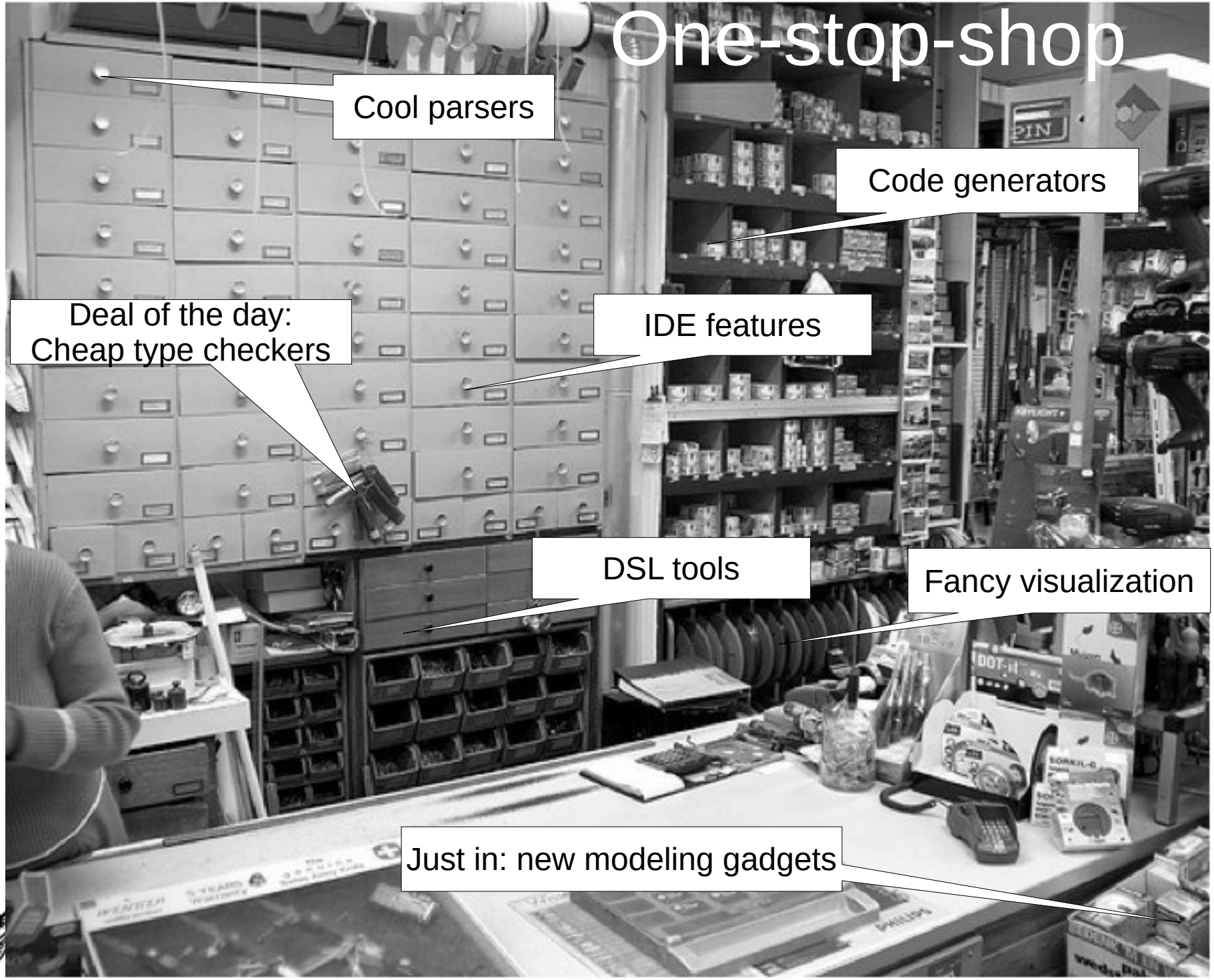
EASY Meta-Programming with Rascal

Paul Klint

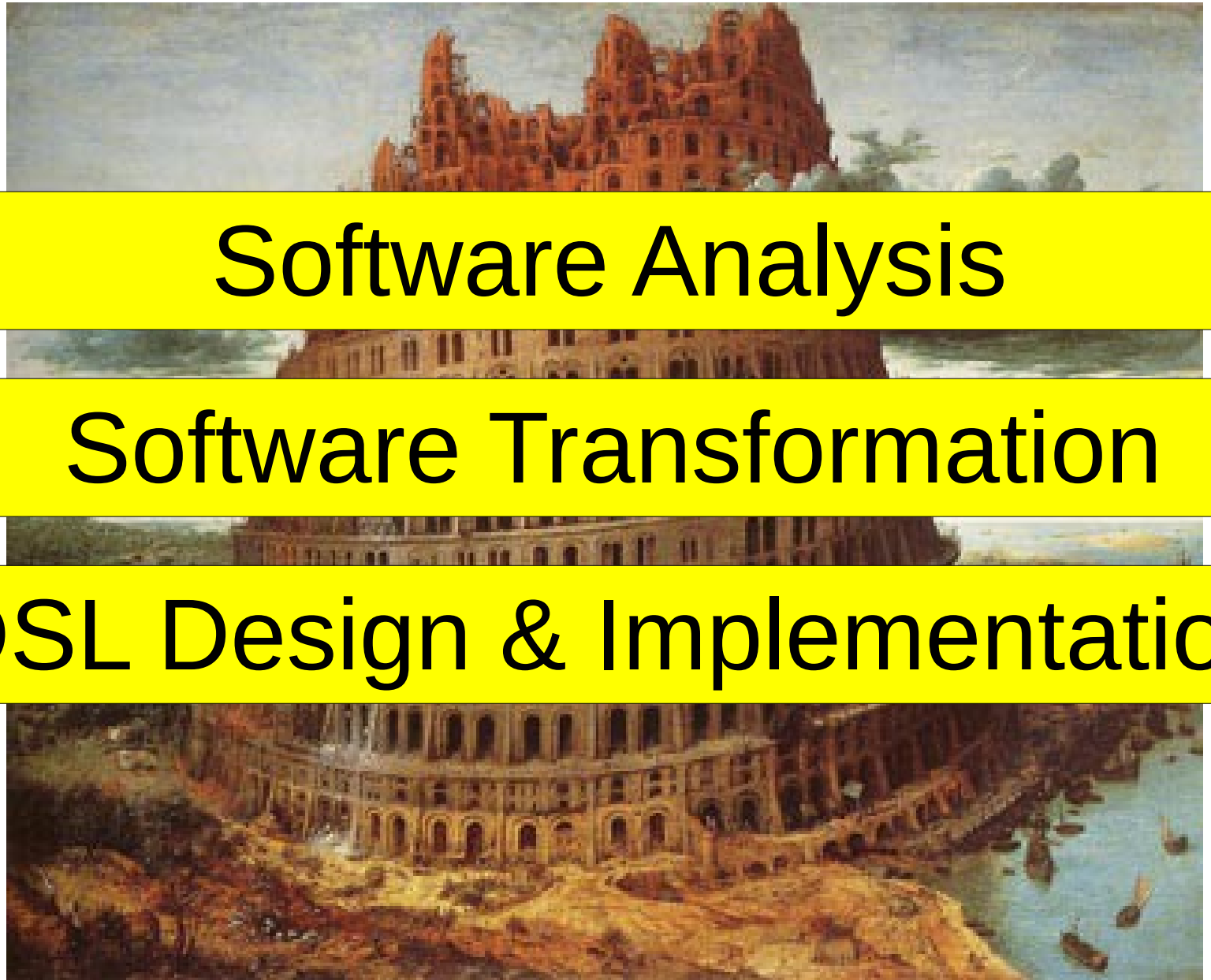
CWI
Centrum Wiskunde & Informatica

Joint work with (amongst others):
Bas Basten, Mark Hills, Anastasia Izmaylova, Davy Landman, Arnold Lankamp, Bert Lisser, Atze van der Ploeg, Michael Steindorfer, ***Tijs van der Storm****,* ***Jurgen Vinju****, Vadim Zaytsev*

One-stop-shop



Meta-Programming in Rascal



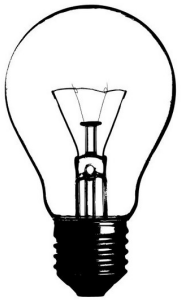
