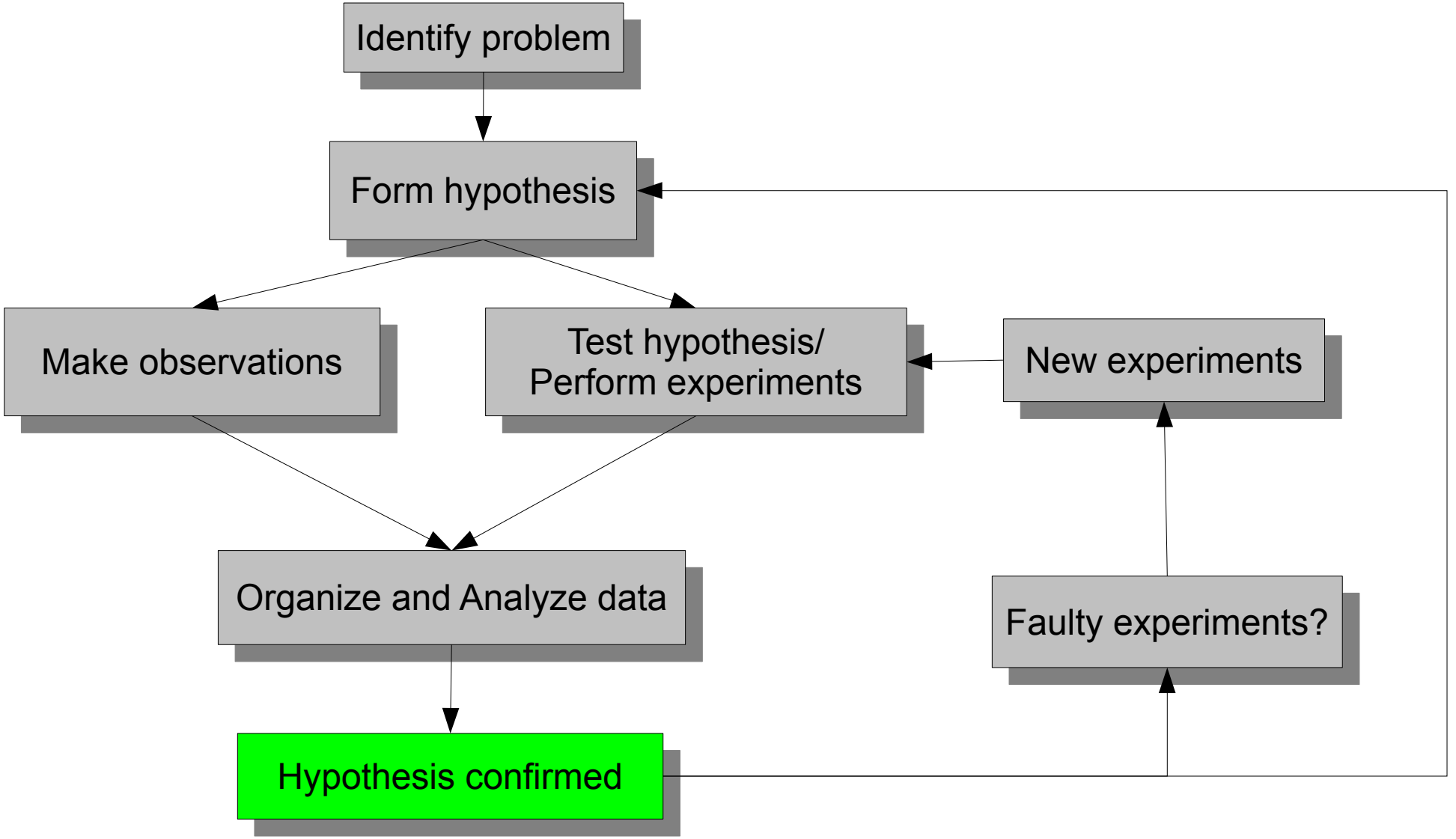
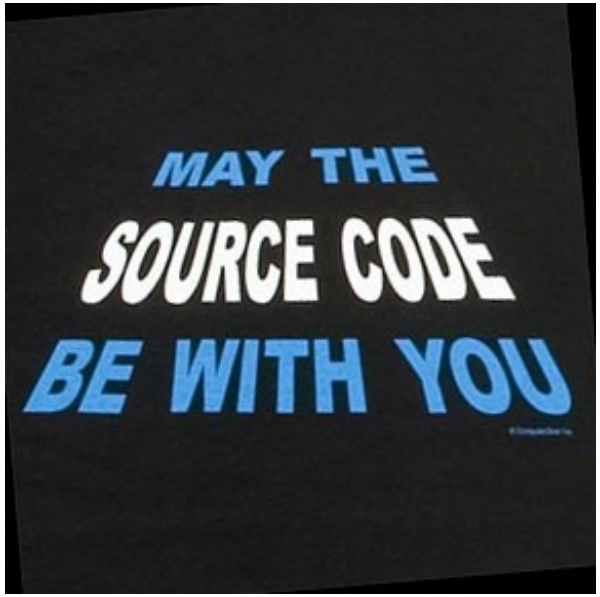
Building Academic Software Tools
Do's and Don'ts

