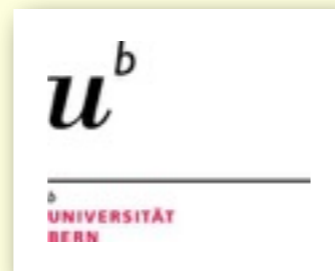


Domain-Specific Tooling

Oscar Nierstrasz
Software Composition Group
scg.unibe.ch



Roadmap



Agile Software Assessment



Agile Modeling



Agility in Moose



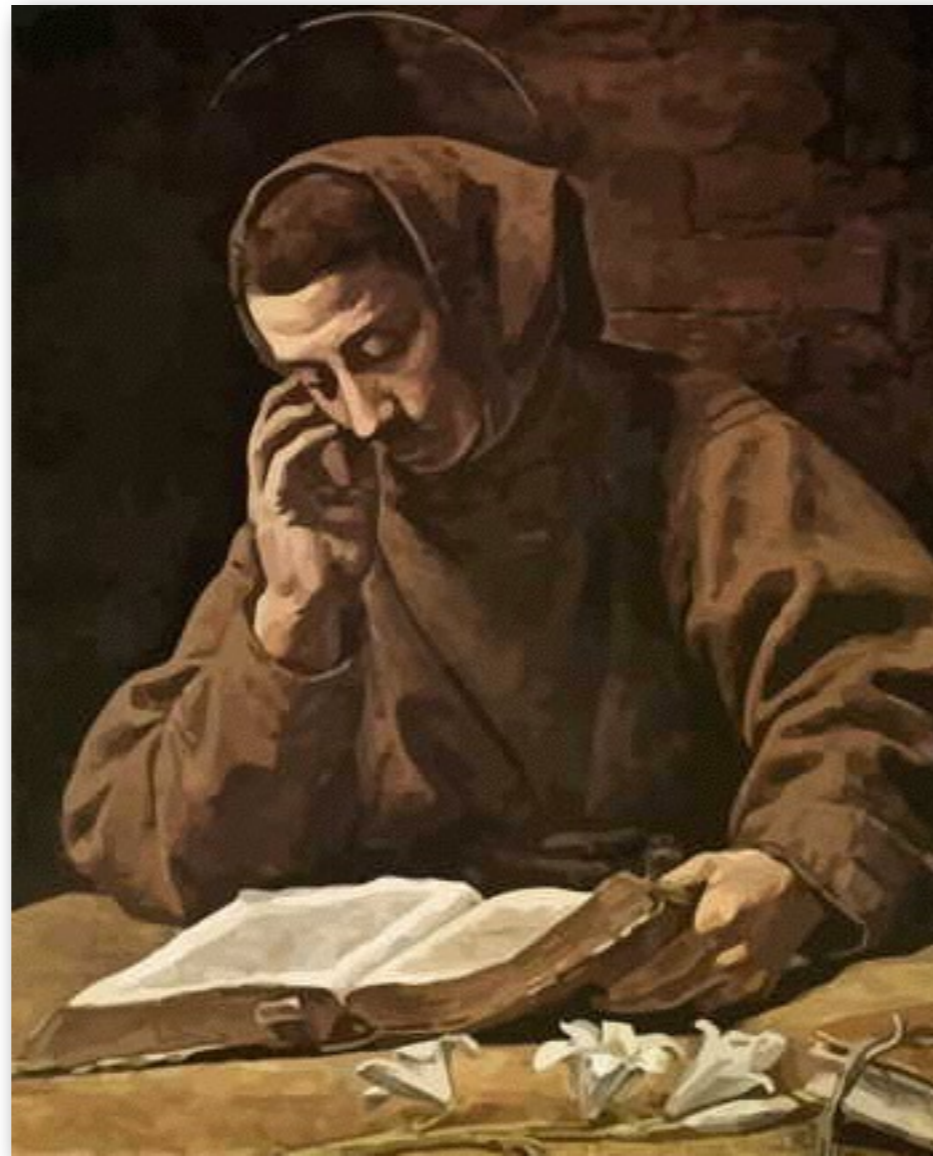
Architectural Monitoring



Moldable Tools

Agile Software Assessment



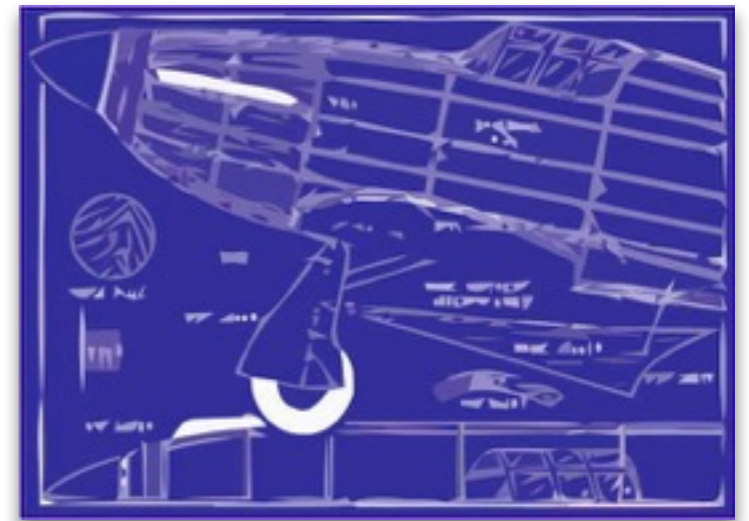


Developers spend more time reading than writing code