Software Analysis

Software Transformation

DSL Design & Implementation

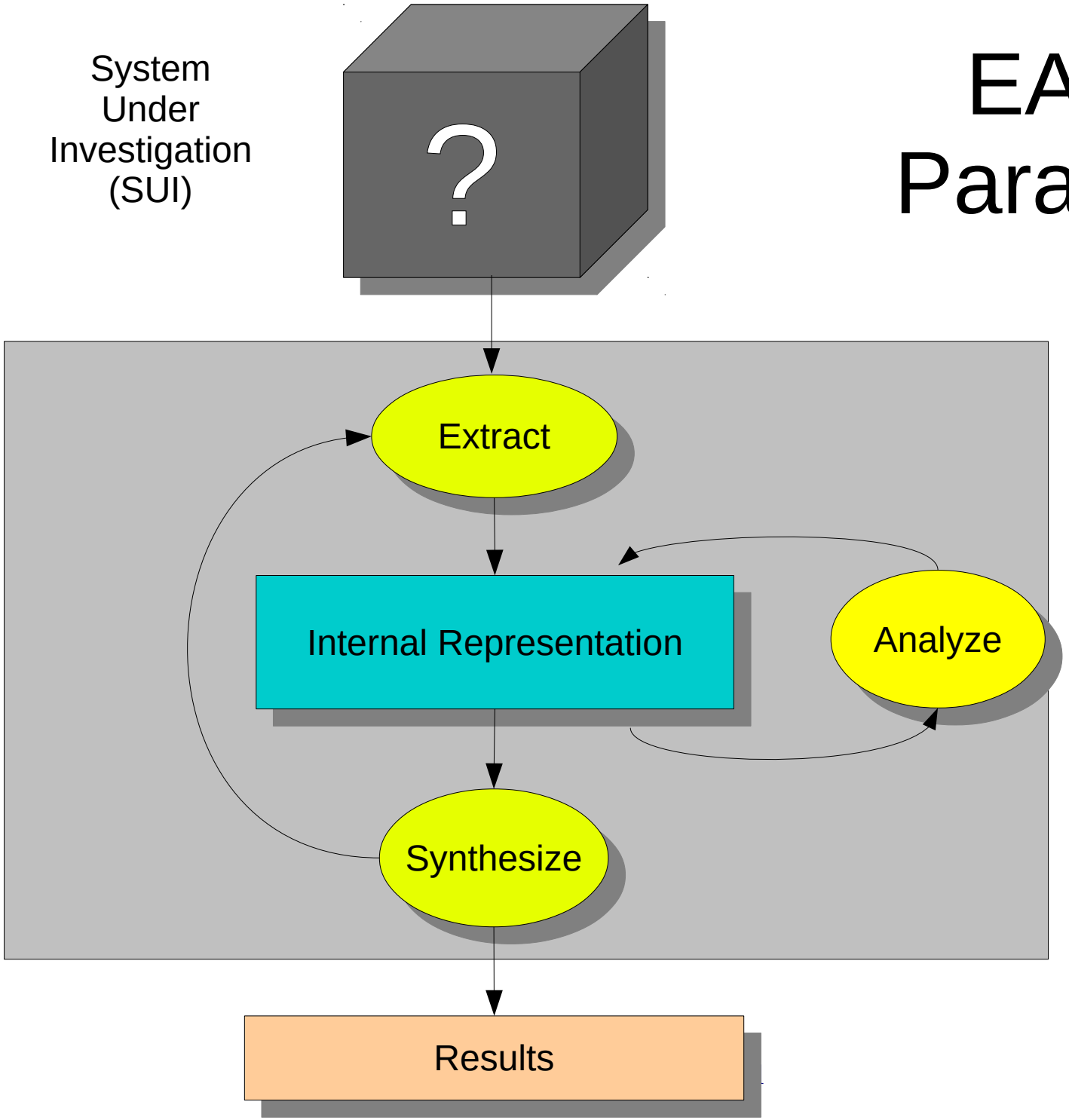
What are the *Technical Challenges*?

- How to parse source code/data files/models?
- How to extract facts from them?
- How to perform computations on these facts?
- How to generate new source code (trafo, refactor, compile)?
- How to synthesize other information?



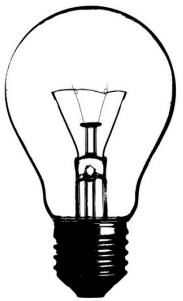
EASY: Extract-Analyze-SYnthesize Paradigm

