Rascal Lab: Sustainable Research Software Infrastructure for Software Engineering

Jurgen J. Vinju

TU Eindhoven
NWO-I Centrum Wiskunde & Informatica
Swat.engineering BV
Sustainable Research Software Infrastructure
[something we would all like to have]

Research Infrastructure (RI): national consortia
round 2021-2022

Participants

<table>
<thead>
<tr>
<th>Title, first name, initials, surname</th>
<th>Affil.</th>
<th>Expertise</th>
</tr>
</thead>
<tbody>
<tr>
<td>prof. dr. Jurgen J. Vinju</td>
<td>CWI, TUE</td>
<td>SE, PL, Software Analysis</td>
</tr>
<tr>
<td>prof. dr. Tijs van der Storm</td>
<td>CWI, RUG</td>
<td>PL, MDE</td>
</tr>
<tr>
<td>prof. dr. Joost Visser</td>
<td>Uni. Leiden</td>
<td>SE, AI</td>
</tr>
<tr>
<td>prof. dr. Alexander Serebrenik</td>
<td>TUE</td>
<td>SE, Empirical Methods</td>
</tr>
<tr>
<td>Prof. dr. Michel R.V. Chaudron</td>
<td>TUE</td>
<td>Software Architecture</td>
</tr>
<tr>
<td>dr. Magiel Brunink</td>
<td>SIG</td>
<td>Software Quality Assessment</td>
</tr>
<tr>
<td>dr. ir. Arjan Mooij</td>
<td>ESI (TNO)</td>
<td>MDE</td>
</tr>
<tr>
<td>dr. Feliemme Hermans</td>
<td>Uni. Leiden</td>
<td>PL, Education</td>
</tr>
<tr>
<td>dr. Bastiaan Heeren</td>
<td>OU</td>
<td>PL, Software Analysis</td>
</tr>
<tr>
<td>prof. dr. ir. Bedir Tekinerdogan</td>
<td>Uni. Wagening.</td>
<td>PL, Internet of Things</td>
</tr>
<tr>
<td>dr. Andrea Capiluppi</td>
<td>Uni. Groningen</td>
<td>SE, Software Analysis</td>
</tr>
<tr>
<td>dr. Fernando Castor</td>
<td>Uni. Utrecht</td>
<td>SE, Energy Efficiency</td>
</tr>
<tr>
<td>dr. Ana-Maria Oprescu</td>
<td>UvA</td>
<td>SE, Distributed Computing</td>
</tr>
<tr>
<td>dr. Georgios Gousios</td>
<td>TU Delft</td>
<td>SE, Machine Learning</td>
</tr>
<tr>
<td>prof. dr. Andy Zaidman</td>
<td>TU Delft</td>
<td>SE, Software Testing</td>
</tr>
<tr>
<td>prof. dr. Patricia Lago</td>
<td>VU</td>
<td>SE, Energy</td>
</tr>
</tbody>
</table>
Consensus

- **Good news**: Opportunity
  - for sharing hard work on research methods
  - for more and better empirical research output
- **Bad news**: No funding
  - for the engineering of a lab
  - for the maintenance of a lab
- **Challenge**: Creating a lab requires highly educated expertise
  - Highly educated expertise is hard to find
  - Highly educated expertise is hard to keep
Aiming farther and higher

excellent research infrastructure is really really really expensive
RASCAL-LAB overview

Raw data sources
- Primary Source Code
- Secondary Code & Project Metadata
- Project Activity Data
- Ecosystem Metadata

Fact extraction instruments
- Parsing
- Name & Type Resolution
- Dependency & Configuration Extractors
- Event monitors
- Log file collectors
- Web crawlers

Core Rascal infrastructure
- Debugger
- Notebooks
- Statistics
- Cloud computing
- Source lookup

M3 hierarchical and relational software meta-model
- Cleanly ordered linkable factual data (safe)
- (annotated) syntax trees
- relational (graph) model

Reusable data enrichment, linkage and cleansing
- Well-defined and well researched data enhancement algorithms (threats to validity)
- Generated Classifiers
detectionLanguage Processing

Empirical research
- Open innovative research both “nounal” (field studies) and “verbal” (tool development)
  (exploratory, innovative, risky)

Raw data
- Many adaptors, parsers

trusted tools
M3 = URI + Relations + ADTs

Locations
java+class://java/util/List
project://myProject/src/java/util/List.java

Language agnostic core
- **containment** loc \(\times\) loc
- **declarations** loc \(\times\) loc
- **use** loc \(\times\) loc

Language specific extension
- **inheritance** loc \(\times\) loc
- **invocation** loc \(\times\) loc
- **overriding** loc \(\times\) loc

Sorts
- Exp, Stat, Decl, Type

Sigs
- If, Add, While
Managing variety by uniformity

Java

PHP

C++ code

...python, ada, bash

M3 Model

CC metric

Commit analysis

Call Graph analysis

... 