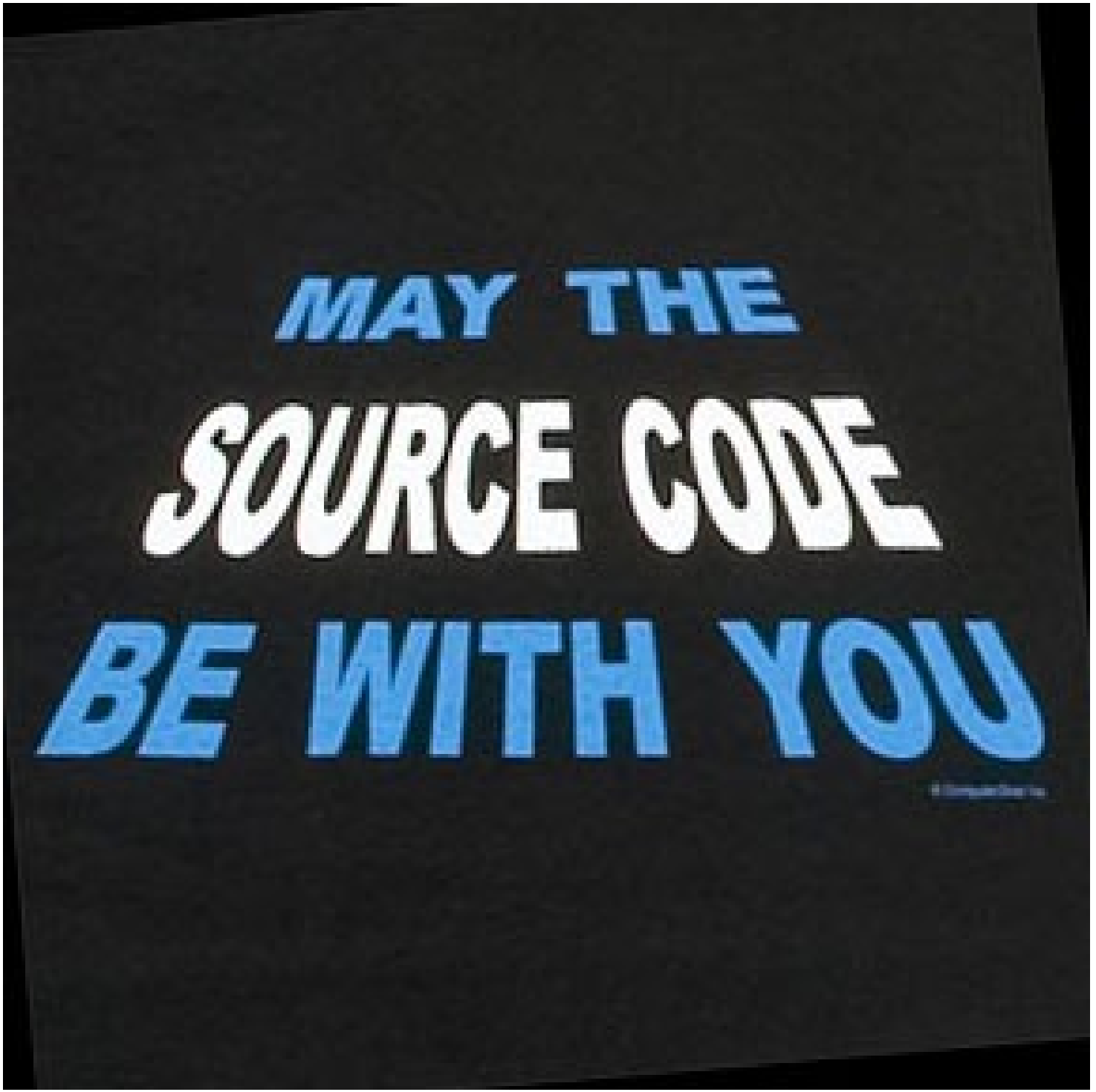
Paul Klint

The Scientific Method



Subjects of Study in *Software Engineering Research*

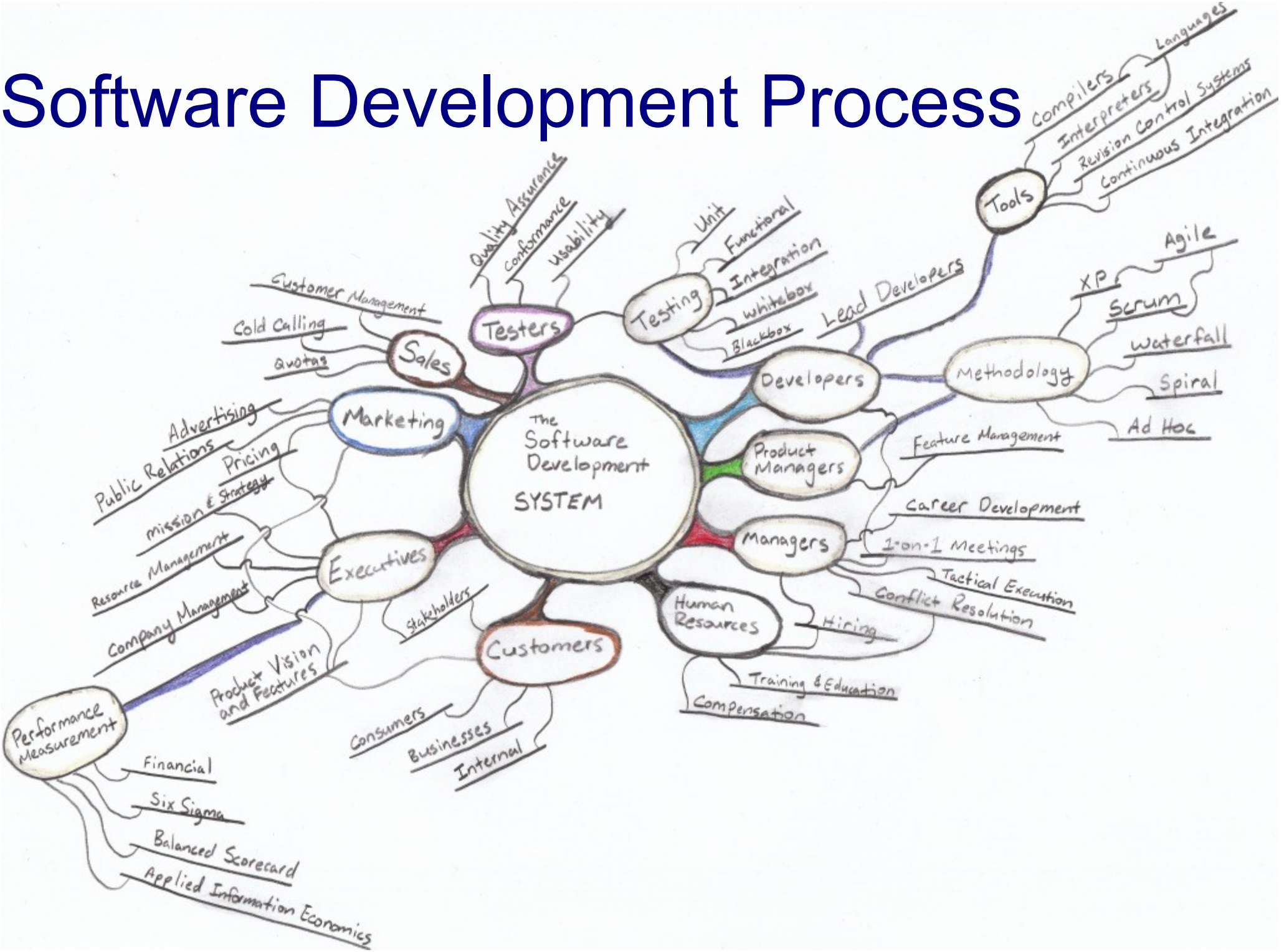



MAY THE
SOURCE CODE
BE WITH YOU

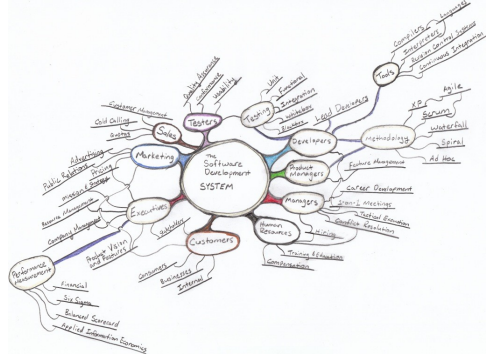
Source code

- **Goal**: determine properties of existing source code or create techniques for building better source code
- **Examples**:
  - Source code properties (metrics, clones, use of APIs, ...)
  - New programming language or DSL
- **Tools**: parsing, compiling, source code analysis, metrics, statistical analysis, ...

Software Development Process



Software Development Process



- **Goal**: study effects of (steps in) the software development process
- **Use**: version histories, management data, bug trackers, test logs, ...
- **Examples**:
  - Compare different processes, e.g. XP vs DSDM
  - Quality of specific step, e.g., test process
- **Tools**: analyzing version histories, bug trackers, productivity data, etc.

People

People

- **Goal**: study how usable, understandable, effective a technology is
- **Use**: interviews, user observation, controlled experiments, ...
- **Examples**:
  - Are OO/imperative programs more understandable?
  - Is bug finding easier in language X or Y?
- **Tools**: (online) interviews, user monitoring, statistical analysis, ...