EASY Paradigm



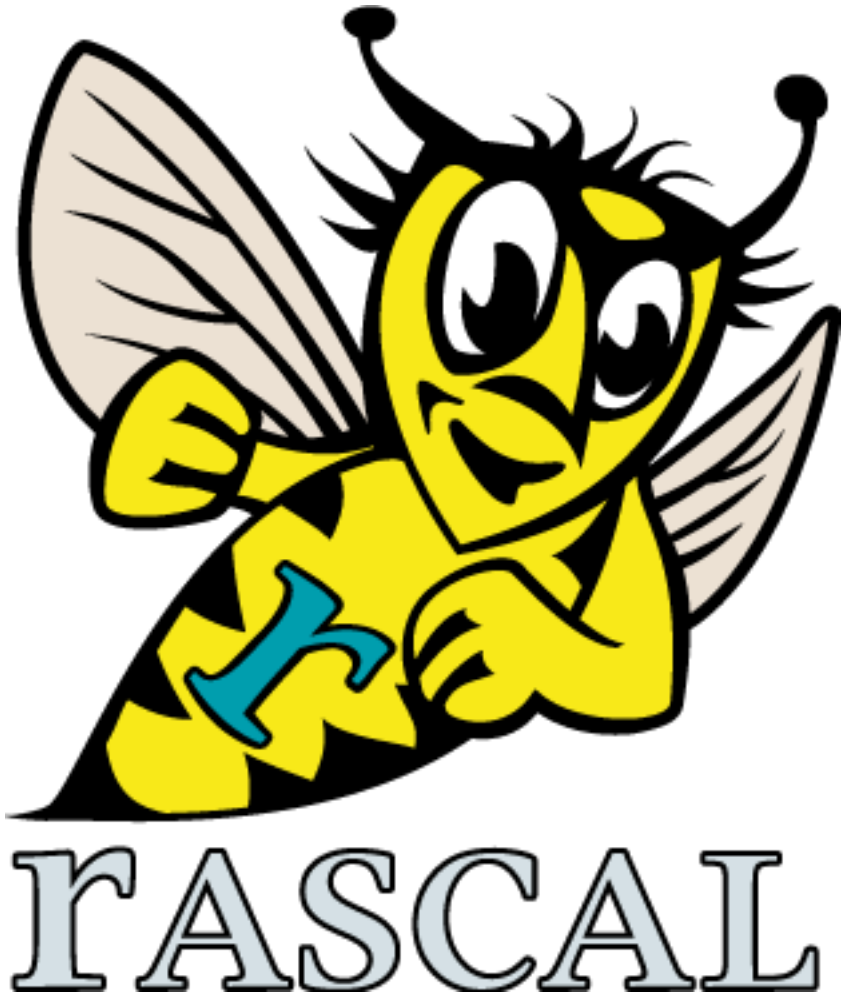
Why a new Language?

- No current technology spans the full range of EASY steps
- There are many fine technologies but they are
  - highly specialized with steep learning curves
  - hard to learn unintegrated technologies
  - not integrated with a standard IDE
  - hard to extend



Goal: Create a
new, **unified**, **extensible**, **teachable**
framework for meta-programming

Here comes Rascal to the Rescue



- “One-stop shop” for
  - Meta-programming
  - Data analysis
  - Visualization
- Lab infrastructure
- Transfer medium
  - Academia
  - Industry (IBM, Eclipse)
- http://www.rascal-mpl.org/

Rascal design based on ...

- **Principle of least surprise**
  - Familiar (Java-like) syntax
  - Imperative core
- **What you see is what you get**
  - No heuristics (or at least as few as possible)
  - *Explicit* preferred over *implicit*
- **Learnability**
  - Layered design
  - Low barrier to entry

Rascal ...

- is a new language for meta-programming
- is based on *Syntax Analysis*, *Term Rewriting*, *Relational Calculus*
- relations used for sharing and merging of facts for different languages/modules
- embedded in the Eclipse IDE
- easily extensible with Java code

Rascal provides

Source locations

- Usual elementary types and datetime and loc
- Rich (immutable) data: lists, sets, maps, tuples, relations with comprehensions and operators
- Syntax definitions & parser generation
- Syntax trees, tree traversal
- Pattern matching (text, trees, lists, sets, ...) and pattern-directed invocation
- Code generation (string templates & trees)
- Java and Eclipse (IMP) integration

Rascal in One Minute

- Sophisticated built-in data types
- Immutable data
- Static safety
- Generic types
- Local type inference
- Pattern Matching
- Syntax definitions and parsing
- Concrete syntax
- Visiting/traversal
- Comprehensions
- Higher-order
- Familiar syntax
- Java and Eclipse integration
- Read-Eval-Print (REPL)

Example

Even numbers:
In many flavors

Even0: initial version

```
list[int] even0(int max) {
  list[int] result = [];
  for (int i <- [0..max])
    if (i % 2 == 0)
      result += i;
  return result;
}
```

```
rascal>even0(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Even1: remove type declarations

```
list[int] even0(int max) {
  list[int] result = [];
  for (int i <- [0..max])
    if (i % 2 == 0)
      result += i;
  return result;
}
```

Even1: remove type declarations

```
list[int] even1(int max) {
  result = [];
  for (i <- [0..max])
    if (i % 2 == 0)
      result += i;
  return result;
}
```

```
rascal>even1(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Even2: merge for and if

```
list[int] even1(int max) {
  result = [];
  for (i <- [0..max])
    if (i % 2 == 0)
      result += i;
  return result;
}
```

Even2: merge for and if

```
list[int] even2(int max) {
  result = [];
  for (i <- [0..max], i % 2 == 0)
     result += i;
  return result;
}
```

```
rascal>even2(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Even3: for returns the list (using append)

```
list[int] even3(int max) {
  result = [];
  for (i <- [0..max], i % 2 == 0)
     result += i;
  return result;
}
```

Even3: for returns the list (using append)

```
list[int] even3(int max) {
  result =  for (i <- [0..max], i % 2 == 0)
                  append i;
  return result;
}
```

```
rascal>even3(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Even4: eliminate result variable

```
list[int] even3(int max) {
  result =  for (i <- [0..max], i % 2 == 0)
                  append i;
  return result;
}
```

Even4: eliminate result variable

```
list[int] even4(int max) {
  return  for (i <- [0..max], i % 2 == 0)
                  append i;
}
```

```
rascal>even4(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Even5: use comprehension

```
list[int] even4(int max) {
  return  for (i <- [0..max], i % 2 == 0)
                  append i;
}
```

Even5: use comprehension

```
list[int] even5(int max) {
  return  [i | i <- [0..max], i % 2 == 0];
}
```

```
rascal>even5(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Even6: use abbreviated function declaration

```
list[int] even5(int max) {
  return  [i | i <- [0..max], i % 2 == 0];
}
```

Even6: use abbreviated function declaration

```
list[int] even6(int max) =  [i | i <- [0..max], i % 2 == 0];
```

```
rascal>even5(25);
list[int]: [0,2,4,6,8,10,12,14,16,18,20,22,24]
```

Example

Generating getters and setters

Generating Getters and Setters (1)

- Given:
  - A class name
  - A mapping from names to types

Required:

  - Generate the named class with getters and setters

Input

```
map[str, str] fields = (
  "name" : "String",
  "age" : "Integer",
  "address" : "String"
);
```

Field name of type String

Field age of type Integerr

Field address of type String

```
genClass("Person", fields)
```

Generate class person with these fields

Expect Output

```
public class Person {

    private Integer age;
    public void setAge(Integer age) {  this.age = age;   }
    public Integer getAge() { return age;   }


    private String name;
    public void setName(String name) { this.name = name;   }
    public String getName() {  return name;   }


    private String address;
    public void setAddress(String address) {  this.address = address;   }
    public String getAddress() {  return address;   }
  }
```

Generating Getters and Setters, 1

```
str capitalize(str s) =
  toUpperCase(s[0 .. 1]) + s[1 .. ] ;
```

String with computed interpolations

```
str genSetter(map[str,str] fields, str x) =
  "public void set<capitalize(x)>(<fields[x]> <x>) {
  '  this.<x> = <x>;
   }";
```

Red is interpolated

Text before ' is ignored

```
str genGetter(map[str,str] fields, str x) =
  "public <fields[x]> get<capitalize(x)>() {
  '  return <x>;
  '}";
```

Generating Getters and Setters, 2

```
str genClass(str name, map[str,str] fields) =
   "public class <name> {
   '  <for (x <- sort([f | f <- fields])) {>
   '  private <fields[x]> <x>;
   '  <genSetter(fields, x)>
   '  <genGetter(fields, x)><}>
   '}";
```

Delimiters of the **for** body

Wrapping Up the Rascal Overview

Other Rascal Features

- Visit arbitrary values (including trees)
- Get/set fields of ADTs
- Parameterized types
- Exception handling
- Annotations
- Higher order functions
- Built-in random testing
- Many libraries ...

