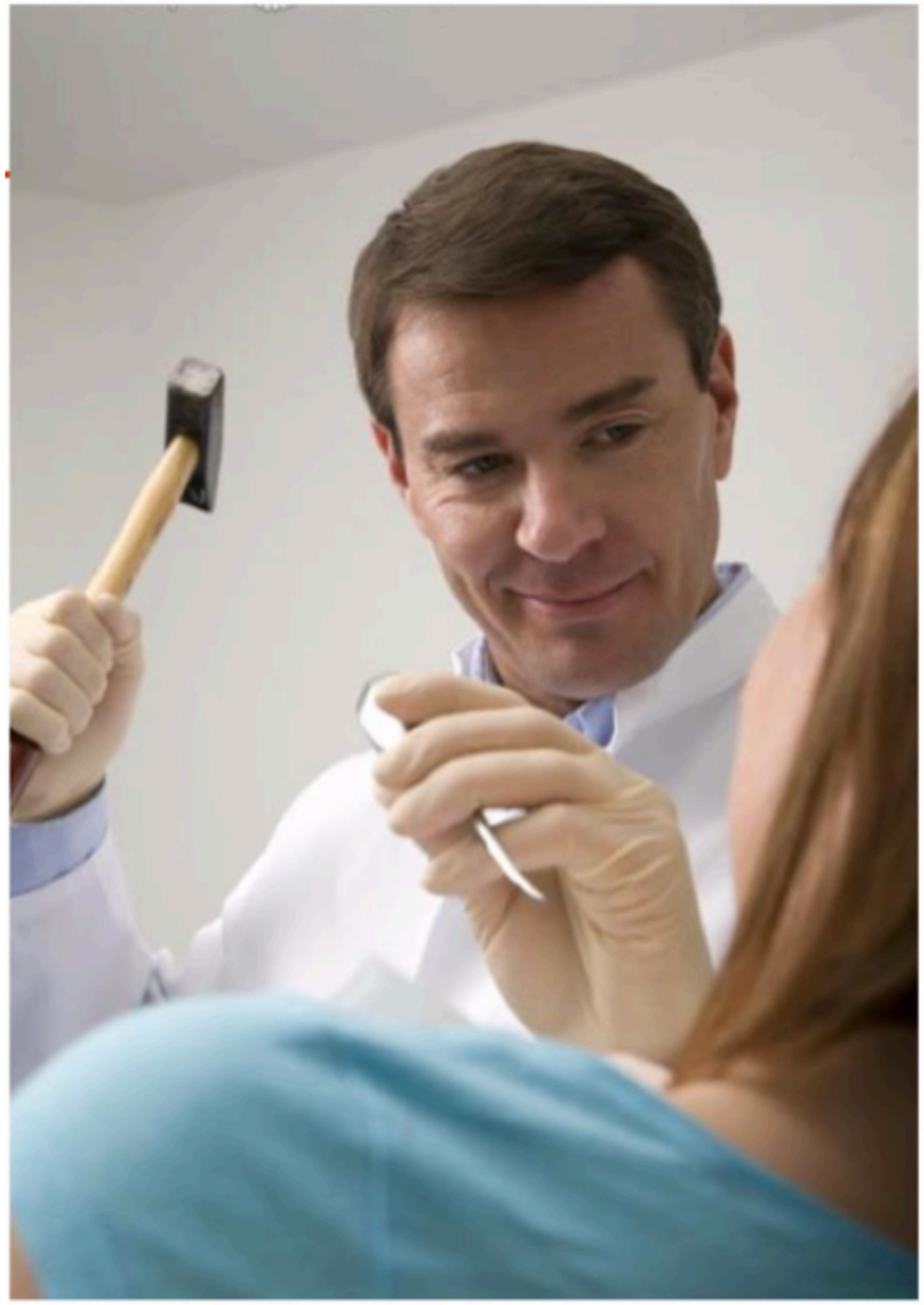
**Collaborations**

Pleiad (Chile), UFMG (Brazil),
SCG (Switzerland), Soft-VUB (Belgium)