Pitfalls in Software Engineering Research

Paul Klint

Pitfalls in Software Engineering Research

- Source code:
  - Size matters: small examples do not scale
- Software development process:
  - Size matters: impossible to redo a multi-million project with a new software process
- People:
  - Hard to get number of subjects that gives statistically relevant experiments
  - Unaware of existing methodologies in sociology and psychology

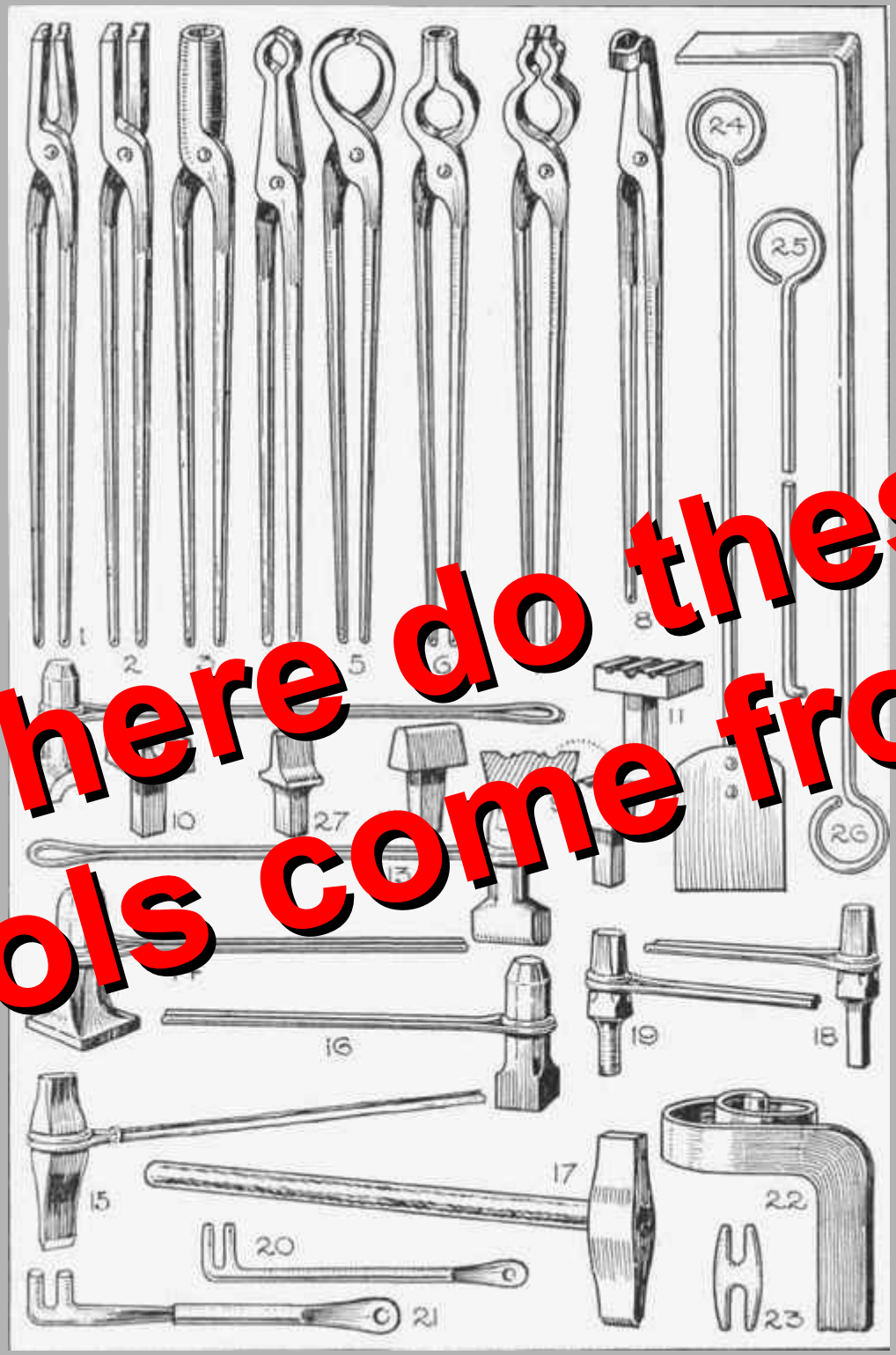
Pitfalls in Software Engineering Research

- Validation
- Validation
- Validation
- Validation
- Validation
- Validation
- Validation
- Validation

Suppose You have Determined ...

- Subject of study
- Hypothesis
- Research method
- Experimental setup
- Validation method
- Needed tools

Needed Tools



Where do these tools come from?

Three Strategies

Reuse existing tool

Write a throw-away tool

Write reusable tool

Reuse Existing Tool

- Low investment, but ...
  - Documentation and usability of many (research) tools is poor
  - Many existing tools are broken and you end up fixing them
  - Combining different tools can be a challenge
- Limited to what is available
- Your results are reproducible
- Easy transfer of results

Write a Throw-Away Tool

- Low investment
  - But may be larger than anticipated
  - Quick method to get results
- Results not reproducible
  - Limited to the examples in your paper
  - Hard for others to build on your work

Write a Re-Usable Tool

- High investment
  - Amortize investment over more research projects
  - Explore new technical approaches
  - Management in a research setting
  - Maintenance
- Get ***real*** software engineering experience!
- Results are reproducible
- When successful: more visible impact on research community and industry