Typical Applications

- Analysis
  - Metrics
  - Java, PHP, ECORE
  - Repositories (SVN,GIT)
  - Assertion checking
  - Type checking
- Transformation
  - Refactoring
  - Compilation
- DSL design & implementation
  - Forensics (NFI)
  - Games (IC3MEDIA)
  - Questionaires
  - Auditing
- Education:
  - UvA, OU, TU/e, U Bergen U Namur, ...

In Focus:

- IDE customization
- Language processors
- Metrics

IDE customization

IDE Customization

- Starting point: Eclipse + IMP
- Use Rascal to create:
  - Parsing, syntax highlighting
  - Typechecking, constraint checking, code generation
  - Outlining, annotations
  - Connection with external tools

Code Outlining

collatz.l3 | test.l1 | test.l2 | testConstants.l4 | sample.l4 | »5

```
PROCEDURE Multiply;
  VAR x, y, z: INTEGER;
BEGIN
  Read(x);
  Read(y);
  z := 0;
  WHILE x > 0 DO
    IF x MOD 2 = 1 THEN
      z := z + y
    END ;
    y := 2*y;
    (* Dag *)
    x := x DIV 2 END;
   Write(x);
   Write(y);
   Write(z);
   WriteLn
END Multiply;

(* def *)

PROCEDURE Divide;
  VAR x, (* Q *) y, r, q, w: INTEGER;
BEGIN
```

Outline

- Constants
- Types
  - Y
- Variables
- Procedures
  - Nesting
  - Multiply
    - Constants
    - Types
    - Variables
      - x, y, z
  - Divide
  - BinSearch

Annotations

*collatz.l3

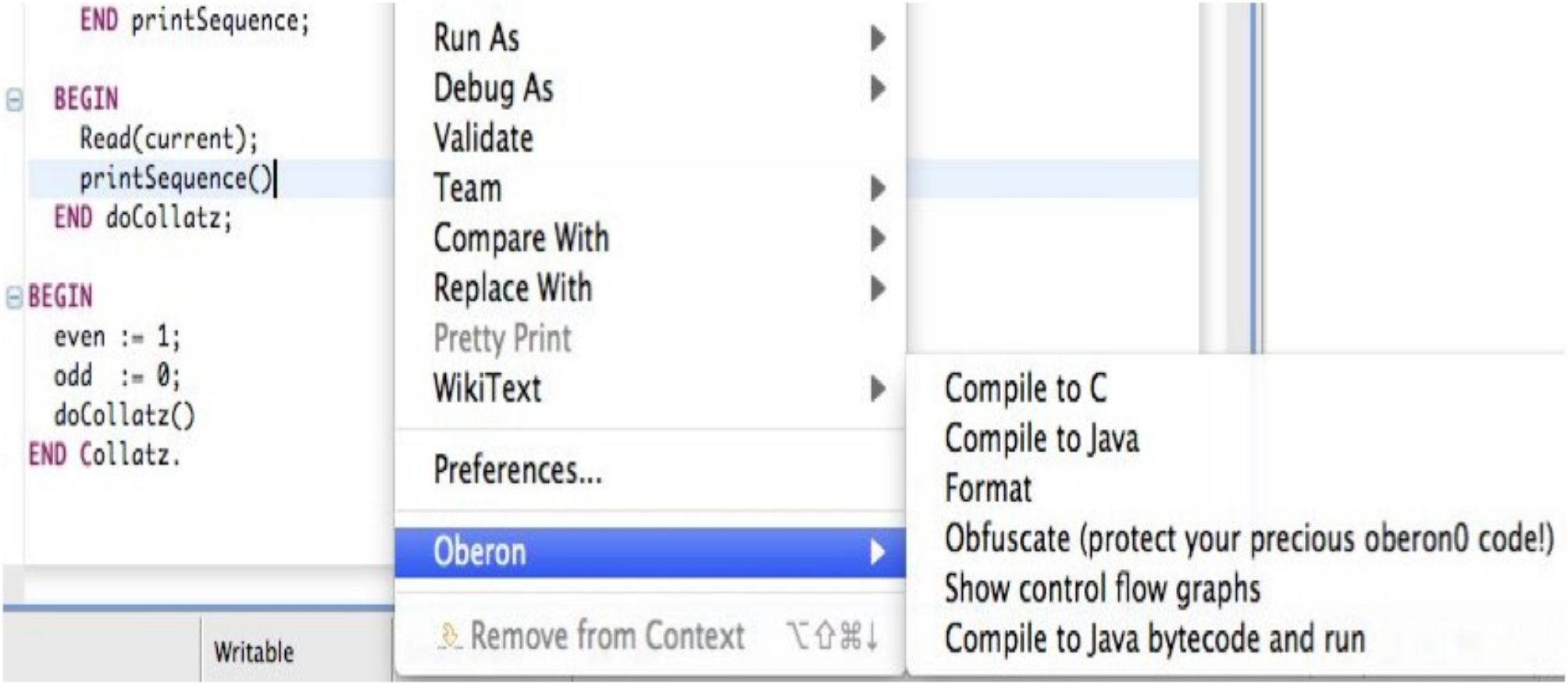
```
1  MODULE Collatz;
2
3  VAR even,odd : INTEGER;
4
5  PROCEDURE doCollatz();
6    VAR current : INTEGER;
7        currentEven : BOOLEAN;
8
9    PROCEDURE computeEven();
10     BEGIN
11       IF current MOD 2 = 0 THEN
12         currentEven := even
13       ELSE
14         currentE
15       END
16     END computeEven;
17
```

Cannot assign value of type INTEGER, expected type BOOLEAN

Outline

- Constants
- Types
- Variables
  - even,odd
- Procedures
  - doCollatz

User-Defined Menus



Creating Language Processors and IDEs

Creating Language Processors and IDEs

- How can we use the Rascal language for creating language processors and IDEs?
- The Pico language (since 80's already used as example in ASF, ASF+SDF and many others)

The Toy Language Pico

- Has a single purpose: being so simple that its specification fits on a few pages
- We can define various operations on Pico programs: parse, typecheck, compile, ...)
- We integrate the Pico tools with Eclipse

A Pico Program

```
begin declare input : natural,
              output : natural,
              repnr : natural,
              rep : natural;
      input := 14;
      output := 1;
      while input - 1 do
          rep := output;
          repnr := input;
          while repnr - 1 do
             output := output + rep;
             repnr := repnr - 1
          od;
          input := input - 1
      od
end
```

- No input/output
- No multiplication
- What does this program do?

