Understanding the Quality of Open Source Projects

Paul Klint

Joint work with many others

SATToSE 2013, July 8-10, 2013, Bern, Switzerland
Understanding the Quality of OSS Projects

Rascal in a Nutshell

Applying Rascal in OSSMETER

WORK IN PROGRESS
Help decision makers to assess and monitor the quality of Open Source Software (OSS) projects
Understanding the Quality of OSS Projects

- Quick replies to questions
- Frequent source code updates
- Bugs xed quickly
- Frequent releases
- Large number of downloads
- Many external references

- Late/no replies to questions
- Infrequent source code updates
- Bugs not xed
- Infrequent releases
- Small number of downloads
- Few external references
OSS Adopters

- Need support with
  - Selecting OSS software
  - Monitoring OSS software they have previously selected
Selecting OSS

• **Is a challenging task**
  - Time, cost of error, stress

• **Approaches**
  - Use over-simplified selection criteria
  - Try to avoid making a decision altogether
  - Choose arbitrary default options
  - Overestimate their expertise
  - Become highly risk averse
Monitoring OSS

• The health of OSS needs to be monitored after initial selection has taken place
  – The company behind an OSS project may fold/switch focus
  – OSS developers may change jobs (or simply lose interest)
  – A newer/better OSS may make the project obsolete

• Adopters need to be warned early enough to set up a transition plan
OSSMETER Vision

- Develop a platform that will support decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of open-source software.

- Quick replies to questions
  - Frequent source code updates
  - Bugs xed quickly
  - Frequent releases
  - Large number of downloads
  - Many external references

- Late/no replies to questions
  - Infrequent source code updates
  - Bugs not xed
  - Infrequent releases
  - Small number of downloads
  - Few external references
Information Sources

- Meta-data from various forges (SourceForge, GitHub, Eclipse, ...)
- Source code repositories
- Version management systems
- Bug trackers
- Wikis and newsgroups
Method

- Develop quality indicators for:
  - Source code quality
  - Developer activity
  - Development community
  - User community
Expected Results

- Metamodels for capturing meta-information relevant to OSS projects (e.g. types and details of source code repositories, communication channels and bug tracking systems, types of licences, number of downloads etc.)
OSSMETER Platform

- **Platform**
  - Storage, analysis
- **API**
  - Interoperability
- **Web application**
  - Presentation
- **Available both as a service and as open source software for local deployment**
  - Can also be used to measure proprietary/internal software development projects
Partners

- The Open Group (coordination)
- Univ. York (meta-models + platform)
- Univ. Manchester (text mining)
- CWI (source code & VCS metrics)
- Univ. L'Aquila (forge models)
- Tecnalia (Eclipse plugins)
- Softeam (OSS consultancy)
- Uninova (OSS for developing buildings)
Project Overview

OSSMETER aims to extend the state-of-the-art in the field of automated analysis and measurement of open-source software (OSS), and develop a platform that will support decision makers in the process of discovering, comparing, assessing and monitoring the health, quality, impact and activity of open-source software.

To achieve this, OSSMETER will compute trustworthy quality indicators by performing advanced analysis and integration of information from diverse sources including the project metadata, source code repositories, communication channels and bug tracking systems of OSS projects.

OSSMETER does not aim at building another OSS forge but instead at providing a meta-platform for analysing existing OSS projects that are developed in existing OSS forges and foundations such as SourceForge, Google Code, GitHub, Eclipse, Mozilla and Apache.