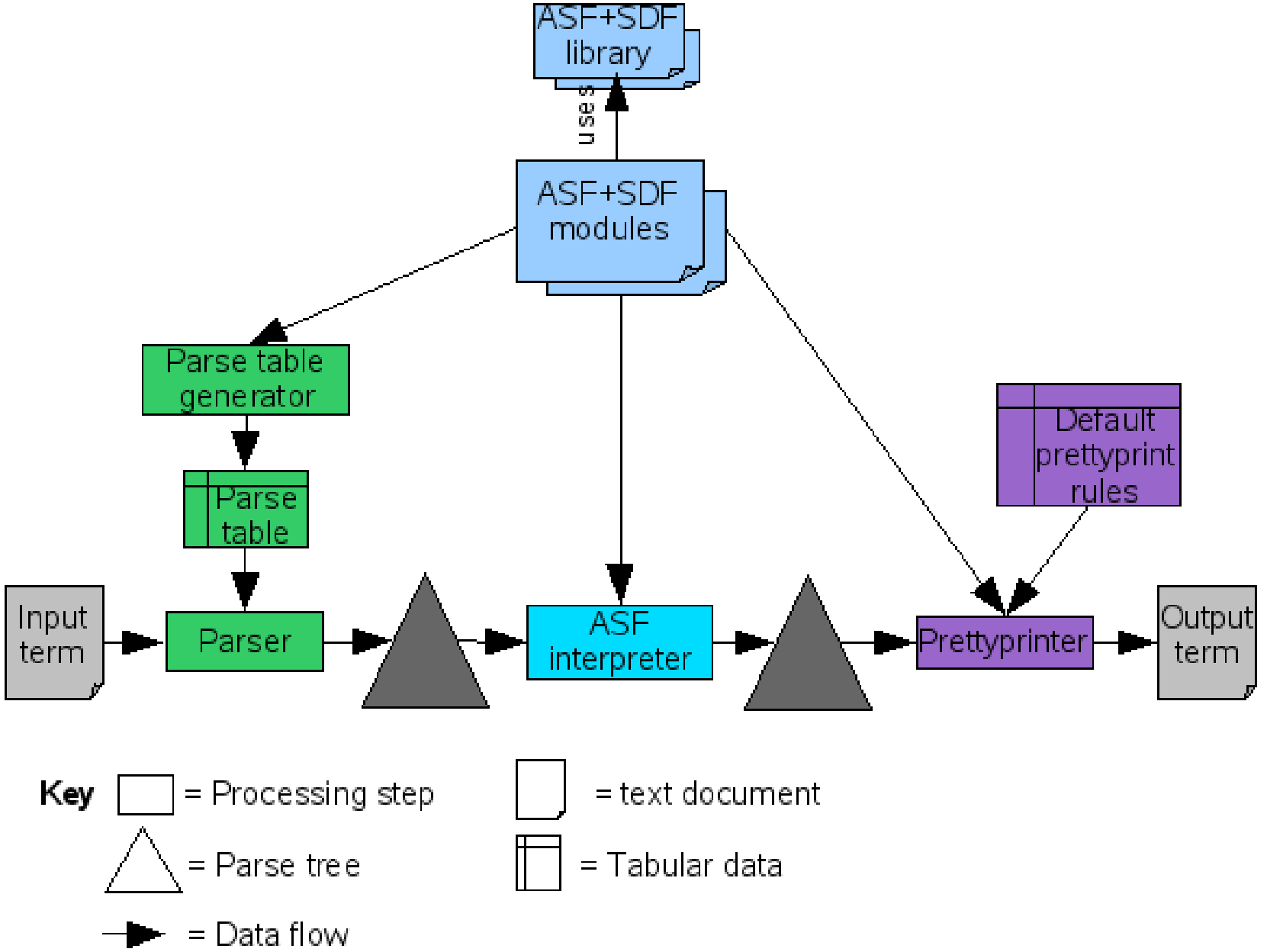
Three Cases of Tool Development

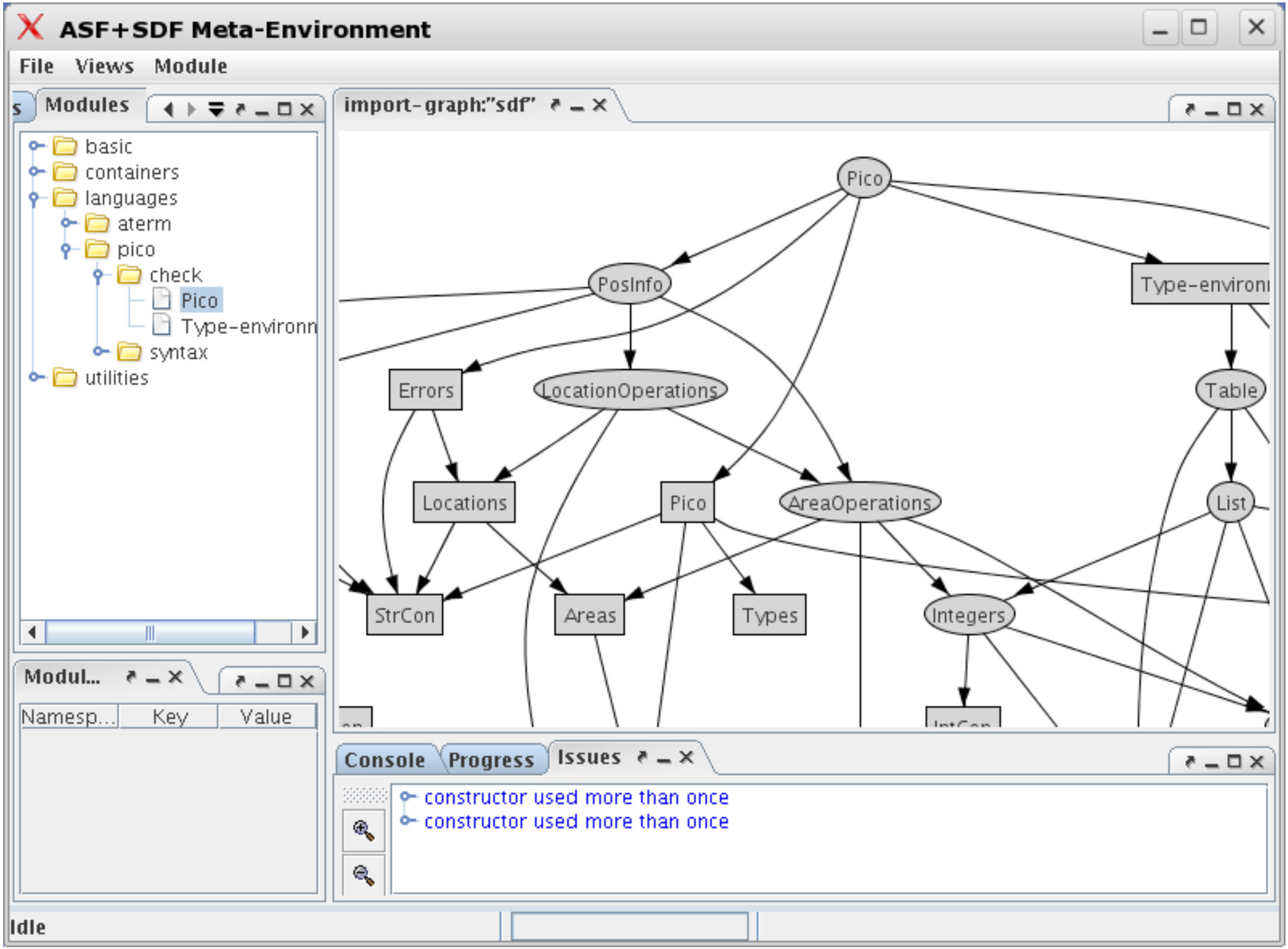
- ASF+SDF Meta-Environment, see www.meta-environment.org
- ToolBus, see www.meta-environment.org
- Rascal, see www.rascalmpl.org