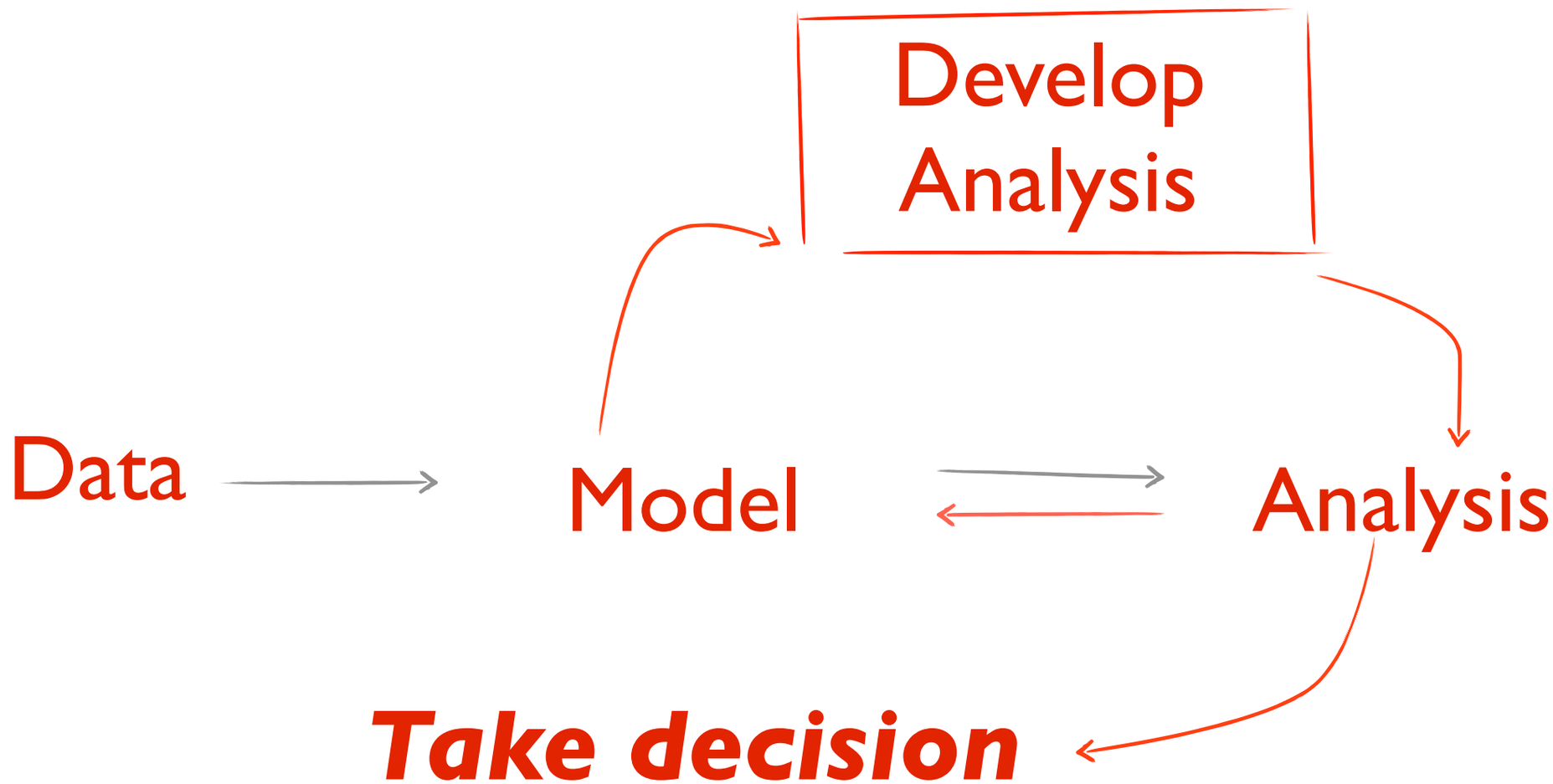


# You would not see this dentist!

# Why doing it for your software?

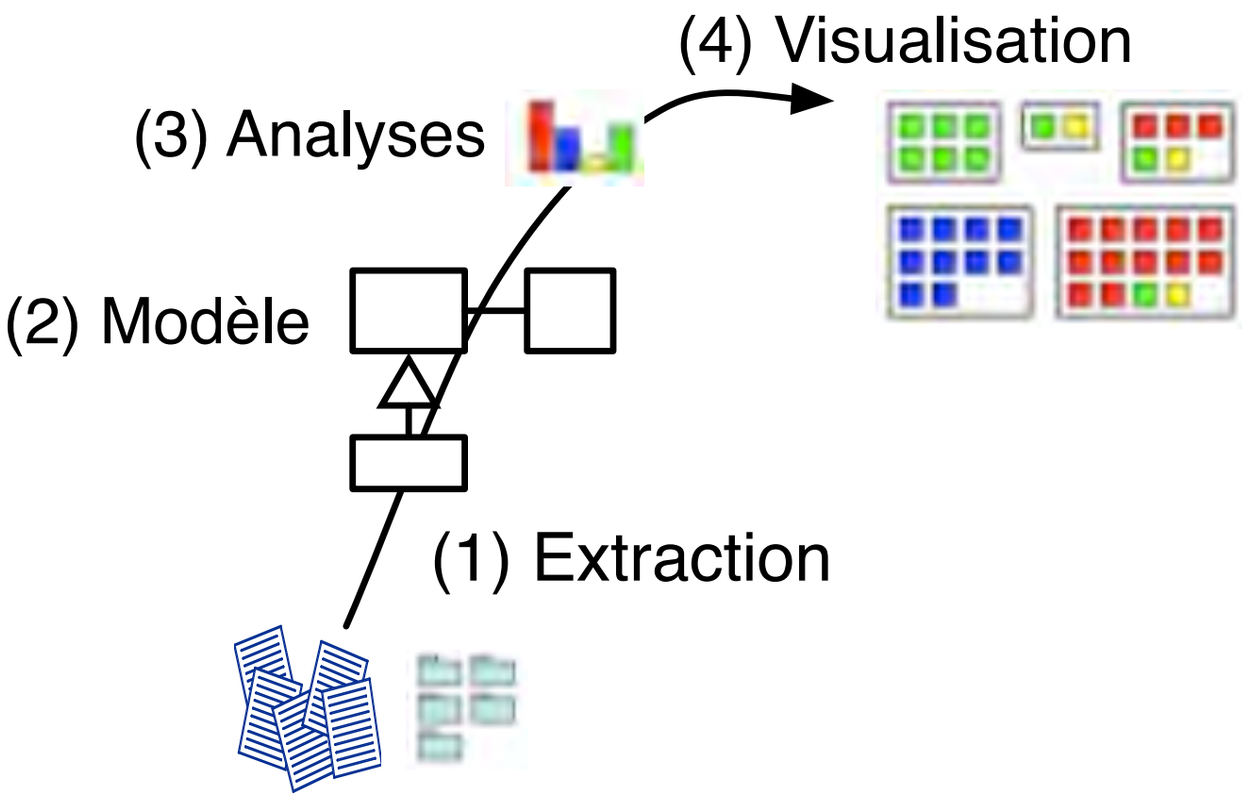


# Building dedicated tools

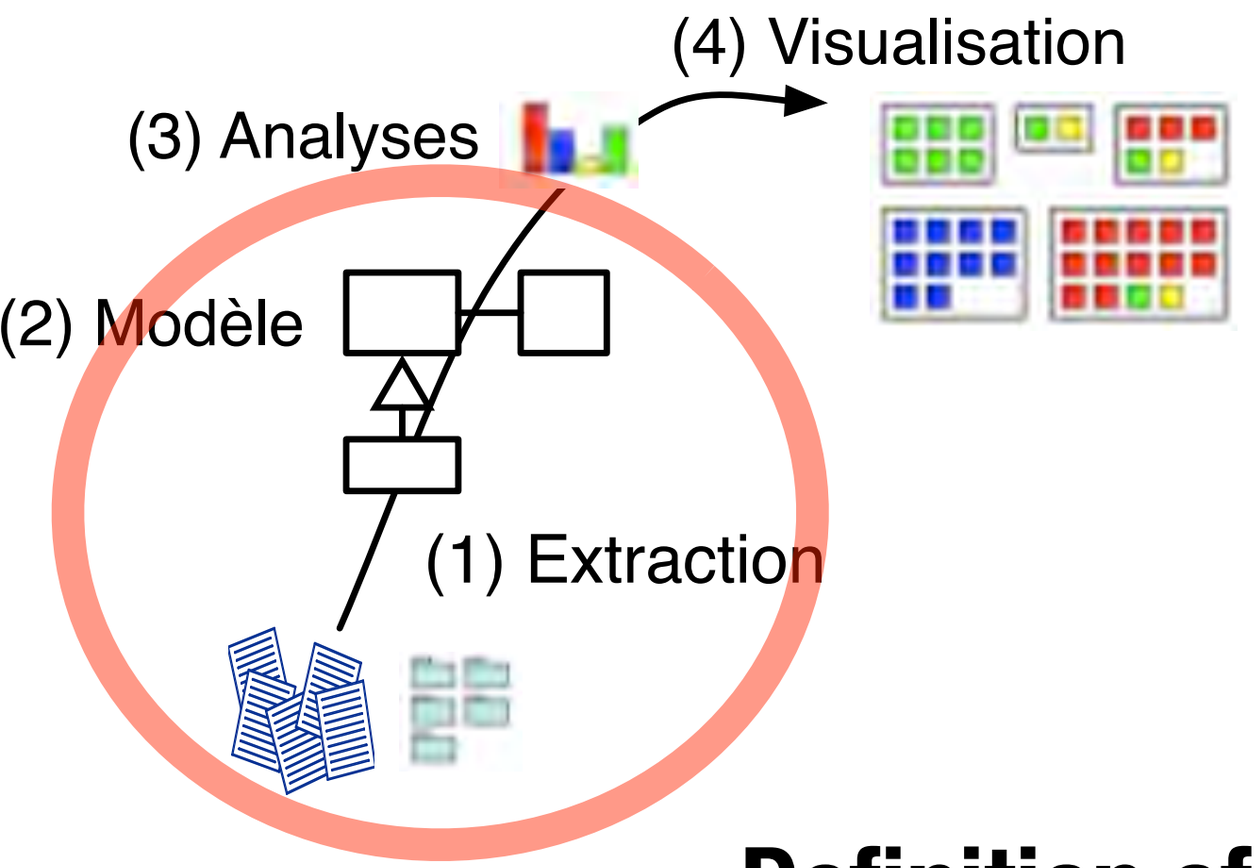



**an analysis should lead to a decision**

# Example : Who is behind package X ?



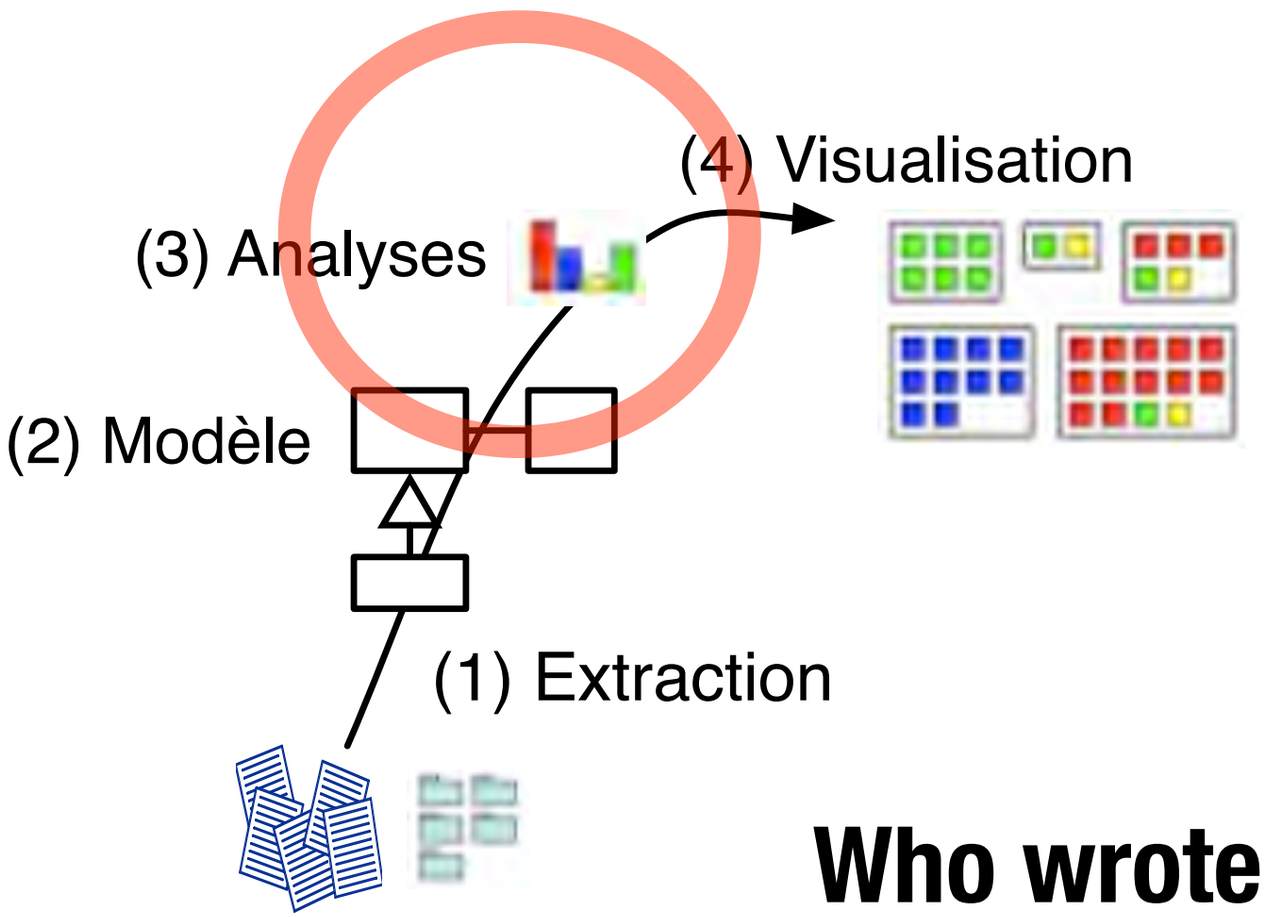
# Step 1 - Model Creation/Import




**Definition of a model to represent entities**

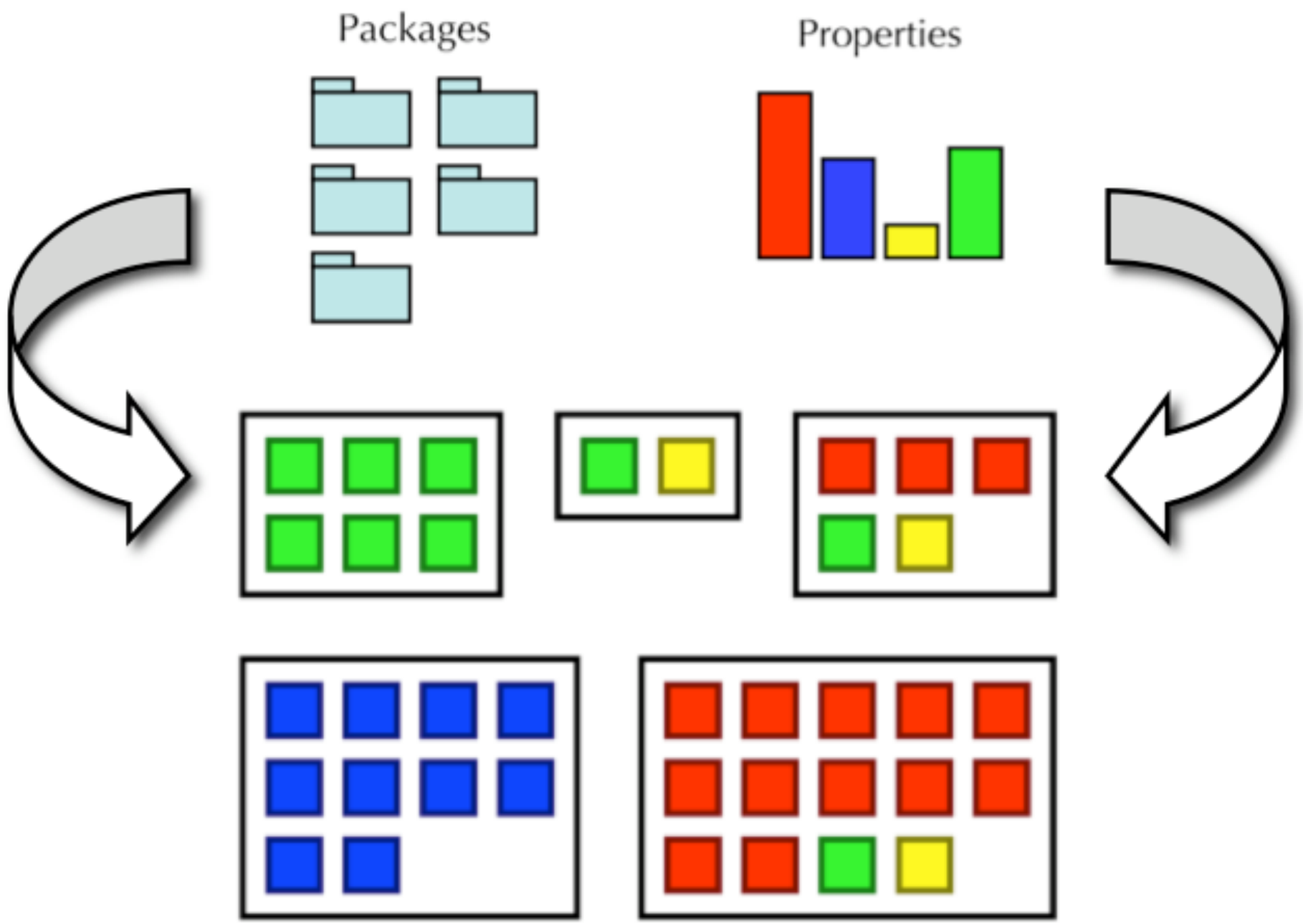
**Data Extraction (CVS...)**

# Step 2 - Analyses




**Who wrote how many lines of code?**

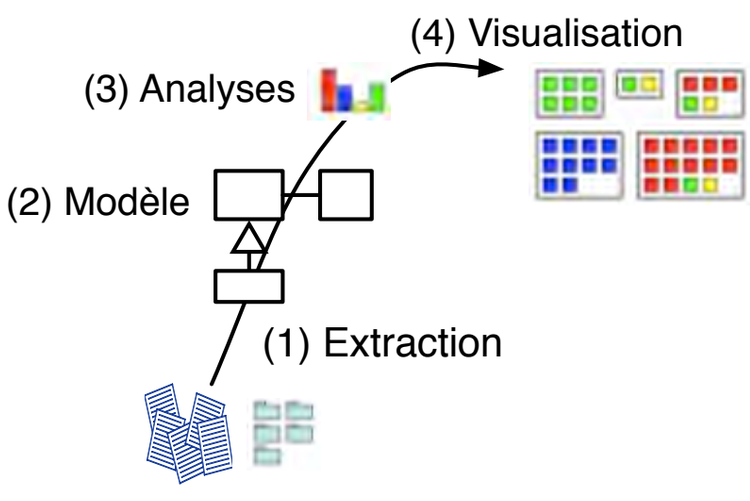
# Step : 3 - Creating the Map



# All JBoss at a glance

## Interactive tool

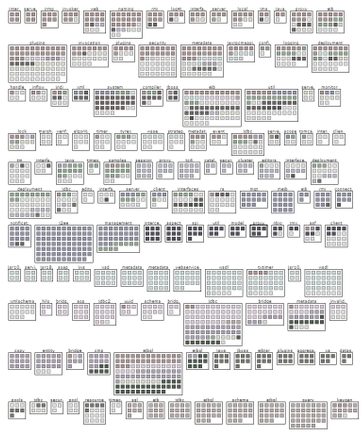
## Data in perspective

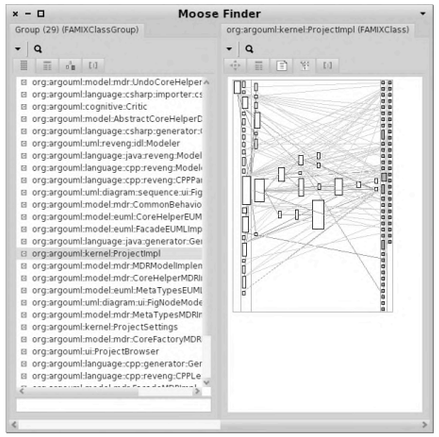



classes select: #isGod

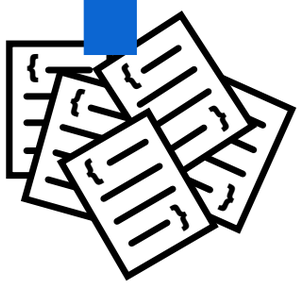
McCabe = 21

LOC = 753,000



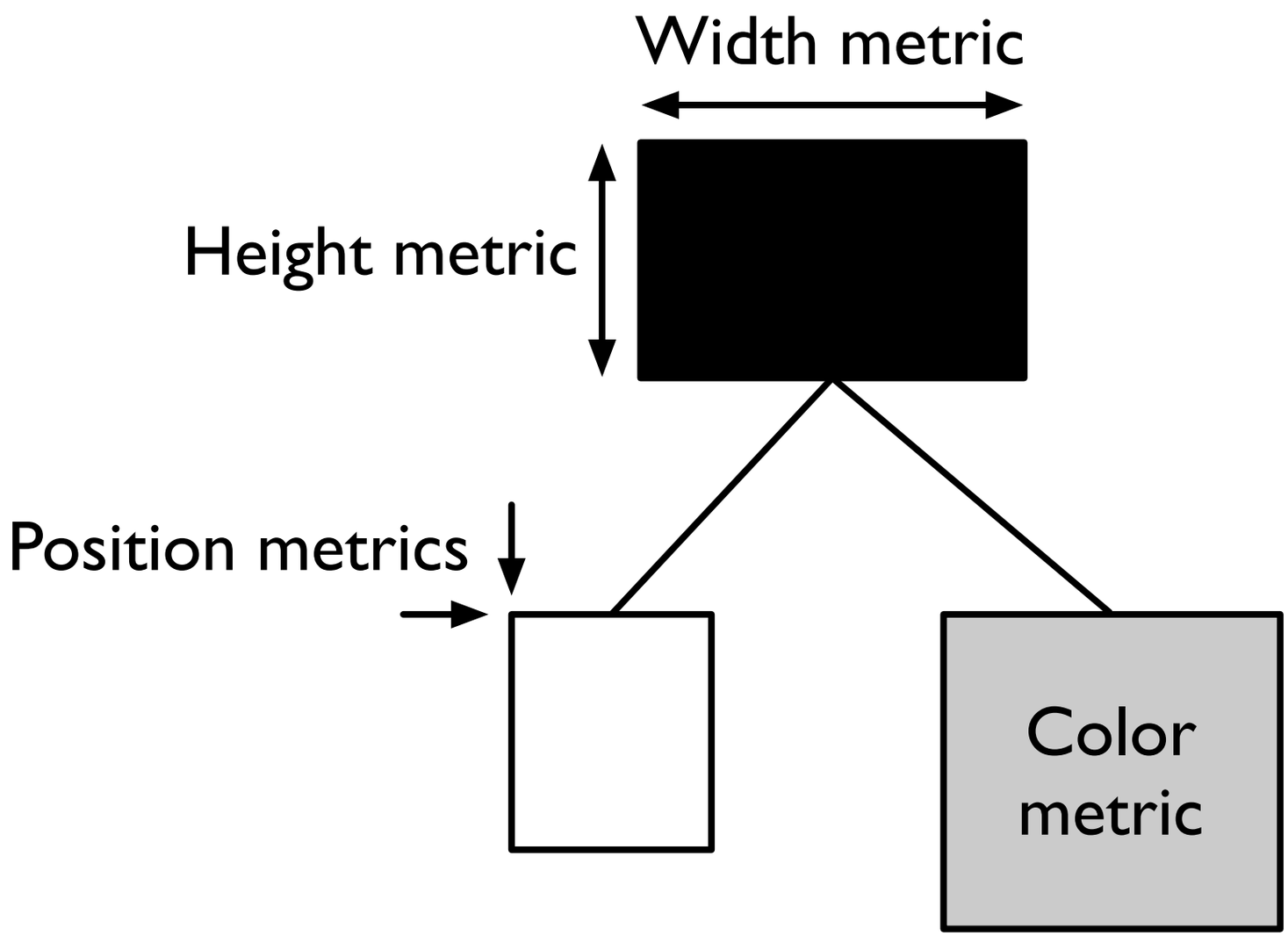



...

