How to build an automatic refactoring and migration toolkit?

Juliette Tisseyre

Juliette Tisseyre

Software Engineer in Code Mining
Lead R&D @Margo

juliette.tisseyre@gmail.com

@zanoellia

#FOSDEM #MLonCode #CodeMining

MARGO

MODERNISATION PROJECTS: BLOOD, SWEAT AND TEARS

Developers are not equipped enough

Modernisation to tackle:

- Technical debt
- Obsolescence
- Legacy

Margo's clients

- Finance / bank
- Huge applications (> 1 million LoC)
- Typical modernisation project: 10M €, 3 years

Crucial need for automation

MARGO

AGENDA

Code as DATA

Anatomy of a non conventional kind of document and technical approaches

Transformation

Let's start to play for real

Focus on Refactoring

How client's projects shaped our toolkit

MARGO

Code as DATA

Transformation

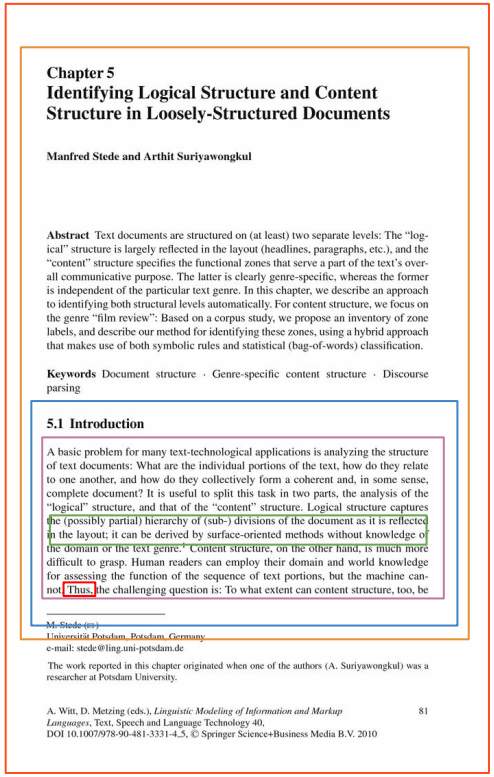
Focus on Refactoring

Anatomy of a non conventional

kind of document and technical approaches

STRUCTURE OF CODE SOURCE

Document
Chapter
Section
Paragraph
Sentence
Word



Chapter 5
Identifying Logical Structure and Content Structure in Loosely-Structured Documents

Manfred Stede and Arthit Suriyawongkul

Abstract Text documents are structured on (at least) two separate levels: The "logical" structure is largely reflected in the layout (headlines, paragraphs, etc.), and the "content" structure specifies the functional zones that serve a part of the text's overall communicative purpose. The latter is clearly genre-specific, whereas the former is independent of the particular text genre. In this chapter, we describe an approach to identifying both structural levels automatically. For content structure, we focus on the genre "film review": Based on a corpus study, we propose an inventory of zone labels, and describe our method for identifying these zones, using a hybrid approach that makes use of both symbolic rules and statistical (bag-of-words) classification.

Keywords Document structure · Genre-specific content structure · Discourse parsing

5.1 Introduction

A basic problem for many text-technological applications is analyzing the structure of text documents: What are the individual portions of the text, how do they relate to one another, and how do they collectively form a coherent and, in some sense, complete document? It is useful to split this task in two parts, the analysis of the "logical" structure, and that of the "content" structure. Logical structure captures the (possibly partial) hierarchy of (sub-) divisions of the document as it is reflected in the layout; it can be derived by surface-oriented methods without knowledge of the domain or the text genre.[1] Content structure, on the other hand, is much more difficult to grasp. Human readers can employ their domain and world knowledge for assessing the function of the sequence of text portions, but the machine cannot. Thus, the challenging question is: To what extent can content structure, too, be

M. Stede (✉)
Universität Potsdam, Potsdam, Germany
e-mail: stede@ling.uni-potsdam.de

The work reported in this chapter originated when one of the authors (A. Suriyawongkul) was a researcher at Potsdam University.

