Modularity

Jurgen Vinju

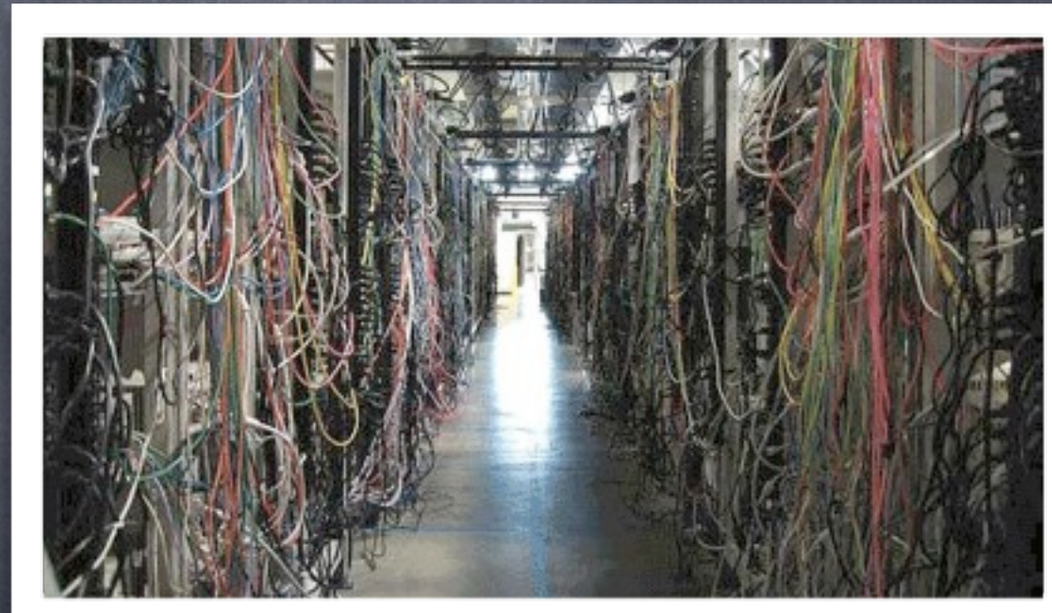
January 13th 2013

Plan

- Motivation
- Conceptual exploration
- A story of three designs of the same system
- Discussion

Software Engineering

- What's the big issue anyway?
- Programming!
- Design?
- Collaboration?
- Goals:
  - Efficiency
  - Quality
  - Continuity

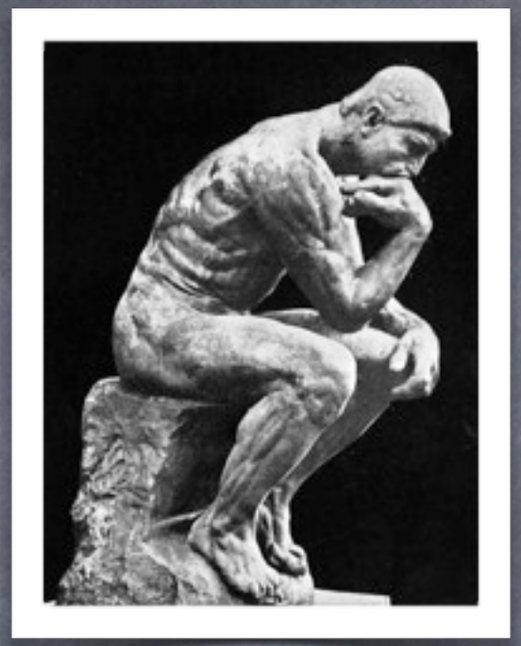


The source code of "ls"

3894 lines

367 ifs

174 cases



Software Engineering is a immensely complex, interesting and above all multi-dimensional domain

(once you open your mind to **all** of it)

Name three solutions to the software engineering puzzle

There is but one

- **It's called "modularity"!**
  - High-level programming languages
  - Abstraction
  - Information hiding
  - Reuse
  - Separation of concerns
  - Aspects
  - Functions
  - Objects
  - Components
  - …

Modularity is para-paradigmatic, ubiquitous, fundamental, the bomb, … [please add your own superlative qualifications here]

What is modularity?

- What is modularity?
- What is modular?
- What is not modular?
- What is a module?
- What is not a module?
- What is a good module?
- What is a bad module?

Examples of modules, or not?