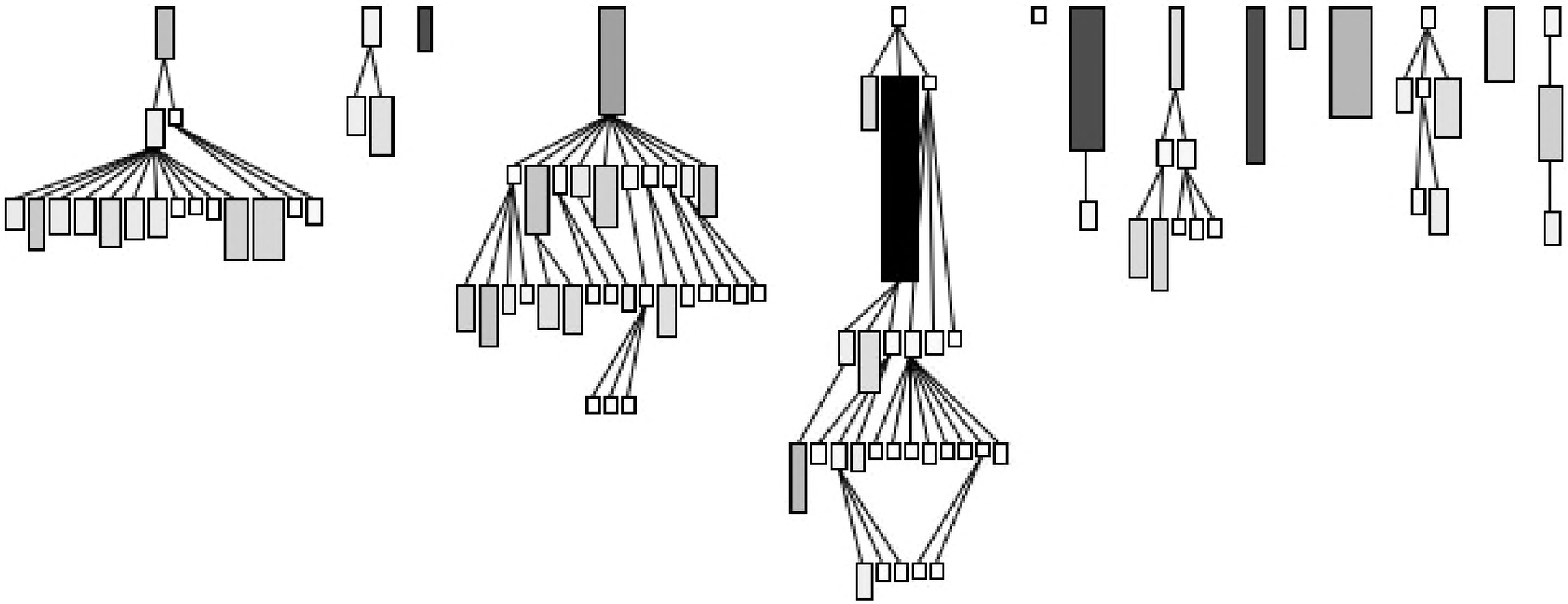


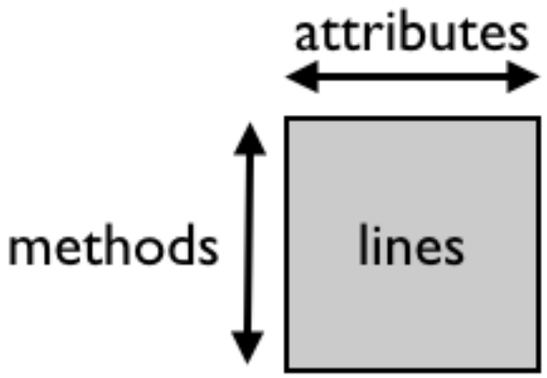
# Some software maps

# Understanding large systems [PhD Lanza]

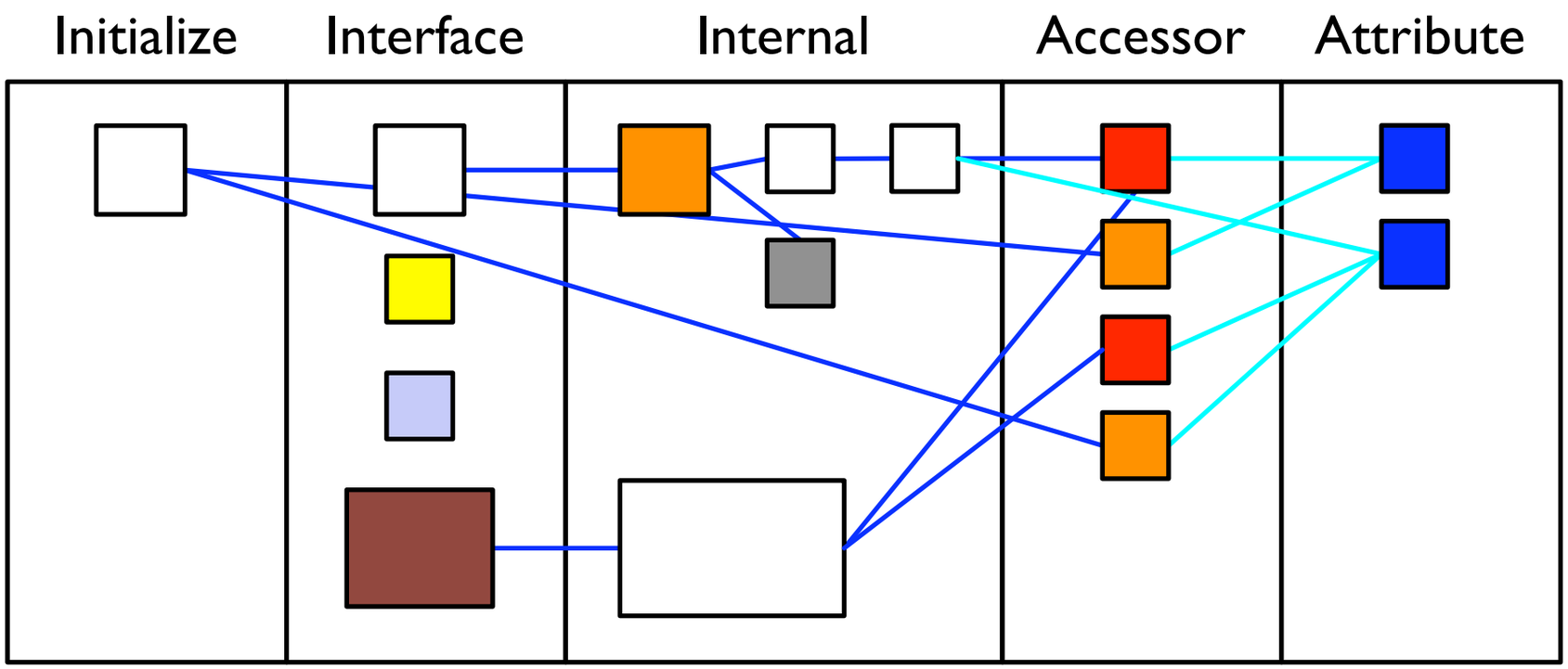


# Understanding systems [PhD M. Lanza]

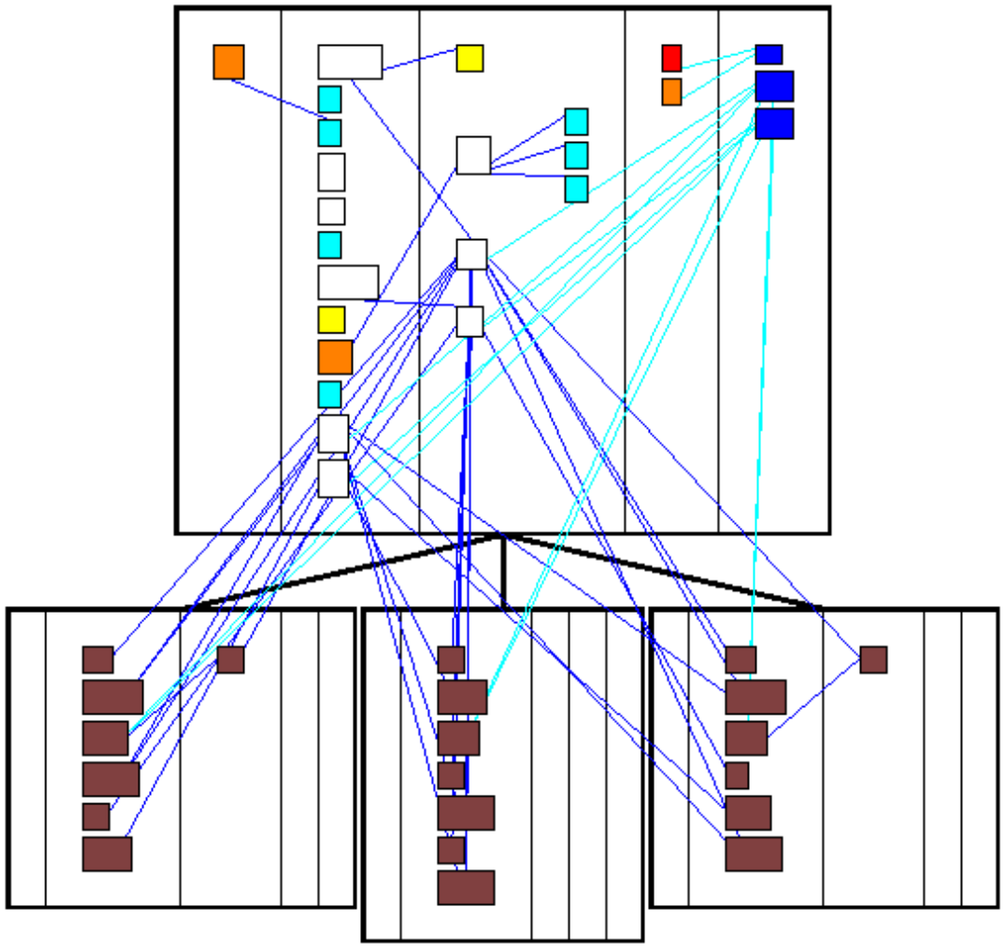


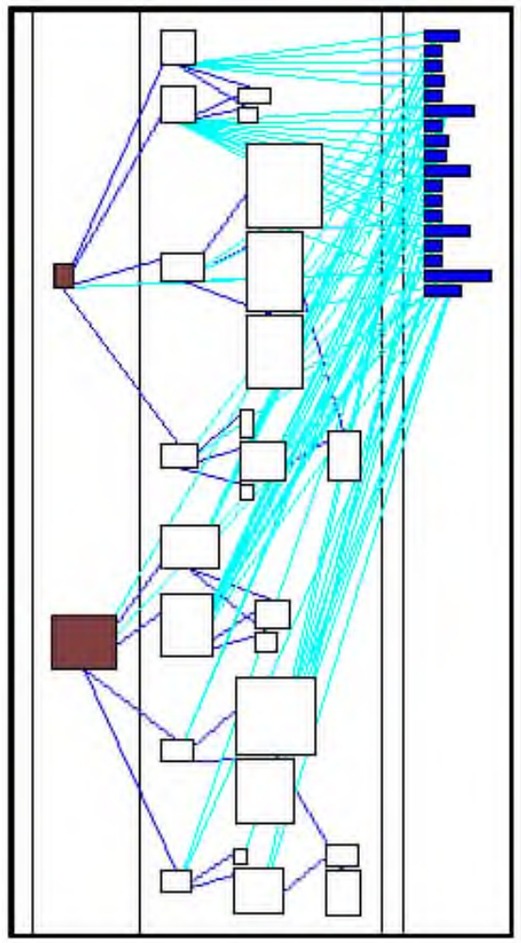


# Understanding classes [PhD M. Lanza]

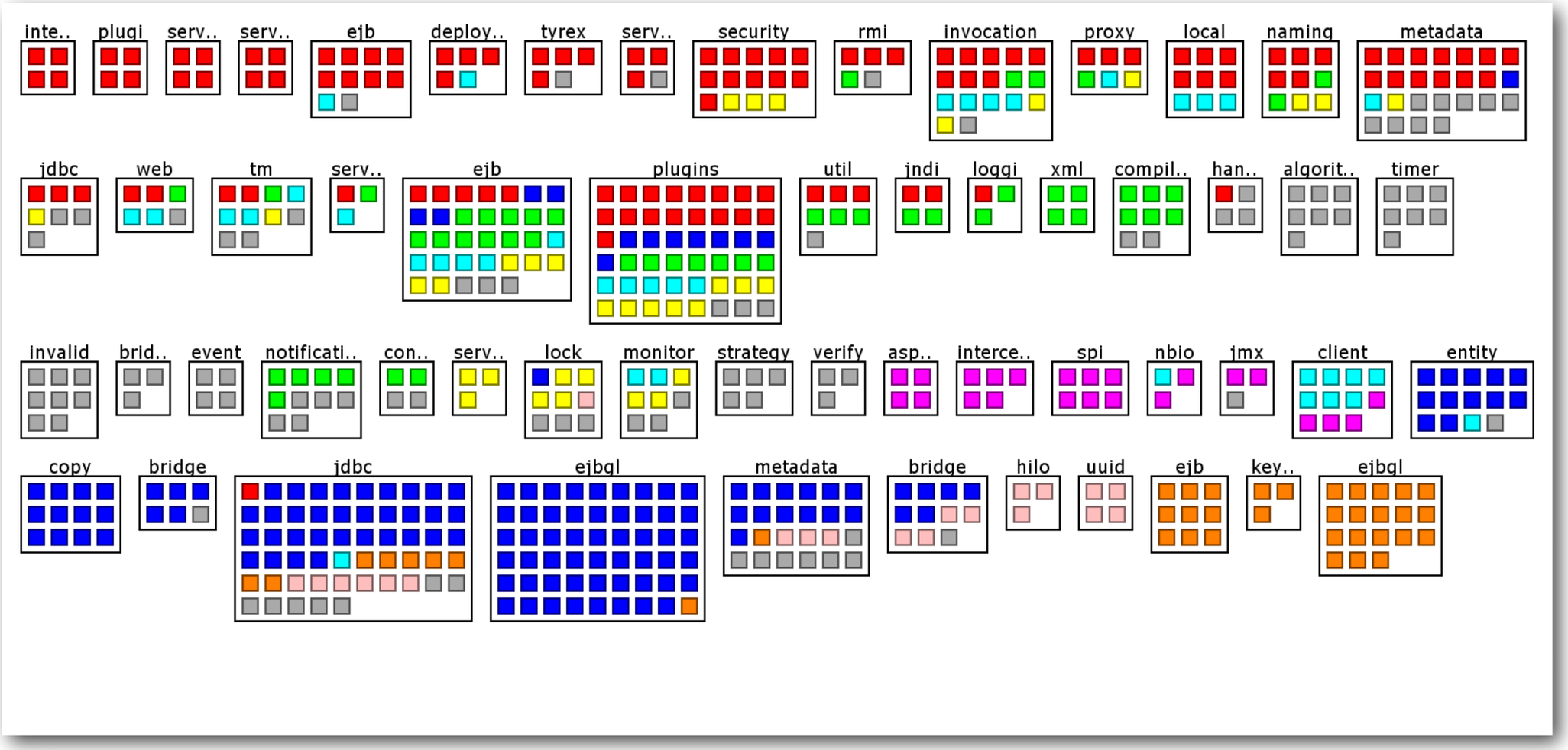


# Understanding classes [PhD M. Lanza]

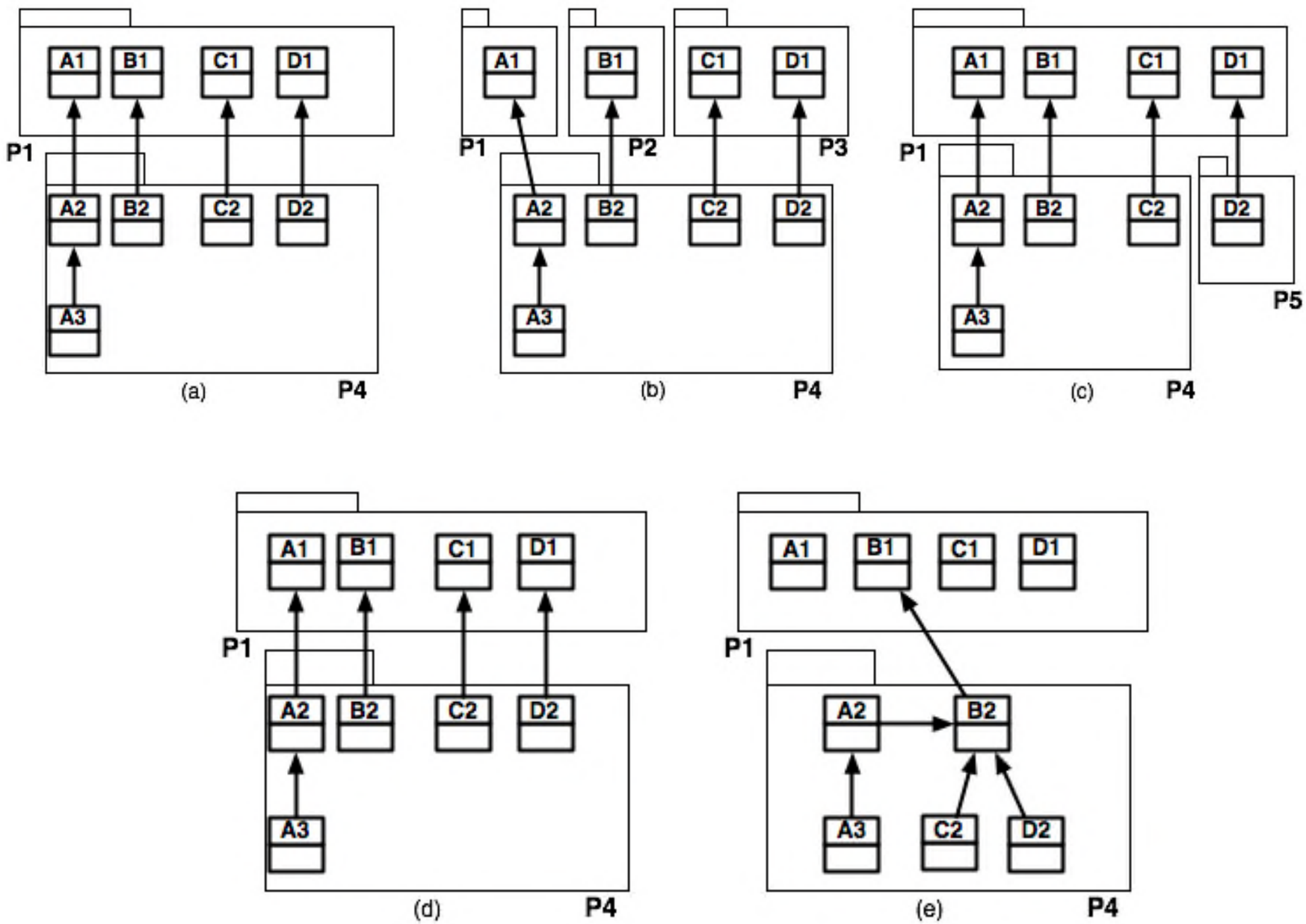


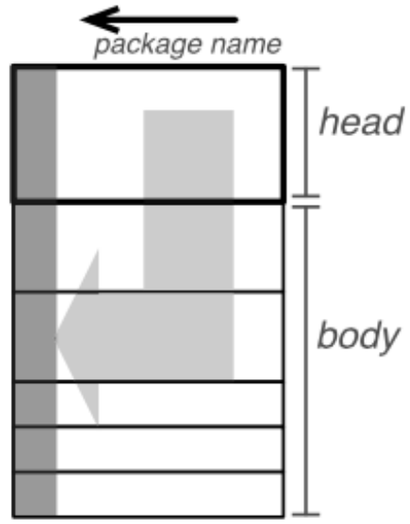


# How a property spread on a system?

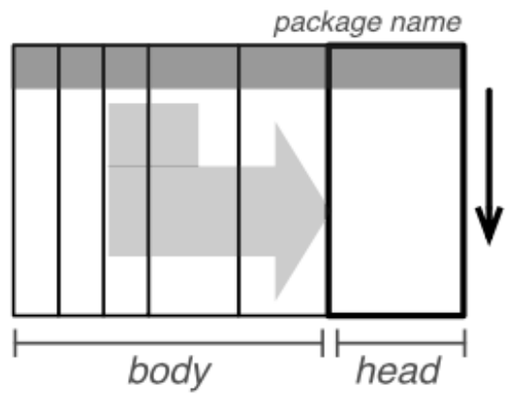


# How to support remodularisation? [PhD Hany Abdeen]

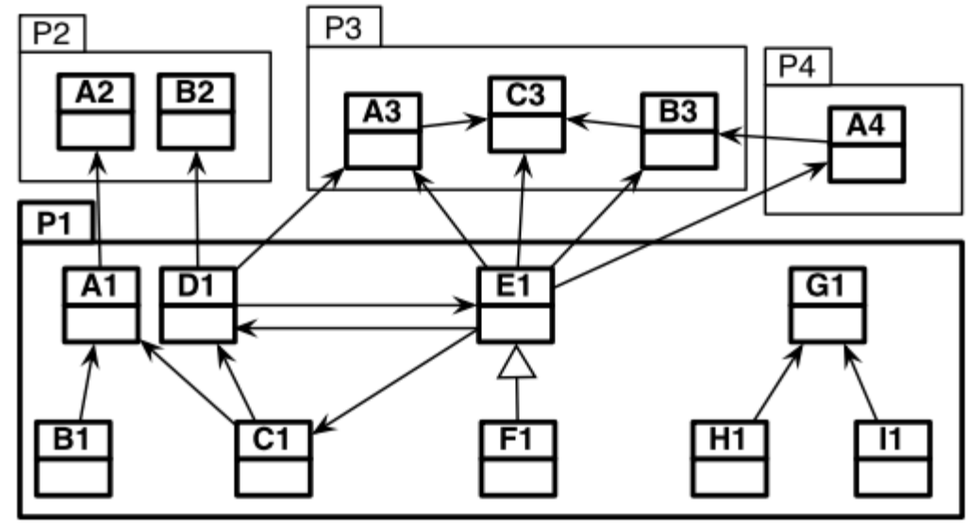


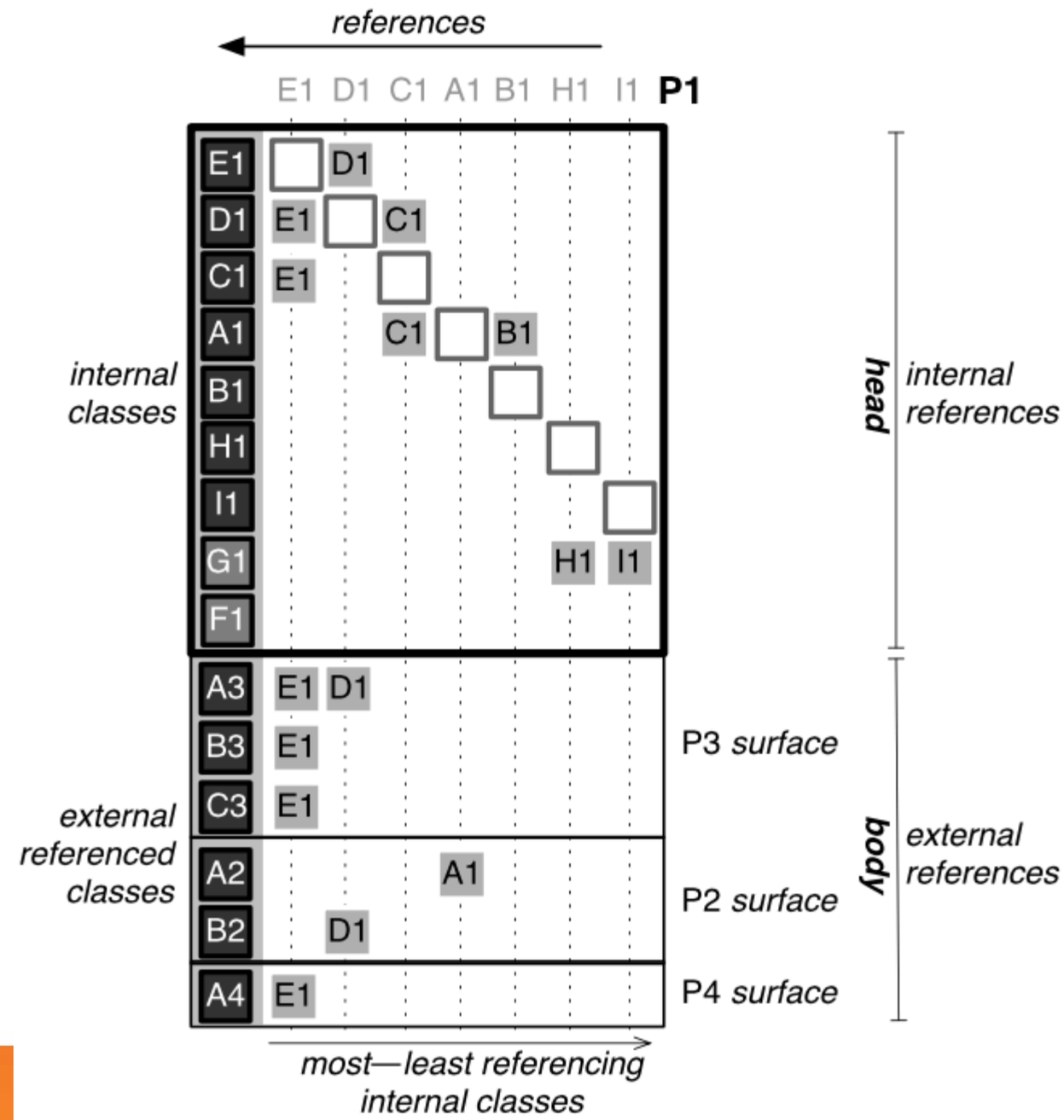



Outgoing references reading



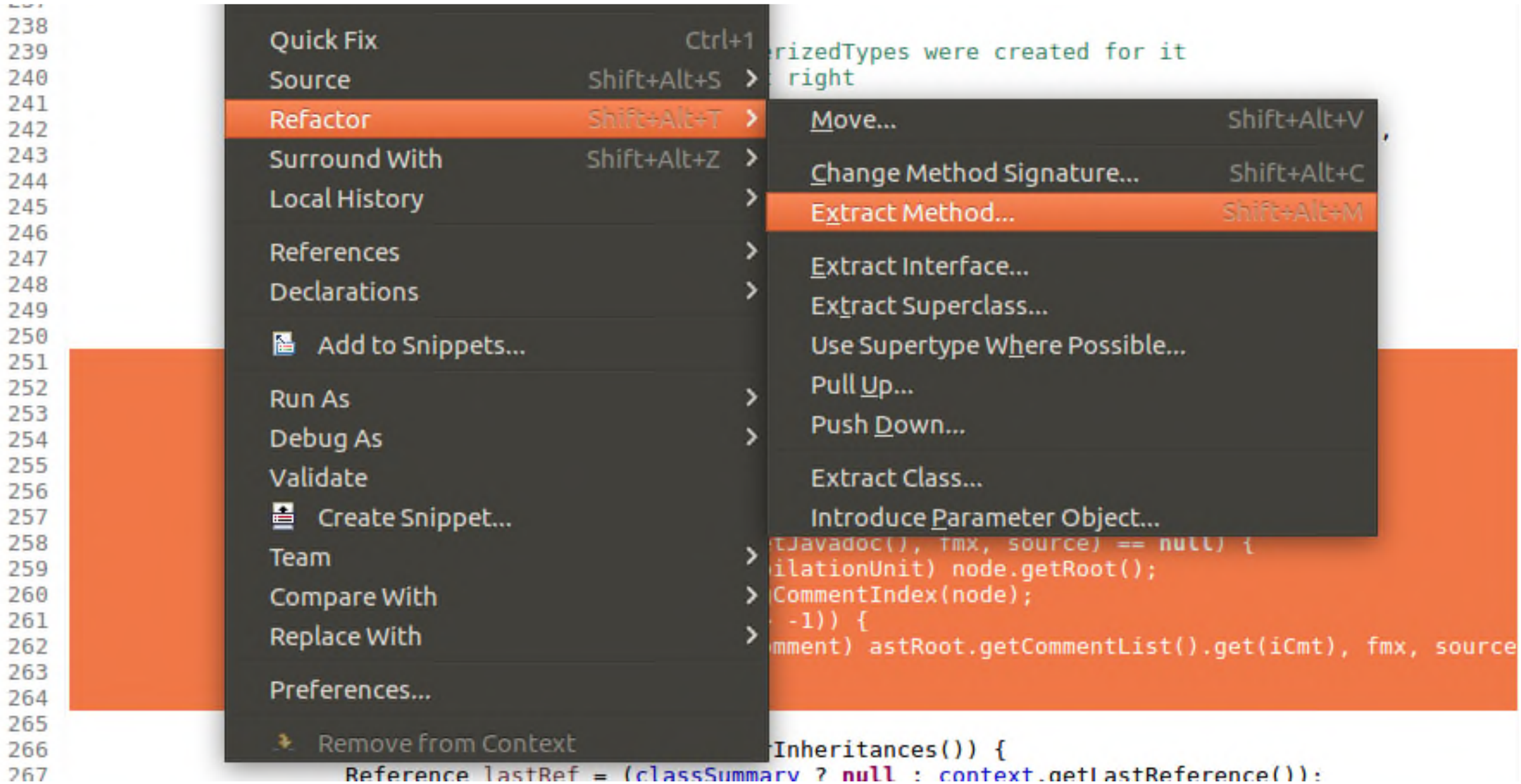

Incoming references reading



# Software Evolution in the Large

# Evolution in the small



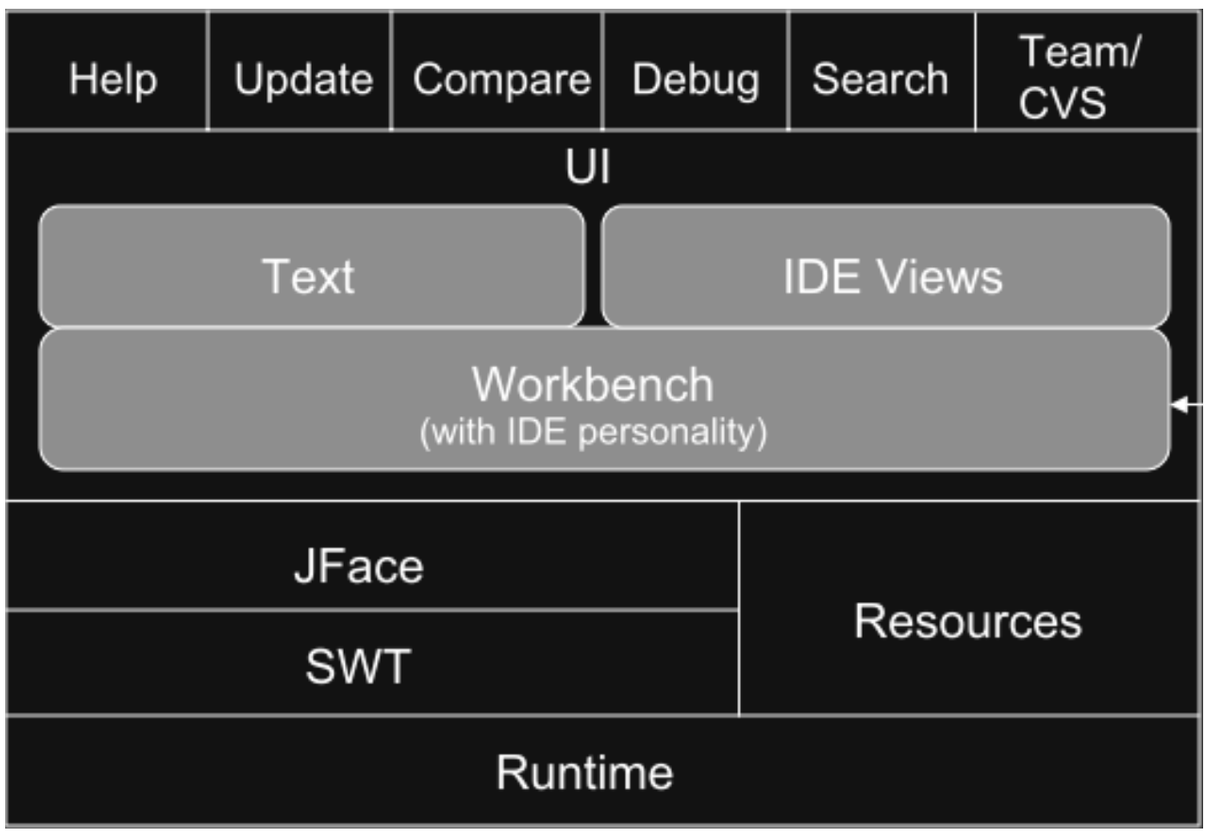
# Evolution in the small

Modification (refactoring)