A. Witt, D. Metzing (eds.), *Linguistic Modeling of Information and Markup Languages*, Text, Speech and Language Technology 40,
DOI 10.1007/978-90-481-3331-4_5, © Springer Science+Business Media B.V. 2010

81

```
package texasholdem;

/**
 * A generic game card in a deck (without jokers).
 * Its value is determined first by rank, then by suit.
 */
public class Card {
    private final int rank;
    private final int suit;

    // Constructor based on rank and suit.
    public Card(int rank, int suit) {
        // Handle rank
        if (rank < 0 || rank > NO_OF_RANKS - 1) {
            throw new IllegalArgumentException("Invalid rank");
        }
        this.rank = rank;

        // Handle suit
        if (suit < 0 || suit > NO_OF_SUITS - 1) {
            throw new IllegalArgumentException("Invalid suit");
        }
        this.suit = suit;
    }

    public String toString() {
        return rank + " " + suit;
    }
}
```

Document
Class
Method
Bloc
Instruction
(Key)word

MARGO

CODE SOURCE'S DUALITY

View as a programmer

```java
package texasholdem;

/**
 * A generic game card in a deck (without jokers).
 * Its value is determined first by rank, then by suit.
 */
public class Card {
    private final int rank;
    private final int suit;

    // Constructor based on rank and suit.
    public Card(int rank, int suit) {
        // Handle rank
        if (rank < 0 || rank > NO_OF_RANKS - 1) {
            throw new IllegalArgumentException("Invalid rank");
        }
        this.rank = rank;

        // Handle suit
        if (suit < 0 || suit > NO_OF_SUITS - 1) {
            throw new IllegalArgumentException("Invalid suit");
        }
        this.suit = suit;
    }

    public String toString() {
        return rank + " " + suit;
    }
}
```

View as a machine

```
000001 package texasholdem ; \r\n
000002 \r\n
000003 /**\r\n
000003 * A generic game card in a deck (without jokers). \r\n
000003 * Its value is determined first by rank, then by suit. \r\n
000003 */ \r\n
000007 public class Card { \r\n
000008 private final int rank ; \r\n
000009 private final int suit ; \r\n
000010 \r\n
000011 // Constructor based on rank and suit. \r\n
000012 public Card ( int rank , int suit ) { \r\n
000013 \t // Handle rank \r\n
000014 if ( rank < 0 || rank > NO_OF_RANKS - 1 ) { \r\n
000015 throw new IllegalArgumentException ( "Invalid rank" ) ; \r\n
000016 } \r\n
000017 this . rank = rank ; \r\n
000018 \r\n
000019 \t // Handle suit \r\n
000020 if ( suit < 0 || suit > NO_OF_SUITS - 1 ) { \r\n
000021 throw new IllegalArgumentException ( "Invalid suit" ) ; \r\n
000022 } \r\n
000023 this . suit = suit ; \r\n
000024 } \r\n
000025 \r\n
000026 public String toString ( ) { \r\n
000027 return rank + " " + suit ; \r\n
000028 } \r\n
000029 } \r\n
```

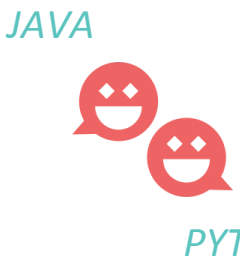

MARGO

DATA CLEANING

Same business logic "*open a file*" but nothing alike

Example.java

```java
import java.io*;

// ...

BufferedReader myFile=
    new BufferReader(
        new FileReader( filename )
    );
```



Example.py

```python
myFile = open( filename )
```

What do we ***need*** to keep?

MARGO

DATA CLEANING

What is **relevant** or **not** in the source code?

- ➔ Generated code
- ➔ Technical frameworks
- ➔ Comments and names
- ➔ Useful code vs meaningless code

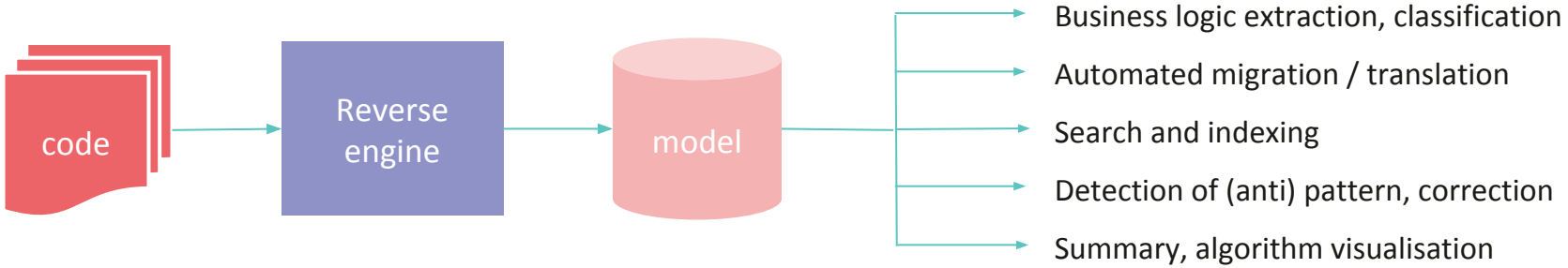
Not a **trivial** task, **depends** on the **objective**