Project Beneficiaries

- **Developers and Project Managers** who are responsible for deciding on the adoption of OSS, as it will enable them to make decisions on hard facts and uniform quality indicators;
- **Developers of OSS** as it will enable them to monitor the quality of the OSS projects they contribute to, promote the OSS they contribute to using independently-calculated and trustworthy quality indicators, and identify related projects for establishing synergies with;
- **Funding Bodies** that are funding ICT projects which produce OSS, as it will allow them to monitor the quality and assess the impact of the produced software even after the end of the projects.
Rascal in a Nutshell
Rascal Contributors

- Ali Afroozeh
- Anya Bagge
- Bas Basten
- Jeroen van den Bos
- Mark Hills
- Paul Klint
- Anastasia Izmaylova
- Davy Landman
- Arnold Lankamp
- Bert Lisser
- Atze van der Ploeg
- Ashim Shahi
- Michael Steindorfer
- Tijs van der Storm
- Jurgen Vinju
- Vadim Zaytsev
- MSc students
- Incidental contributors
Rascal ...

- “One-stop shop” for
  - Meta-programming
  - Data analysis
  - Visualization
- Lab infrastructure
- Transfer medium
  - Academia
  - Industry
- http://www.rascal-mpl.org/
Understanding the Quality of OSS Projects

One-stop-shop

- Cool parsers
- Code generators
- IDE features
- DSL tools
- Fancy visualization

Deal of the day: Cheap type checkers

Just in: new modeling gadgets
Which Technical Challenges does Rascal address?

- Parse source code/data files/models
- Extract facts & perform computations on them
- Generate new source code
  - transform, refactor, compile
- Synthesize other information
  - reports, visualization
- Define DSLs and create tools including IDE support (domain-specific workbenches)
Rascal is ...

- A new language for meta-programming
- Based on Syntax Analysis, Term Rewriting, Relational Calculus
- Extended super set (regarding features not syntax!) of ASF+SDF and Rscript
- Relations used for sharing and merging of facts for different languages/modules
- Embedded in the Eclipse IDE
- Easily extensible with Java code
Rascal design based on ...

- **Principle of least surprise**
  - Familiar (Java-like) syntax
  - Mostly functional but with some imperative features

- **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 provides ...

- Rich (immutable) data: lists, sets, maps, tuples, relations, ... with comprehensions and many operators
- Syntax definitions & parser generation
- Syntax trees, tree traversal
- Pattern matching (text, trees, lists, sets, ...) and pattern-directed invocation
- Code generation (string templates & trees)
Rascal's USPs

- A single, unified, linguistic framework for:
  - **Source code analysis and transformation**
    - EASY – Extract, Analyze, Synthesize
    - Includes processing of heterogeneous data from many sources
  - **Creating Language Workbenches**
    - DSL design, implementation, workbench
- Grammars are first class citizens
- Very strong pattern matching
Grammars and Parse Trees are first class

- A BNF-like modular grammar notation
- Supports all context-free grammars
- Non-terminals integrated in the type system
- Automatic construction of parse tree or abstract syntax tree
- Typed and untyped views on all trees
- Traversal/pattern matching on all trees
- Generic tree annotations
One or more letters “a” ...

... and we call them as $a^n b^n c^n$

**syntax**

```plaintext
ABC = "a" + as "b" + bs "c" + cs;
```

```plaintext
bool properABC(ABC s) = length(s.as) == length(s.bs) &&
                        length(s.bs) == length(s.cs);
```

```plaintext
bool properABC(str s) = properABC(parse(#ABC, s));
```

```plaintext
rascal> properABC("aaabbbccc")
bool: true

rascal> properABC("aaabccc")
bool: false
```
Expressions

layout Whitespace = [\ \t\n]*;

lexical Integer = [0-9]+;

syntax E = Integer
  | E "*" E
  | E "+" E
  | "(" E ")"

int eval(str s) = eval(parse(#E, s));

int eval((E) `<Integer n>`) = toInt("<n>");
int eval((E) `<E lhs> * <E rhs>`) = eval(lhs) * eval(rhs);
int eval((E) `<E lhs> + <E rhs>`) = eval(lhs) + eval(rhs);

rascal>eval("2*3")
int: 6

rascal>eval("2*3+5")
int: 11
Sophisticated Pattern Matching with local backtracking