The heavy lifting is in the front-ends
But, **no analysis reuse** is guaranteed: analysis is often language specific
Versatility, extensibility of “M3”

- \( \text{rel}[\text{commitId}, \text{loc email}] \circ \text{rel}[\text{loc email}, \text{loc githubid}] \)
  
  infer committer identity for commit

- \( \text{rel}[\text{loc patchLocation}, \text{commitId}] \circ \text{rel}[\text{commitId}, \text{loc email}] \)
  
  infer committer spread over files

- \( \text{rel}[\text{loc parent}, \text{loc child}] \circ \text{rel}[\text{loc patch}, \text{loc email}] \)
  
  lift syntactic code patches to “semantic patches”

- relations are sets of tuples: closed under composition
  
  - IF the locations are indeed Universal Resource Identifiers
  
  - incremental extraction per file
  
  - compose and query over packages, projects, systems, ecosystems
Enough “bragging rights”

- Rascal has been field-tested in research since 2009; some highlights:
  - FP7 OSSMETER - OSS project analysis and reporting (code, activity, sentiments)
  - H2020 CROSSMINER - Cross-project, cross-language OSS project analysis on the ecosystem level (L. Ochoa, T. Degueule, et al.)
  - SP&E 2022 - Migration of C++ legacy code (M.T.W. Schuts, R.T.A. Aarssen, P. M. Tielemans)
  - ICSE 2017 - best paper on reflection in the Java ecosystem (D. Landman, A. Serebrenik)
  - EMSE 2021 - breaking changes in Maven grand central (L. Ochoa, T. Degueule, J-R. Falleri)
  - JSEP 2016 - on (non-existent) correlation between CC and SLOC (D. Landman, A. Serebrenik, E. Bouwers)
- IEEE SCAM 2019 Most influential paper award
- Rascal has been applied in education, bachelors’, masters’ courses and thesis projects since 2010, for example:
  - Universiteit van Amsterdam - Software Evolution, Software Construction
  - TU Eindhoven - part of Software Evolution, and Generic Language Technology
  - Open Universiteit - Software Quality Management
  - Rijksuniversiteit Groningen - Software Language Engineering
  - Hundreds of master thesis projects (UvA, TUE, OU, RUG)
- Swat.engineering BV (2017) - industrial software language engineering expertise based on Rascal
Ingredients: learned from the best

- “Software Knowledge Base” - centralized, integrated, persistent, makes tacit facts explicit
- Web and Semantic Web - addressable, linkable, compositional
- FAMIX, KDM, Rigi: accurate facts about source code. Clean intermediate formats.
- ASF+SDF: Algebraic Specification of Programming Languages: query syntax trees
- Datalog/RScript/Soul/Doop: relational query for analysis of graph-like data
- Scripting languages (Python, Ruby): versatility for a wide audience
- Functional programming: immutable data, type-safety

Rascal integrates analysis and transformation primitives with intermediate representations linguistically.
Based on open-source

- Open compilers:
  - Java M3 (Shahi, Basten), based on Eclipse JDT
  - CIAiR (Aarssen), based on Eclipse CDT
  - Ada-AiR (Decampos, vd Laar), based on libadalang
  - PHP-Analysis (Hills), based on PHP-Parser

- Libraries:
  - ASM
  - JGit
  - JSoup
  - Lucene
  - Gson
  - NanoHttpD
  - ICU
  - JDBC
  - ... all not [L]GPL

Mapping external data sources can be optimized using analysis of the open compiler’s code (R. Aarssen, vd Storm, PEPM 2020, <Programming> 2019)
false notion:
**factor** the “generally usable parts” **from** existing research methods.