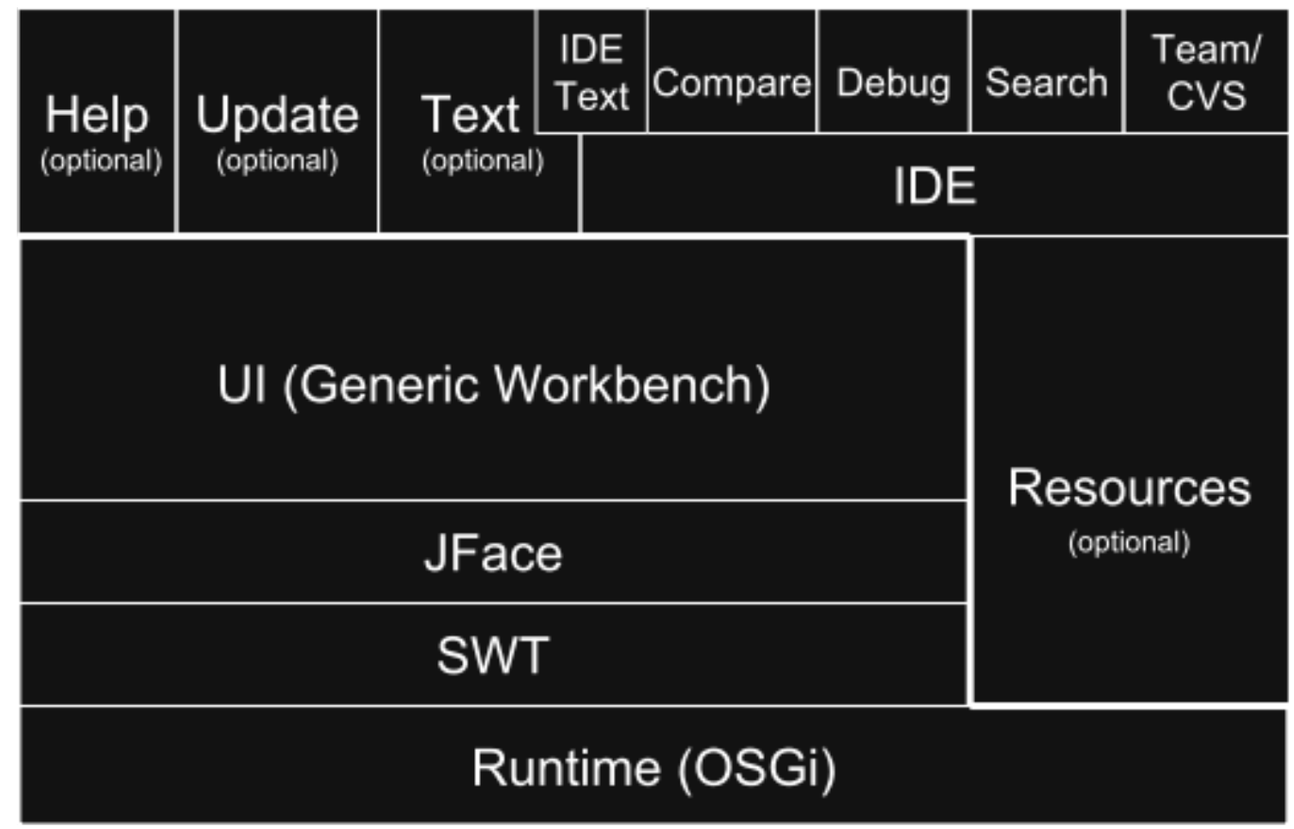
- On 1 entity
- Small number of parameters
- Well defined behavior
- “Preserve code semantic”
- Generic (constrained only by the programming language)

# Evolution in the LARGE

- Eclipse v2.1 → v3.0




v2.1 (Extensible IDE)




v3.0 (Rich Client Platform)

# Evolution in the LARGE

Restructure architecture

Large refactor

Break a big class

Introduce a design pattern (e.g. MVC, Hybernate)

Migrate to a new library version

…

# Evolution in the LARGE

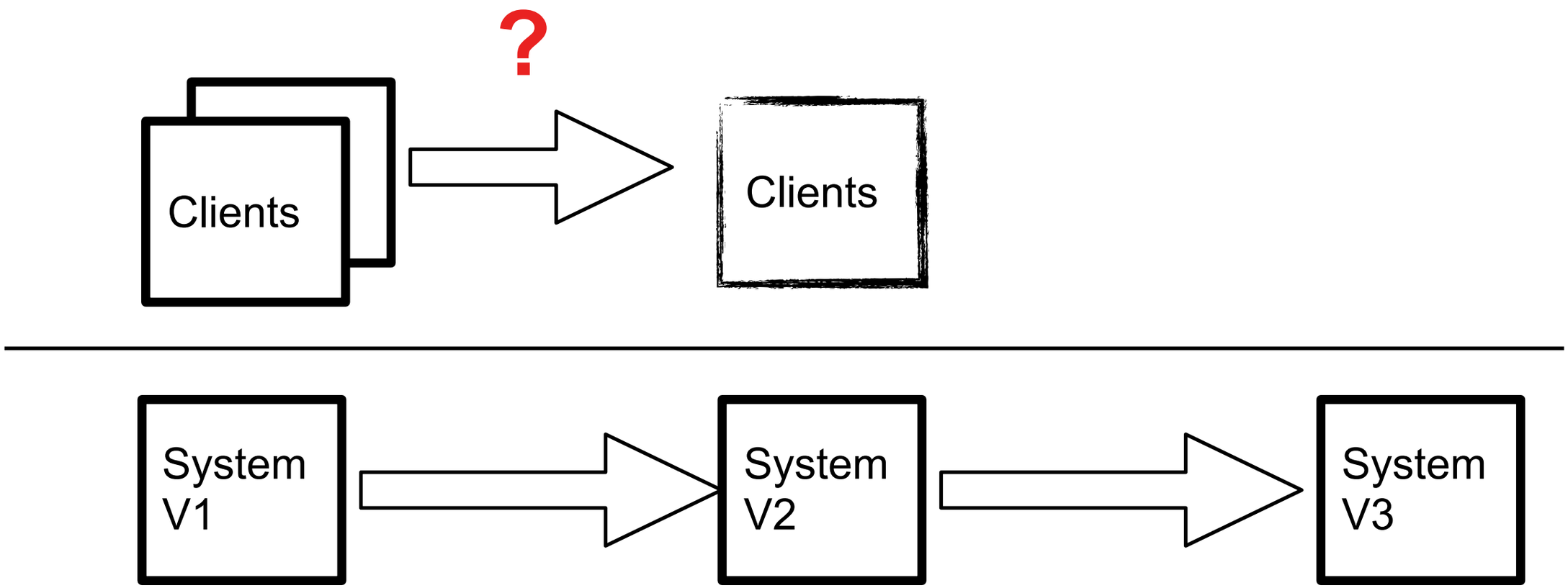
- Can/Should occur regularly (≠often) in the life of a system
- Evolution
  - On several entities
  - Complex behavior
  - Specific to the domain, system, and task
  - May break code semantic temporarily

# Evolution in the large: Automated Rules to support migration

# Mining API Change Rules

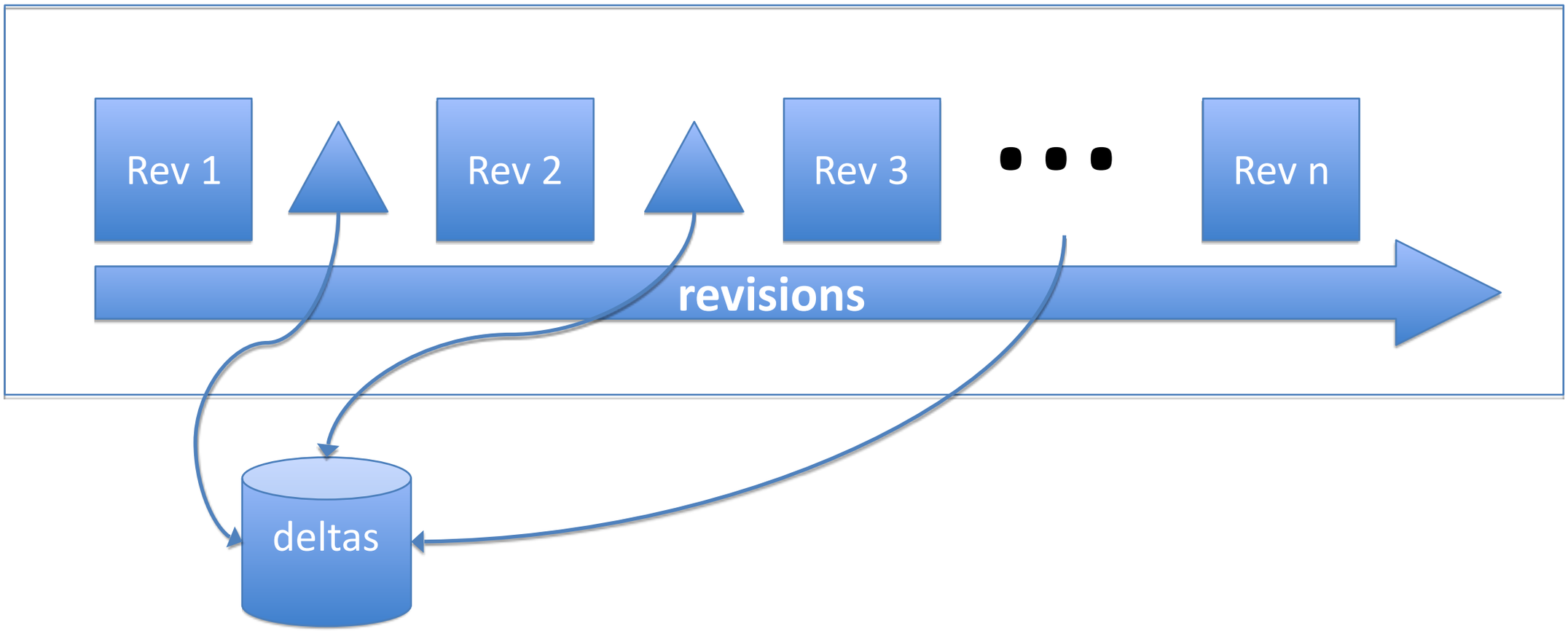
- In Eclipse, 42% of the methods in version 1 were not in v.2
- In Pharo, a simple API change affected thousands of clients
- Clients should not but *do* call internal API

# How to help migrate from version to version?



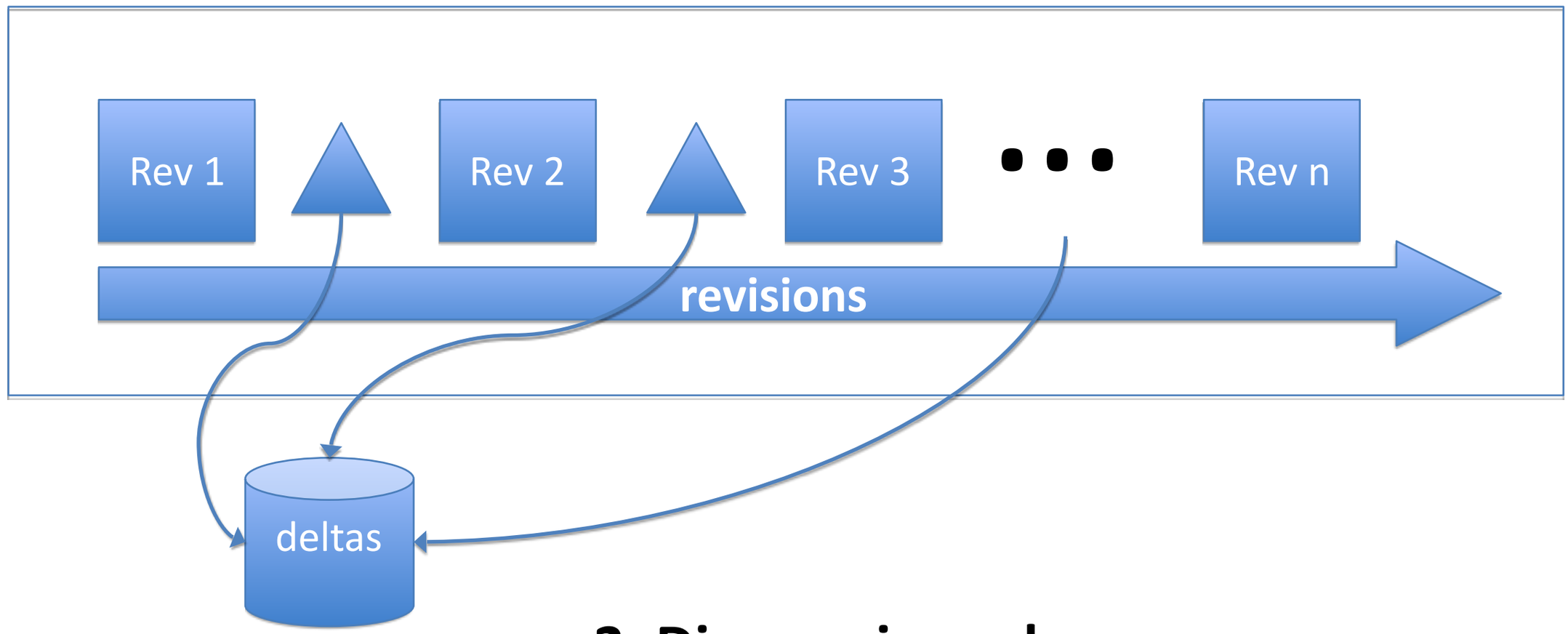
# Mining API Change Rules

## 1. Extracting deltas

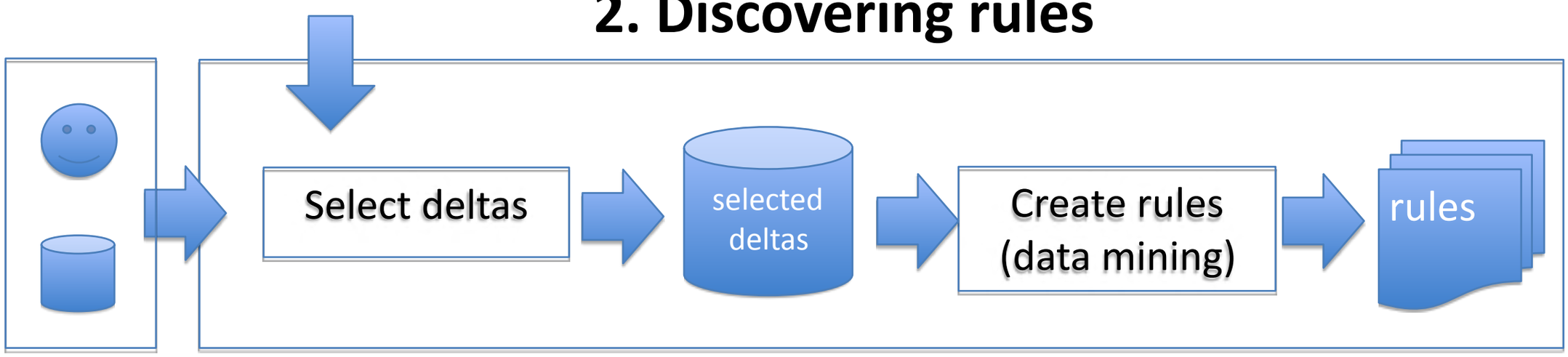


# Mining API Change Rules

## 1. Extracting deltas




## 2. Discovering rules



# Mining API Change Rules

## Format of the changes


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

**Diff of method foo() between version 1 and 2**

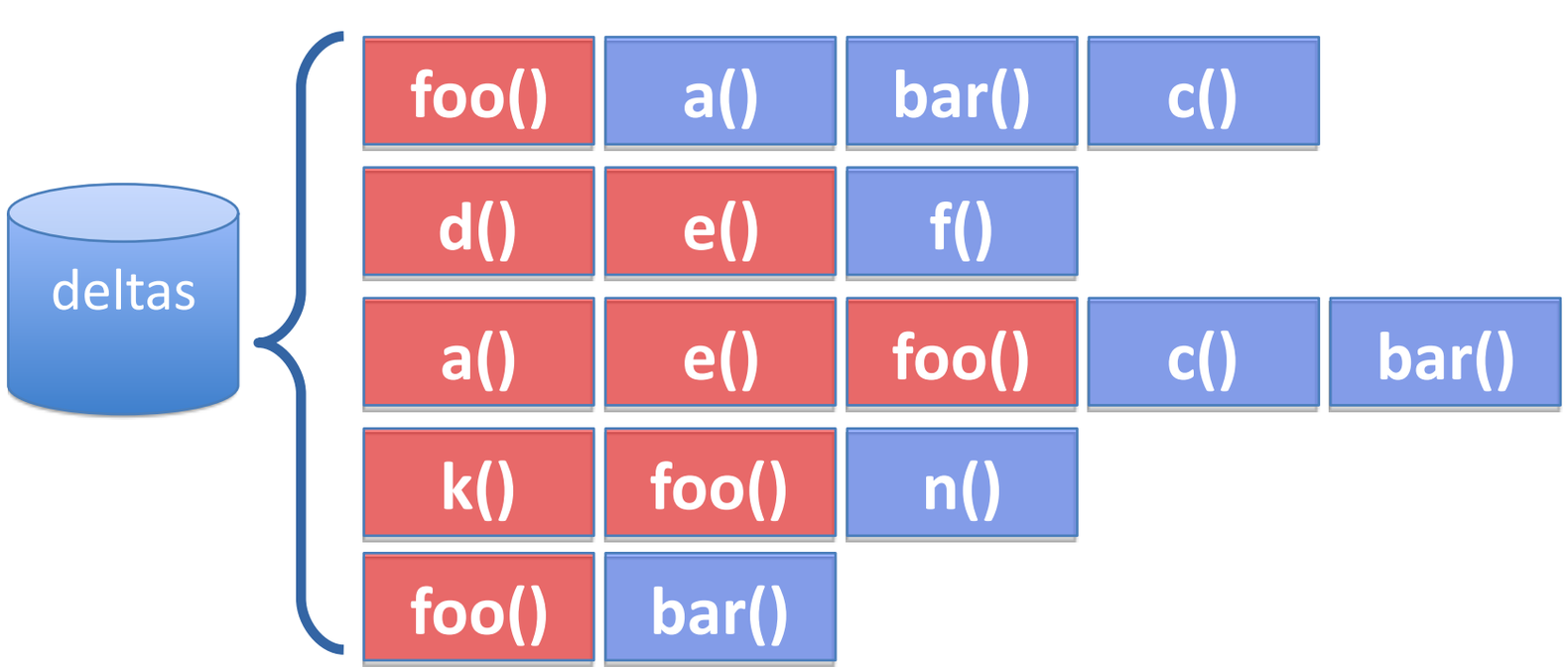
```
– self.add(MooseModel.root().add(model));
+ self.add(model.install());
```

**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()



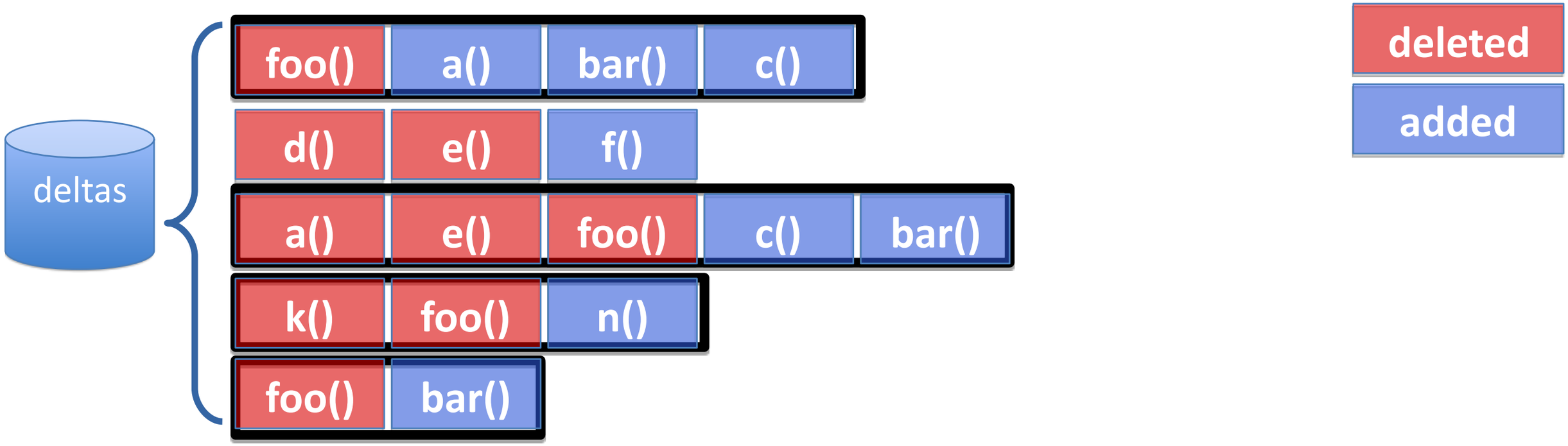

deleted

added

# Mining API Change Rules

Request: foo()