Parsing, Editing, Syntax highlighting

Assembly.rsc | Compile.rsc | Fac.pico » 37

```
begin declare input : natural,
              output : natural,
              repnr : natural,
              rep : natural;
      input := 14;
      output := 1;
      while input - 1 do
          rep := output;
          repnr := input;
          while repnr - 1 do
             output := output + rep;
             repnr := repnr - 1
          od;
          input := input - 1
      od
end
```

Signaling Parse Errors

Assembly.rsc | Compile.rsc | Fac.pico | »37

```
1  begin declare input : natural,
2                output : natural,
3                repnr : natural,
4                rep : nat;
5        input := 14;
6        output := 1;
7        while input - 1 do
8            rep := output;
9            repnr := input;
10           while repnr - 1 do
11               output := output + rep;
12               repnr := repnr - 1
13           od;
14           input := input - 1
15       od
16 end
```

Signaling Type Checking Errors

Compile.rsc | P1.pico | Fac.pico | »37

```
1 begin
2   declare
3      x : natural,
4      y : string;
5   x :=  "AA";
6   while "abc" do y := 1 od
7 end
```

Pico Metrics
(LOC including comments and blank lines)

Component	LOC
AST	34
Assembly	20
Compile	89
Control Flow Graph	54
Eval	81
Load	7
Plugin	84
Syntax	53
Typechecker	92
Uninit	22
UseDef	28
Visualize	56
Total	**620**

Read the full story at:
http://tutor.rascal-mpl.org/Recipes/Recipes.html#/Recipes/Languages/Pico/Pico.html

Metrics: Scaling to Java, PHP, ...

M^3: The Metrics Meta-Model

- Goal: collect relevant information about source code in
  - Relations between source code entities
  - Abstract Syntax Trees (ASTs)
- Idea: use Rascal's source locations to identify source code entities
- Implement specific metrics using the above info

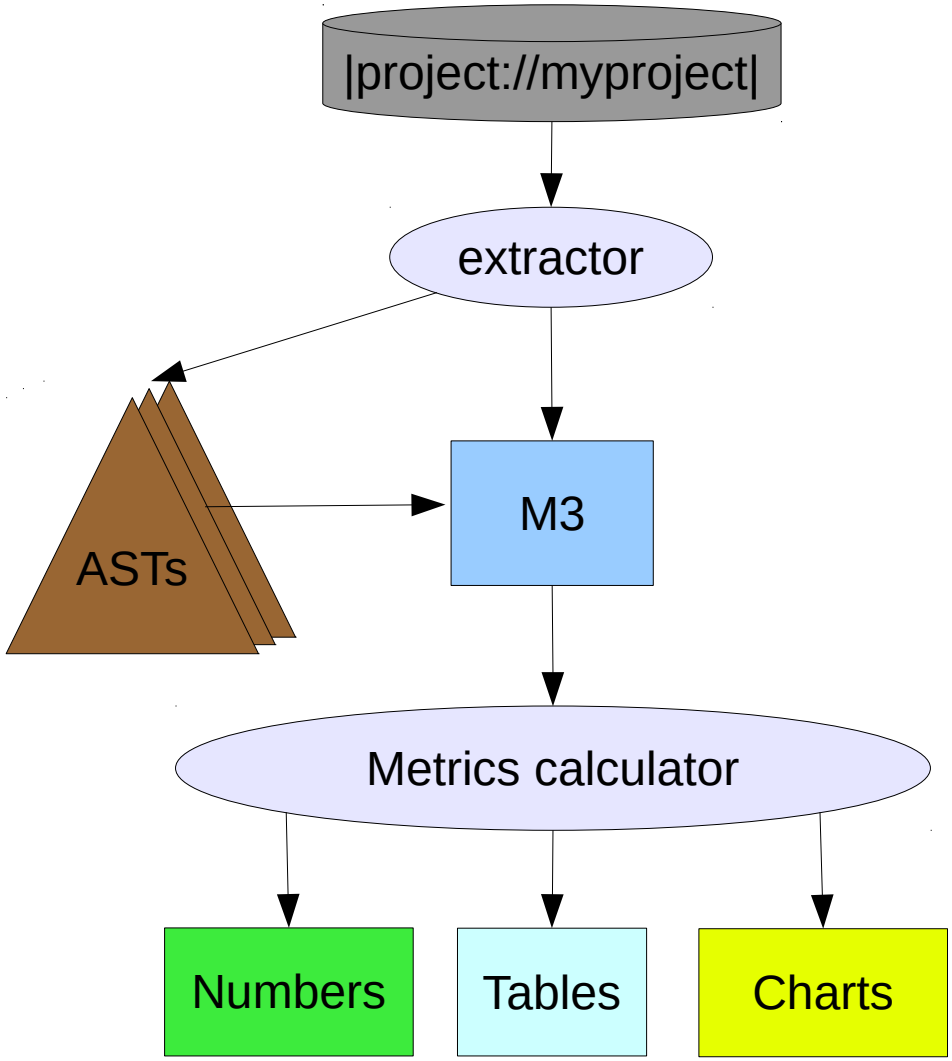
Identifying Source Code entities

- Structure of a location (extends URIs):
  - |<scheme>://<auth>/<path>?<qry>|(<off>,<len>)
- Examples of physical locations:
  - |file:///tmp/Hello.java| – absolute path
  - |http://foo.com/index.html|
  - |project://MyPrj/Hello.java| – relative to Eclipse project MyPrj

Identifying Source Code Entities

- Introduce new schemes to represent Java entities:
  - |java+class://myPrj/java/util/List|
- Notions like containment, inheritance can now be represented by a relation of type `rel[loc, loc]`
- Given a project an M^3 model can be extracted automatically