There is a gap between Models

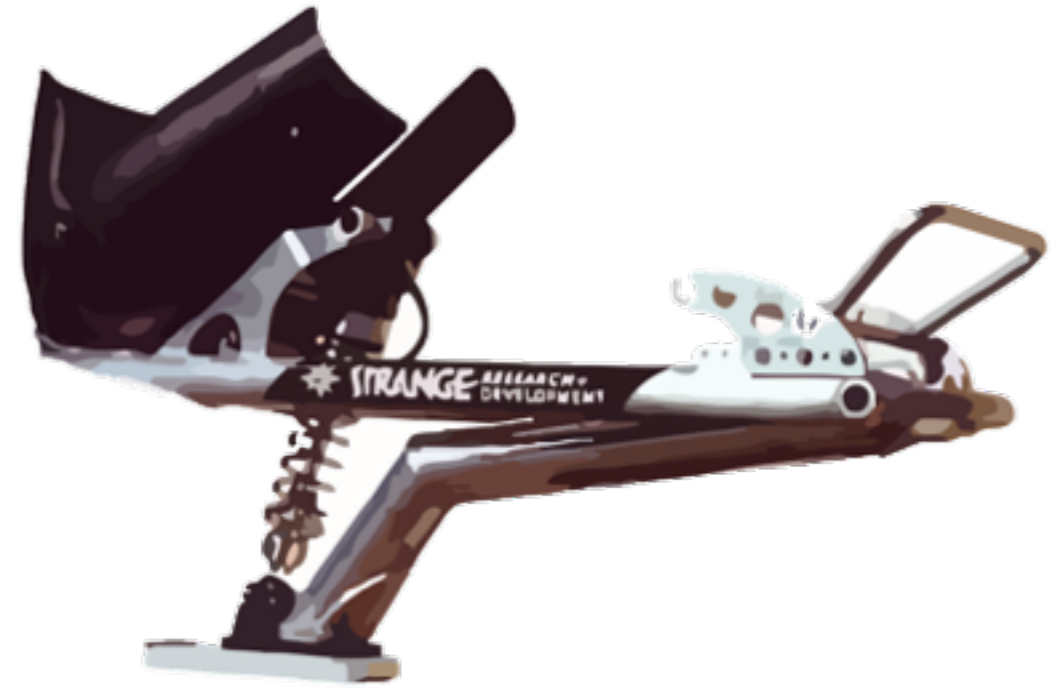
and Code



The architecture



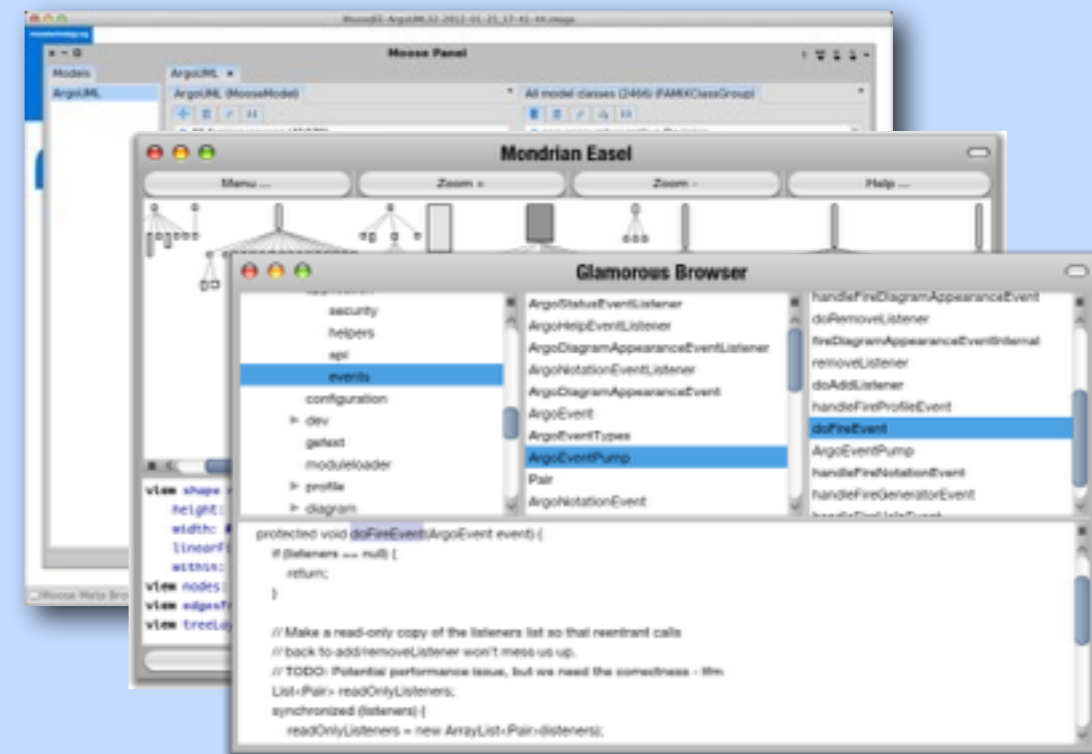
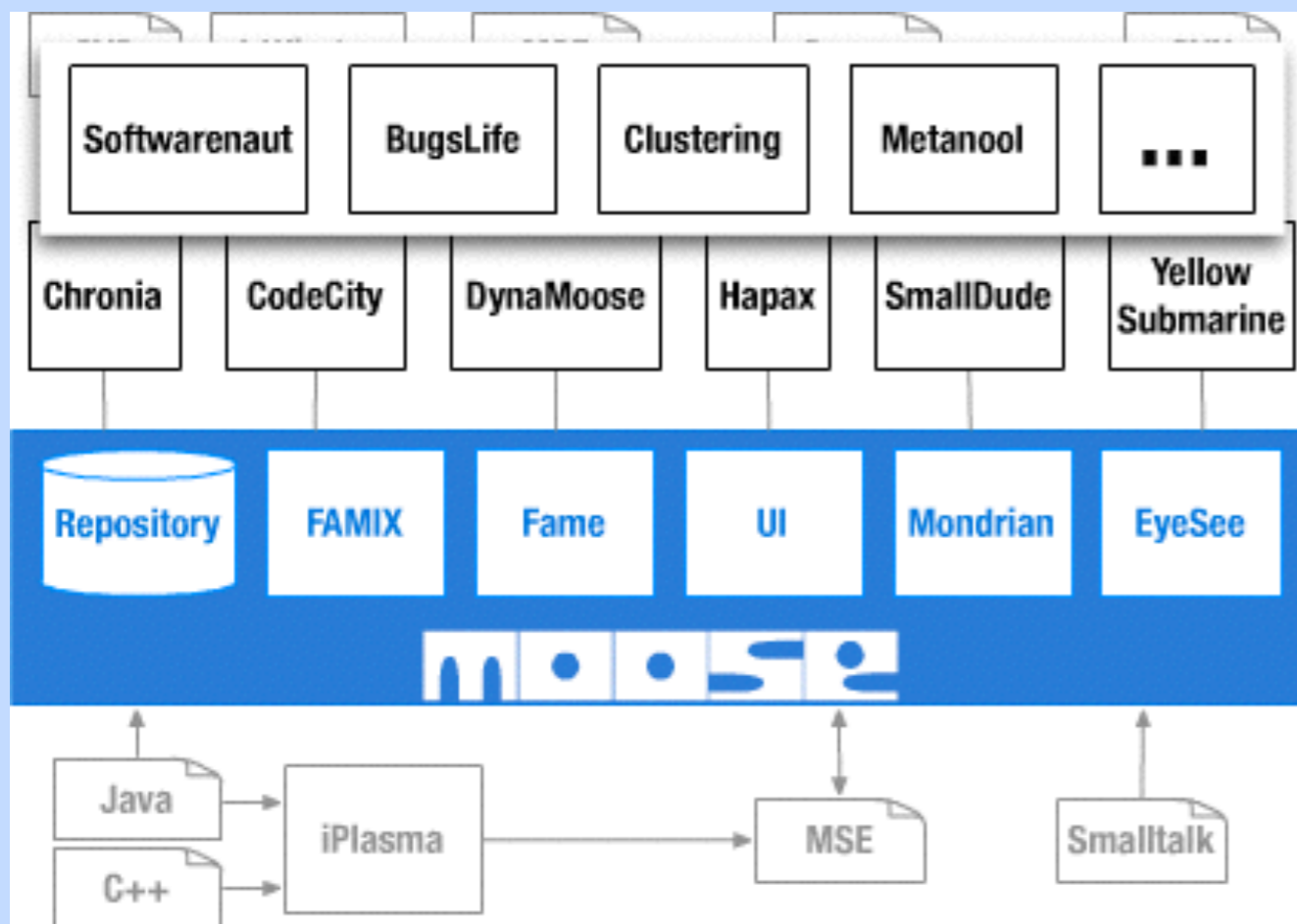
... is not in the code