1st step: selecting deltas

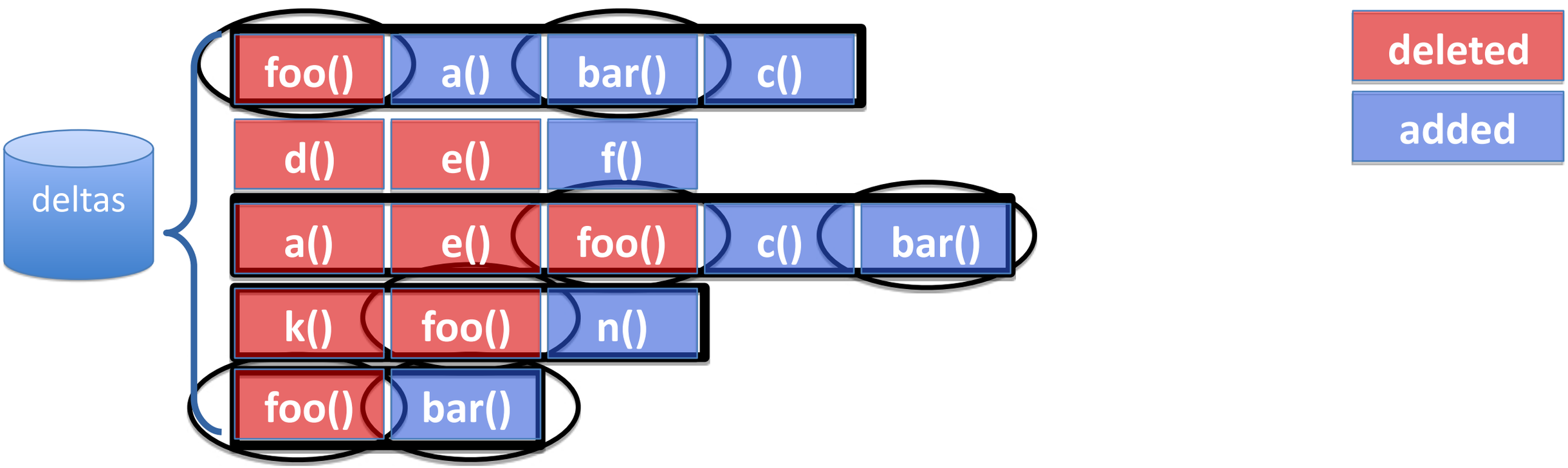


# Mining API Change Rules

Rule: foo() → bar() Confidence = 75%

1st step: selecting deltas

2nd step: discovering rules



# Mining API Change Rules

isNil().ifTrue(*) → ifNil(*)

keys().do(*) → keysDo(*)

intersect(*) → intersectIfNone(*,*)

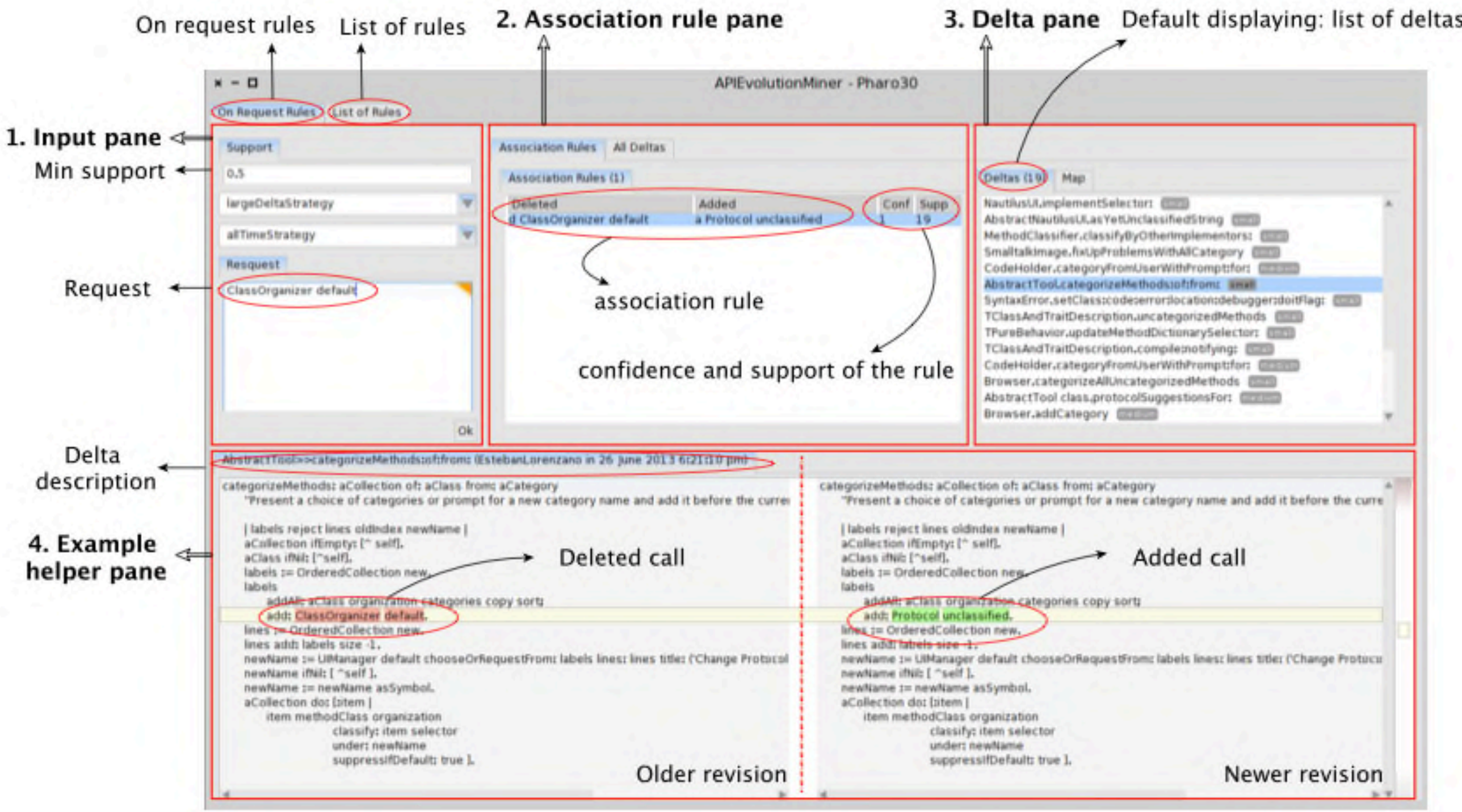
Scanner.new().scanTokens(*) → parseAsLiteralToken()

RegisterAsApplication(*) →

WAAdmin.registerAsApplicationAt(*,*)

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

# Tool Support



# 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

```
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";
  }

  // …
}
```

Repeated 9 times

# 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

## PackageManager 0.58 → 0.59

```
class GreasePharo30CoreSpec

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

```
class GreasePharo30CoreSpec

platformRequirements
    ^ #( #'pharo' )

provisions
    ^ #( #'Grease-Core-Platform' )
```

Repeated 19 times

```
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

| Java systems | Found |
|---|---|
| Eclipse I | 26 |
| Eclipse II | 72 |
| JHotDraw | 9 |
| MyWebMarket | 7 |
| VerveineJ | 3 |

| Pharo systems | Found |
|---|---|
| PetitDelphi | 21 |
| PetitSQL I | 6 |
| PetitSQL II | 98 |
| PackageManager I | 50 |
| PackageManager II | 19 |
| PackageManager III | 64 |
| PackageManager IV | 7 |
| Jenkins | 7 |
| MooseQuery I | 16 |
| MooseQuery II | 8 |

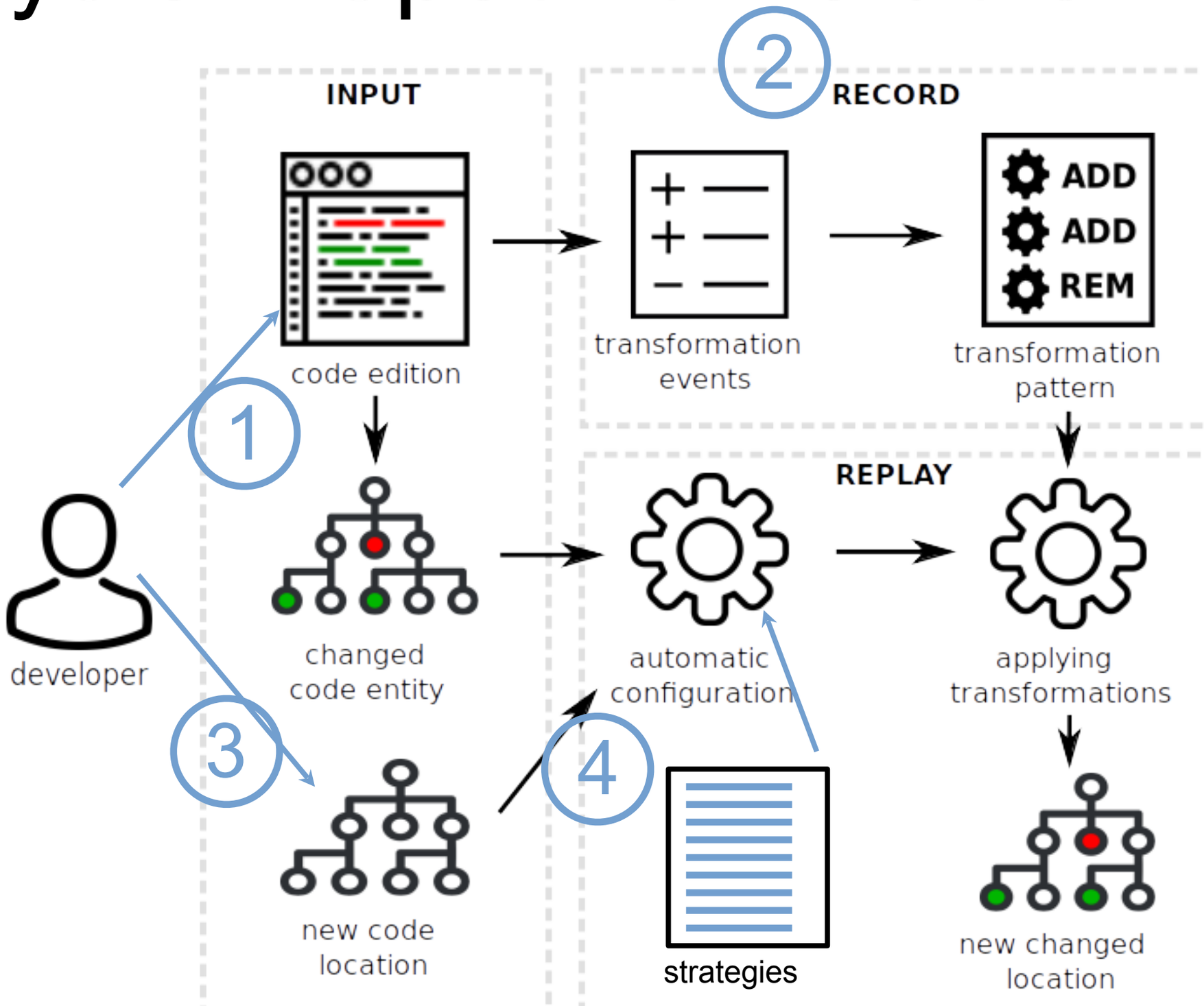
# System Specific Refactorings

- System/task specific
- “Incomplete” (may break semantics)

- Complex
- Tedious
- Error prone

} Automate ?

# System Specific Refactorings

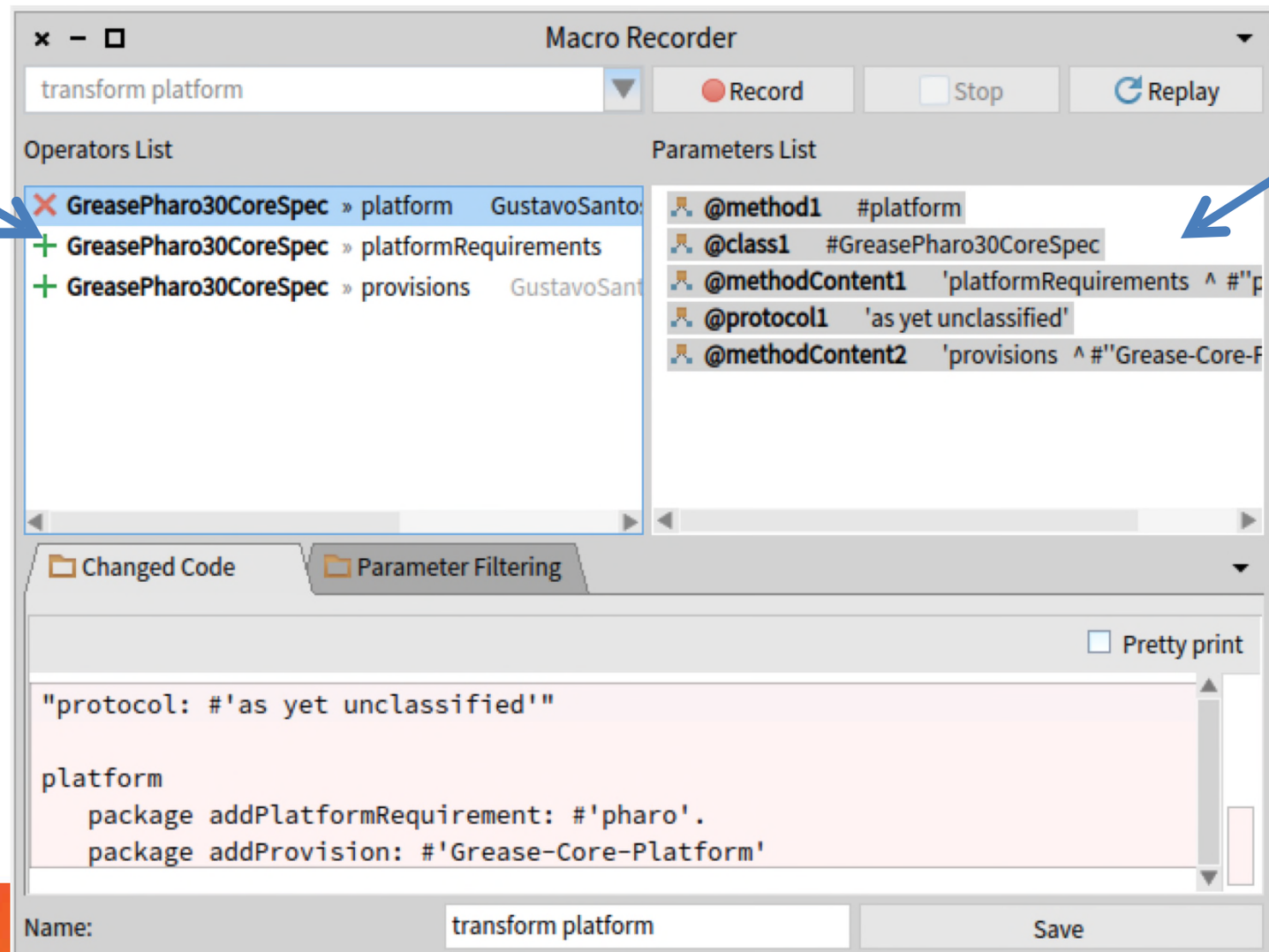


# System Specific Refactorings

- Record

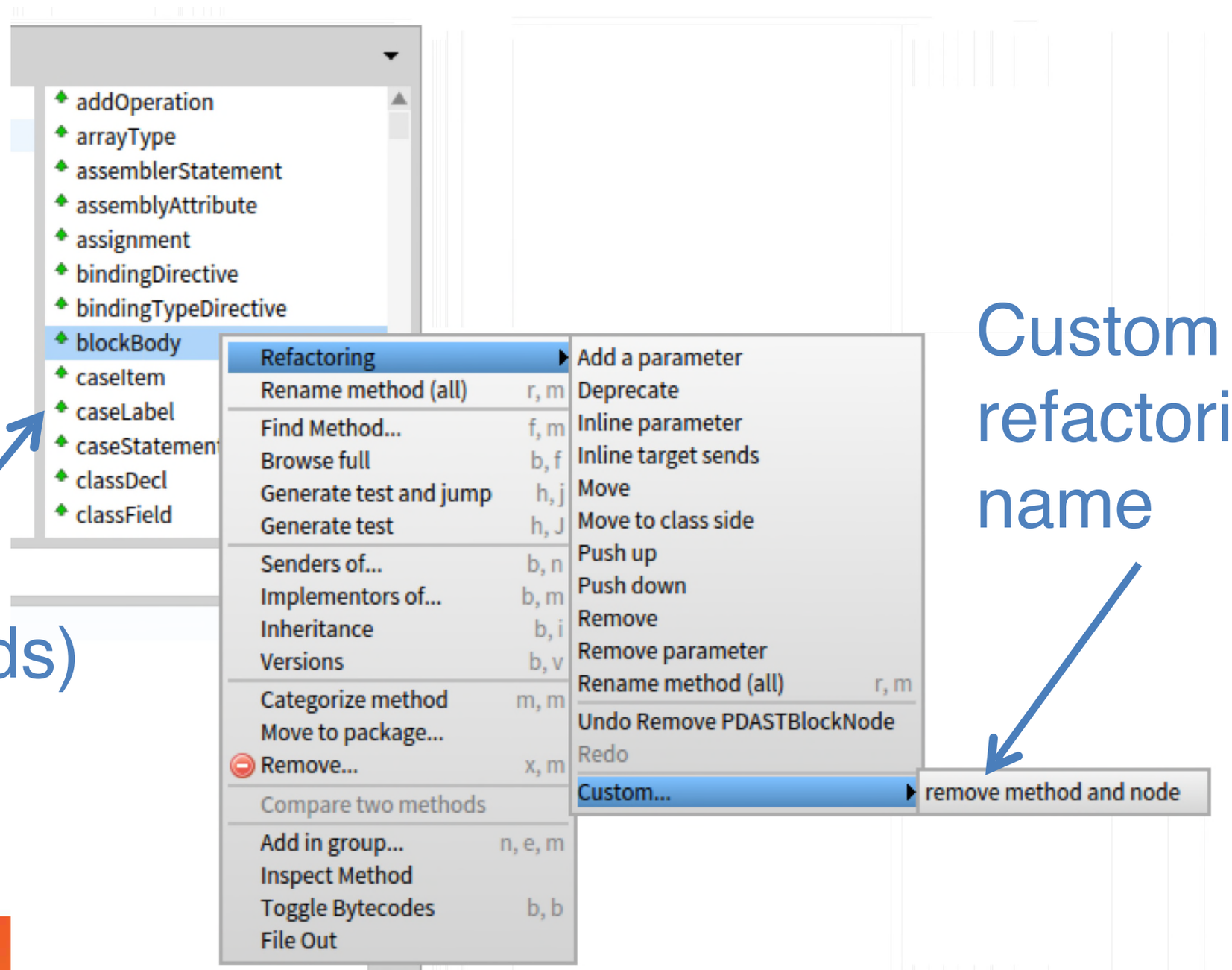
Recorded actions (operators)

Operators' parameters (to adapt)



# *System Specific Refactorings*

- Replay



# System Specific Refactorings

- Case studies

| Pharo systems | #Repetition | #Operators | #Parameters |
|---|---|---|---|
| PetitDelphi | 21 | 2 | 3 |
| PetitSQL I | 6 | 3 | 5 |
| PetitSQL II | 98 | 3 | 6 |
| PackageManager I | 50 | 5 | 4 |
| PackageManager II | 19 | 3 | 5 |
| PackageManager III | 64 | 2 | 4 |
| PackageManager IV | 7 | 4 | 7 |
| Jenkins | 7 | 1 | 3 |
| MooseQuery I | 16 | 1 | 3 |
| MooseQuery II | 8 | 4 | 5 |

# System Specific Refactorings

- “Can the tool configure  parameters?”

| Pharo systems | #repetitions | #Operators | #Parameters | %Correc |
|---|---|---|---|---|
| PetitDelphi | 21 | 2 | 3 | 99% |
| PetitSQL I | 6 | 3 | 5 | 67% |
| PetitSQL II | 98 | 3 | 6 | 83% |
| PackageManager I | 50 | 5 | 4 | 29% |
| PackageManager II | 19 | 3 | 5 | 54% |
| PackageManager III | 64 | 2 | 4 | 72% |
| PackageManager IV | 7 | 4 | 7 | 79% |
| Jenkins | 7 | 1 | 3 | 49% |
| MooseQuery I | 16 | 1 | 3 | 36% |
| MooseQuery II | 8 | 4 | 5 | 28% |

Arc

# System Specific Refactorings

- Efficiency depends on type of operator parameter

| Param. type | Correct | Incorrect | %Correct |
|---|---|---|---|
| Class | 21 | 3 | 77% |
| Method | 6 | 5 | 98% |
| Pragma (*annotation*) | 98 | 6 | 13% |
| Prototype | 50 | 4 | 99% |
| Source code | 19 | 5 | 9% |
| variable | 64 | 4 | 100% |