- Matching for all datatypes including Tree/List/Set/Tuple/Relation
- Backtracking extends over Boolean operators
- Descendant matching (nested patterns at arbitrary depth)
- Regular expressions
- Pattern-directed function invocation
- Built-in support for the visitor pattern
Rascal is Still Close to Rewriting

**ASF+SDF**

```plaintext
module PEANO
  exports
  sorts Nat
  context-free syntax
    "z"        -> Nat
    s(Nat)     -> Nat
    plus(Nat, Nat) -> Nat
  variables [nm] -> Nat
  equations
    [1] plus(s(n), m) = s(plus(n,m))
    [2] plus(z(), m) = m;
endfm
```

**Maude**

```plaintext
fmod PEANO is
  sort Nat .
  op z : -> Nat [ctor] .
  op s : Nat -> Nat [ctor] .
  vars N M : Nat .
  op plus : Nat Nat -> Nat .
  eq plus(s(N),M) = s(plus(N,M)) .
  eq plus(z,M) = M .
endfm
```

**Rascal**

```plaintext
module PEANO
  data Nat = z() | s(Nat);
  Nat plus(s(Nat n), Nat m) = s(plus(n,m));
  Nat plus(z(), Nat m) = m;
endfm
```
ColoredTrees: CTree

```java
data CTree = leaf(int N)
  | red(CTree left, CTree right)
  | black(CTree left, CTree right);

public CTree rb =
  red(black(leaf(1), red(leaf(2), leaf(3))),
      black(leaf(4), leaf(5)));
```

![Diagram of a colored tree structure](image-url)
public int cntBlack1(CTree t){
    switch(t) {
        case leaf(_): return 0;
        case black(l,r): return 1 + cntBlack1(l) + cntBlack1(r);
        case red(l,r): return cntBlack1(l) + cntBlack1(r);
    }
}

cntBlack1( ) ==> 2
Counting Black Nodes, 2

```java
public int cntBlack2(CTree t) {
    int c = 0;
    visit(t) {
        case black(,_): c = c + 1;
    };
    return c;
}
```

cntBlack2(               )  ==>  2
Counting Black Nodes, 3

Deep match of \( t \) for subtrees \( \text{black}(\_, \_) \) and bind each to \( n \)

Pattern match with subject \( t \)

\[
\text{public int} \ \text{cntBlack3}(\text{CTree} \ t) = \text{size}( [n \mid n:\text{black}(\_, \_):= t] )
\]

Collect in list comprehension and determine size

\( \text{cntBlack3}( \ ) \Rightarrow 2 \)
Dedicated Language Workbenches

- Starting point: Eclipse + IMP
- Use Rascal to create:
  - Parser, Editor with syntax highlighting
  - Typechecker, constraint checker, code generator
  - Outliner
  - Annotator
  - Connection with external tools
Parsing, Editing, Syntax highlighting

```
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
```
Code Outlining

```pascal
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;

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

Understanding the Quality of OSS Projects
User-Defined Menus
Annotations

```
MODULE Collatz;

VAR even, odd : INTEGER;

PROCEDURE doCollatz();
    VAR current : INTEGER;
    currentEven : BOOLEAN;