**Specialized analyses
require custom tools**



Agility in Moose



Moose is a platform for software and data analysis

The screenshot displays the Moose Finder application window titled "Moose Finder - igeEnt86-2009-05-25 (MooseModel)". The interface is divided into two main panes. The left pane, titled "igeEnt86-2009-05-25 - MooseModel", contains a list of model classes with their counts in parentheses. The right pane, titled "ClassGroup - 1814 items", shows a complexity diagram with nodes and connecting lines, and tabs for "Properties", "Complexity", and "Evaluator".

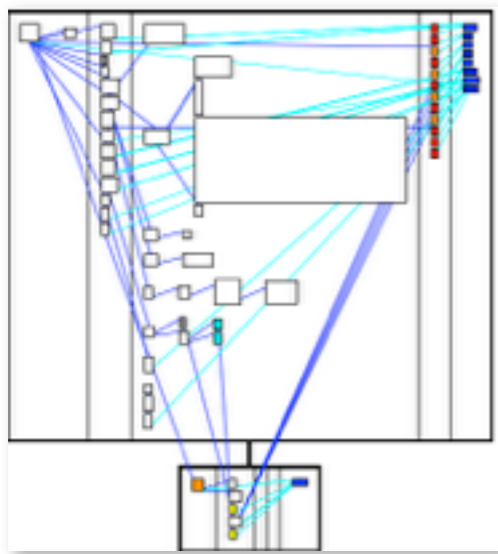
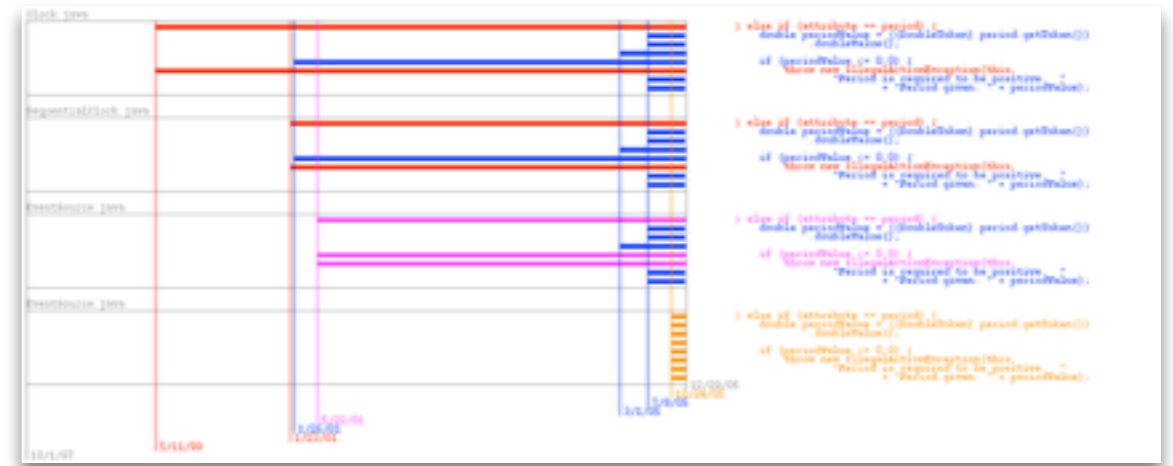
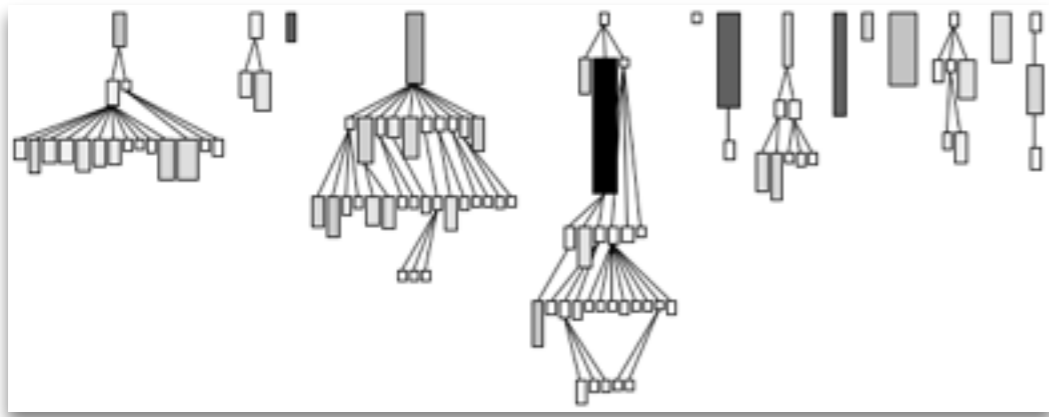
Left Pane: igeEnt86-2009-05-25 - MooseModel

