CWI SWAT & Rascal

NWO Special Interest Group Software Engineering
Nov 14th, 2013
Jurgen Vinju
• Centrum Wiskunde & Informatica
• **Programming** languages and systems
  • Algol
  • Python
  • ASF+SDF, Rascal
  • MonetDB
• Software Improvement Group (spin-off)
  • Software Quality Assessment & Monitoring
  • Reverse Engineering
• CWI SWAT ⇔ INRIA ATEAMS
  • all about source code
  • supporting the tasks of programmers
• Master Software Engineering @{Universiteit van Amsterdam, VU, HvA}
Today

- What and why do we research software at CWI?
- Discussions
  - What is “software engineering” to you?
  - Quality for scientific software
SWAT team
The problem with software is not in constructing it (given sufficiently experienced architects & engineers)
The problem is in understanding existing software in order to improve it (and a lot of software exists)
We study software systems: their design, their construction and their inevitable evolution.

- learning to **understand** software systems
- learning to **improve** them
- focusing on **complexity** as the quality attribute
- studying the **causes** of software complexity
- studying **solutions** to get simpler software

- helping programmers to be more effective while
  - designing, constructing, maintaining, testing, debugging
- shaping the future of programming languages and IDEs

*(NASA mission control, apollo 13)*
Software is not so difficult to understand, but it is extremely complex
Software - large and complex structures of computer instructions, written and read by man, executed by computers.

“marked by a senseless, disorienting, often menacing complexity...” (Infoplease.com)
The source code of “ls”

3894 lines

367 ifs

174 cases
Solution...

Tools
Transformation & Analysis

- (de)optimization
- GOTO removal
- Bug fixing (Y2K)
- Porting
- Refactoring
- Model-to-code
- Domain specific languages
- Code-to-model

(etc)

- Quality assessment
- Mining trends
- Dead code detection
- Bug detection
- Model checking
- Impact analysis
- Guided random testing
- Visualization

Raphael (1509)
“every week a new tool”
Rascal is a DSL for meta programming

(Bruygel, Tower of Babel)
The three challenges

Diversity

Multi-disciplinary

Precision vs Efficiency
Example Rascal Tasks
Example Rascal Tasks

- Measure aspects of source code
Example Rascal Tasks

- Measure aspects of source code
- Compare different versions of the same system
Example Rascal Tasks

- Measure aspects of source code
- Compare different versions of the same system
- Consistently transform a big system to another
Example Rascal Tasks

- Measure aspects of source code
- Compare different versions of the same system
- Consistently transform a big system to another
- Translate between different formalisms
Example Rascal Tasks

- Measure aspects of source code
- Compare different versions of the same system
- Consistently transform a big system to another
- Translate between different formalisms
- Consolidate data about software systems (bug trackers) with source code analysis results.
Example Rascal Tasks

- Measure aspects of source code
- Compare different versions of the same system
- Consistently transform a big system to another
- Translate between different formalisms
- Consolidate data about software systems (bug trackers) with source code analysis results.
- Generate code from a new kind of model (DSL)
Example Rascal Tasks

- Measure aspects of source code
- Compare different versions of the same system
- Consistently transform a big system to another
- Translate between different formalisms
- Consolidate data about software systems (bug trackers) with source code analysis results.
- Generate code from a new kind of model (DSL)
- Translate from code to a model to generate test cases
Rascal for Digital Forensics
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
Rascal for Digital Forensics

• “Derric” [Jeroen van den Bos, Tijs van der Storm]
• Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
- Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
- State-of-the-art: hand optimized/specialized code
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
- Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
- State-of-the-art: hand optimized/specialized code
- Problem: variety of file formats [dialects]
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
- Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
- State-of-the-art: hand optimized/specialized code
- Problem: variety of file formats [dialects]
- Solution: generate and optimize automatically
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
- Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
- State-of-the-art: hand optimized/specialized code
- Problem: variety of file formats [dialects]
- Solution: generate and optimize automatically
- Model: detailed file format description
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
- Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
- State-of-the-art: hand optimized/specialized code
- Problem: variety of file formats [dialects]
- Solution: generate and optimize automatically
- Model: detailed file format description
- Code: plugins for file carver in Java
Rascal for Digital Forensics