PROCEDURE computeEven();
    BEGIN
        IF current MOD 2 = 0 THEN
            currentEven := even
        ELSE
            currentEven := even
        END
    END
END computeEven;
```
Viewing Result of Transformation

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
end
Other Rascal Features

- Dedicated support for *source code locations*
- *Resources* allow typed access to external data sources (≈ F# type providers)
- String templates
- Read-Eval-Print (REPL)
- Extensive libraries:
  - Language front-ends: Java, PHP, XML, Lua, ...
  - Visualization, Statistics, Concept Analysis, ...
Examples of Rascal Applications
Language Analysis and Transformation

- Java
  - Metrics, control flow patterns, naming bugs, evolution, refactoring
  - OSSMETER (www.ossmeter.eu)
- PHP
  - Use of language features, metrics, dynamic includes, eval, variable variable, ...
- Lua:
  - Static analysis, interface checking (Riemer van Rozen)

Minor/planned: C#, C, Python, ...
DSLs

- **DERRIC**: a DSL for Forensics Applications
  - Jeroen van den Bos & Tijs van der Storm (CWI & NFI)
- **Magnolia**: a DSL for Scientific Computing
  - Anya Bagge (Univ Bergen, Norway)
- **Micro-Machinations**: a DSL for Game Economies
  - Riemer van Rozen (CWI, HvA, 3DMedia)
- **Pacioli**: a DSL for Computational Auditing
  - Paul Griffioen (CWI, PwC, Dutch Tax Authority)
- **QL**: a DSL for Interactive Questionnaires
  - Tijs van der Storm (Language Workbench Competition PWC 2012)
- **DSL for GPU Programming**
  - Pieter Hijma, VU
Using Rascal in OSSMETER

(joint work with Ashim Shahi and Jurgen Vinju)
Source Code Quality
and Activity Analysis
Towards a Meta-Model for Metrics Calculation

Point of Departure: Famix
Towards a Meta-Model for Metrics Calculation

We want:

- A small, flat, core
- Good extensibility
M3 is Based on Three Ideas

- **Use annotations** to extend the core
  - Annotations are mostly used to add coordinates, types and other meta-information to syntax trees
  - Now we use them to attach new source code relations to the M3 model
- **Use locations** to identify source code elements
  - `loc` is a built-in data type, based on URIs, can be used to denote files, external sources etc.
- **Represent facts as relations between locations** when possible
Annotations

Declare the annotation `containment`, it is attached to values of type M3 and has type `rel[loc from, loc to]`:

```
anno rel[loc from, loc to] M3@containment;
```

Retrieve the annotation `containment`, From the model `my_m3_model`:

```
my_m3_model@containment;
```
Understanding the Quality of OSS Projects

Locations: Anatomy of an URI

- We use the scheme-part to encode language-specific elements.
- We use the query-part to encode coordinates within a resource.

```
foo://example.com:8042/over/there?name=ferret#nose
```

<table>
<thead>
<tr>
<th>scheme</th>
<th>authority</th>
<th>path</th>
<th>query</th>
<th>fragment</th>
</tr>
</thead>
</table>

We use the scheme-part to encode language-specific elements.
Use URI to Represent Language Specific Relations (e.g., Java)

- java+compilationUnit
- java+package
- java+class
- java+package
- java+method
- java+parameter
- java+variable
- java+field
- java+interface
M3: Modular Meta-Model

M3 Core

data M3 = m3();

anno rel[loc name, loc src] M3@source;
anno rel[loc from, loc to] M3@containment;
anno list[str errorMessage] M3@projectErrors;
M3: Modular Meta-Model
M3 Core Extended for Java

```
extend experiments::m3::Core;

anno rel[loc from, loc to] M3@inheritance;
anno rel[loc from, loc to] M3@invocation;
anno rel[loc from, loc to] M3@access;
anno rel[loc from, loc to] M3@reference;
anno rel[loc from, loc to] M3@imports;
anno map[loc definition, Type typ] M3@types;
anno rel[loc definition, loc comments] M3@documentation;
anno rel[loc definition, Modifiers modifier] M3@modifiers;
```
Analyzing a Java Project

rascal>my_m3_model =
    createM3FromProject(|project://org.eclipse.imp.pdb.values|);

M3: m3()[
    @imports={
        <|java+compilationUnit:
            //org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/
            impl/util/collections/ShareableValuesHashSet.java|>,
        <|java+interface:
            //org.eclipse.imp.pdb.values/org.eclipse.imp.pdb.facts.IValue|>,

        <|java+compilationUnit:
            //org.eclipse.imp.pdb.values/src/org/eclipse/imp/pdb/facts/
            IConstructor.java|>,
        <|java+class:
            type.TypeStore|>,

        ...
    }
]
Example: Methods per Class