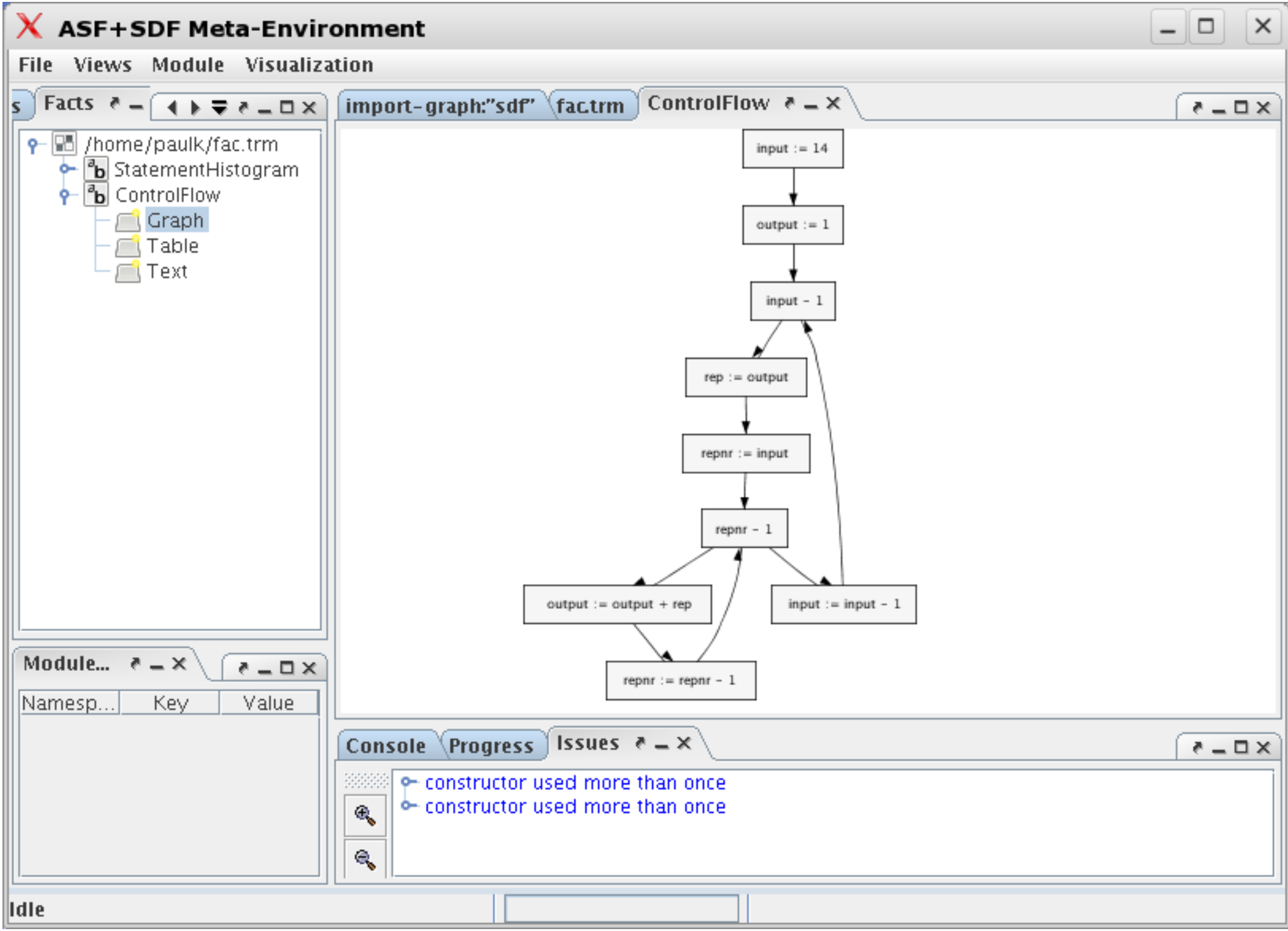
ASF+SDF Meta-Environment

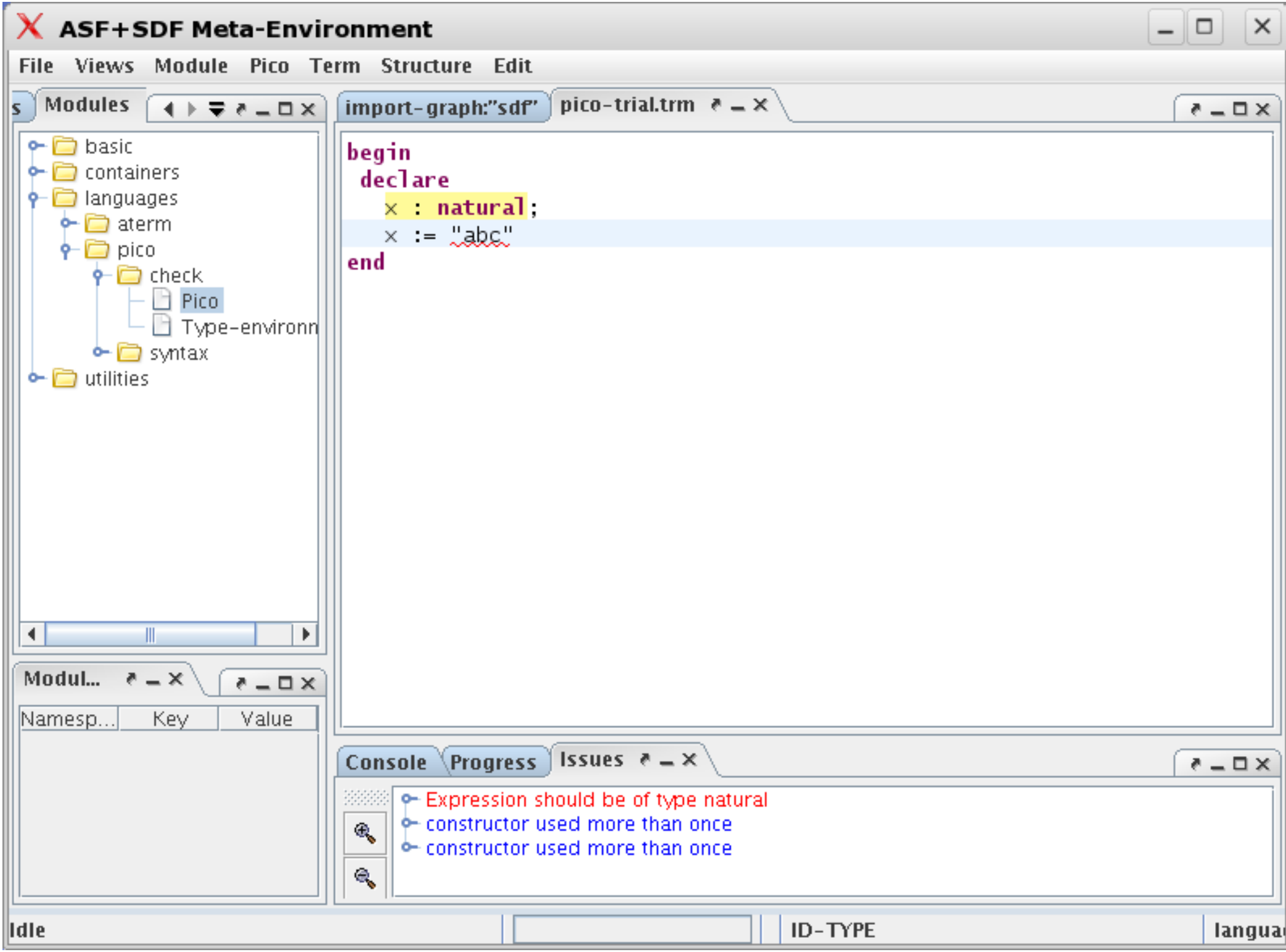
- System for interactive development of algebraic specifications
  - software analysis and transformation
  - DSL implementation
- Size ~250 Kloc, developed over more than 15 years by many different people
- Many shifting technologies:
  - Lisp -> C -> Java
  - User-interface toolkits

ASF+SDF Meta-Environment




ASF+SDF Meta-Environment
File
Views
Module
Modules
basic
containers
languages
aterm
pico
check
Pico
Type-environn
syntax
utilities
Modul...
Namesp...
Key
Value
import-graph:"sdf"
Pico
PosInfo
Type-environ
Errors
LocationOperations
Table
Locations
Pico
AreaOperations
List
StrCon
Areas
Types
Integers
Console
Progress
Issues
constructor used more than once
constructor used more than once
Idle


ASF+SDF Meta-Environment
File Views Module Visualization
Facts
/home/paulk/fac.trm
StatementHistogram
ControlFlow
Graph
Table
Text
import-graph:"sdf"
fac.trm
ControlFlow
input := 14
output := 1
input - 1
rep := output
repnr := input
repnr - 1
output := output + rep
input := input - 1
repnr := repnr - 1
Module...
Namesp...
Key
Value
Console
Progress
Issues
constructor used more than once
constructor used more than once
Idle


ASF+SDF Meta-Environment
File Views Module Pico Term Structure Edit
Modules
basic
containers
languages
aterm
pico
check
Pico
Type-environn
syntax
utilities
import-graph:"sdf"
pico-trial.trm
begin
declare
x : natural;
x := "abc"
end
Modul...
Namesp...
Key
Value
Console
Progress
Issues
Expression should be of type natural
constructor used more than once
constructor used more than once
Idle
ID-TYPE