- Java class
- Function
- Jar file
- C header file
- HTML file
- Haskell function

- Dll
- Object
- Eclipse project
- GNU project
- Prolog clause
- Haskell module
- BNF grammar
- ANTLR grammar

Build-time

Deployment-time

Release-time

Run-time

Test-time

Original Modularity

- David Parnas "On the criteria being used in decomposing systems into modules" (1972)
- Gauthier & Pont "Designing Systems Programs" (1970)
- Motivations
  - Portability
  - Reuse
  - Scaling to more programmers
- Key concept: **information hiding**

A module is...

a weird kind of box

- An encapsulation of software artifacts
- With certain properties
  - separate
  - independent
  - (re)usable
  - composable (ergo, dependent)
- At a certain stage in the life-cycle, or more, or all (build, release, deploy, test, run)

like lego, or yet...maybe not quite like lego.

Home grown modularity

- "Module Algebra", Bergstra, Heering & Klint
- Modules for algebras
- Algebra for modules
- BTW, algebra in itself is about orthogonality, compositionality

Modules are separate

- Function
- Method
- Class
- Clause
- Dll
- Jar

they have
identity (a name)

Modules are independent

they hide things

- The body of function can change
- The private parts of a class can change
- The internals of a library can change

Modules are usable

they expose an interface

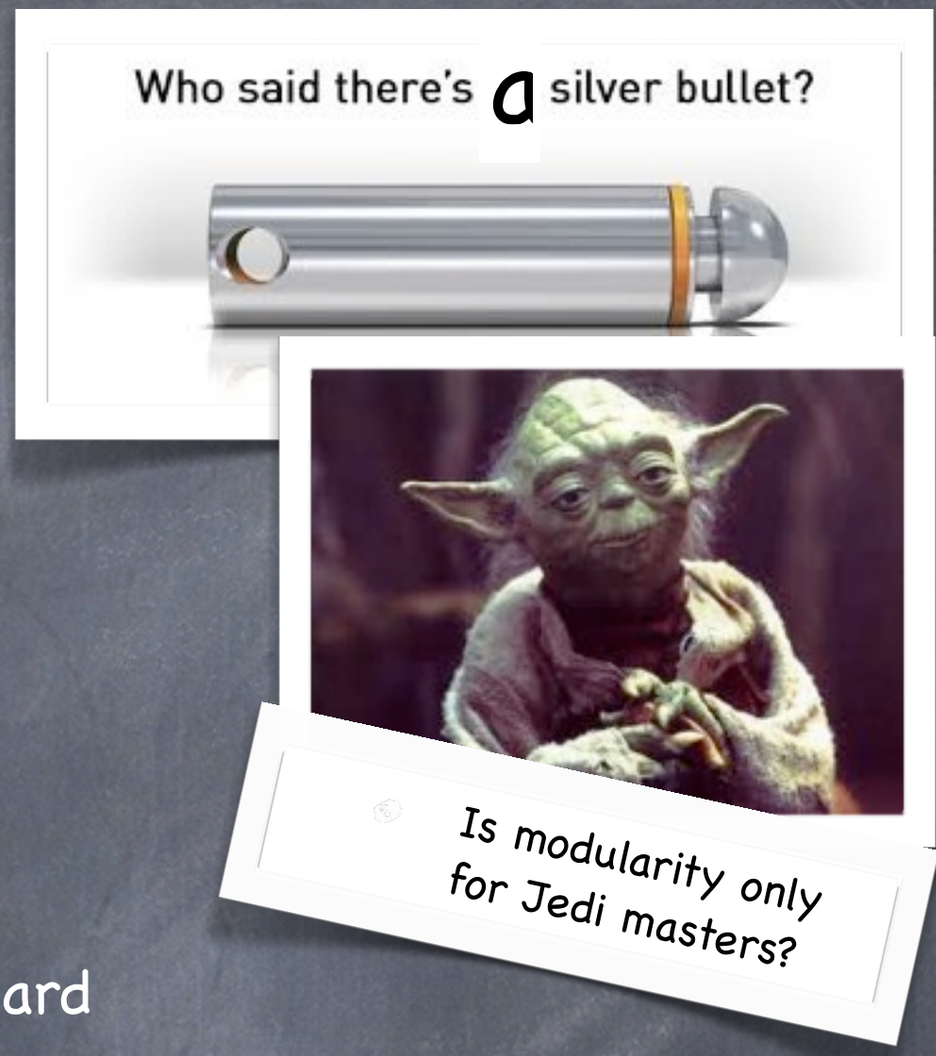
- Function calling
- Class importing, inheritance, referencing
- Clause application
- Dll loading
- Jar loading
- ...