M^3: The Metrics Meta-Model

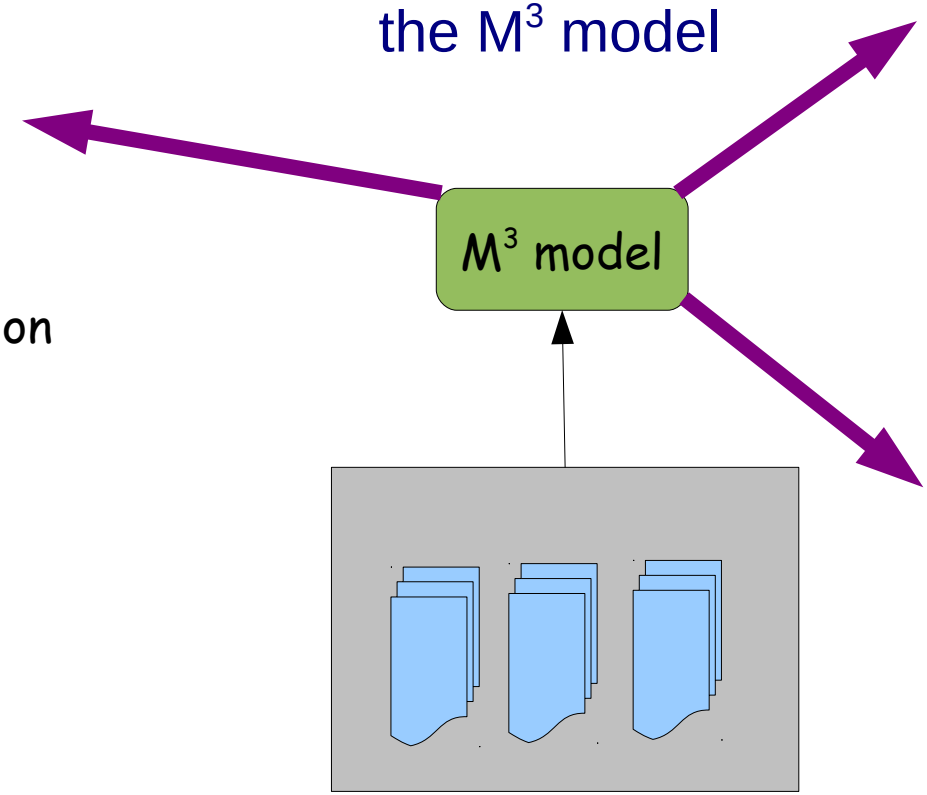


Scaling to Java

M^3 Core

@declarations
@types
@uses
@containment
@messages
@names
@documentation
@modifiers

Access Java facts via annotations and functions on the M^3 model



Java Project

M^3 Java Extension

@extends
@implements
@methodInvocation
@fieldAccess
@typeDependency
@methodOverrides
@annotations

M^3 Java convenience functions

classes
interfaces
packages
variables
parameters
fields
methods

Creating an M^3 Model

Take a project in your Eclipse workspace

```
rascal>loc project = |project://org.eclipse.imp.pdb.values|;
loc: |project://org.eclipse.imp.pdb.values|

rascal>import lang::java::jdt::m3::Core;
ok
rascal>model = createM3FromEclipseProject(project)
```

Create M3 model for the project

Creating an M^3 Model

```
M3: m3(|project://org.eclipse.imp.pdb.values|)[
  @annotations={
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieMap/Map2To2Node/sizePredicate()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieMap/Map0To0Node/sizePredicate()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/type/SetType/carrier()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieSet/Set0To2Node/hasPayload()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/Map2/asTransient()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieSet/Set0To3Node/nodeIterator()|,
     |java+interface:///java/lang/SuppressWarnings|>,
    <|java+method:///org/eclipse/imp/pdb/facts/type/ValueType/isSubtypeOfNode(org.eclipse.imp.pdb.facts.type.Type)|,
     |java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieSet/HashCollisionSetNode/containsKey(java.lang.Object,int,int)|,
     |java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/impl/reference/ValueFactory/listRelationWriter(org.eclipse.imp.pdb.facts.type.Type)|,
     |java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieSet/Set1To3Node/hasPayload()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/impl/fast/RelationViewOnSet/range()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/Map1/keySet()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieMap/Map2To2Node/findByKey(java.lang.Object,int,int)|,
     |java+interface:///java/lang/Override|>,
...
```

Method `sizePredicate` has an `Override` annotation

Model elements are hyperlinked

Rascal - org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java - Eclipse - /Users/paulklint/Documents/workspace-level1

Plug-in Development | Java | Debug

TrieMap.java

```
9089                throw new IllegalStateException("Index out of range.");
9090            }
9091        }
9092
9093        @Override
9094        byte sizePredicate() {
9095            return SIZE_MORE_THAN_ONE;
9096        }
9097
9098        @Override
9099        public int hashCode() {
9100            final int prime = 31;
9101            int result = 1;
```

Console | Progress | Problems | Tutor | History | Store history | Terminate | Interrupt | Trace

Rascal [DEBUG, M3Metrics]

```
rascal>model = createM3FromDirectory(project);
M3: m3(|project://org.eclipse.imp.pdb.values|)[
  @annotations={
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieMap/Map2To2Node/sizePredicate()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieMap/Map0To0Node/sizePredicate()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/type/SetType/carrier()|,|java+interface:///java/lang/Override|>,
    <|java+method:///org/eclipse/imp/pdb/facts/util/TrieSet/Set0To2Node/hasPayload()|,|java+interface:///java/lang/Override|>,
```

Exploring an M^3 Model

```
rascal>import Prelude;
ok
rascal> size(model@annotations);
int: 3035
```

Import the Rascal standard library

```
rascal>classes(model);
set[loc]: {
  |java+class:///org/eclipse/imp/pdb/facts/exceptions/NullTypeException|,
  |java+class:///org/eclipse/imp/pdb/test/persistent/TestList|,
  |java+class:///org/eclipse/imp/pdb/facts/type/IntegerType|,
  |java+class:///org/eclipse/imp/pdb/facts/type/SourceLocationType|,
  |java+class:///org/eclipse/imp/pdb/facts/util/Set4|,
  |java+class:///org/eclipse/imp/pdb/test/random/DataIterable|,
…
rascal> size(classes(model));
int: 429
```

Printing Methods per Class

```
void methodsPerClass(M3 model){
    for(c <- classes(model)){
        println("<c>: <size(methods(model, c))>");
    }
}
```

Iterate over all classes

String interpolation

The set of all methods in class c

Compute the size of that set

```
rascal>countMethods(model)
|java+class:///org/eclipse/imp/pdb/facts/exceptions/NullTypeException|: 1
|java+class:///org/eclipse/imp/pdb/test/persistent/TestList|: 1
|java+class:///org/eclipse/imp/pdb/facts/type/IntegerType|: 14
|java+class:///org/eclipse/imp/pdb/facts/type/SourceLocationType|: 12
|java+class:///org/eclipse/imp/pdb/facts/util/Set4|: 14
|java+class:///org/eclipse/imp/pdb/test/random/DataIterable|: 2
...
```

Find Classes with Most Methods

Map comprehension

All methods in c

All classes c

```
set[loc] classesWithMostMethods(M3 model){
    methodCount = ( c : size(methods(model, c)) | c <- classes(model));
    largest = max(range(methodCount));
    return { c | c <- classes(model), methodCount[c] == largest };
}
```

Maximum element in set

Range of a key/value map returns a set of the values

The set of all classes with the largest method count

```
rascal>classesWithMostMethods(model)
set[loc]: {|java+class:///org/eclipse/imp/pdb/facts/type/Type|}
```

The Java AST datatype `lang::java::m3::AST`

- It is possible to create Abstract Syntax Trees (ASTs) for all declarations in a project.
- Described by the datatypes:
  - `Declaration`
  - `Expression`
  - `Statement`
  - `Type & Modifier`