ASF+SDF Meta-Environment

- Used in many research projects word-wide for
  - Compilation, language translation, refactoring
  - DSL implementation
  - Studies in programming language semantics
  - Term rewriting
- Used in industry for
  - COBOL migration, source code analysis, ...
  - DSL for financial products
  - DSL for business modelling

ASF+SDF Meta-Environment

- Researchers are interested in problems and general solutions, but not in completing a specific software project
- Writing papers conflicts with writing software
- PhD students want to write a thesis, not *maintainable* software
- Choice of programming language is crucial (LeLisp)
- Before common IDEs => program for obsolence

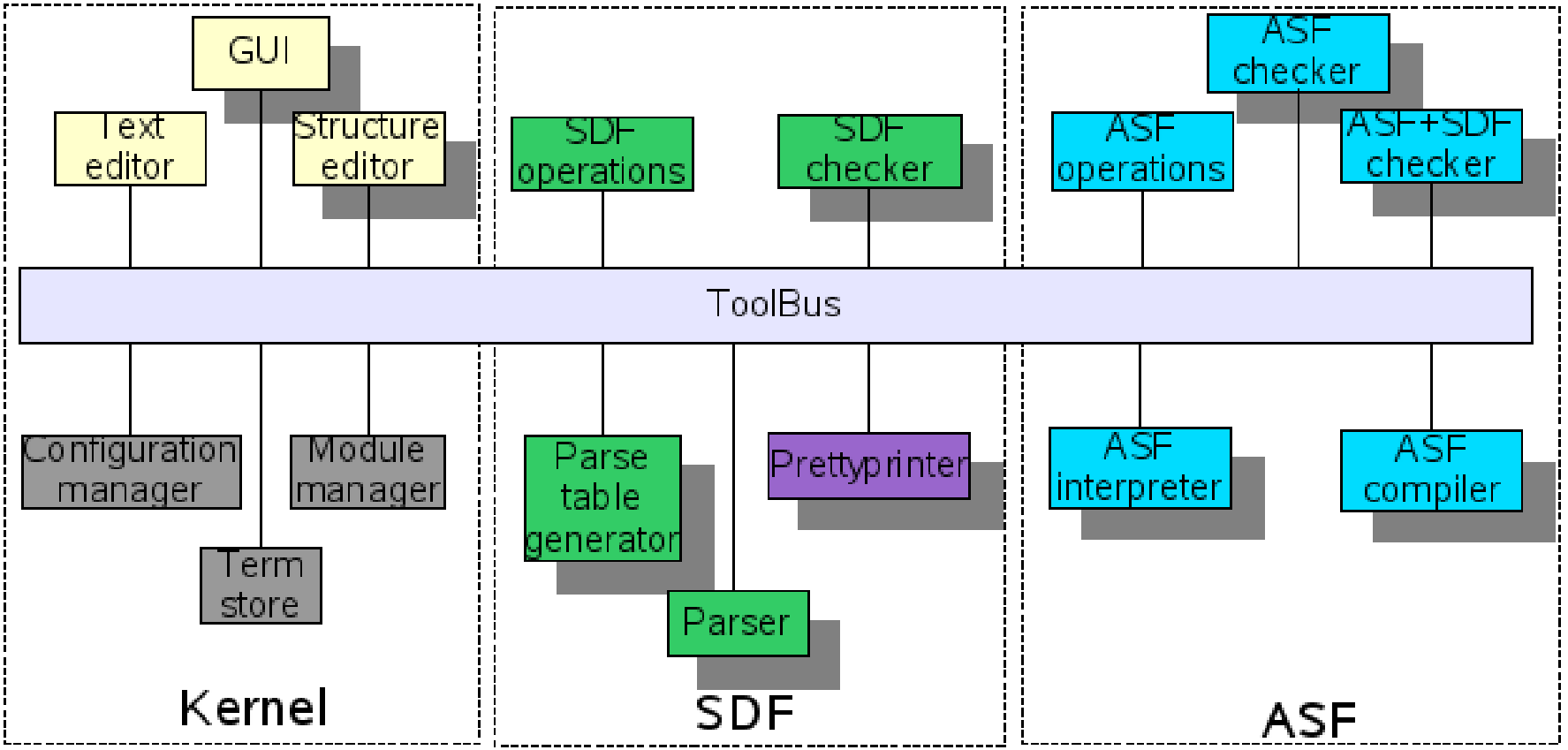
ASF+SDF Meta-Environment

- Software engineering ≠ programming
- Overreaction to problems encountered:
  - Lack of modularity/reusability => refactor into too many small units
- Configuration management is very hard to get right
  - Autoconf, automake, ...
- Continuous evolution of the software landscape creates lot of overhead

ToolBus

- A system for the coordination of heterogeneous, distributed, components
- A coordination script (based on process algebra) controls the execution of programs written in different languages runnng on different machines
- Size: ~15 Kloc
- Developed/improved in ~ 2 years
- Used in several projects over many years

ToolBus in Action



ToolBus
Lessons Learned

- Elegant system that has resulted in several well-cited publications
- Example of turning a problem during software development into a research problem
- The solution is probably more general than what we needed at the time
- The maintenance of ToolBus scripts became a problem on its own

Rascal

- A meta-programming language for
  - Meta-programming (yes, what else :-)
  - Software analysis and transformation
  - Design and implementation of domain-specific languages
- Design 2007, implementation started end 2008
- Size: ~60-70 Kloc
- Mostly Java, integrated in Eclipse

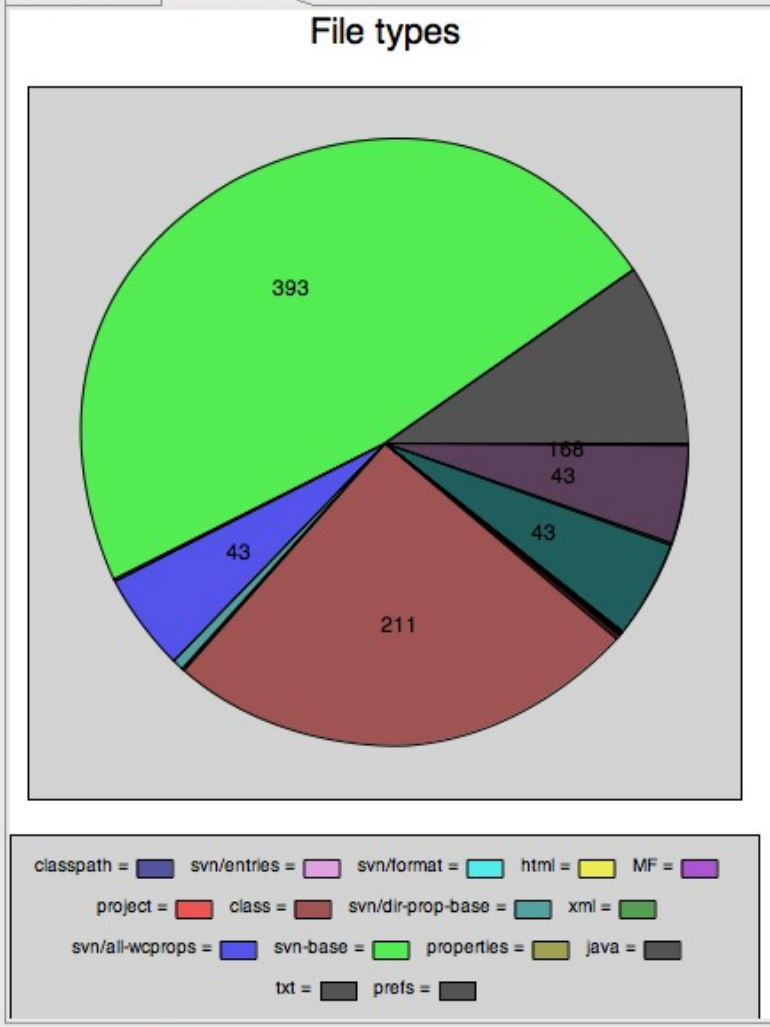
Rascal – MyRascal/src/FileTypes.rsc – Eclipse Platform