```cpp
facts = createM3FromProject(project);
class_with_methods =
    {<e.lhs, e.rhs> |
        tuple[loc lhs, loc rhs] e <- facts@containment,
        isClass(e.lhs),
        isMethod(e.rhs)
    };

for cls <- domain(class_with_methods))
    println("<cls> : <size(class_with_methods[cls])>");
```
Mini Demo
# Some Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Attributes per class</td>
<td>Count of attributes in a class</td>
</tr>
<tr>
<td>Number of Methods per class</td>
<td>Count of methods in a class</td>
</tr>
<tr>
<td>Weighted Methods per Class</td>
<td>( WMC = \sum_{i} C_i ), where ( C_i ) is complexity for method ( M_i )</td>
</tr>
<tr>
<td>RFC for a Class</td>
<td>( RFC =</td>
</tr>
<tr>
<td>Coupling Between Objects</td>
<td>Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.</td>
</tr>
<tr>
<td>Data Abstraction Coupling</td>
<td>Number of Abstract Data Types defined in a class</td>
</tr>
<tr>
<td>Message Passing Coupling</td>
<td>Number of send statements defined in a class</td>
</tr>
<tr>
<td>Coupling Factor</td>
<td>( CF = \sum_{i=1}^{TC} \sum_{j=1}^{TC} [is_client(C_i, C_j)] / TC^2 - TC ), where ( TC ) = total number of classes</td>
</tr>
<tr>
<td>Lack of Cohesion in Methods</td>
<td>( LCOM =</td>
</tr>
<tr>
<td>Tight Class Cohesion</td>
<td>Percentage of pairs of public methods of the class with common attribute usage (directly)</td>
</tr>
<tr>
<td>Loose Class Cohesion</td>
<td>Percentage of pairs of public methods of the class with common attribute usage (directly and indirectly)</td>
</tr>
<tr>
<td>Depth of Inheritance Tree</td>
<td>Number of ancestor classes</td>
</tr>
<tr>
<td>Number of Children</td>
<td>Number of immediate subclasses</td>
</tr>
</tbody>
</table>
### Some Metrics contd.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method Inheritance Factor</strong></td>
<td>(MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}), where (M_a(C_i) = M_i(C_i) + M_d(C_i)), (TC = \text{total number of classes}), (M_d(C_i) = \text{the number of methods declared in a class}), (M_i(C_i) = \text{the number of methods inherited in a class})</td>
</tr>
<tr>
<td><strong>Attribute Inheritance Factor</strong></td>
<td>Same as above, replace methods with attributes</td>
</tr>
<tr>
<td><strong>Method Hiding Factor</strong></td>
<td>(MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{TC} \left(1 - V(M_{mi})\right) M_{mi}(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}), where (M_d(C_i) = \text{number of methods defined in a class}), (V(M_{mi}) = \frac{\sum_{j=1}^{TC} \text{is_visible}(M_{mi}, C_j)}{(TC - 1)}), where (TC = \text{the total number of classes}) and (\text{is_visible}(M_{mi}, C_j) = \begin{cases} 1 &amp; \text{iff } j \neq i \land C_j \text{ may call } M_{mi} \ 0 &amp; \text{otherwise} \end{cases})</td>
</tr>
<tr>
<td><strong>Attribute Hiding Factor</strong></td>
<td>Same as above, replace methods with attributes</td>
</tr>
<tr>
<td><strong>Polymorphism Factor</strong></td>
<td>(PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}), where (M_n(C_i) = \text{Number of new methods}), (M_o(C_i) = \text{Number of Overriding Methods}), (DC(C_i) = \text{Descendants Count})</td>
</tr>
</tbody>
</table>
## Metrics contd.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reuse Ratio</strong></td>
<td>( U = \frac{\text{Number of superclasses}}{\text{Total Number of classes}} )</td>
</tr>
<tr>
<td><strong>Specialization Ratio</strong></td>
<td>( S = \frac{\text{Number of subclasses}}{\text{Number of superclasses}} )</td>
</tr>
<tr>
<td><strong>Lines Of Code</strong></td>
<td></td>
</tr>
<tr>
<td><strong>McCabe’s Cyclomatic Complexity</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Afferent Coupling</strong></td>
<td>Incoming dependencies</td>
</tr>
<tr>
<td><strong>Efferent Coupling</strong></td>
<td>Outgoing dependencies</td>
</tr>
<tr>
<td><strong>Instability</strong></td>
<td>( I = \frac{\text{Efferent Coupling}}{\text{Efferent Coupling} + \text{Afferent Coupling}} )</td>
</tr>
<tr>
<td><strong>Abstractness</strong></td>
<td>( A = \frac{\text{Number of abstract classes}}{\text{Number of concrete classes}} )</td>
</tr>
</tbody>
</table>
Understanding the Quality of OSS Projects
Wrapping up ...
Summary

- **OSSMETER** aims at measuring the quality of OSS projects
- Rascal is a language for meta-programming
- In **OSSMETER** Rascal is being used to compute:
  - Source code metrics
  - Activity metrics
Information

See:

- http://www.rascal-mpl.org
- http://tutor.rascal-mpl.org
- http://www.cwi.nl/~paulk
Bonus Slides
Random Testing in Rascal ...
Points of Departure

- **QuickCheck (Koen Claassen & John Hughes, 2000):**
  - Specify a property of a program as a Boolean function with one or more parameters
  - Test that the property holds for randomly generated parameter values
- **Integrated in Rascal by Wietse Venema (MSc project)**
Reversing a List