Modules are composable

closed under one or more composition operations

- Functions can call other functions
- Classes can use, inherit from other classes
- Jar files can be composed of other jar files
- Yet there are so many software artifacts that are separate, usable, **but not easily composable**...
- Example: frameworks are not modules themselves

The dark side



- Composition (a.k.a. "integration") is hard!
- Making actually composable modules is hard
- Making actually independent modules is hard
- Finding the right module in a large collection is hard
- Understanding an existing module is hard
- Testing a module in isolation is hard
- Predicting the quality of a composition is hard
- **Does modularity solve a problem, or just shift it? or does it even make life harder than it used to be?**

Clemens Szyperski

"Maximizing reuse, minimizes use"

"It's a DLL Hell"

"Modules schmodules"

"The standard library uses way to much memory"

"All this added indirection is slow and above all confusing"

"When abstraction fails..."

Andreas Zeller

Think about the trade-off(s) that modularity is involved in

Coffee!

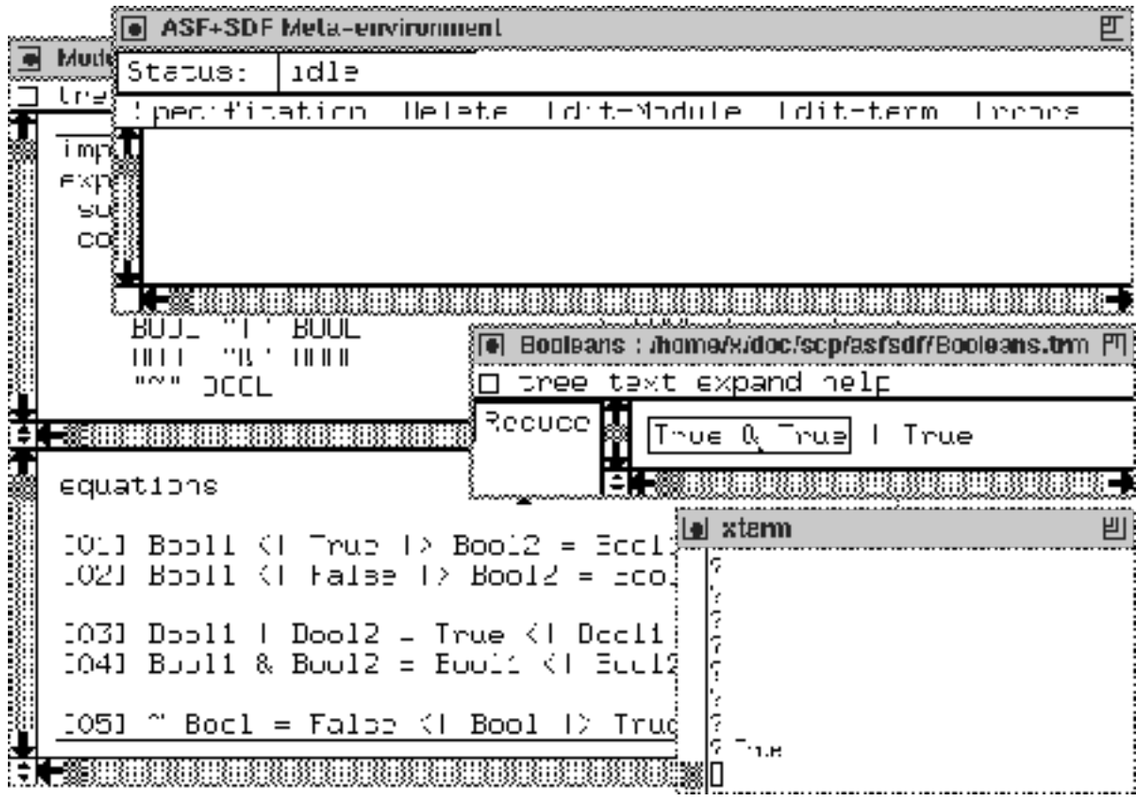
Recap

- Module = identity, encapsulation, hiding, composition
- Modules at build-time, run-time, deployment-time, ...
- One thing I skipped about modules... what?
- Modules are the only solution
- Modules are hard and introduce cost
- Now, a story about 3 systems that do the same thing.