Debug

Package Explorer

- jspwiki
- MyRascal
  - eclipse
  - src
    - Users
      - Exp.rsc
      - FileTypes.rsc
  - std
    - demo
    - experiments
      - DDA
      - GrammarTools
      - Lexicals
      - ModelTransformatic
      - Parsing
      - PrettyPrinting
      - Processing
      - RascalTypechecker
      - StandardLibrary
      - viz
      - VL
        - Chart.rsc
        - Examples.rsc
        - Examples1.rsc
        - Examples2.rsc
        - Examples3.rsc
        - Examples4.rsc
        - VLCore.rsc
        - VLRender.rsc
      - Dashti.rsc
      - DateVars.rsc
      - DocGenFDL.rsc
      - StringTemplate.rsc
      - Subversion.rsc
      - TestStrategy.rsc
      - Visitors.rsc

Exp.rsc | VL



FileTypes.rsc

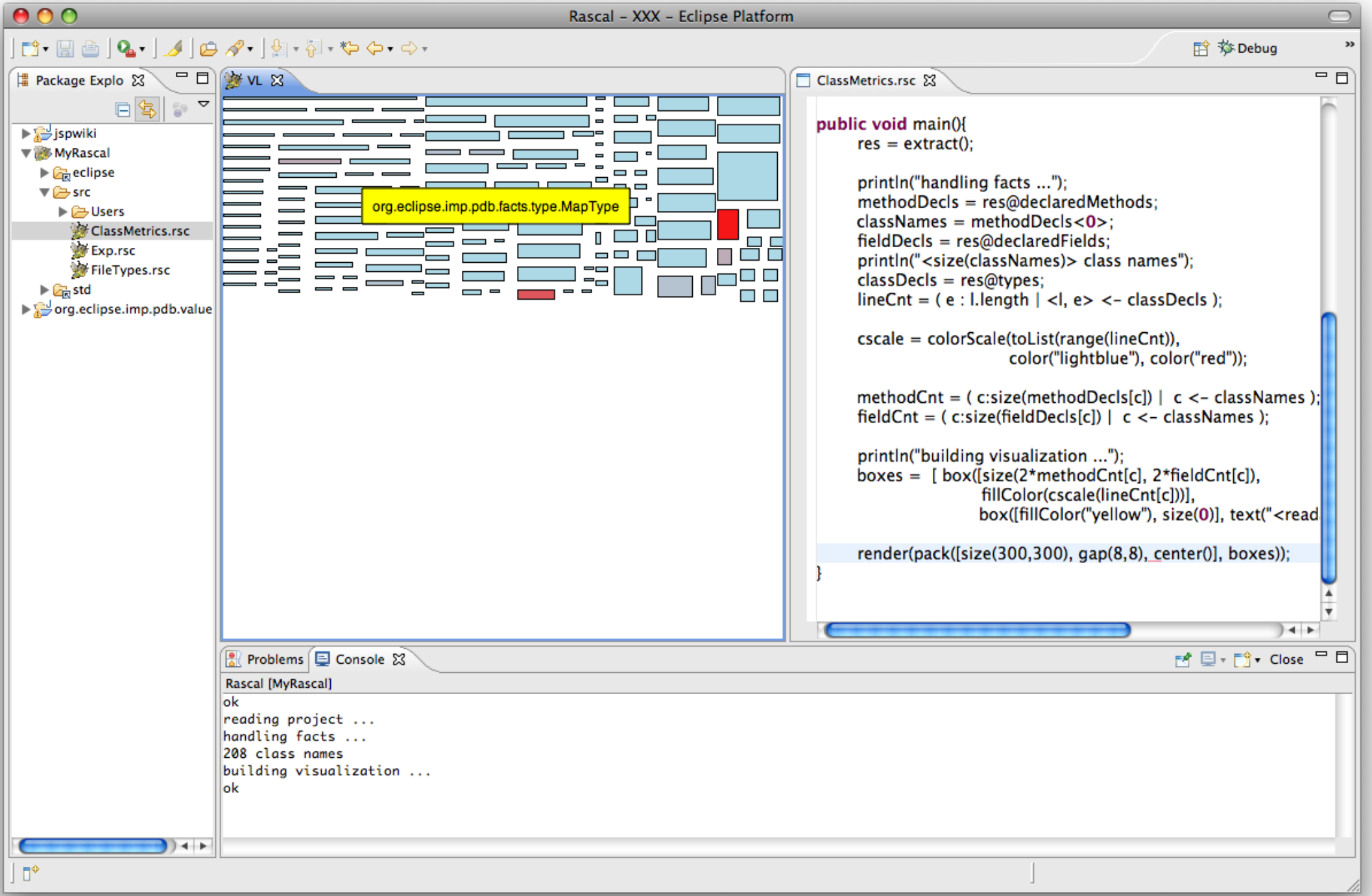
```
module FileTypes

import experiments::VL::VLCore;
import experiments::VL::Chart;
import viz::VLRender;
import JDT;
import Java;
import Resources;
import IO;
import Set;
import Map;
import Relation;
import Graph;

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

Resource extract(){
	println("reading project ...");
	return extractProject(project);
}

public void main(){
	res = extract();
	extCnt = ();
	visit(res){ case file(loc l):
			if(l.extension != "") extCnt[l.extension]?0 += 1;
		}

	render(pieChart("File types", extCnt));
}
```

Problems | Console

Close

Rascal [MyRascal]

```
ok
reading project ...
```


Rascal – XXX – Eclipse Platform
Debug
Package Explo
jspwiki
MyRascal
eclipse
src
Users
ClassMetrics.rsc
Exp.rsc
FileTypes.rsc
std
org.eclipse.imp.pdb.value
VL
org.eclipse.imp.pdb.facts.type.MapType
ClassMetrics.rsc
public void main(){
res = extract();
println("handling facts ...");
methodDecls = res@declaredMethods;
classNames = methodDecls<0>;
fieldDecls = res@declaredFields;
println("<size(classNames)> class names");
classDecls = res@types;
lineCnt = (e : l.length | <l, e> <- classDecls);
cscale = colorScale(toList(range(lineCnt)),
color("lightblue"), color("red"));
methodCnt = (c:size(methodDecls[c]) | c <- classNames);
fieldCnt = (c:size(fieldDecls[c]) | c <- classNames);
println("building visualization ...");
boxes = [box([size(2*methodCnt[c], 2*fieldCnt[c]),
fillColor(cscale(lineCnt[c]))],
box([fillColor("yellow"), size(0)], text("<read
render(pack([size(300,300), gap(8,8), center()], boxes));
}
Problems
Console
Close
Rascal [MyRascal]
ok
reading project ...
handling facts ...
208 class names
building visualization ...
ok

Applications So Far

- Java analysis, verification and refactoring
- Rascal type checking
- Parser generation
- Mining of version repositories
- DSL for forensics
- DSL for computer aided instruction: RascalTutor
- DSL for computational auditing (just started)

Rascal
Lesson Learned, so far

- Over-reliance on certain design patterns:
  - Visitor pattern gives problems with understandability and performance
- Underestimation of effort
- Excellent test suite
- Modular concepts and design
- Performance, performance, performance

Some More Lessons

- Use version management (of course)
- Use refactoring to continuously improve your code (of course)
- Use test-driven design (of course)
- **Manage your software project!**

Tool Development in Academia

Tool Development in Academia
Conflicts of Interest

How to keep everybody happy?