# Evolution in the large: GWT to angular

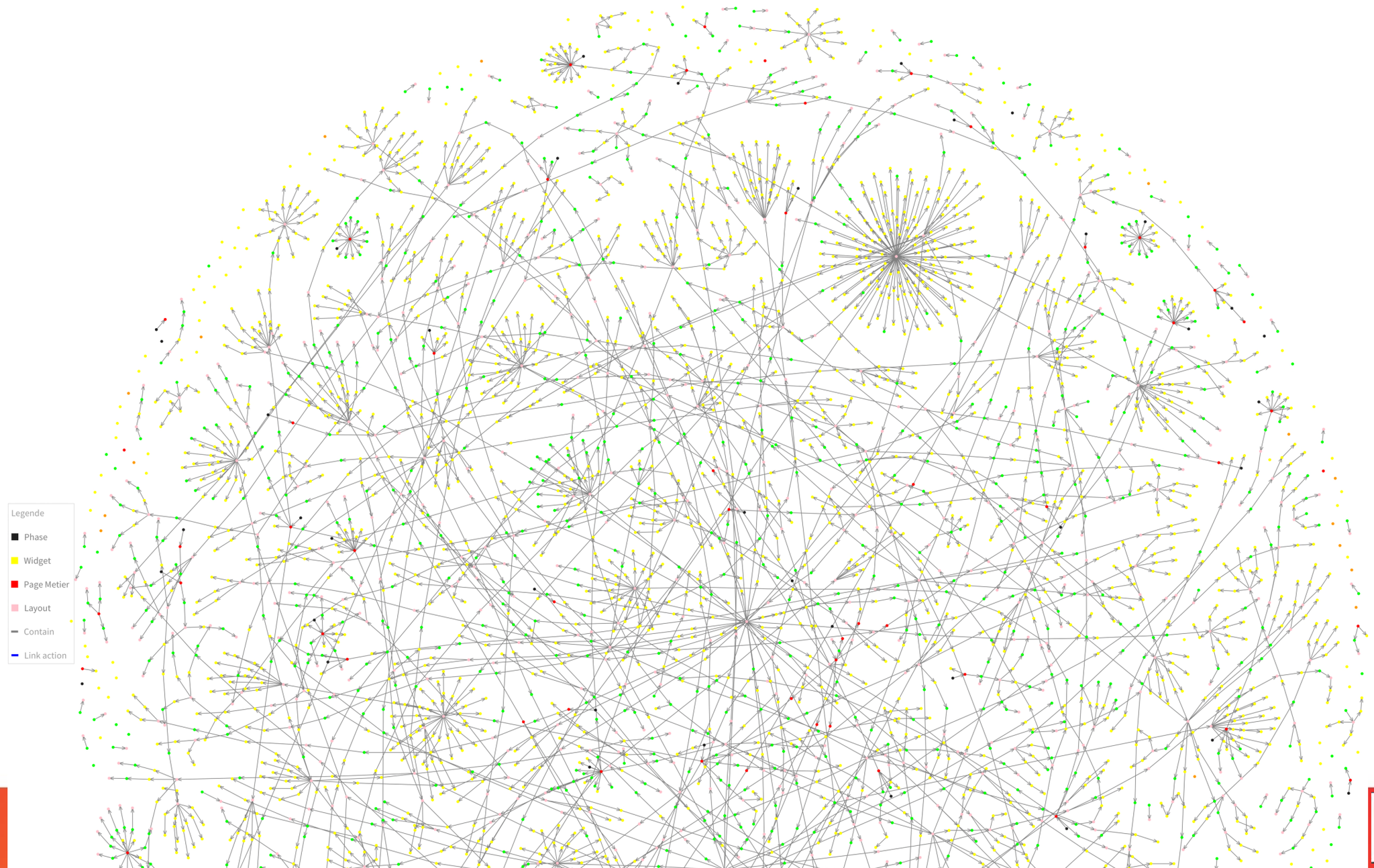
PhD CIFRE of Benoit Verhaeghe for Berger-Levrault

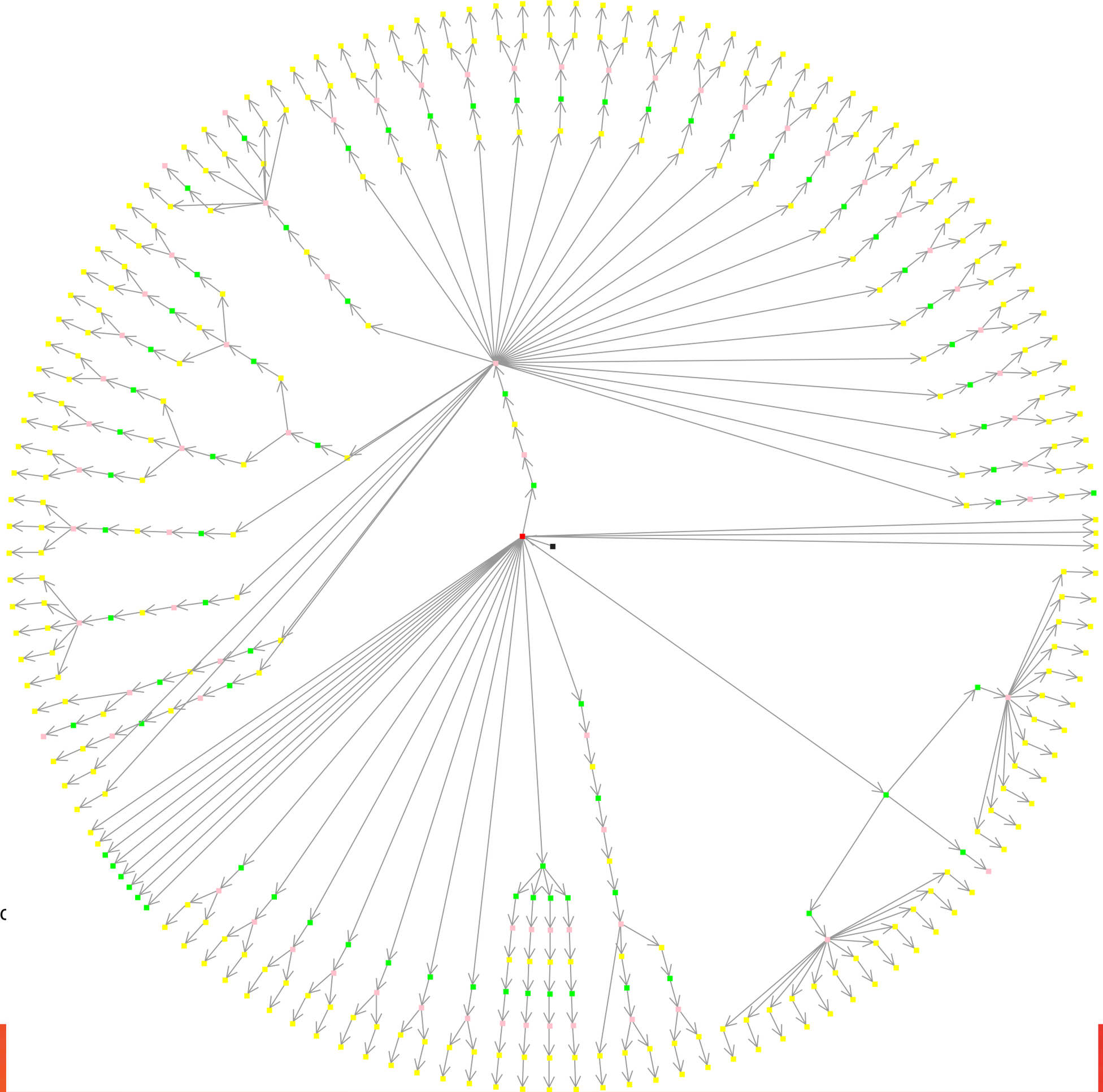
# GWT Typical Application

- 26 000 Classes
- 450 web pages
- 974 000 invocations
- 1 063 163 LOC (UI)

# GWT big kitchen (bac a sable)

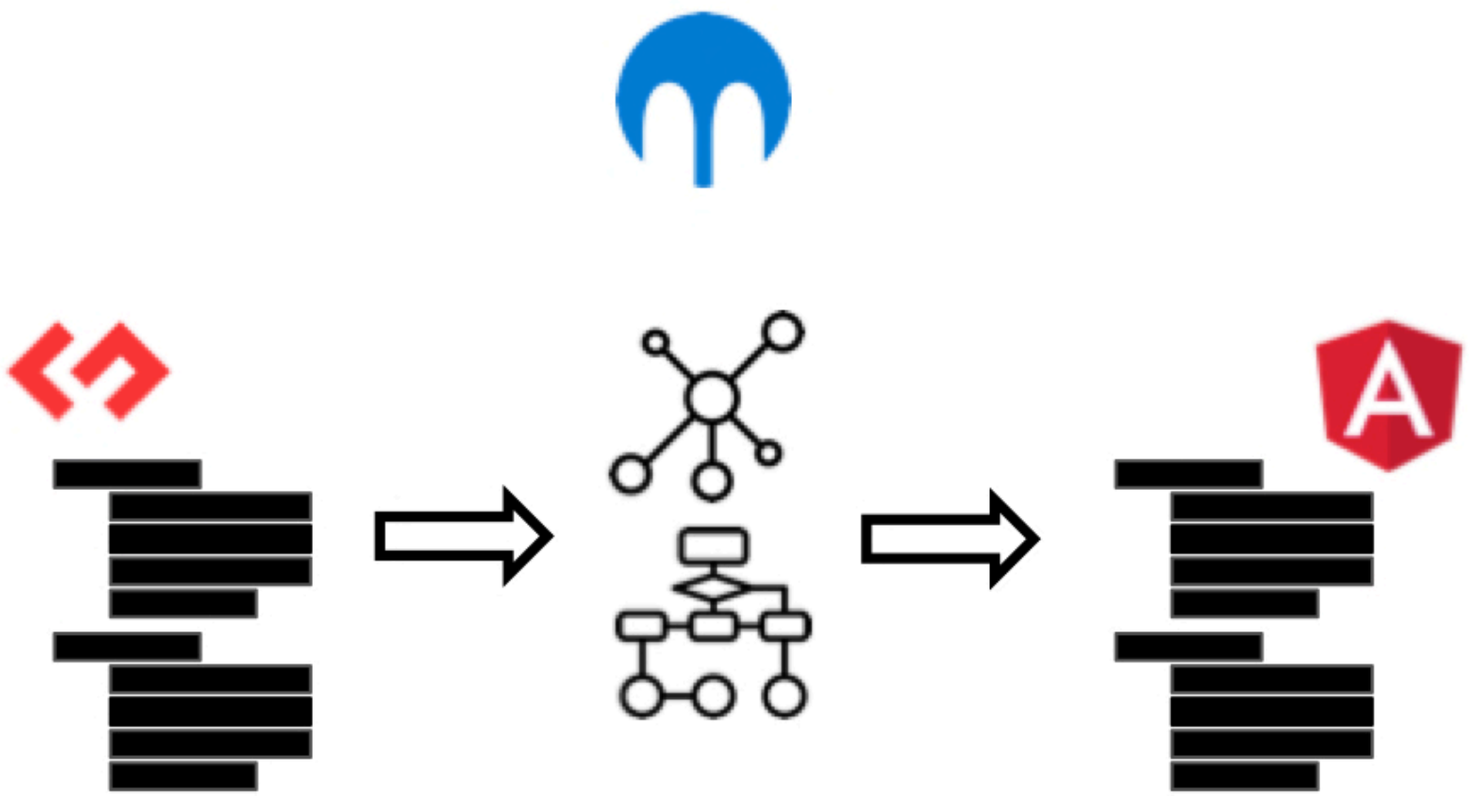
bigKitchen ~ 8979 éléments



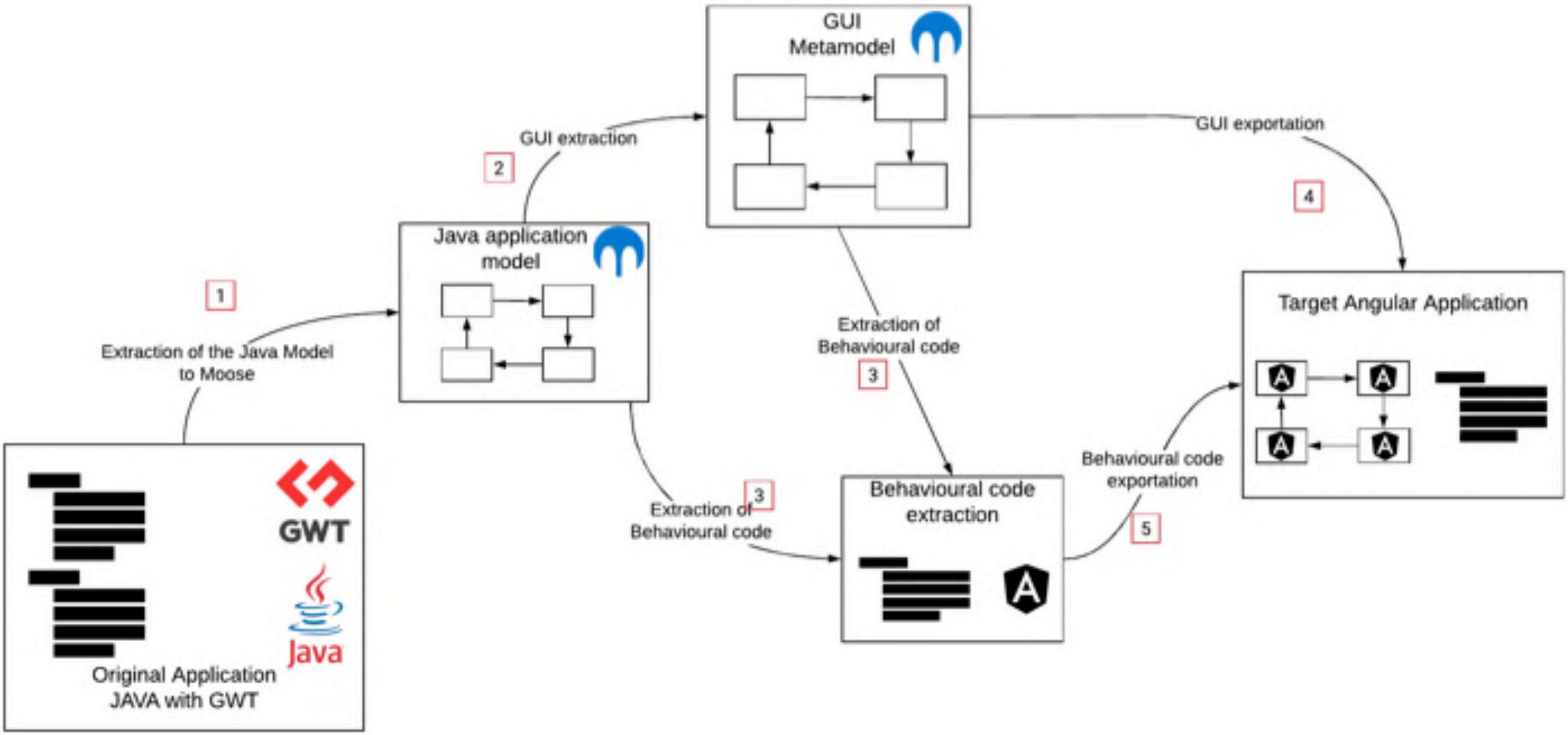


Noir-> des phase/page web
Rouge -> Business Page (grc
Gris -> Layout
Vert -> Widget container
Jaune -> Widget leaf

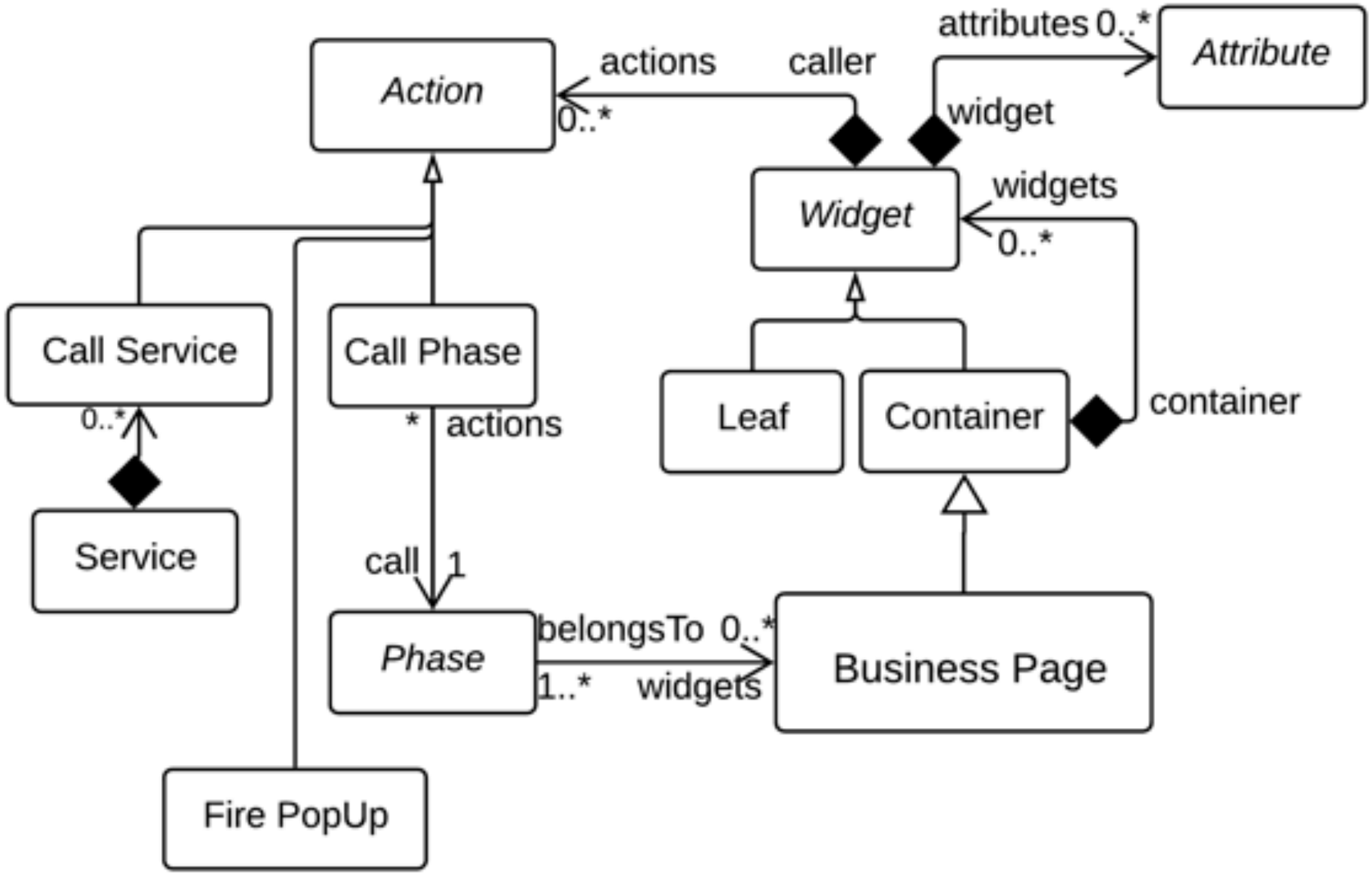
# Approach



# Approach


GUI
Metamodel
GUI extraction
2
GUI exportation
4
Java application
model
1
Extraction of the Java Model
to Moose
Extraction of
Behavioural code
3
Target Angular Application
Extraction of
Behavioural code
3
Behavioural code
extraction
Behavioural code
exportation
5
GWT
Java
Original Application
JAVA with GWT

# Metamodel


Action
actions
0..*
caller
attributes 0..*
Attribute
widget
Widget
widgets
0..*
Call Service
Call Phase
0..*
*
actions
Leaf
Container
container
Service
call
1
Phase
belongsTo 0..*
1..*
widgets
Business Page
Fire PopUp

# Results

Export

Navigation par phase

- Ouvrir un onglet
- Ouvir une boite de dialogue modale
- Ouvrir une boite de dialogue non modale
- Ouvrir une boite de dialogue unique non modale

Boites de dialogue (hors phase)

- Boite de dialogue modale
- Boite de dialogue non modale
- Message d'information
- Message d'avertissement
- Message d'erreur (métier)
- Message de confirmation
- Message d'erreur (exception)

Navigation par phase

- Ouvrir un onglet
- Ouvir une boite de dialogue modale
- Ouvrir une boite de dialogue non modale
- Ouvrir une boite de dialogue unique non modale

Boites de dialogue (hors phase)

- Boite de dialogue modale
- Boite de dialogue non modale
- Message d'information
- Message d'avertissement
- Message d'erreur (métier)
- Message de confirmation
- Message d'erreur (exception)

# Results

Export

**Etiquettes formattées (pour les listes)**

| | |
|---|---|
| Montant : | 10 000 000,00 € |
| Pourcentage : | 1,50% |
| Booléen (1) : | Oui |
| Booléen (0) : | Non |
| Date : | 26/06/2018 |
| Durée : | 3 ans, 2 mois et 1 jour |
| Enumération : | jour(s) |
| Entier : | 999999999 |
| Entier long : | 999999999999999999 |
| Entier short : | 999 |

**Etiquettes formattées (pour les listes)**

Montant :Pourcentage :Booléen (1) :Booléen (0) :Date :Durée :Enumération :Entier :Entier long :Entier short :

# Future work

Behavior code model

Specific representation of layout

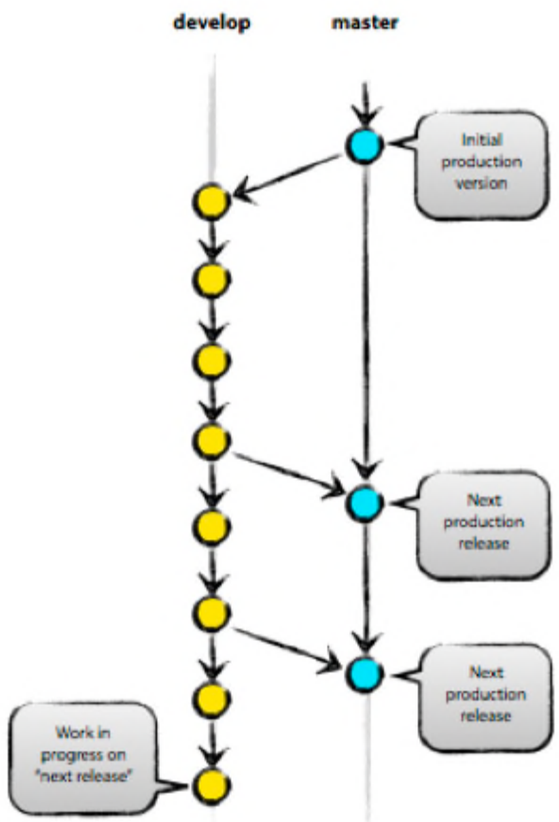
Extract complex structure (such table data structure)

How to evaluate GUI migration?