- All famixaccess (32789 FAMIXAccesses)
- All famixannotationinstance (3351 FAMIXAnnotationInstances)
- All famixannotationtype (11 FAMIXAnnotationTypes)
- All famixattribute (7036 FAMIXAttributes)
- All famixcaughtexception (2279 FAMIXCaughtExceptions)
- All famixclass (2447 FAMIXClasses)
- All famixdeclaredexception (5209 FAMIXDeclaredExceptions)
- All famixfunction (2 FAMIXFunctions)
- All famixinheritance (3338 FAMIXInheritances)
- All famixinvocation (35864 FAMIXInvocations)
- All famixlocalvariable (14303 FAMIXLocalVariables)
- All famixmethod (13827 FAMIXMethods)
- All famixnamespace (307 FAMIXNamespaces)
- All famixparameter (11958 FAMIXParameters)
- All famixprimitivetype (9 FAMIXPrimitiveTypes)
- All famixsessionbean (39 FAMIXSessionBeans)
- All famixthrownexception (869 FAMIXThrownExceptions)
- All model classes (1814 FAMIXClasses)**
- All model namespaces (238 FAMIXNamespaces)
- Group (515 FAMIXMethods)

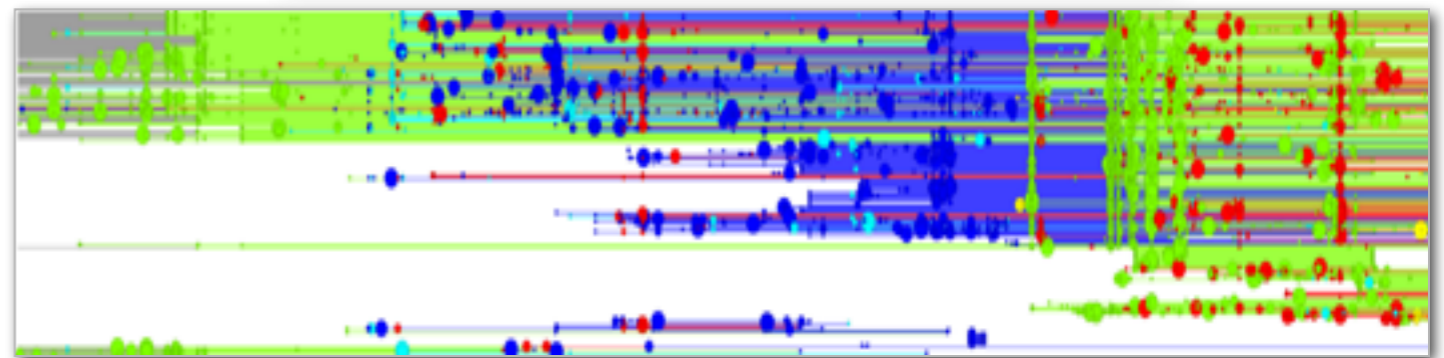
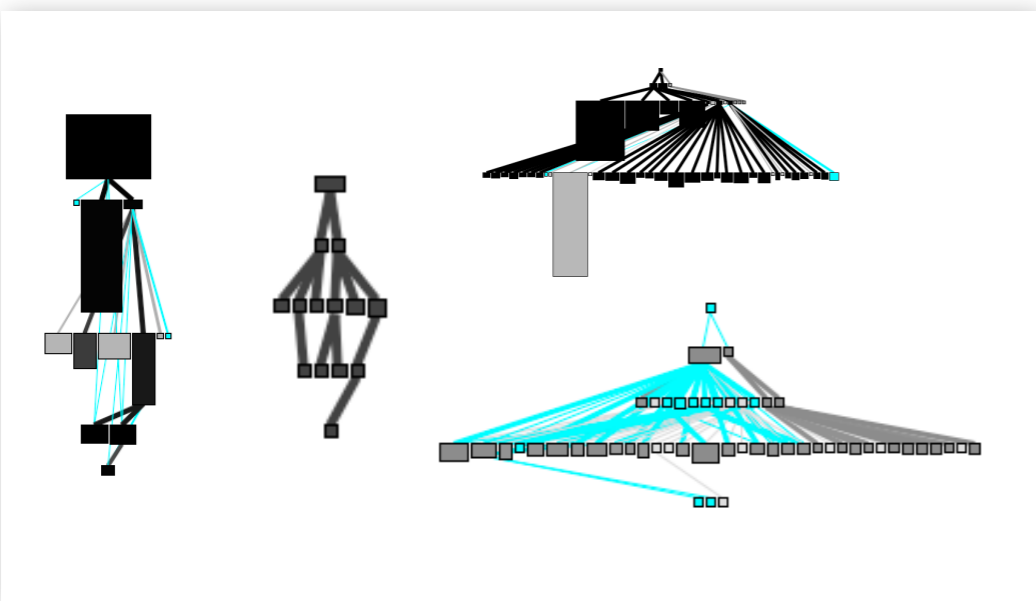
Right Pane: ClassGroup - 1814 items