Part 2: Three Modular Designs

1984 Generating Interactive Programming Environments

- Take a language definition
- Generate a full blown IDE
- Which includes everything a programmer may need to program in this language (domain specific, general purpose, whtvr)
- Solution space:
  - Generating components
  - Generic (parametric) components
  - Grammars and algebra and term rewriting


ASF+SDF Meta-environment
Status: idle
Specification Delete Edit-Module Edit-term Traces
Booleans : /home/x/doc/scp/asfsdf/Booleans.trm
tree text expand help
Reduce
True & True | True
equations
xterm

Version 1: Sparc & Lisp

- Centaur LeLisp: great GUI programming (for those days)
- SUN Sparc only, 16Mhz, 1Mb (perhaps even 4!)
- Lisp is the beginning and end of programming
- Lisp has macros
- Lisp has functions
- Lisp has side-effects
- The Lisp language is simple and elegant
- Yet, Lisp programs do not necessarily inherit those qualifications...

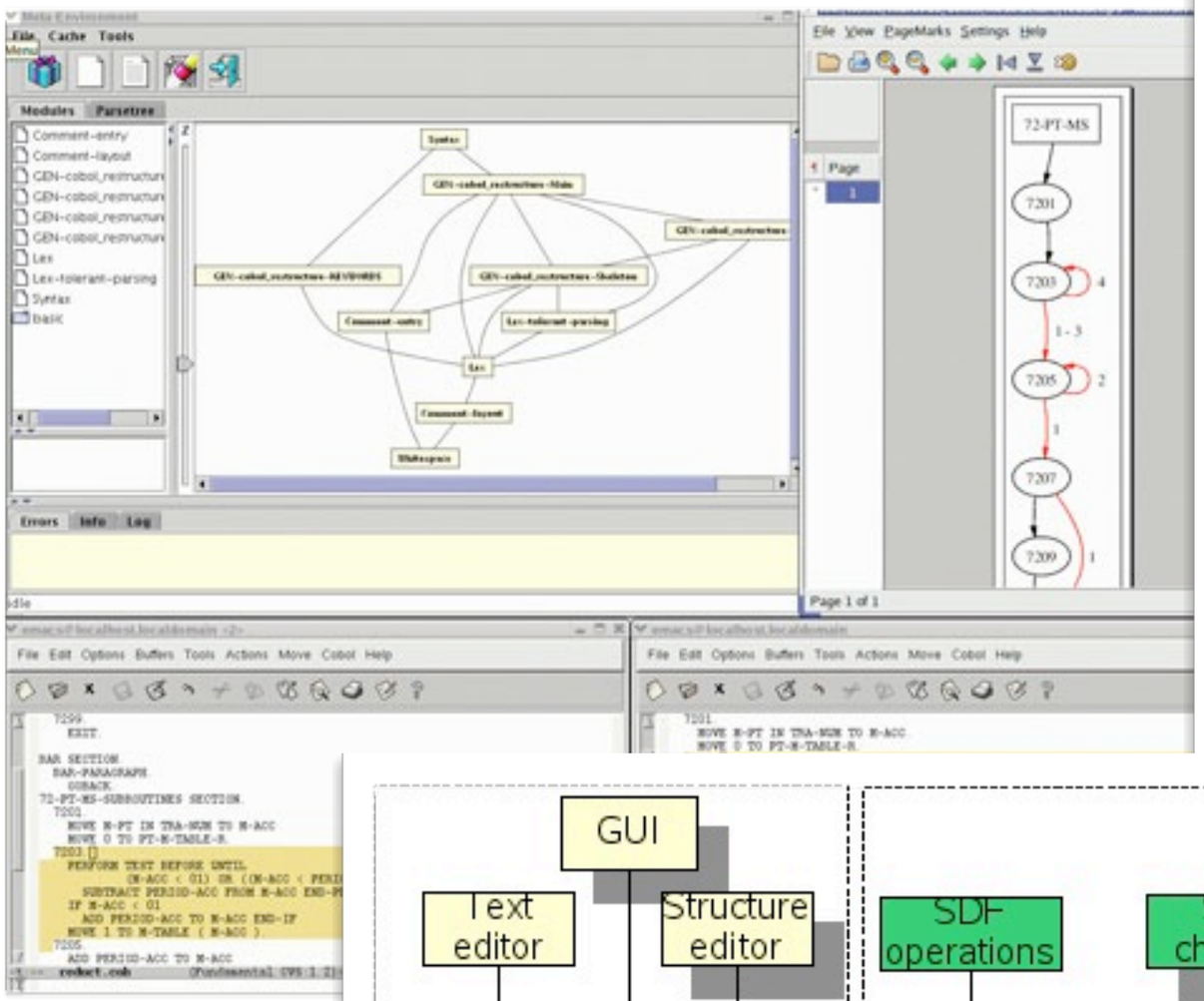
Version 1: result

- A bunch of PhD theses
- A usable system
- > 100.000 LOC
- A big ball of inter-dependent, incremental, state-full, highly optimized, LeLisp programs
- Incomprehensible
- Not portable
- Really fast
- Not modularly deployable
- The end of a road

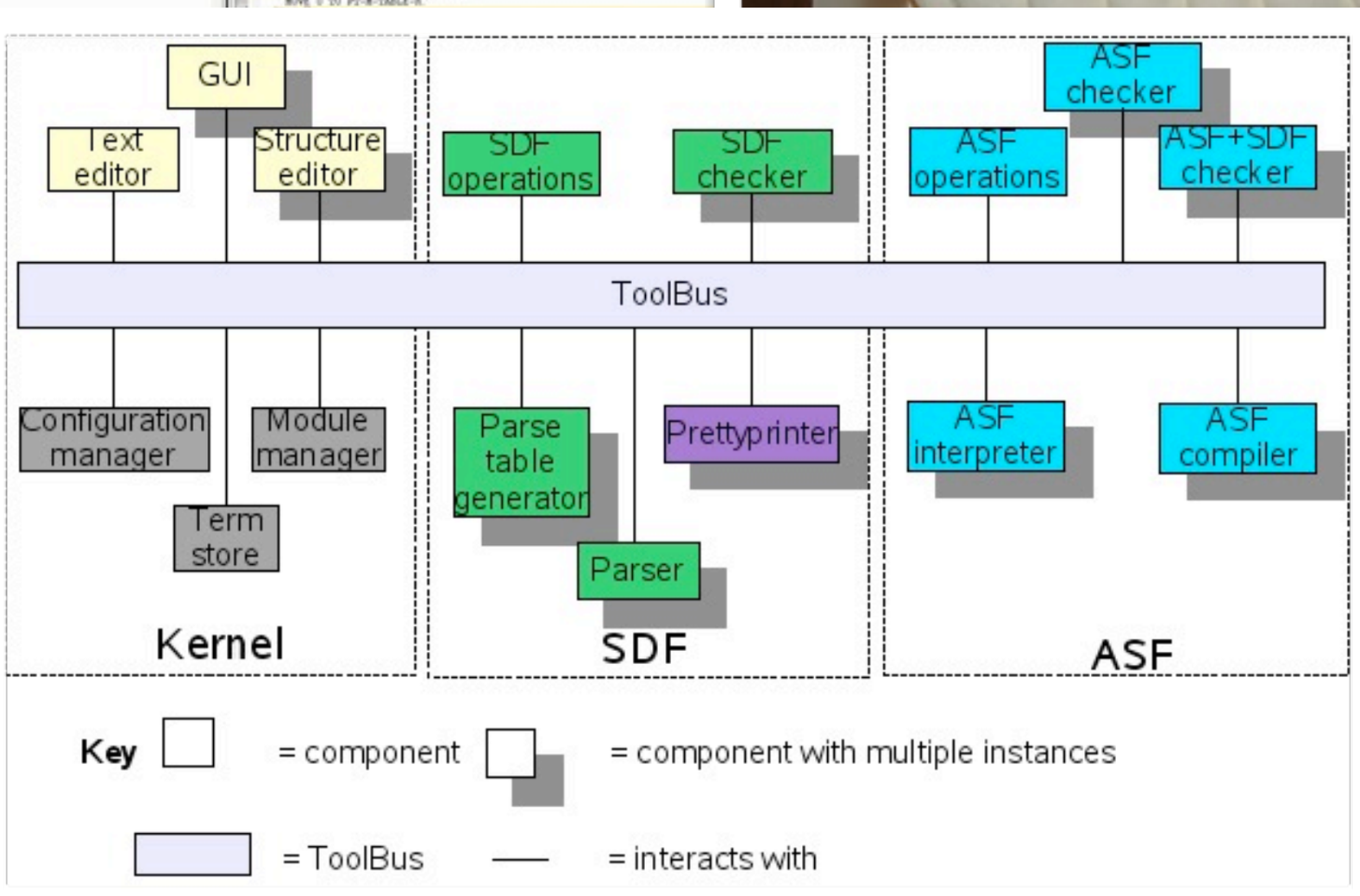
modularity everywhere though... but not of the good kind