- Balance between cleaning and information loss
- Code structure and coding conventions can help to make a choice

MARGO

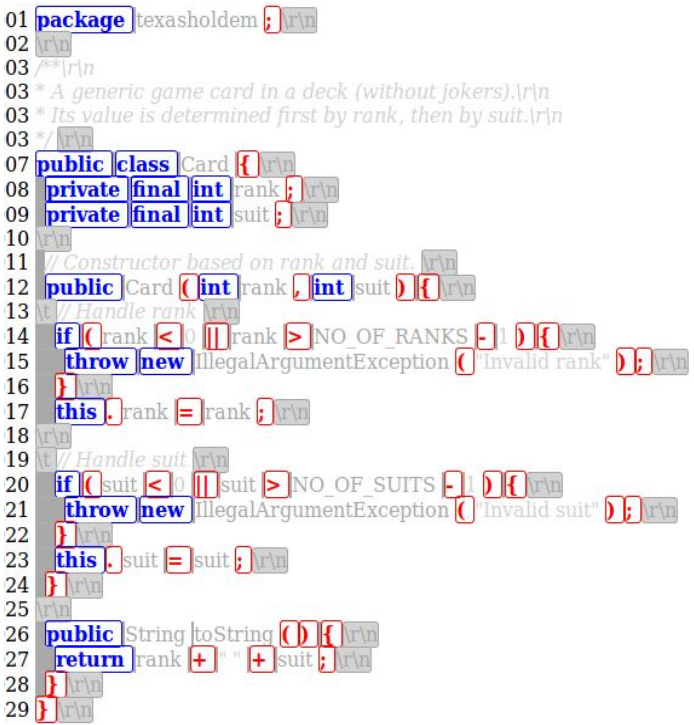
GLOBAL PROCESS

Before applying smart algorithms, the source code must be **transformed** into a **model**



MARGO

FORMAL APPROACH



- Treat code as a **structure**, no interest in naming and comments
- Based on **programming language grammars**:
  - Set of well defined and unambiguous *lexical*, *syntactic* and *semantic* rules
- Modelisation as AST or graph

MARGO

FORMAL APPROACH

Good starting point:

- Only few ambiguities thanks to **internal relationship knowledge**
- Existing tools and algorithms for graph analysis

But tough limitations:

- Unable to understand the meaning
- Poor results on business logic extraction

NATURAL APPROACH

```
01 package texasholdem; \r\n
02 \r\n
03 /**\r\n
03 * A generic game card in a deck (without jokers).\r\n
03 * Its value is determined first by rank, then by suit.\r\n
03 */\r\n
07 public class Card { \r\n
08 private final int rank; \r\n
09 private final int suit; \r\n
10 \r\n
11 // Constructor based on rank and suit. \r\n
12 public Card ( int rank, int suit) { \r\n
13 \t // Handle rank \r\n
14 if ( rank < 0 || rank > NO_OF_RANKS - 1) { \r\n
15 throw new IllegalArgumentException ( "Invalid rank" ) ; \r\n
16 } \r\n
17 this . rank = rank; \r\n
18 \r\n
19 \t // Handle suit \r\n
20 if ( suit < 0 || suit > NO_OF_SUITS - 1) { \r\n
21 throw new IllegalArgumentException ( "Invalid suit" ) ; \r\n
22 } \r\n
23 this . suit = suit; \r\n
24 } \r\n
25 \r\n
26 public String toString ( ) { \r\n
27 return rank + " " + suit; \r\n
28 } \r\n
29 } \r\n
```

- Treat code as **simple text**
- Extract **natural language** elements
  - Name of code entities (variables, functions…)
  - Comments, string content.
- Reuse of **Text Mining** techniques

MARGO

NATURAL APPROACH

Powerful level of analysis:

- Enable business logic extraction
- Understand the developper’s intention

Similar challenges as Text Mining:

- Unable to solve all ambiguities
- Infinite vocabulary, mix of languages can occur
- Strong noise
- Not always understandable for a human

HYBRID APPROACH

What if we take the best of the two worlds?

Bottom up process:

- Rely on the code structure
- Text mining techniques to consolidate meaning

MARGO


Hybrid approach: early stage... ...amazing lands yet to be explored!