Complexity diagram showing a hierarchical structure of nodes and connections.

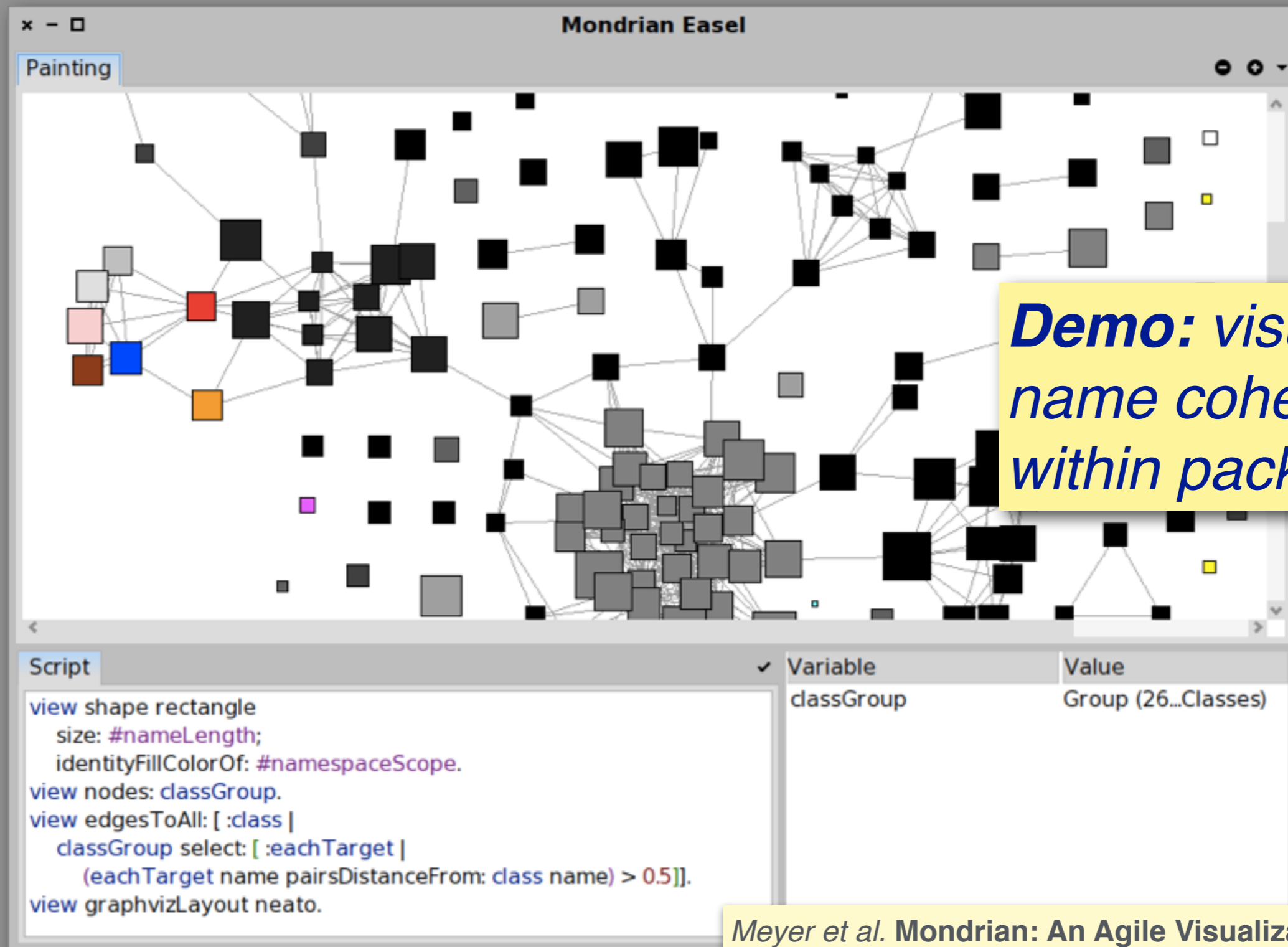
Moose is a platform for modeling software artifacts to enable software analysis. Moose has been developed for well over a decade. It is the work of dozens of researchers, and has been the basis of numerous academic and industrial projects.