```
public list[int] rev1(list[int] L) = size(L) <= 1 ? L : rev1(L[1..]) + [L[0]];

public list[int] rev2([]) = [];
public list[int] rev2([int X, *L]) = rev2(L) + [X];

public list[int] rev3(list[int] L) {
    res = [];
    for(x <- L)
        res += [x];
    return res;
}

public list[int] rev4(list[int] L) {
    res = [];
    for(i <- [size(L)-1 .. -1])
        res += [L[i]];
    return res;
}
```

```
test bool t1a() = rev1([1,2]) == [2,1];
test bool t1b() = rev1([]) == [];
test bool t2a() = rev2([1,2]) == [2,1];
test bool t2b() = rev2([1]) == [1];
test bool t3a() = rev3([1,1]) == [1,1];
test bool t3b() = rev3([]) == [];
test bool t4a() = rev4([1,2]) == [2,1];
test bool t4b() = rev4([1]) == [1];
```
Reversing a List

```java
public list[int] rev1(list[int] L) = size(L) <= 1 ? L : rev1(L[1..]) + [L[0]];

public list[int] rev2([]) = [];
public list[int] rev2([int X, *L]) = rev2(L) + [X];

public list[int] rev3(list[int] L) {
    res = [];
    for(x <- L)
        res += [x];
    return res;
}

public list[int] rev4(list[int] L) {
    res = [];
    for(i <- [size(L)-1 .. -1])
        res += [L[i]];
    return res;
}

test bool prop1(list[int] L) = L == rev1(rev1(L));
test bool prop2(list[int] L) = L == rev2(rev2(L));
test bool prop3(list[int] L) = L == rev3(rev3(L));
test bool prop4(list[int] L) = L == rev4(rev4(L));
```
Reversing a List