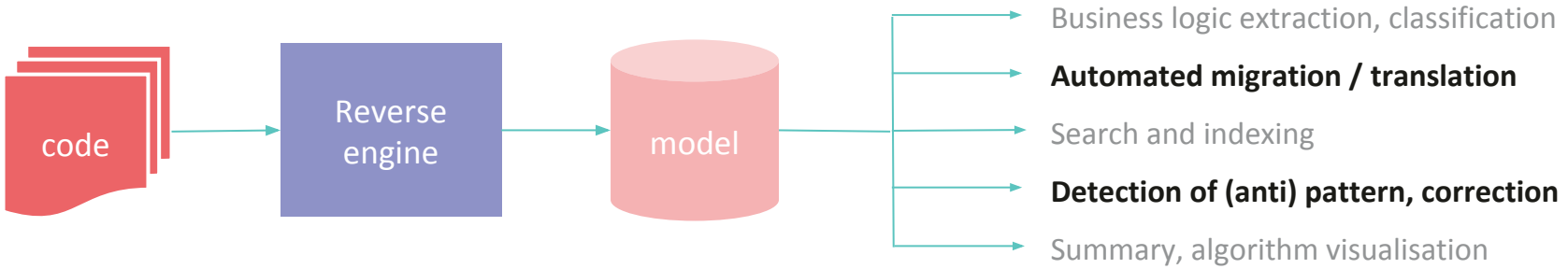
Code as DATA

Transformation

Focus on Refactoring

Let’s start to play for real

OUR PROCESS

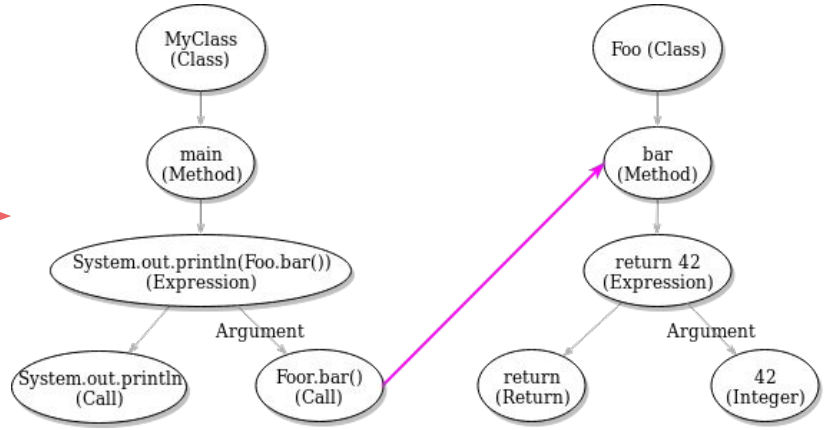


MARGO

REVERSE ENGINE: CHOSEN MODEL

```
public class Foo {
    public static int bar() {
        return 42;
    }
}

public class MyClass {
    public static void main() {
        System.out.println(Foo.bar());
    }
}
```

transformed into



MARGO

REVERSE ENGINE: PARSING

Scan

Project's file handling

Post process

```
**************************************************************************************************
*                                          Parsing                                               *
**************************************************************************************************
SMILE (INFO) 2019-02-03:12:10:21 :
 *******************************************************
 *                   Step Parsing                      *
 *                      Scan                           *
 *******************************************************
SMILE (INFO) 2019-02-03:12:10:21 : Detected languages are:
 ["   * Java(1 files, 11 locs)\r\n"]
SMILE (INFO) 2019-02-03:12:10:21 : Preproc in 0.085899061 (s)
SMILE (INFO) 2019-02-03:12:10:21 :
 *******************************************************
 *                   Step Smile                        *
 *                 Components Only                     *
 *******************************************************
SMILE (INFO) 2019-02-03:12:10:21 :
 *******************************************************
 *                   Step Smile                        *
 *                Component Sources                    *
 *******************************************************
SMILE (INFO) 2019-02-03:12:10:21 :
 *******************************************************
 *                   Step Smile                        *
 *              Unreferenced Sources                   *
 *******************************************************
SMILE (INFO) 2019-02-03:12:10:21 : Proceeding UnreferencedSources: :
SMILE (INFO) 2019-02-03:12:10:21 :  + file /home/juliette/test.java
SMILE (INFO) 2019-02-03:12:10:21 :
 *******************************************************
 *                   Step Smile                        *
 *                 Post Process                        *
 *******************************************************
SMILE (INFO) 2019-02-03:12:10:21 : 28 post processes found.
SMILE (INFO) 2019-02-03:12:10:21 : Parsing : 0.176615805 (s)
UMM (INFO) 2019-02-03:12:10:21 : Starting persistence
UMM (INFO) 2019-02-03:12:10:21 : Persisted 61 objects in : 0.0006728 (s)
UMM (INFO) 2019-02-03:12:10:21 : end reloading 61 in 0.001848223 (s)
```