System complexity - Clone evolution view
 Class blueprint - Topic Correlation Matrix - Distribution Map
 for topics spread over classes in packages
 Hierarchy Evolution view - Ownership Map



Mondrian Demo



Demo: visualizing name cohesion within packages

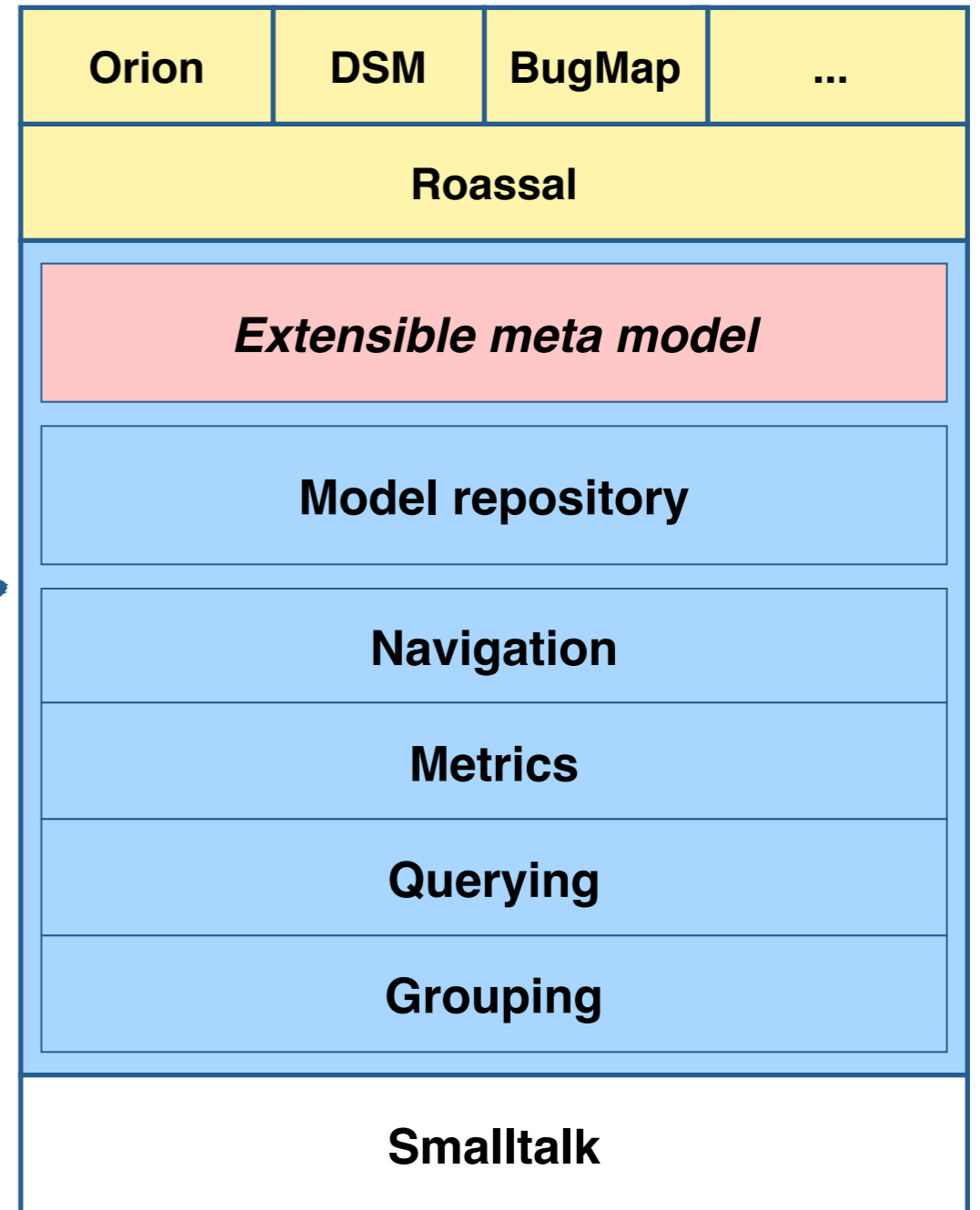
Meyer et al. Mondrian: An Agile Visualization Framework. SoftVis 2006. DOI: 10.1145/1148493.1148513