Tool Development in Academia
Conflicts of Interest

- Time: tools versus papers
  - Long term continuity versus short term results
  - Usability versus new functionality
- Brownie points:
  - Individual versus group
  - Short term versus long term
  - Internal versus external
- Conflict resolution
  - Joint software development ► joint papers

Tool Development in Academia
Programming versus Management

Tool Development in Academia
Programming versus Management

- Researchers hate management
- Research ≠ Software Engineering Project
- Software Engineering ≠ Programming
- There are many roles: manager, sales, architect, programmers, testers, documentation writers, ...
- In a research team few people have to play all these roles.
- A small team with large obligations ...

Tool Development in Academia
Programming versus Management

- Be aware of your own roles and of the roles of others
- Rethink your coding efficiency:
  - Use code generators where possible
- Re-use tools, libraries, algorithms whenever you can (instead of reinventing them)

Tool Development in Academia
Requirements Engineering and Design

- Usually overlooked
- Requirements engineering/design is also a role
- No real stakeholders (yet).
- Requirements can dramatically change during the project.
- Feature creep due to desire to invent and publish

Tool Development in Academia
Who does the Programming?

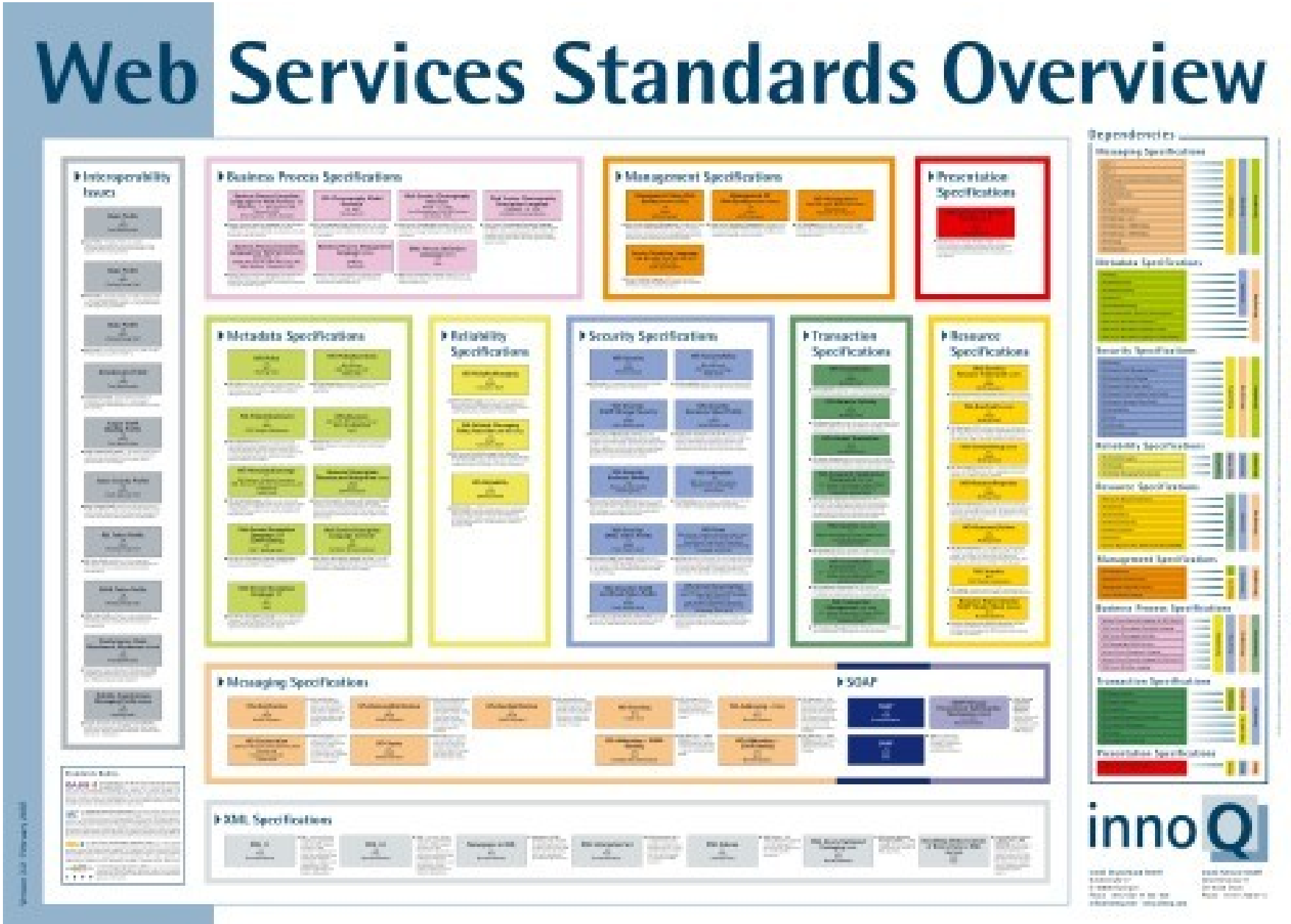
- PhD students
  - Distracts from writing research papers
  - Lack of continuity
  - Increases value on the job market
- Senior staff
  - Distracts from writing research papers
  - Good continuity
  - Increases awareness of real problems
- Scientific programmers

The Career Perspective of a Scientific Programmer

The Career Perspective of a Scientific Programmer

- Just out of college:
  - The coolest job: programming without the hassles of writing papers
- After 10 years:
  - Younger programmers start to appear with knowledge that you miss; time for re-education
- After 20 years:
  - Either you are at the same experience level as researchers and are co-author of papers, or
  - You ended up in a dead alley ....

Tool Development in Academia
Using Standards



Tool Development in Academia
Using Standards

- A good idea, but depends on your objectives
- Which standard?
- Standards are bulky and may lead to a lot of extra implementation effort
- No standards for many topics in our domain:
  - Meta-data about programs
    - Structure (packages, module, classes, methods, ...)
    - Versioning, bug reports
  - Metrics

Tool Development in Academia
Writing Documentation

- A tool is useless without documentation
- Scientific paper ≠ documentation
- Writing for paper documentation versus writing for online documentation:
  - No sequential story to tell
  - Independent knowledge units that can be read in arbitrary order
  - Judging the maturity of your audience

Experiment: RascalTutor

- A *concept* is a learning unit with a fixed number of sections
- Concepts are organized in a hierarchy:
  - Rascal/Expressions/Values/Set/Union
- Browsing/searching/linking
- Wiki style editing
- Written in Rascal

Tool Development in Academia
Tools and Education

- Academic tools can be used in academic courses as case study for
  - Architecture documentation
  - Code quality
  - Test quality
  - Usability experiments
- This helps to improve tool quality

Do's and Don'ts?

Don'ts have already been

suggested in the presentation

Do

Carefully reflect on
your tool strategy
before embarking on
any
software engineering research
project