1999 Version 2: Generic, Language Independent, Service-Oriented

- Separation of concerns
  - coordination from computation
  - programming language independent
    - small tools connected to a generic bus
- C, Java, TCL, Perl, Python, ASF+SDF, you name it
- Release of parts (sum of the parts more than the whole)
- Bootstrapped on previous system


Meta Environment
File Cache Tools
Modules
Parsetree
Errors
Info
Log
File View PageMarks Settings Help
Page
Page 1 of 1
72-PT-MS
7201
7203
7205
7207
7209
File Edit Options Buffers Tools Actions Move Cobol Help
7299
EXIT.
BAR SECTION.
BAR-PARAGRAPH.
GOBACK.
72-PT-MS-SUBROUTINES SECTION.
7201.
MOVE M-PT IN TNA-NUM TO M-ACC
MOVE 0 TO PT-M-TABLE-R.
7203.
PERFORM TEST BEFORE UNTIL
IF M-ACC < 01
ADD PERIOD-ACC TO M-ACC END-IF
MOVE 1 TO M-TABLE (M-ACC)
7205.
ADD PERIOD-ACC TO M-ACC
reduct.cob


Meta-Environment 2.0
Release Candidate 2


GUI
Text editor
Structure editor
SDF operations
SDF checker
ASF checker
ASF operations
ASF+SDF checker
ToolBus
Configuration manager
Module manager
Term store
Parse table generator
Prettyprinter
Parser
ASF interpreter
ASF compiler
Kernel
SDF
ASF
Key
= component
= component with multiple instances
= ToolBus
= interacts with

Version 2: modularity everywhere

- Tools connected to bus: build-time and run-time modules
- Processes: composable coordination scripts
- Packages: GNU build, test and deployment interfaces (automake, autoconf)
- GUI plugins (via Java reflection and jars)
- Libraries, libraries, libraries
- Code generators for C, Java, etc..
- > 65 packages, > 150 tools, > 300.000 LOC (200.000 generated)