- “Derric” [Jeroen van den Bos, Tijs van der Storm]
- Recover deleted, permuted, lost, evidence from data sources in the terabyte range, and quickly!
- State-of-the-art: hand optimized/specialized code
- Problem: variety of file formats [dialects]
- Solution: generate and optimize automatically
- Model: detailed file format description
- Code: plugins for file carver in Java
- Success factors: expert knowledge + rascal implementation
One slide DSL
metro {
    Centraal Waterloo Weesperplein Wibautstraat Amstel;
    Amstel Spaklerweg Overamstel Rai Zuid;
    Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
    Centraal Rokin FerdinandBol Zuid;
}
metro {
    Centraal Waterloo Weesperplein Wibautstraat Amstel;
    Amstel Spaklerweg Overamstel Rai Zuid;
    Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
    Centraal Rokin FerdinandBol Zuid;
}

{ <"Centraal", "Waterloo"},
One slide DSL

```
metro {
    Centraal Waterloo Weesperplein Wibautstraat Amstel;
    Amstel Spaklerweg Overamstel Rai Zuid;
    Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
    Centraal Rokin Ferdinand Bol Zuid;
}

{ <"Centraal", "Waterloo">,
  <"Waterloo"," Weesperplein">, ... }
```
One slide DSL

```plaintext
metro {
    Centraal Waterloo Weesperplein Wibautstraat Amstel;
    Amstel Spaklerweg Overamstel Rai Zuid;
    Rai Zuid Amstelveenseweg Lelylaan Sloterdijk;
    Centraal Rokin FerdinandBol Zuid;
}

{ <"Centraal", "Waterloo">,
  <"Waterloo"," Weesperplein">, ... }

digraph Metro {
    node [shape=box]
    Centraal -> Waterloo
    Waterloo -> Weesperplein ...
    Centraal [shape=ellipse]
}
```
module Metro
module Metro

start syntax System = "metro" "{" Track* tracks "}";
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
module Metro

start syntax System = "metro" "{{ Track* tracks }}";

syntax Track = Id+ stations ";";

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\ \t\n\r]*;
module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";";

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =


module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),}
module Metro

start syntax System = "metro" "{“ Track* tracks “}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [
\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
{<from, to> | /
Track t := parse(#start[System], source),
(Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` `:= t};
module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z][A-Za-z0-9]*;

layout WS = [	

rel[Id,Id] extractMetroGraph(loc source) =

  {<from, to> | /Track t := \parse(#start[System], source),
    (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};

bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;
module Metro

start syntax System = "metro" "{ " Track* tracks " }";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {


module Metro

start syntax System = "metro" "{" Track* tracks "}";

syntax Track = Id+ stations ";" ;

lexical Id = [A-Za-z] [A-Za-z0-9]* ;

layout WS = [\ \t\n\r]* ;

rel[Id,Id] extractMetroGraph(loc source) =
  {<from, to> | /Track t := parse(#start[System], source),
    (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` `:= t};

bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1 ;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,"digraph Metro { node [shape=box] 

  ...
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]+;
layout WS = ['\t' '\n' '\r' '\r']*;

rel[Id,Id] extractMetroGraph(loc source) =
{<from, to> | /Track t := parse(#start[System], source),
 (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro { node [shape=box]
        '<for (<from, to> <- metro) {
        


module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\ \t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
   {<from, to> | /Track t := parse(#start[System], source),
      (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` `:= t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
   writeFile(target,"digraph Metro { node [shape=box]
         `<for (<from, to> <- metro) {>
            `<from> \-> <to><>
         
   

Friday, November 15, 13
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =

{<from, to> | /Track t := parse(#start[System], source),
 (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};

bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
  writeFile(target,"digraph Metro {
    node [shape=box]
      `<for (<from, to> <- metro) {>
      `  <from> -> <to><}>
    `<for (st <- metro<from>, isHub(metro, st)){>

module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";" ;
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),
    (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro { node [shape=box]
        '<for (<from, to> <- metro) {>
        '    <from> -> <to></>
        '<for (st <- metro<from>, isHub(metro, st)){>
        '    <st> [shape=ellipse]<}
    