MARGO

REVERSE ENGINE: SOLVING & BINDING

Solving

Binding

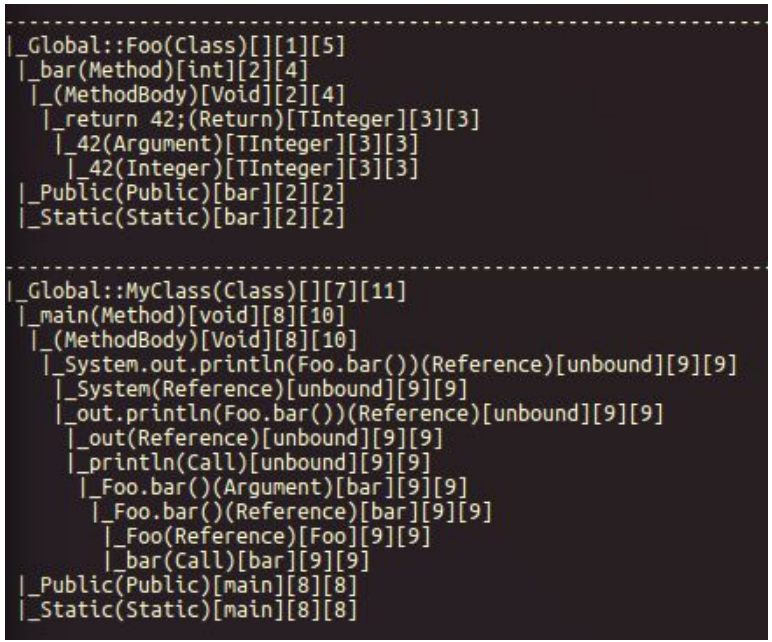
```
*************************************************************************************************************
*                                                Type Solving                                               *
*************************************************************************************************************
UMM_TS (INFO) 2019-02-03:12:10:21 : There are 8 existing types (0 typerefs)
UMM_TS (INFO) 2019-02-03:12:10:21 : Making solving plan...
UMM_TS (INFO) 2019-02-03:12:10:21 : Solving plan made in 0.001144811 (s)
UMM_TS (INFO) 2019-02-03:12:10:21 : Applying solving plan...
UMM_TS (INFO) 2019-02-03:12:10:21 : Merging 0 types...
UMM_TS (INFO) 2019-02-03:12:10:21 : Merges finished in 0.000111275 (s)
UMM_TS (INFO) 2019-02-03:12:10:21 : There are 8 existing types (0 typerefs)
UMM_TS (INFO) 2019-02-03:12:10:21 : Moving 0 types...
UMM_TS (INFO) 2019-02-03:12:10:21 : Moves finished in 1.3087e-05 (s)
UMM_TS (INFO) 2019-02-03:12:10:21 : Solving plan applied in 0.000189485 (s)
UMM_TS (INFO) 2019-02-03:12:10:21 : Type solving finished in 0.001427379 (s)
SMILE (INFO) 2019-02-03:12:10:21 : Solving : 0.001592265 (s)
UMM (INFO) 2019-02-03:12:10:21 : Starting persistence
UMM (INFO) 2019-02-03:12:10:21 : Persisted 61 objects in : 0.000802234 (s)
SMILE (INFO) 2019-02-03:12:10:21 : Total Smile Solve : 0.002905168 (s)
UMM (INFO) 2019-02-03:12:10:21 : end reloading 61 in 0.001535346 (s)
SMILE_TASK (INFO) 2019-02-03:12:10:21 : loading db with option `dbname=test` and `prod=true`.
BIND (INFO) 2019-02-03:12:10:21 :
*************************************************************************************************************
*                                                Binding                                                    *
*************************************************************************************************************
BIND (INFO) 2019-02-03:12:10:21 : Start Building Binding Context ...
BIND (INFO) 2019-02-03:12:10:21 : Binding context built in 0.000152277 (s)
BIND (INFO) 2019-02-03:12:10:21 : Bind Without MultiThreading.
BINDER_THREAD_13950660 (INFO) 2019-02-03:12:10:21 : Starting binding thread with 2 boundables.
BINDER_THREAD_13950660 (INFO) 2019-02-03:12:10:21 : Progress 50%.
BINDER_THREAD_13950660 (INFO) 2019-02-03:12:10:21 : Progress 100% - DONE
BIND (INFO) 2019-02-03:12:10:21 : Time behaviour binding 0.000489494 (s)
BIND (INFO) 2019-02-03:12:10:21 : Comment binding ... Go!
BIND (INFO) 2019-02-03:12:10:21 : Time comment binding 0.00029448 (s)
SMILE (INFO) 2019-02-03:12:10:21 : Binding : 0.00118179 (s)
UMM (INFO) 2019-02-03:12:10:21 : Starting persistence
UMM (INFO) 2019-02-03:12:10:21 : Persisted 62 objects in : 0.000704673 (s)
SMILE (INFO) 2019-02-03:12:10:21 : Total Smile Bind : 0.002313952 (s)
SMILE (INFO) 2019-02-03:12:10:21 : Total Smile Run : 0.337379616 (s)
```