Declaration

```
data Declaration
    = \compilationUnit(list[Declaration] imports, list[Declaration] types)
    | \compilationUnit(Declaration package, list[Declaration] imports, list[Declaration] types)
    | \enum(str name, list[Type] implements, list[Declaration] constants, list[Declaration] body)
    | \enumConstant(str name, list[Expression] arguments, Declaration class)
    | \enumConstant(str name, list[Expression] arguments)
    | \class(str name, list[Type] extends, list[Type] implements, list[Declaration] body)
    | \class(list[Declaration] body)
    | \interface(str name, list[Type] extends, list[Type] implements, list[Declaration] body)
    | \field(Type \type, list[Expression] fragments)
    | \initializer(Statement initializerBody)
    | \method(Type \return, str name, list[Declaration] parameters, list[Expression] exceptions, Statement impl)
    | \method(Type \return, str name, list[Declaration] parameters, list[Expression] exceptions)
    | \constructor(str name, list[Declaration] parameters, list[Expression] exceptions, Statement impl)
    | \import(str name)
    | \package(str name)
    | \package(Declaration parentPackage, str name)
    | \variables(Type \type, list[Expression] \fragments)
    | \typeParameter(str name, list[Type] extendsList)
    | \annotationType(str name, list[Declaration] body)
    | \annotationTypeMember(Type \type, str name)
    | \annotationTypeMember(Type \type, str name, Expression defaultBlock)
    // initializers missing in parameter, is it needed in vararg?
    | \parameter(Type \type, str name, int extraDimensions)
    | \vararg(Type \type, str name)
    ;
```

Expression

```
data Expression
    = \arrayAccess(Expression array, Expression index)
    | \newArray(Type \type, list[Expression] dimensions, Expression init)
    | \newArray(Type \type, list[Expression] dimensions)
    | \arrayInitializer(list[Expression] elements)
    | \assignment(Expression lhs, str operator, Expression rhs)
    | \characterLiteral(str charValue)
    | \newObject(Expression expr, Type \type, list[Expression] args, Declaration class)
    | \newObject(Expression expr, Type \type, list[Expression] args)
    | \newObject(Type \type, list[Expression] args, Declaration class)
    | \newObject(Type \type, list[Expression] args)
    | \conditional(Expression expression, Expression thenBranch, Expression elseBranch)
    | \fieldAccess(bool isSuper, Expression expression, str name)
    | \fieldAccess(bool isSuper, str name)
    | \methodCall(bool isSuper, str name, list[Expression] arguments)
    | \methodCall(bool isSuper, Expression receiver, str name, list[Expression] arguments)
    | \number(str numberValue)
    | \booleanLiteral(bool boolValue)
    | \stringLiteral(str stringValue)
    | \type(Type \type)
    | \variable(str name, int extraDimensions)
    | \variable(str name, int extraDimensions, Expression \initializer)
    | \bracket(Expression expression)
    | \this(Expression thisExpression)
    | \infix(Expression lhs, str operator, Expression rhs)
    | \postfix(Expression operand, str operator)
    | \prefix(str operator, Expression operand)
    | \simpleName(str name)
    ...
```

Statement

```
data Statement
    = \assert(Expression expression)
    | \assert(Expression expression, Expression message)
    | \block(list[Statement] statements)
    | \break()
    | \continue()
    | \do(Statement body, Expression condition)
    | \empty()
    | \foreach(Declaration parameter, Expression collection, Statement body)
    | \for(list[Expression] initializers, Expression condition, list[Expression] updaters, Statement body)
    | \for(list[Expression] initializers, list[Expression] updaters, Statement body)
    | \if(Expression condition, Statement thenBranch)
    | \if(Expression condition, Statement thenBranch, Statement elseBranch)
    | \return(Expression expression)
    | \return()
    | \switch(Expression expression, list[Statement] statements)
    | \case(Expression expression)
    | \defaultCase()
    | \synchronizedStatement(Expression lock, Statement body)
    | \throw(Expression expression)
    | \try(Statement body, list[Statement] catchClauses)
    | \try(Statement body, list[Statement] catchClauses, Statement \finally)
    | \catch(Declaration exception, Statement body)
    | \declarationStatement(Declaration declaration)
    | \while(Expression condition, Statement body)
    | \expressionStatement(Expression stmt)
    | \constructorCall(bool isSuper, Expression expr, list[Expression] arguments)
    | \constructorCall(bool isSuper, list[Expression] arguments)
    ...
```

Type & Modifier

```
data Type
    = arrayType(Type \type)
    | parameterizedType(Type \type)
    | qualifiedType(Type qualifier,
                    Expression simpleName)
    | simpleType(Expression name)
    | unionType(list[Type] types)
    | wildcard()
    | upperbound(Type \type)
    | lowerbound(Type \type)
    | \int()
    | short()
    | long()
    | float()
    | double()
    | char()
    | string()
    | byte()
    | \void()
    | \boolean()
    ;
```

```
data Modifier
    = \private()
    | \public()
    | \protected()
    | \friendly()
    | \static()
    | \final()
    | \synchronized()
    | \transient()
    | \abstract()
    | \native()
    | \volatile()
    | \strictfp()
    | \annotation(Expression \anno)
    | \onDemand()
    ;
```

Creating ASTs

```
rascal>import lang::java::jdt::m3::AST;
ok
rascal>decls = createAstsFromEclipseProject(project, true);
set[Declaration]: {
  compilationUnit(
    package(
      package(
        package(
          package(
            package(
              package("org")[
                @decl=|java+package:///org|
              ],
              "eclipse")[
              @decl=|java+package:///eclipse|
            ],
            "imp")[
            @decl=|java+package:///imp|
          ],
          "pdb")[
          @decl=|java+package:///pdb|
        ],
        "facts")[
        @decl=|java+package:///facts|
      ],
      "visitors")[
      @decl=|java+package:///org/eclipse/imp/pdb/facts/visitors|,
      @src=|project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/visitors/IValueVisitor.java|(0,514,<1,0>,<12,43>)
    ],
    [
      import("org.eclipse.imp.pdb.facts.IBool")[
        @src=|project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/visitors/IValueVisitor.java|(516,39,<14,0>,<14,39>)
      ],
...
```

Count number of casts

```
int countCasts1(set[Declaration] decls){
    int cnt = 0;
    visit(decls){
        case \cast(_, _): cnt += 1;
    }
    return cnt;
}
```

```
rascal>countCasts1(decls)
int: 1169
```

```
int countCasts2(set[Declaration] decls) =
    size([c | /c:\cast(_,_) := decls]);
```

pattern match operator

/ does a deep match, assigns every match to c

```
rascal>countCasts2(decls)
int: 1169
```

Compute If nesting

```
set[loc] ifNesting(set[Declaration] decls, int limit){
    results = {};
    visit(decls){
        case m: \method(_, _, _, _, Statement impl):
                if(countIfNesting(impl) > limit)
                    results += m@src;
        case c: \constructor(_, _,_, Statement impl):
                if(countIfNesting(impl) > limit)
                    results += m@src;
    }
    return results;
}
```

```
int countIfNesting(Statement stat){
    top-down-break visit(stat){
        case \if(_, Statement thenBranch):
            return 1 + countIfNesting(thenBranch);
        case \if(_, Statement thenBranch, Statement elseBranch):
            return 1 + max(countIfNesting(thenBranch),
                           countIfNesting(elseBranch));
    }
    return 0;
}
```