```
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;`' := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro {
        node [shape=box]
        `<for (<from, to> <- metro) {>
            '<from> -
            '<to></>
            `<for (st <- metro<from>, isHub(metro, st)){>
                '<st> [shape=ellipse]<}
        '}
    ");
module Metro

start syntax System = "metro" "{" Track* tracks "}";
syntax Track = Id+ stations ";";
lexical Id = [A-Za-z][A-Za-z0-9]*;
layout WS = [\t\n\r]*;

rel[Id,Id] extractMetroGraph(loc source) =
    {<from, to> | /Track t := parse(#start[System], source),
        (Track) `<Id+ _> <Id from> <Id to> <Id+ _>;` := t};
bool isHub(rel[Id,Id] metro, Id station) = size(metro[Id]) > 1;

void synthesizeDotGraph(loc target, rel[Id from,Id to] metro) {
    writeFile(target,"digraph Metro { node [shape=box]
        ‘<for (<from, to> <- metro) {>
            ‘<from> -\> <to><}
        ‘<for (st <- metro<from>, isHub(metro, st)){>
            ‘<st> [shape=ellipse]<}
        ‘}”);
A one-slide DSL

- What is the point?
  - Rapid tool development
  - No boilerplate
  - No glue
  - No magic
  - Done. Next!
- This works for
  - all kinds of meta-programming tools
  - all kinds of languages
IDE “generation”
Current applications

- PHP, Lua static analysis of dynamic languages
- Modular/Language parametric refactoring
- Grammar engineering
- Domain specific languages
  - Pacioli - Computational auditing
  - Derric - Digital Forensics
  - QL - Complex Questionnaires
- GPU programming
Rascal

- http://www.rascal-mpl.org
- open-source on github
- tools for tools
- documented: http://tutor.rascal-mpl.org
- supported: http://ask.rascal-mpl.org
- “alpha” = under development (language & libraries)
- active: compiler & static checker, adding support for units & dimensions, scripting languages, grammars for legacy languages, libraries for SMT solvers, etc. etc.
D.I.Y.

- That’s the goal
- We teach and use it
- Caveat emptor
Discussion(s)
Discussion(s)

- What does “software engineering” mean to you?
  - critical or non-critical? has this changed?
  - stakeholders, requirements, deadlines
  - art, science or engineering (or all)
Discussion(s)

• What does “software engineering” mean to you?
  • critical or non-critical? has this changed?
  • stakeholders, requirements, deadlines
  • art, science or engineering (or all)
• Quality for scientific software
  • what are important quality aspects?
  • which tools are used?
  • which methods?
  • which problems are hard to solve?
  • what does the future look like?
Take home messages

- [ ] http://www.rascal mpl.org
- open-source
- tools for tools
- CWI SWAT
  - studies real software
  - in software domains: bio, finance, forensics, law, ...
- builds and evaluates tools
- Master Software Engineering exists
From coding to declaring

list[int] even(int max) {
    list[int] result = [];

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

    return result;
}

list[int] even(int max)
    = [ i | i <- [0..max], i % 2 == 0];
From coding to declaring

```java
list[int] even(int max) {
    list[int] result = [];

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

    return result;
}
```
From coding to declaring

```java
list[int] even(int max) {
    result = [];
    for (i <- [0..max], i%2 == 0) {
        result += i;
    }
    return result;
}
```
From coding to declaring

```plaintext
list[int] even(int max) {
    r = for (i <- [0..max], i%2 == 0) append i;
    return r;
}
```
From coding to declaring

```python
list[int] even(int max) {
    return for (i <- [0..max], i%2 == 0)
        append i;
}
```
From coding to declaring

```
list[int] even(int max) {
    return [i | i <- [0..max], i%2 == 0];
}
```
From coding to declaring

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