Software is Complex
Software is Complex

* The principles of software are easy
Software is Complex

• The principles of software are easy
  • Just a bunch of computer instructions
The principles of software are easy

- Just a bunch of computer instructions
- IO, arithmetic, control, done.
Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code
Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.

- The practice of software is incomprehensible
  - There is too much code
  - There is too much diversity
Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code
  - There is too much diversity
  - The CPU is too fast
Software is Complex

- The principles of software are easy
  - Just a bunch of computer instructions
  - IO, arithmetic, control, done.
- The practice of software is incomprehensible
  - There is too much code
  - There is too much diversity
  - The CPU is too fast
  - There is too much memory
Instruments
Instruments

- Higher level programming
- Model Driven Engineering
- Domain Specific Languages
Instruments

- Higher level programming
  - Model Driven Engineering
  - Domain Specific Languages
- Program comprehension
  - Static & dynamic analysis
- IDEs
Instruments

- Higher level programming
  - Model Driven Engineering
  - Domain Specific Languages
- Program comprehension
  - Static & dynamic analysis
  - IDEs
- Observing Software
  - Mining Software Repositories
- Measurement & benchmarking
Higher level programming
- Model Driven Engineering
- Domain Specific Languages
Program comprehension
- Static & dynamic analysis
- IDEs
Observing Software
- Mining Software Repositories
Measurement & benchmarking
Problem

- Instruments are costly
PROBLEM

- Instruments are costly
- Diverse
Instruments are costly

Diverse

Multi-disciplinary
PROBLEM

- Instruments are costly
- Diverse
- Multi-disciplinary
- Accurate/Efficient
RASCAL

ENABLES

Research

Application
RASCAL

A new tool every week

ENABLES

Research

Application
RASCAL

A new tool every week

One-stop-shop

ENABLES

Research

Application
Rascal is EASY
Rascal is EASY

- Extract,

- Parse DSLs and (legacy) GPLs

- Create abstract relational models (like call graphs)
Rascal is EASY

- **Extract**,
  - Parse DSLs and (legacy) GPLs
  - Create abstract relational models (like call graphs)
- **Analyze**
  - Query abstract/concrete syntax trees
  - Query relational models
Rascal is EASY

- Extract,
  - Parse DSLs and (legacy) GPLs
  - Create abstract relational models (like call graphs)
- Analyze
  - Query abstract/concrete syntax trees
  - Query relational models
- Synthesize
  - Generate code
  - Transform code
  - Produce data and visuals
Rascal is EASY
Rascal is EASY

- Extract,
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
**Rascal is EASY**

- Extract,
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
- Analyze
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...

(CWI)

Friday, June 15, 12
Rascal is EASY

- **Extract**,
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal

- **Analyze**
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...

- **SYnthesize**
  - String templates
  - Concrete syntax
  - Interactive visualization generator
A one-slide DSL

- Code
- Model
- Picture
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"},
A 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">, ... }
```
A one-slide DSL

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

Code

Model

Picture
A one-slide DSL
module Metro

A one-slide DSL
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 = [\ \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;
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]*;

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><}>
   
   writeFile(target,""");
}


Friday, June 15, 12
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;

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]<}

Friday, June 15, 12
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) {
  readFile(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

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) {>
            '<for (<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!
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
Library development

Diagram:
- Code
- Model
- Picture
Type-safe access to resources such as bug databases, version management systems, spreadsheets, webservices
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages
Library development

- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices
- **Front-ends** for programming languages
- **Generic analyses**: statistics, constraints, satisfiability, …
- **Type-safe** access to resources such as bug databases, version management systems, spreadsheets, webservices

- **Front-ends** for programming languages

- **Generic analyses**; statistics, constraints, satisfiability, …

- **Visualization**: one-stop-library for any visualization (graph, chart, browser, …)
State of the art
Rascal is \textit{beta} quality, \textit{alpha} guarantees
State of the art

- Rascal is beta quality, alpha guarantees

Experience

- Software Evolution/Construction at UvA & OU
- Bachelors Amsterdam University College (AUC)
- Applications in SWAT (see posters)
State of the art

- Rascal is **beta** quality, *alpha* guarantees

Experience

- Software Evolution/Construction at UvA & OU
- Bachelors Amsterdam University College (AUC)
- Applications in SWAT (see posters)
- Currently applying lessons-learned
FOR RESEARCH IN ANALYSIS AND SYNTHESIS OF COMPLEX SOFTWARE ARTIFACTS
Software may be hard but Rascal is easy for research in analysis and synthesis of complex software artifacts.
Software may be hard but Rascal is easy.

For research in analysis and synthesis of complex software artifacts.