MARGO

REVERSE ENGINE: MODEL

```java
public class Foo {
    public static int bar() {
        return 42;
    }
}

public class MyClass {
    public static void main() {
        System.out.println(Foo.bar());
    }
}
```

transformed into



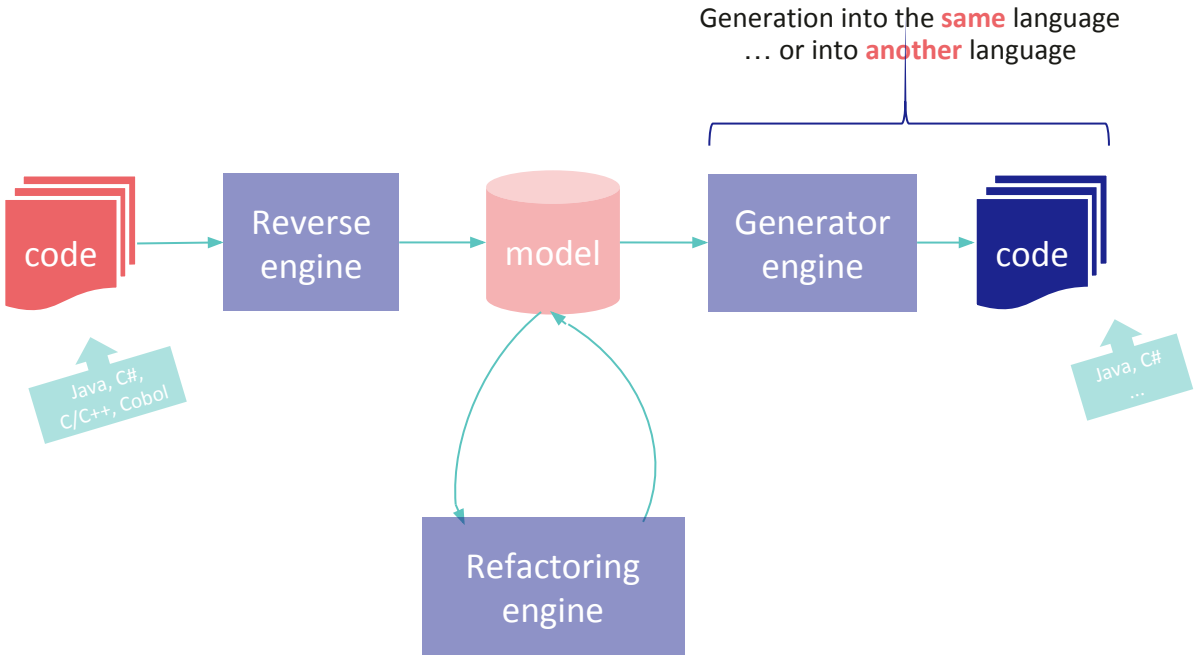
MARGO

REVERSE ENGINE: QUERY

```
> methods = graph.select { |m| m.is_a?(Method) }
=> [bar(Method)[int], main(Method)[void]

> bar_method = methods.first

> bar_method.clients.find { |m| m.is_a?(Call) }.get_fist_context_as(Method)
=> main(Method)[void]

> bar_method.signature
=> Foo.bar() [Public,Static]

> bar_method.cc_complexity
=> 1
```


MARGO

TOOLKIT WORKFLOW



MARGO

MIGRATION EXAMPLE: C++ TO C#

```cpp
double CreditCalculator::GetIssuerPercent(double compositionPercent)
{
    LOG_METHOD_ENTER_LEAVE_INFO();

    double defaultIssuerPercent = 0;

    PIListIterator iLp;
    for (iLp = m_PI_Vector.begin();iLp != m_PI_Vector.end();iLp ++)
    {
        CPercentIssuer* pPI = (*iLp);

        if(pPI->m_min_maturity <= compositionPercent &&
                compositionPercent < pPI->m_max_maturity ||
                compositionPercent == 100 && pPI->m_max_maturity == 100)
            return pPI->m_percentage;

        if(CString(pPI->m_isDefault).CompareNoCase("Y") == 0)
        {
            defaultIssuerPercent = pPI->m_percentage;
        }
    }
    return defaultIssuerPercent;
}
```

```cpp
private double GetIssuerPercent(double compositionPercent)
{
    this.Logger.Info("LOG_METHOD_ENTER_LEAVE_INFO");
    double defaultIssuerPercent = (0);
    foreach (CPercentIssuer iLp in m_PI_Vector)
    {
        CPercentIssuer pPI = new CPercentIssuer((iLp));
        if (pPI.m_min_maturity <= compositionPercent &&
            compositionPercent < pPI.m_max_maturity ||
            compositionPercent == 100 && pPI.m_max_maturity == 100)
        {
            return pPI.m_percentage;
        }

        if (pPI.m_isDefault.ToUpper().CompareTo("Y") == 0)
        {
            defaultIssuerPercent = pPI.m_percentage;
        }

    }

    return defaultIssuerPercent;
}
```