False because, **these parts** (often) **do not exist**, and if they **do** they’re only “**accurate enough**”, or of **unknown** accuracy…

true notion:
**use the experience from earlier research**, to **design** and **validate** new reusable tools.
Challenges of a good SE lab

“no rocket science”

- **Diversity** of data sources: expertise of everything
- **Linking** data in unexpected ways: addressing facts
- **Accuracy**: precision, completeness, noise, bias
- **Well-definedness, meaning** of quantitative analytics and their aggregation
- **Scale** to ridiculous amounts of data (ecosystem scale)
- **Openness** (generality), to any ad-hoc specialized, innovative, measurement, reasoning

**Infrastructure** = isolating data acquisition from analysis steps, into reusable components: profit!

Quality attributes are over-emphasized, now every step must be **high-fidelity** and **always**
Diversity

- **RASCAL-LAB**: 300 proposed components
- Enough for 5 engineers to work for 5 years; almost 2M€
- 25 new *programming/scripting* languages, dynamic and static
- Dozens of (AI) *libraries*, {web,rpc} frameworks modelled
- **Natural language** sources, ownership, authorship, sentiments
- **Time**: versions, differencing, trends
- **Events** (merges, commits, issues opened or closed)
- **Log and trace** data sources
- Cross **linkage** between languages, frameworks, data sources
- etc. etc.

Design of data adapters simply does not scale… it has to be done carefully with attention to detail and validated (tested, tested, tested)
Linking

• The URI is a builtin data-type in Rascal: `loc`
  • `java+interface://java/util/List`
  • `file://Users/jurgen/.bashrc`
  • `mailto://l.m.ochoa.venegas@tue.nl`

• Semantic web-style: all identification of artefacts is via URI

• Avoid any and all confusion about identification (not OO instances!)

• Binary relation of URI is the workhorse: `rel[loc, loc] “many-to-many”`

• Composing and linking data is **union** and **join** of binary relations.

• Immutable data means safety/correctness

Design of linkage does **not scale** either
every link has specific semantics
Java-JNI-C, Makefile-GCC-Python
Accuracy: details, details…

• Requires exact syntax, names and types for code
• But, **name/type analysis** is not always exact
• The C pre-processor… grrrrr, Java’s type inference…
• Almost all data sources have inexact identification
• `rel[loc, loc] => “many-to-many”` to the rescue
• Fact extraction: get every relevant detail (**high resolution**), introduce nothing extra (**low noise**)
• **Naming:** document it clearly “email” != “author”

No free lunch: we must not “abstract from” the accuracy issue for any adapter, or any analysis
This is why analysis reuse “in-the-wild” is a threat-to-validity
Simultaneous under and over-approximation (almost every code analysis)
The meaning of metrics

- Most analyses of code will be **undecidable**, e.g. “method call graph”
- We {over, under} approximate the edges
- Then, when we count the edges, right? “at least” and “at most”
- “Extended subset” leads to **confused metrics**
- Rascal LAB solution:
  - **document** and **test** over- or under-approximation, of everything
  - **educate**: avoid complex linking and combining of over- and under approximated data
- **Reproducible research methods and benchmarks** to the rescue; this is simply really hairy and a threat-to-validity of many research methods in our field. [ICSE 2017, Landman]. So we have to re-iterate.
Openness

• Rascal is a modular GPL scripting language at core, imperative and functional at the same time.
• Any algebra, any relational calculus expression, any algorithm
• Experience: textbook algorithms in pseudocode, imperative, declarative, logical style are translated (almost) 1-to-1 to Rascal
• Fewer internal threats: no marshaling data back-and-forth to a database, logical language, graph analysis toolkit, term rewriter
• The “LAB” is an open collection of independently developed and released components. Add your own, and ignore what you wish.
• BSD2 and Eclipse licenses; free to extend and free to use
• Extensive data-export facilities: online (web)server and offline
  • all major formats and databases
Lots of work to do!
State of the proposal

- RASCAL LAB was submitted last year
- 3 excellent reviews, 1 hater
- **not** invited for a “site visit”
- Expecting final verdict from NWO today
- Next step: **double or nothing**
  - twice as many components
  - twice as many consortium partners
  - twice as many expert engineers
- The community grew in the meantime, and keeps growing
  - ada-air, rascal-git, lua-air revived, salix UIs, …
Rascal Lab: Sustainable Research Software Infrastructure for Software Engineering

- something many of us could use
- it exists already
- it must grow to make more sense
- become a consortium partner!
- become a rascal contributor!
- become a rascal user!

Jurgen.Vinju@cwi.nl