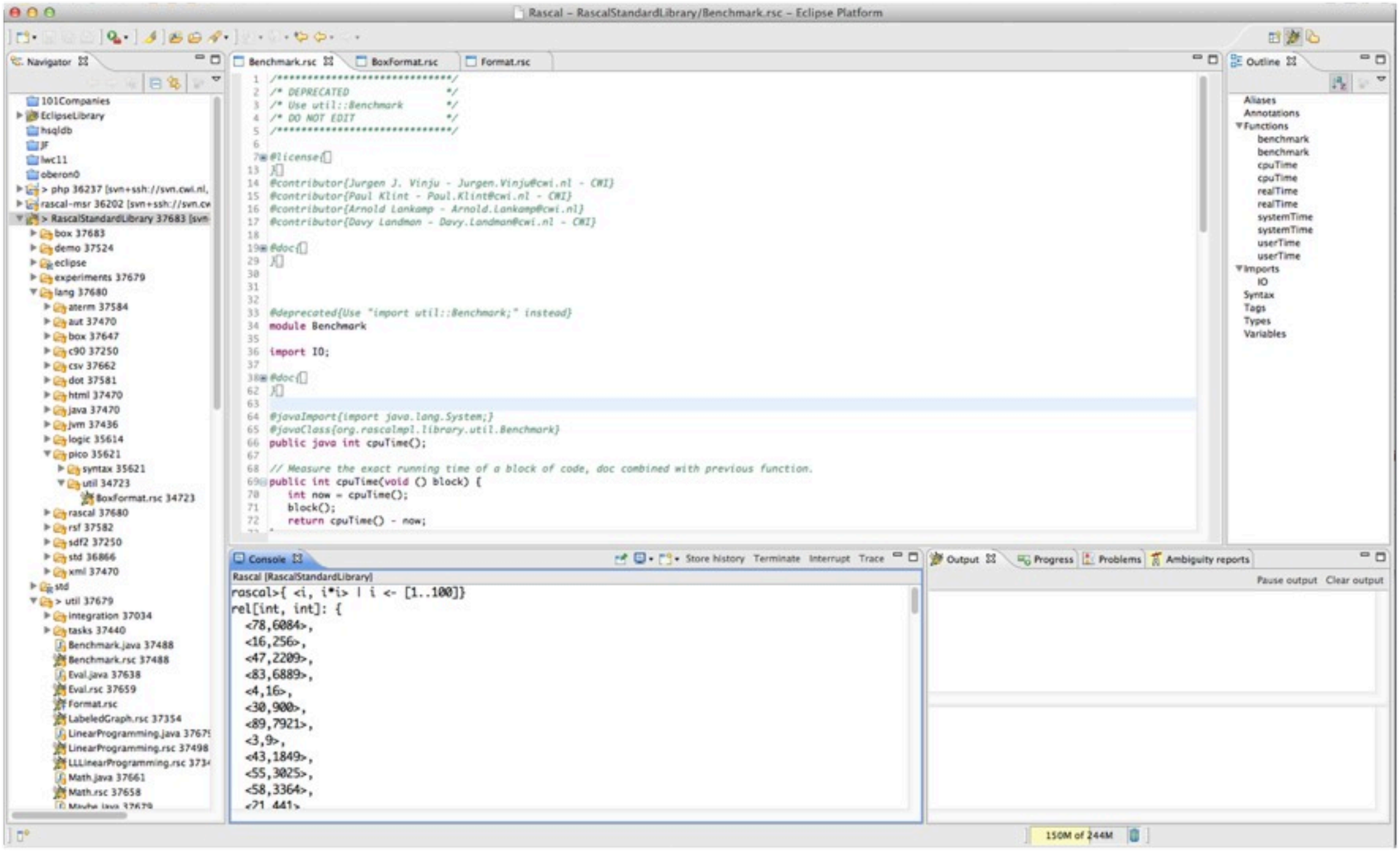
Version 2: results

- **(Re)use**!!! libraries, parser generator, rewriting engines, generic IDE,
- A usable system, no wait: a family of usable systems
- Overhead. M4, autoconf, automake, gcc, shell scripts, ant, you name it!
- Home grown incremental continuous integration system (sisyphus)
- Home grown source code package composition system (autobundle)
- Too much modularity for our own good
- Source code releases only (limited binary support)

2009

Version 3: back to basic

- Everything on the JVM
  - Bootstrapped on previous system
    - Then Java
    - Then Bootstrapped on itself
- Eclipse and IDE meta-tooling platform (IMP)
- Only 3 components: run-time, language, IDE


Rascal – RascalStandardLibrary/Benchmark.rsc – Eclipse Platform
Navigator
Benchmark.rsc
BoxFormat.rsc
Format.rsc
Outline
Console
Store history
Terminate
Interrupt
Trace
Rascal [RascalStandardLibrary]
rascal>{ <i, i*i> | i <- [1..100]}
rel[int, int]: {
<78,6084>,
<16,256>,
<47,2209>,
<83,6889>,
<4,16>,
<30,900>,
<89,7921>,
<3,9>,
<43,1849>,
<55,3025>,
<58,3364>,
Output
Progress
Problems
Ambiguity reports
Pause output
Clear output
150M of 244M

Version 3: results (2011)

- 100.000 LOC
- **more** features than before, **more** users, **more** uses
- Faster and simpler implementation (per feature)
- Completely **documented**
- Many automated **tests**
- **internal libraries no longer sold/exposed so much...**
- Success factors:
  - Uses reflection to decouple front-end from back-end (!)
  - Uses in-memory on-the-fly Java compilation instead of files
  - Uses simple abstract syntax classes and dynamic dispatch
  - Java JIT and GC deal well with the code we write
  - Long live Eclipse (yes really!)

Discussions?

- Modularity at different levels
- Modularity at different times
- Modularity for different purposes
- Cost/Benefit of modularity
- Styles and Standards