MARGO

How client's projects shaped our toolkit

AUTOMATIC REFACTORING

Refactoring is **deeply integrated** within **dev processes**

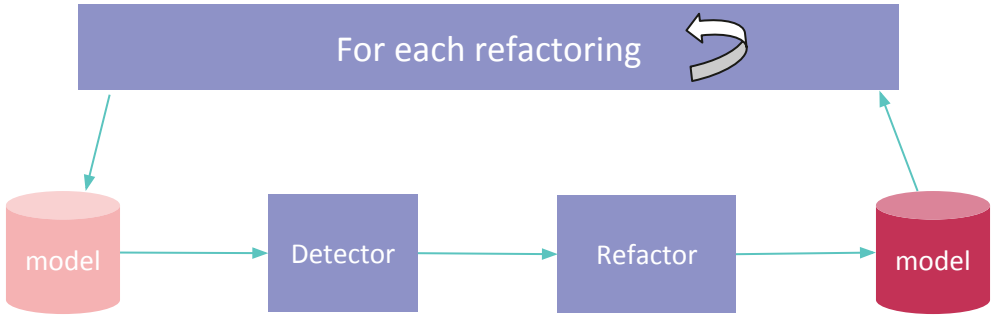
20%+ *of* development *time is dedicated to refactoring*

Time is is **wasted** through refactoring

Risky *1/3 of the manually performed refactorings insert defects*
Same *tedious refactorings all and all over again*

MARGO

REFACTORING ENGINE V1



MARGO

AUTOMATIC REFACTORING

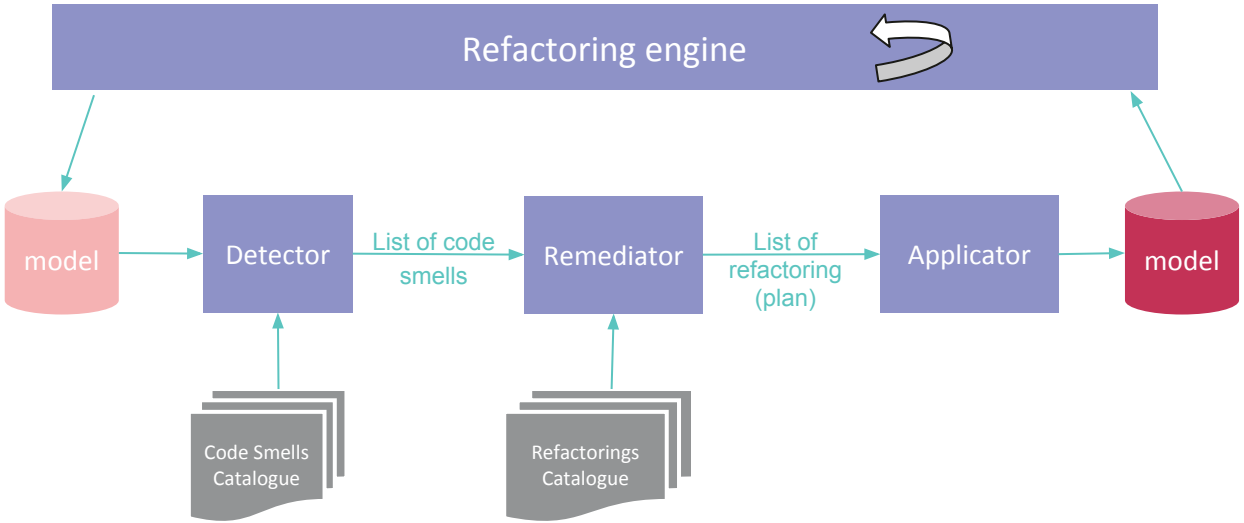
Feedback from our projects

- **Control:**
  - Personnalisation of available refactorings
  - Creation of custom refactoring according to *codebase* needs
  - No Big Bang

- **Industrialisation:**
  - Apply the same refactoring at different locations *at the same time*

MARGO

REFACTORING ENGINE V2



MARGO

WHY NO MACHINE LEARNING YET

Or why finance sector is not always good for innovation...