Compute If nesting

```
rascal>ifNesting(decls, 5)
set[loc]: {
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/io/XMLWriter.java|(2860,1110,<76,1>,<116,2>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/io/XMLReader.java|(4056,1112,<91,1>,<131,2>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/io/StandardTextReader.java|(4004,1641,<110,1>,<178,2>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(140008,2791,<4966,2>,<5042,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(296114,3039,<10430,2>,<10514,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(241276,3483,<8535,2>,<8631,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(277778,3492,<9795,2>,<9892,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(274377,3397,<9696,2>,<9793,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(142803,2835,<5044,2>,<5120,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(198184,3334,<7000,2>,<7089,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(293183,2927,<10344,2>,<10428,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(244763,3561,<8633,2>,<8729,3>),
  |project://org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/util/TrieMap.java|(194907,3273,<6909,2>,<6998,3>)
}
```

Wrapping up ...

Summary

- Rascal: a rich language for meta-programming
- Growing number of libraries and extensions
- Used in real projects
- Rascal type checker and compiler in full development
- Used in teaching: Amsterdam, Eindhoven, Bergen, Namur, Open University, …
- Actively maintained by a growing community

In a Nutshell, Rascal MPL...

... has had 13,720 commits made by 43 contributors
representing 511,538 lines of code

... is mostly written in Java
with a low number of source code comments

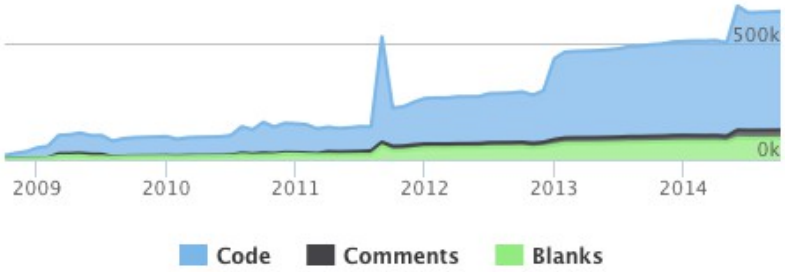
... has a well established, mature codebase
maintained by a large development team
with stable Y-O-Y commits

... took an estimated 135 years of effort (COCOMO model)
starting with its first commit in October, 2008
ending with its most recent commit 4 days ago

Languages

HTML	44%	Java	33%
JavaScript	18%	10 Other	5%

Lines of Code



Activity

30 Day Summary

Sep 25 2014 — Oct 25 2014

110 Commits
8 Contributors

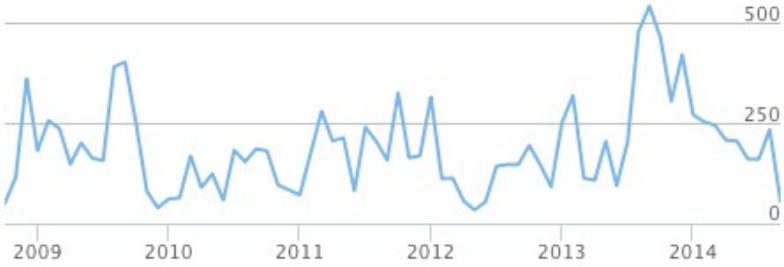
12 Month Summary

Oct 25 2013 — Oct 25 2014

2714 Commits
Down -237 (8%) from previous 12 months

20 Contributors
Down -3 (13%) from previous 12 months

Commits per Month



Rascal on OpenHub

(formerly: Ohloh)

BLACKDUCK | Open HUB

Follow @ OH | SIGN IN | JOIN NOW

PROJECTS PEOPLE ORGANIZATIONS TOOLS CODE BLOG

Projects | Search...

Rascal MPL

Settings | Report Duplicate

Very High Activity

5 I Use This!

Analyzed 4 days ago based on code collected 4 days ago.

Project Summary

Rascal is a domain specific language for source code analysis and manipulation a.k.a. meta-programming. It is currently being developed and tested at CWI. No formal release has been made yet, but there are alpha quality previews available.

Tags

programminglanguage, relationalalgebra, software_languages, metrics, textual_dsl, dsl_tools, idegenerator, languageworkbench, metaprogramming, dsl, parsergenerator, genericprogramming, ide, generalizedparsing, staticanalysis, reverseengineering, patternmatching, functional_language, statictyping

Share

Like 0 | Tweet 0 | g+1 0 | Share

Quick Reference

Project Links: Homepage, Download

Code Locations: (4 Locations)

Licenses: EPL-1.0

Similar Projects: ASF+SDF Meta..., metasharp, Racket, Whiley Compi...

Managers: Vadim Zaytsev and jurgenvinju

Browse Code

stackoverflow Questions Tags Users Badges Unanswered

Tagged Questions

info newest

Rascal is an experimental domain specific language for metaprogramming, such as static code analysis, program transformation and implementation of domain specific languages. It includes primitives from relational calculus and term rewriting. Its syntax and semantics are based on procedural ...

learn more... | top users | synonyms

0 votes
2 answers
38 views

Parsing CSV file with dynamic seperator

I try to solve the following parsing problem, but I fail: I have a CSV file that with a certain command inside the file can change seperators, and the current seperator can be escaped with a slash. ...

context-free-grammar rascal

asked Oct 9 at 14:08
JasperT
101 •1 •10

0 votes
1 answer
24 views

Can't launch Rascal files in Eclipse: 'no such java method: org.rascalmpl.library.Prelude.remove'

When trying to run any rascal file in eclipse (by right-clicking on file in Rascal Navigator and pressing Run as → Rascal Application), I get this error An internal error occurred during: "Launching ...

eclipse rascal

asked Sep 22 at 10:23
Guildenstern
248 •1 •10

0 votes
1 answer
39 views

How does syntax coloring work

What is the right way of doing syntax coloring in a grammar? I believe it was like this: syntax MappingName = @category="Constant" mappingname: Id mapping; But it doesn't work for me. The file is ...

rascal

asked Jul 16 at 9:09
JasperT
101 •1 •10

0 votes
1 answer
30 views

Integrating a Rascal project with a Java Project

I developed a code generator in Rascal and I want to integrate it with a tool developed in Java. I tried to generate a jar file for the Rascal project with eclipse to put into the Java project, but it ...

java eclipse rascal

asked Jul 11 at 18:47
user3537142
8 •2

0 votes
2 answers
62 views

Analyzing Java code across Eclipse projects

We want to use Rascal to find all unused public methods in a collection of Java projects in an Eclipse workspace. I have just learned how to create a model of a Java project in Eclipse using ...

java eclipse rascal

asked Jun 27 at 15:43
Eric Jan Malotaux
4 •1

Further reading

- http://www.rascal-mpl.org
- http://tutor.rascal-mpl.org
- http://stackoverflow.com/questions/tagged/rascal
- Rascal tutor is always one click away