```java
public list[int] rev1(list[int] L) = size(L) <= 1 ? L : rev1(L[1..]) + [L[0]];

public list[int] rev2([]) = [];
public list[int] rev2([int X, *L]) = rev2(L) + [X];

public list[int] rev3(list[int] L) {
    res = [];
    for(x <- L) res += [x];
    return res;
}

public list[int] rev4(list[int] L) {
    res = [];
    for(i <- [size(L)-1 .. -1]) res += [L[i]];
    return res;
}

test bool prop1(list[int] L) = isEmpty(L) || all(i <- index(L), L[i] == rev1(L)[size(L) - i - 1]);
test bool prop2(list[int] L) = isEmpty(L) || all(i <- index(L), L[i] == rev2(L)[size(L) - i - 1]);
test bool prop3(list[int] L) = isEmpty(L) || all(i <- index(L), L[i] == rev3(L)[size(L) - i - 1]);
test bool prop4(list[int] L) = isEmpty(L) || all(i <- index(L), L[i] == rev4(L)[size(L) - i - 1]);
```
Reversing a List

```java
public list[int] rev3(list[int] L) {
    res = [];
    for(x <- L)
        res += [x];
    return res;
}
```

```java
public list[int] rev3c(list[int] L) {
    res = [];
    for(x <- L)
        res = [x] + res;
    return res;
}
```

```rascal
rascal>rev3([1,2,3])
list[int]: [1,2,3]
```

```rascal
rascal>rev3c([1,2,3])
list[int]: [3,2,1]
```

We are rebuilding the result in the same order

```rascal
test bool prop3c(list[int] L) = isEmpty(L) || all(i <- index(L), L[i] == rev3c(L)[size(L) - i - 1]);
```
return res;

public list[int] rev4(list[int] L) {
    res = [];
    for(i <- [size(L)-1 .. -1])
        res += [L[i]];
    return res;
}

test bool t1a() = rev1([1,2]) == [2,1];
test bool t1b() = rev1([]) == [];
test bool t2a() = rev2([1,2]) == [2,1];
test bool t2b() = rev2([1]) == [1];
test bool t3a() = rev3([1,1]) == [1,1];
test bool t3b() = rev3([]) == [];
test bool t4a() = rev4([1,2]) == [2,1];
test bool t4b() = rev4([1]) == [1];

test bool prop1(list[int] L) = L == rev1(rev1(L));
test bool prop2(list[int] L) = L == rev2(rev2(L));
test bool prop3(list[int] L) = L == rev3(rev3(L));
test bool prop4(list[int] L) = L == rev4(rev4(L));

test bool prop1(list[int] L) = size(L) <= 1 ? rev1(L) == L : all(i <- index(L), L[i] == rev1(L));
test bool prop2(list[int] L) = size(L) <= 1 ? rev2(L) == L : all(i <- index(L), L[i] == rev2(L));
test bool prop3(list[int] L) = size(L) <= 1 ? rev3(L) == L : all(i <- index(L), L[i] == rev3(L));
test bool prop4(list[int] L) = size(L) <= 1 ? rev4(L) == L : all(i <- index(L), L[i] == rev4(L));

test bool prop3c(list[int] L) = size(L) <= 1 ? rev3c(L) == L : all(i <- index(L), L[i] == rev3c(L));

Understanding the Quality of OSS Projects
Random Testing and Parameterized Types

```
public list[int] rev1(list[int] L) = size(L) <= 1 ? L : rev1(L[1..]) + [L[0]];
```

```
test bool prop1(list[int] L) =
    isEmpty(L) || all(i <- index(L), L[i] == rev1(L)[size(L) - i - 1]);
```

Can be generalized to

```
public list[&T] rev1(list[&T] L) = size(L) <= 1 ? L : rev1(L[1..]) + [L[0]];
```

```
test bool prop1(list[&T] L) =
    isEmpty(L) || all(i <- index(L), L[i] == rev1(L)[size(L) - i - 1]);
```

The random test framework will call such a test with lists of arbitrary element type.
Testing in the Rascal Implementation

- Circa 3000 Junit tests in Rascal implementation
- Circa 900 random test cases → 180000 tests
- We are gradually moving from manual test cases to random test cases
- Part of continuous integration tool chain etc etc.