- Construction of training datasets is tricky
- Human subjectivity
- Open source vs corporate code
- Volume, access
- Code quality and technical dept is a real obstacle
- No revision history or poor quality of incoming commits

MARGO

NEXT STEPS

3,000 billions of running lines of code in the world to take care of!

Welcome in the era of the
augmented developper

MARGO

If time… demo!

DEMO

Refactoring of *java-design-patterns*

java-design-patterns project

- Examples of design pattern implementations
- Open source: https://github.com/iluwatar/java-design-patterns
- 14 k LoC Java
- ~ 600 classes

MARGO

DEMO EXAMPLE

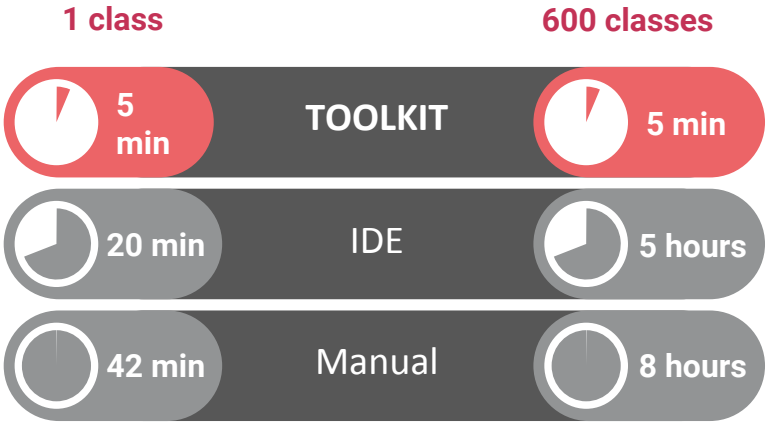
Refactoring of *java-design-patterns*

java-design-patterns project

- Examples of design pattern implementations
- Open source: https://github.com/iluwatar/java-design-patterns
- 14 k LoC Java
- ~ 600 classes

Refactoring has been applied on one smell:

- *Hard Coded String*



MARGO

GO NEXT

Appendices

References - Literature on Refactoring

- M. Fowler, **Refactoring: Improving the Design of Existing Code,** Addison-Wesley, 1999.
- Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "**A quantitative evaluation of maintainability enhancement by refactoring**," presented at Software Maintenance, 2002. Proceedings. International Conference on, 2002.
- J. Ratzinger, M. Fischer, and H. Gall. **Improving evolvability through refactoring**. In Proceedings of the international workshop on Mining software repositories, pages 1–5, 2005.
- R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi. **Does refactoring improve reusability?** In Proceedings of International Conference on Reuse of Off-the-Shelf Components, pages 287–297, 2006.
- M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. **Let's go to the whiteboard: how and why software developers use drawings**. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 557–566, 2007.
- X. Ge, Q. L. DuBose, and E. Murphy-Hill. **Reconciling manual and automatic refactoring**. In Proceedings of the International Conference on Software Engineering, pages 211-221, 2012.
- L. Tokuda and D. Batory, "**Evolving Object-Oriented Designs with Refactorings**," presented at ASE '99: Proceedings of the 14th IEEE international conference on Automated software engineering, 1999.
- R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi. **A case study in refactoring a legacy component for reuse in a product line**. In Proceedings of International Conference on Software Maintenance, pages 369–378, 2005.

MARGO

Why automate refactoring?

Manual	**Refactoring** by editing code **manually** can be a **risky endeavor**	• **1/3** of the manually performed refactorings **insert defects** • Refactoring may take **significant time** • **Waste** of the developper's **skills**
Automation	**Refactoring tools** can perform a **repetitive and monotonous task on behalf** of the programmer	• New **designs** can be explored **quickly** • The use of tools **decreases** restructuring **time** • Developer can focus on **evolutive maintenance** rather than corrective maintenance
Existing	**Refactoring tools in IDE** have **limitations**	• **Limited** configuration • **One** refactoring at a time

MARGO

Examples of refactorings

Some of the most common refactorings in our projects

- **Change visibility of classes, methods or fields**
  - Avoid inappropriate intimacy and indecent exposure
- **Make `private final` all the variables used in synchronized block**
  - Avoid non-deterministic bugs in multithreaded environment.
- **Replace `if/else` statements by a `switch case`**
  - Expose the business logic in a clearer way, simplify the complexity
- **Make `static final` all the constant variables and their names upper-case**
  - Enforce standards and avoid unwanted behaviours
- **Remove deprecated or dead code code**
  - Less code is less code to maintain or migrate