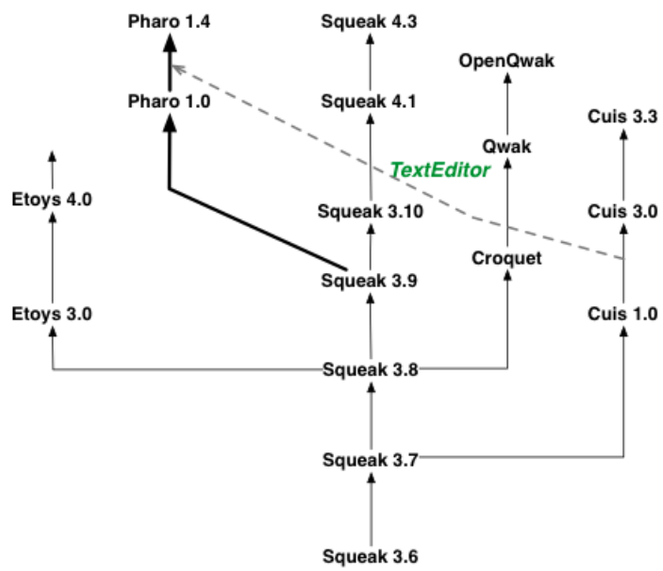
# How can we help merging?

## What is the impact of a change?
## PhD of Veronica Uquillas (Paid VUB)

# How to support merging branches?




Git




Forks

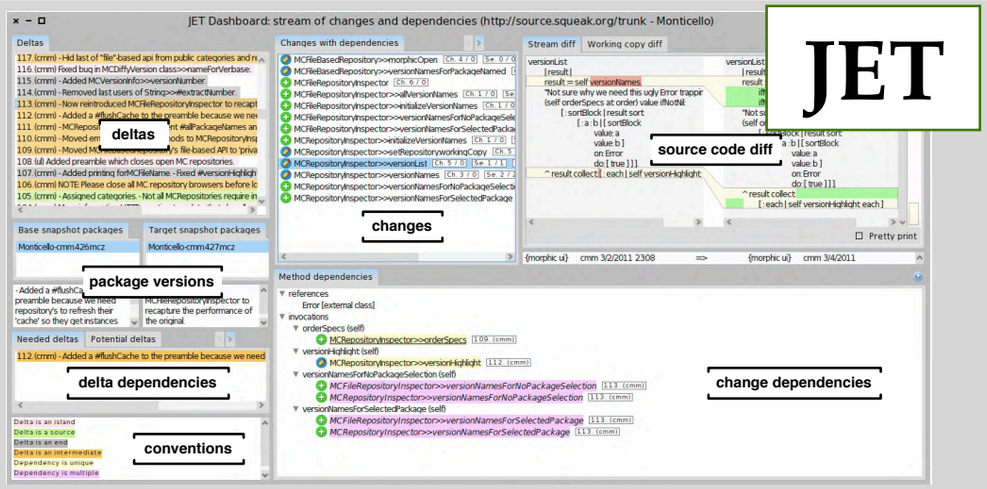
- Manual tasks are needed
- Dependencies between changes
- Integrator is not the author of the changes
- No guarantee that the system will work

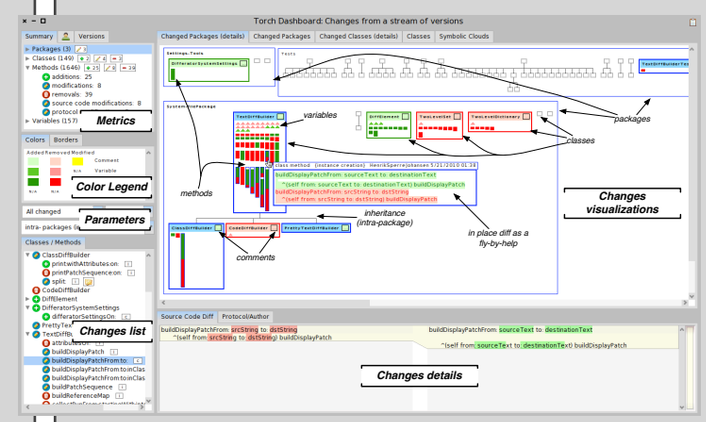
# Assisted Integration

Approach Overview

## Stream of changes (chains of commits)

## Single delta (commit)



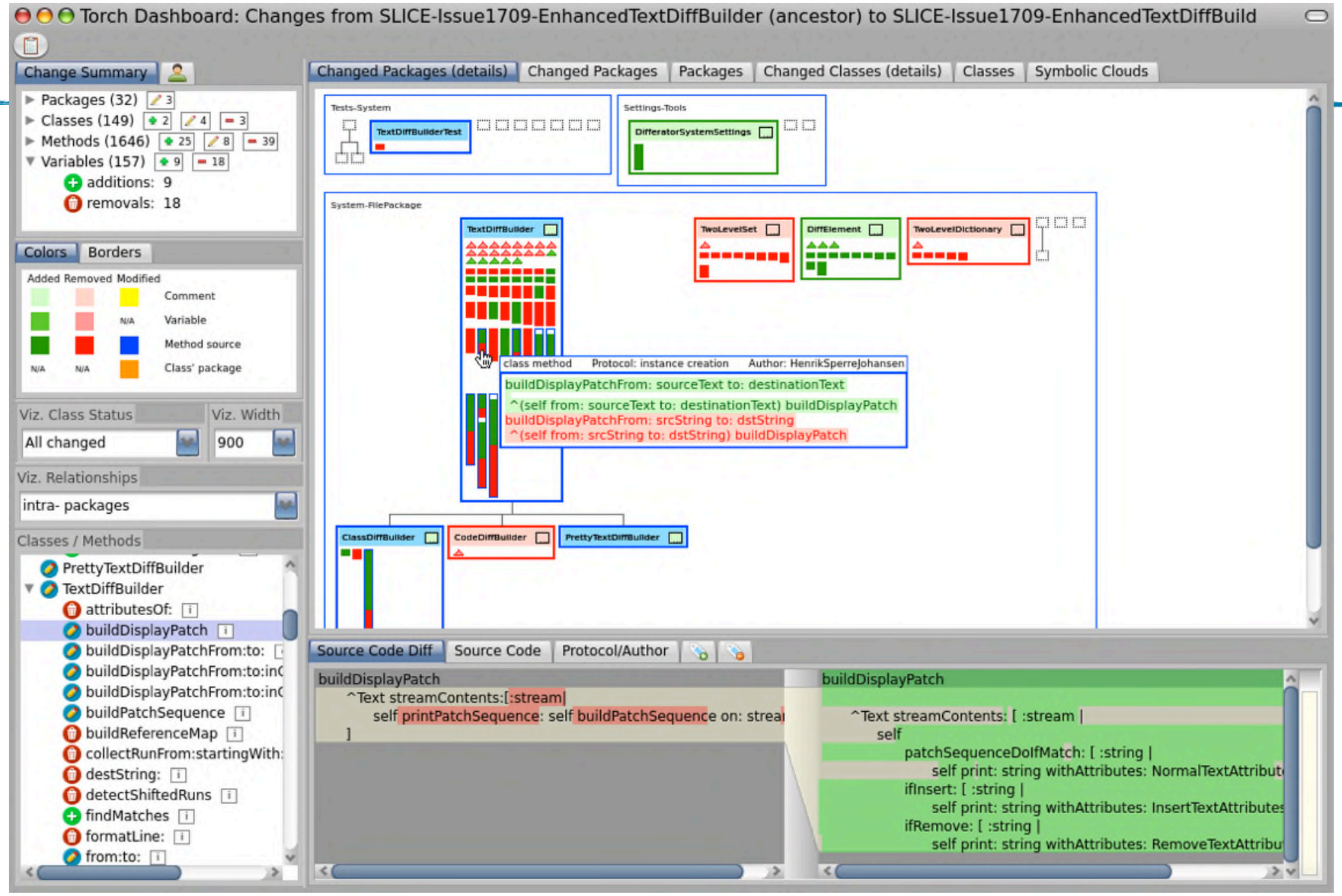

**Change & Dependency Meta-Model and Analyses (RingC)**

**History Meta-Model and Analyses**

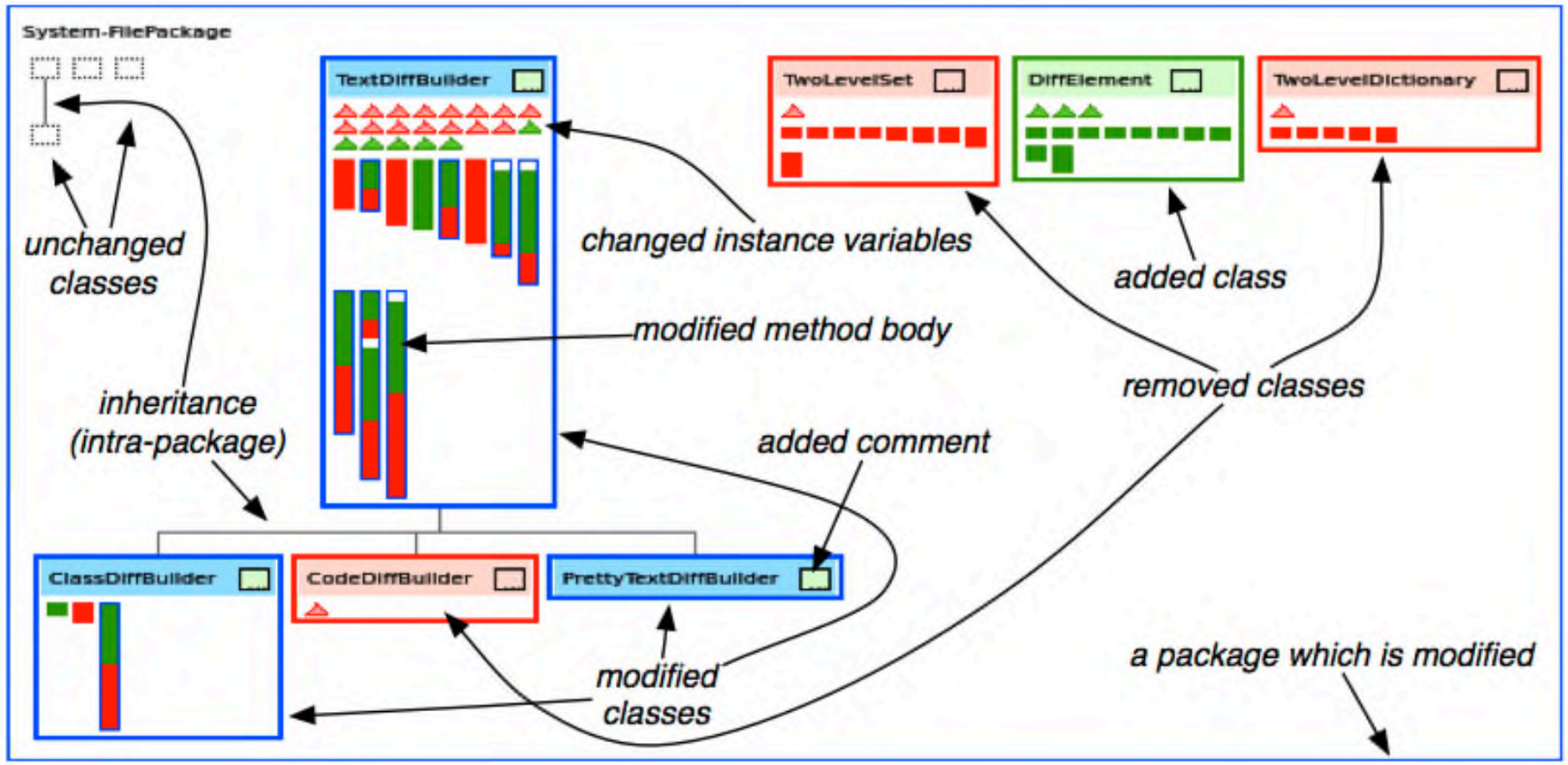
**Single-delta Change Meta-Model and Analyses (RingS)**

**Source Code Meta-Model (Ring)**

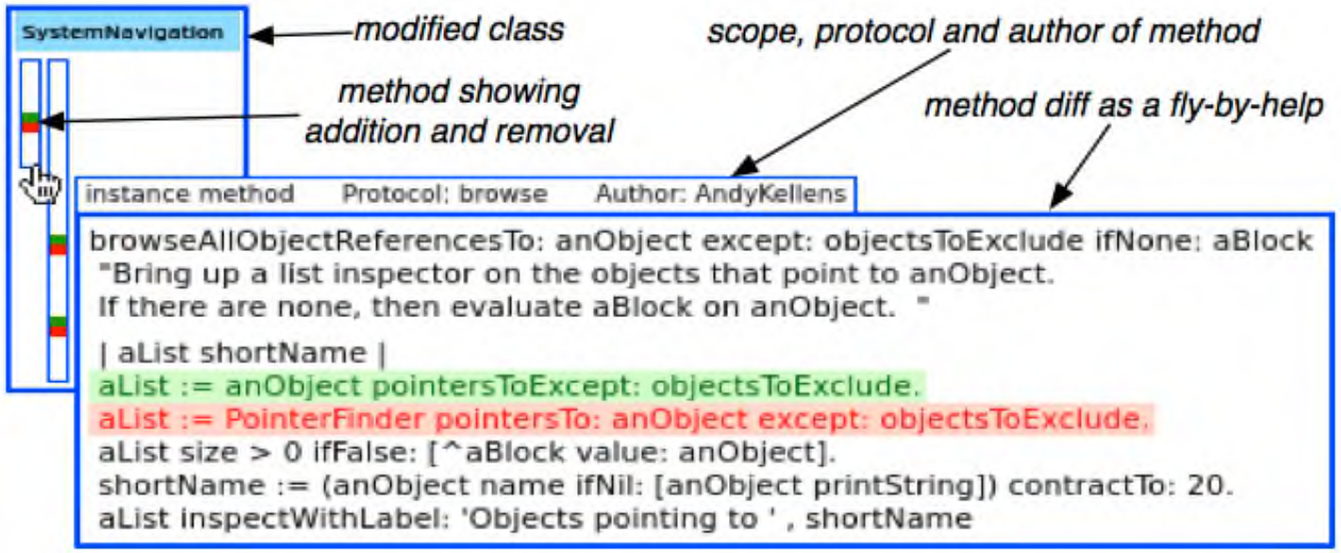
# Torch: Supporting Commit Understanding



# Package Structure

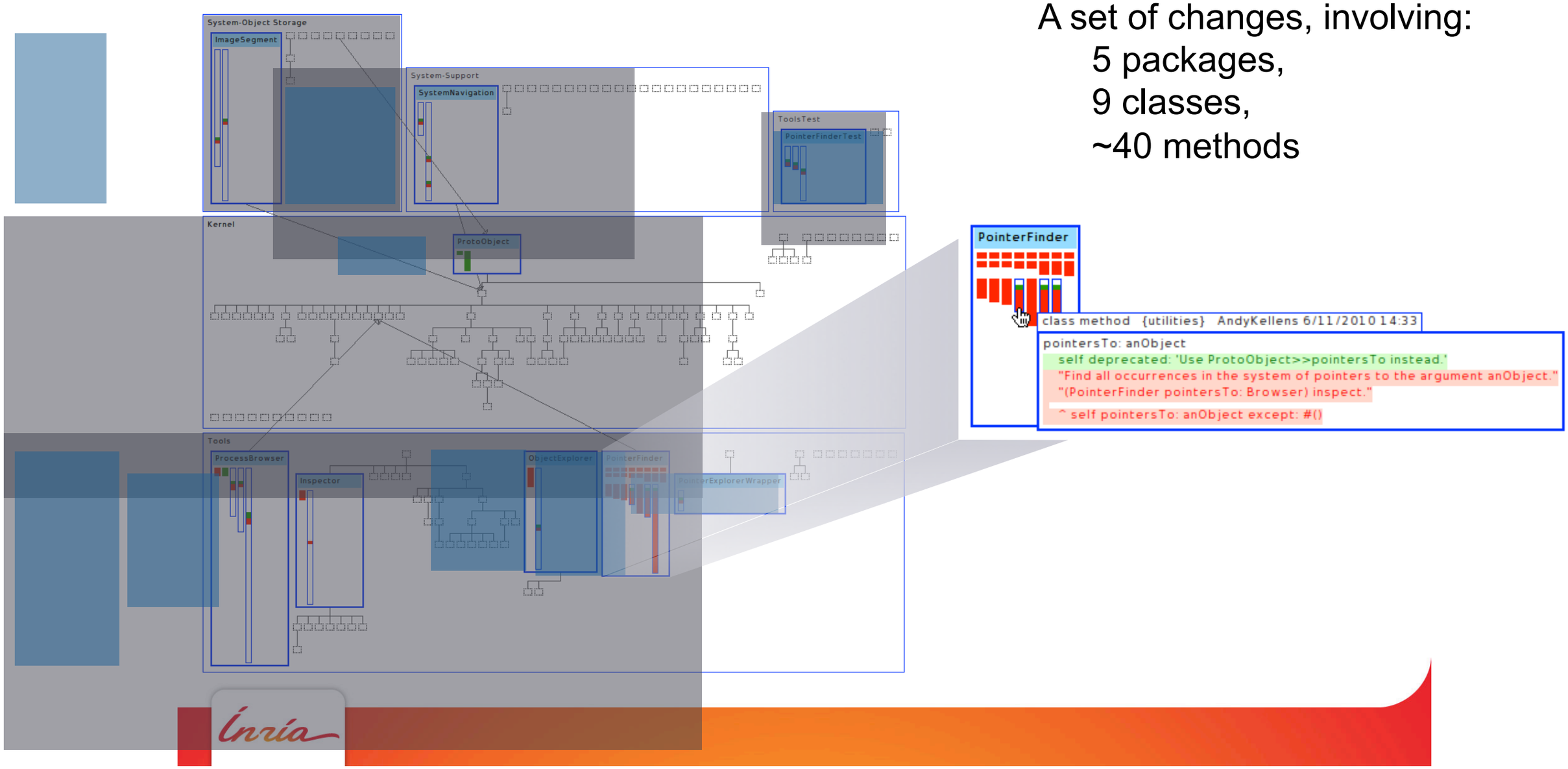


# Omnipresent source code



# Torch: Which changes?

## Where? Who? What?

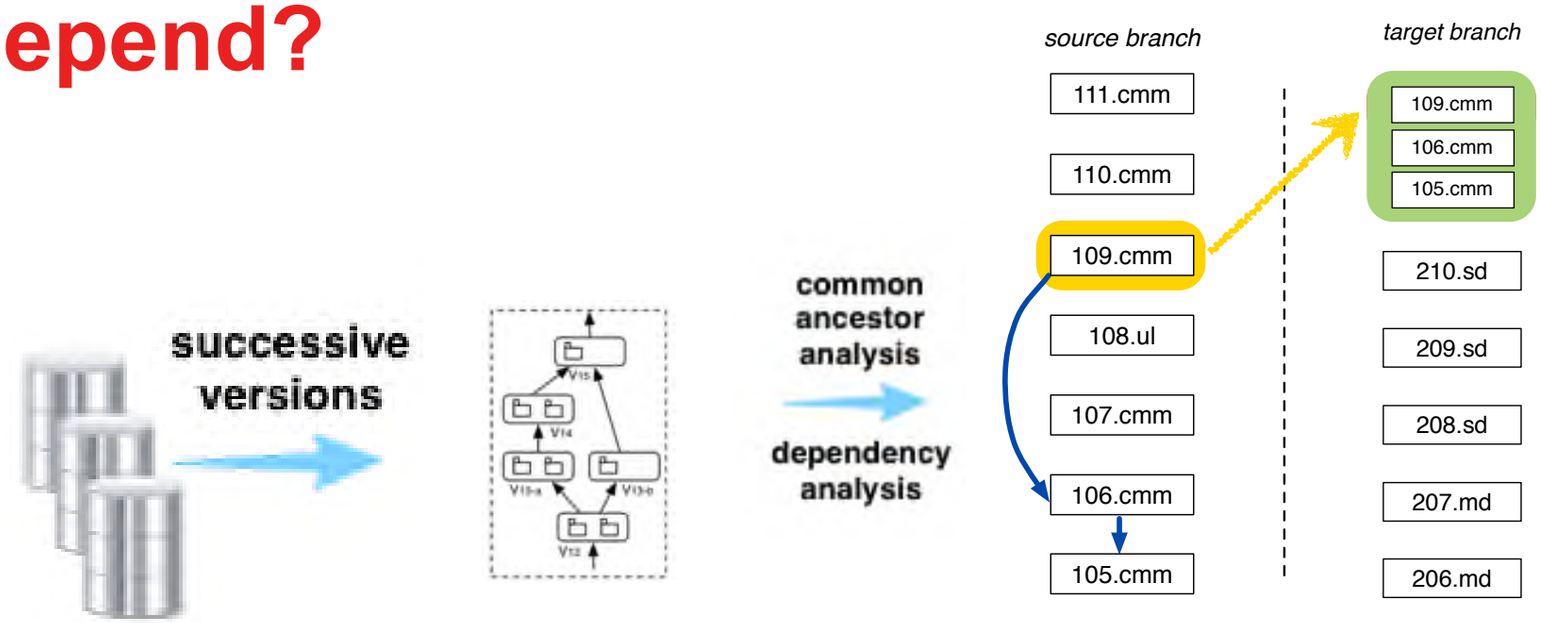



A set of changes, involving:

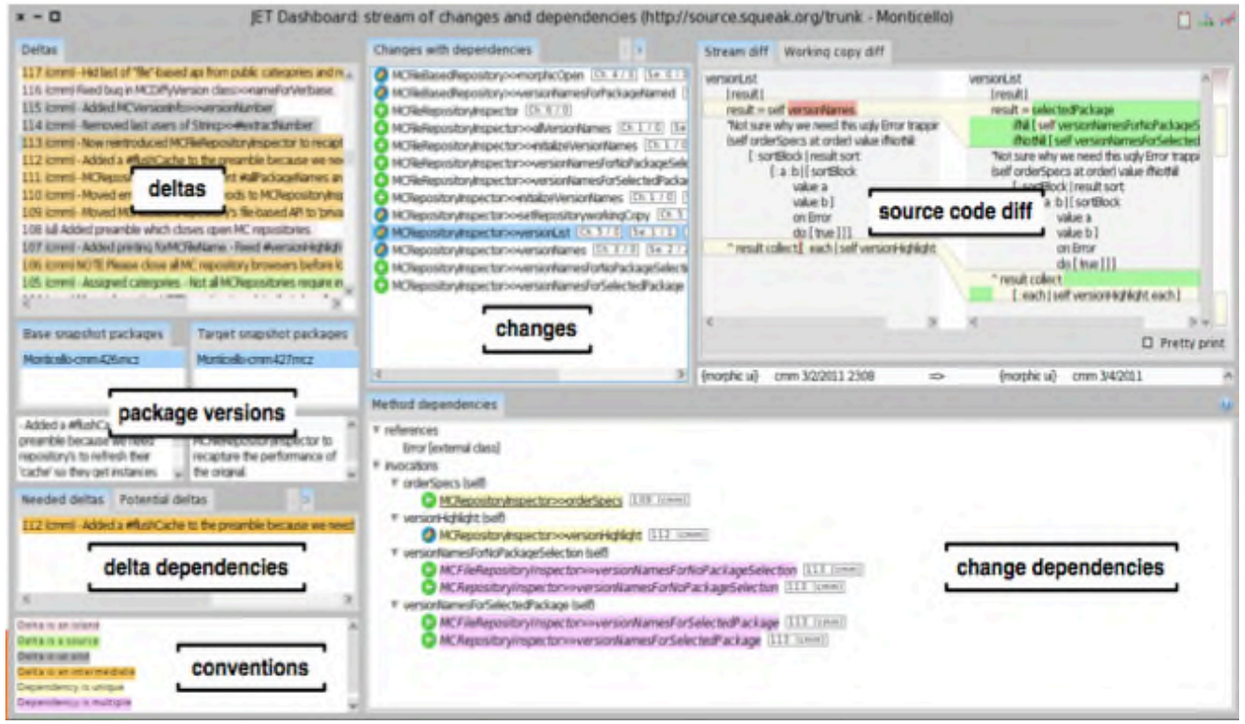
- 5 packages,
- 9 classes,
- ~40 methods

# Streams of Changes:

## On what other changes does this change depend?




characterization of dependencies and deltas



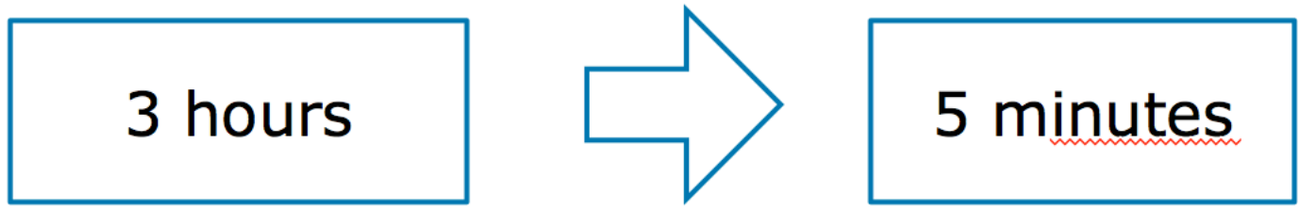
# Other Software Evolution Challenges

**Blockchain, test selection,**

# Which tests to rerun? [PhD CIFRE ATOS Vincent Blondeau]

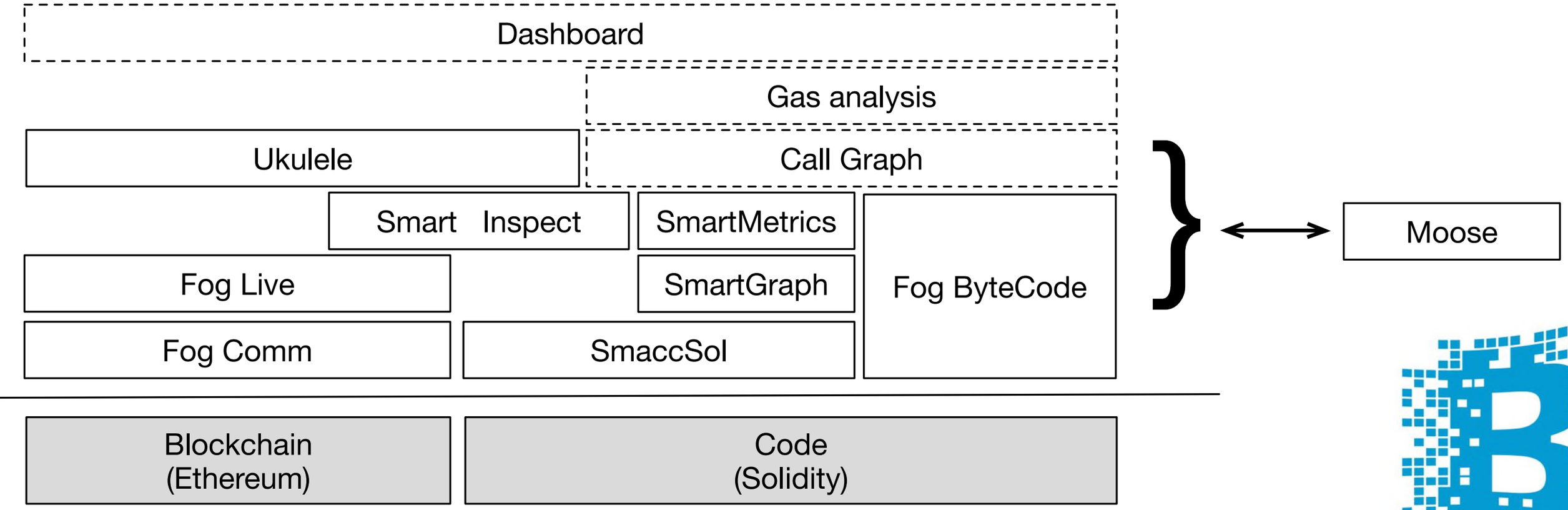
**Test Selection**

**Automatic Test Selection**




Select only the tests related to the changes and run only those ones

# SmartAnvil: Tools for SE-Blockchains



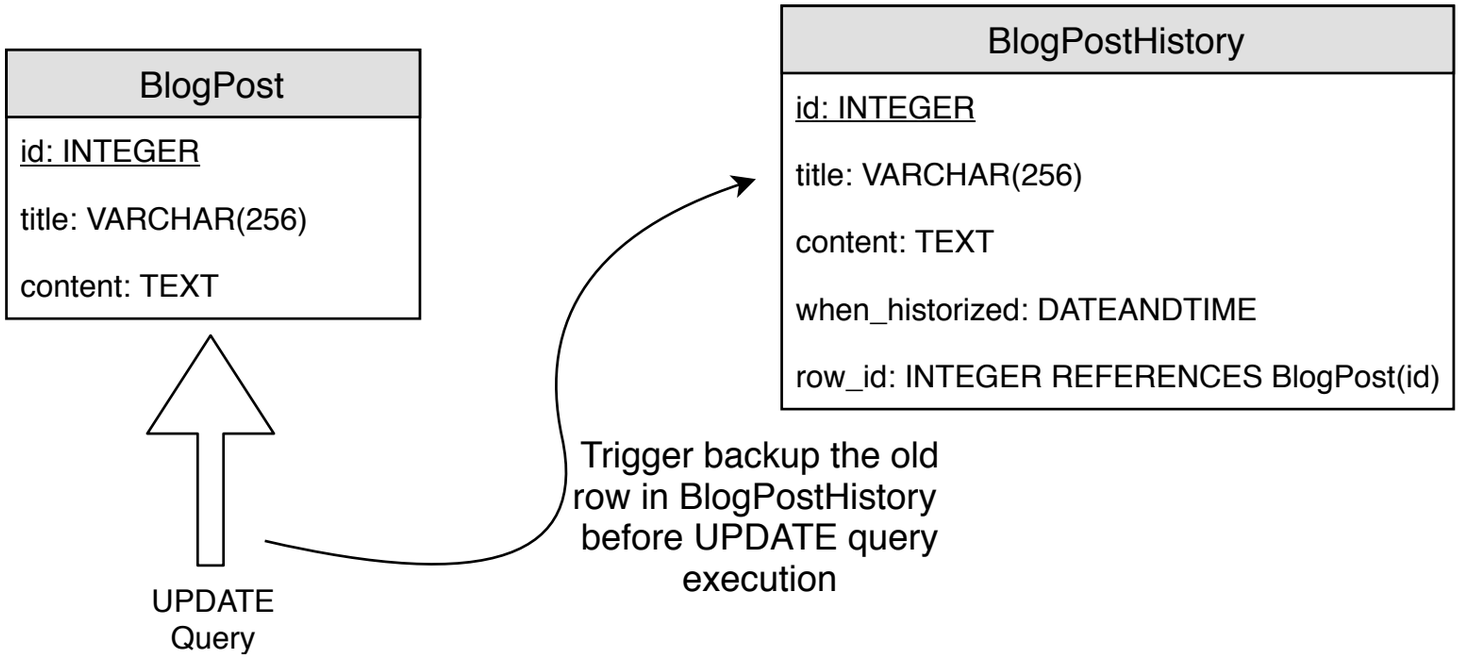

# Blockchain modelisation [PhD - S. Bragagnolo - Berger-Levrault]

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?

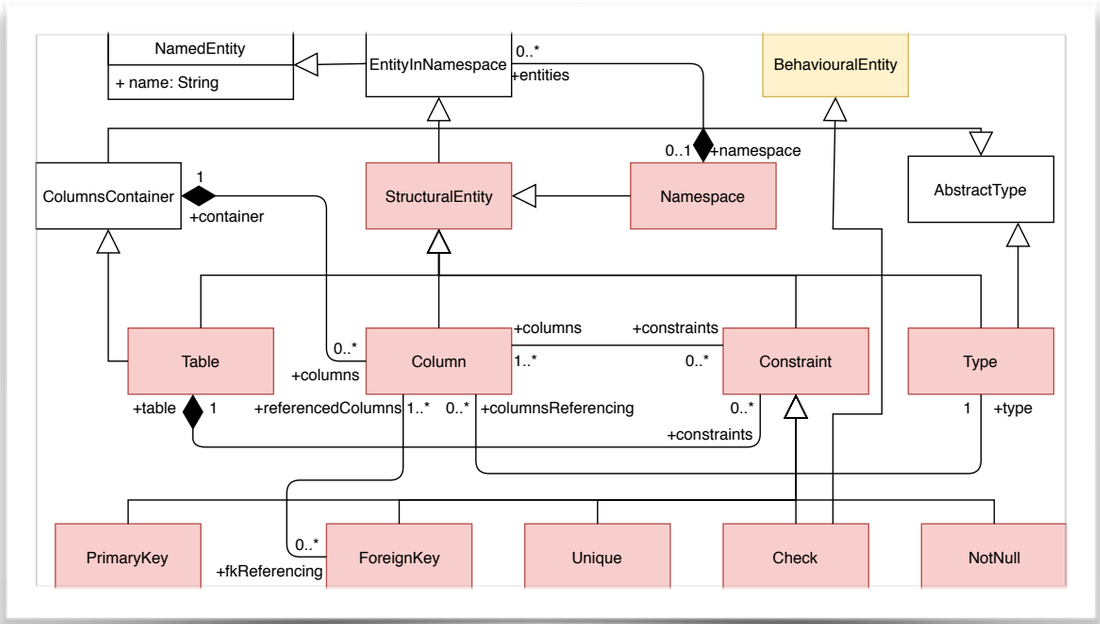
# And embedded procedures? [PhD Julien Delplanque]



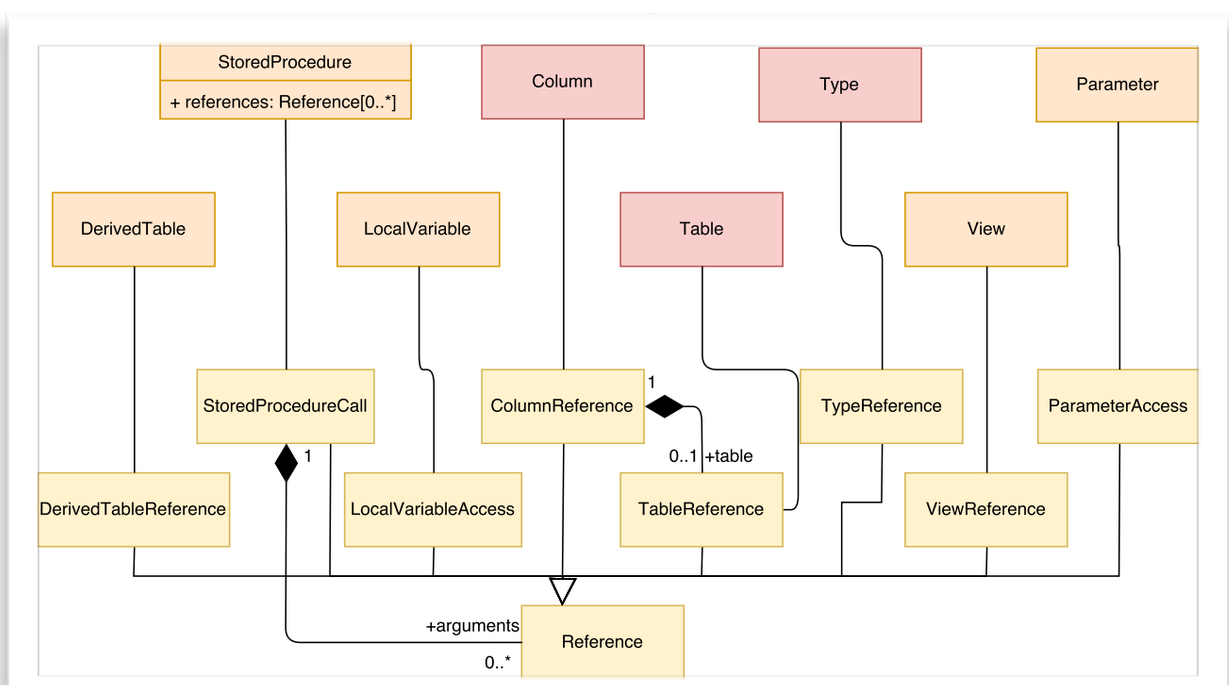

- Dependency Analysis — *How to know which entities potentially need to be changed when an entity is changed?*
- Recommandations — *How to adapt an entity to a change on a entity it depends on?*
- Embedded Behaviour Management — *How to keep the behaviour embedded inside the database working after a change?*

# Refactorings for Embedded procedures

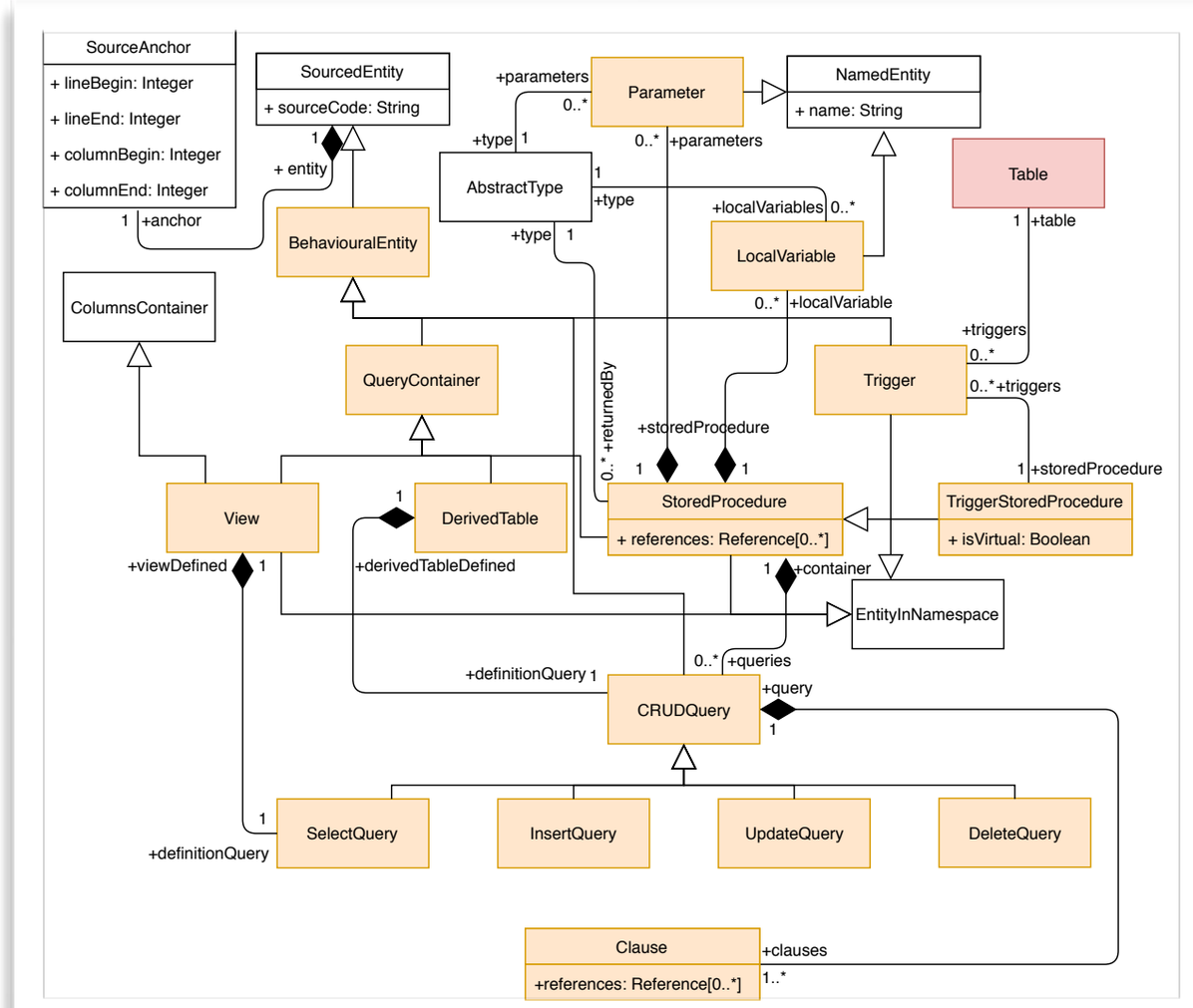
# SQL meta-model




Structural Entities




References Entities




Behavioural Entities

# Creating analyses to tame software

Interested by your challenges

- migration help
- assessment
- rule extraction
- software map
- rearchitecturing
- service/micro service identification
- blockchainisation :)