Agile Modeling





Smalltalk
Java
Python
C++
...

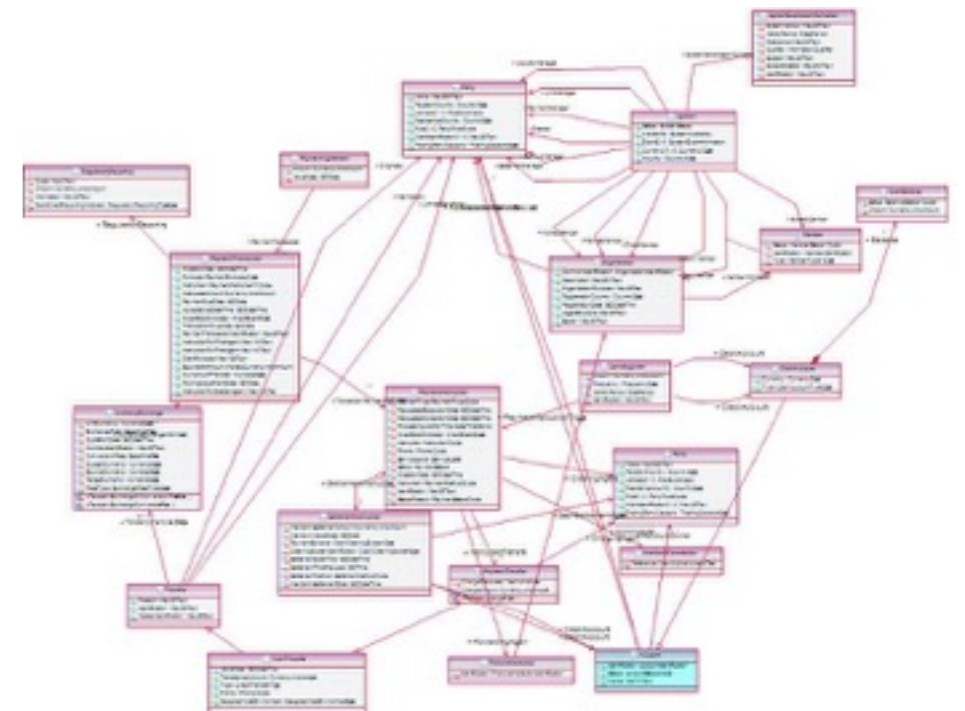


***Moose is a powerful tool
once we have a model ...***

Nierstrasz et al. The Story of Moose.
ESEC/FSE 2005. DOI: 10.1145/1095430.1081707

Challenge

Load the model in the morning, analyze it in the afternoon



The key bottleneck to assessment is creating a suitable model for analysis. If a tool does not already exist, it can take days, weeks or months to parse source files and generate models.

Problems



Unknown languages



Unstructured text



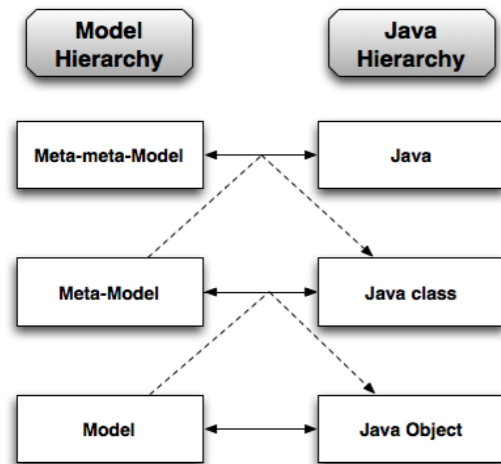
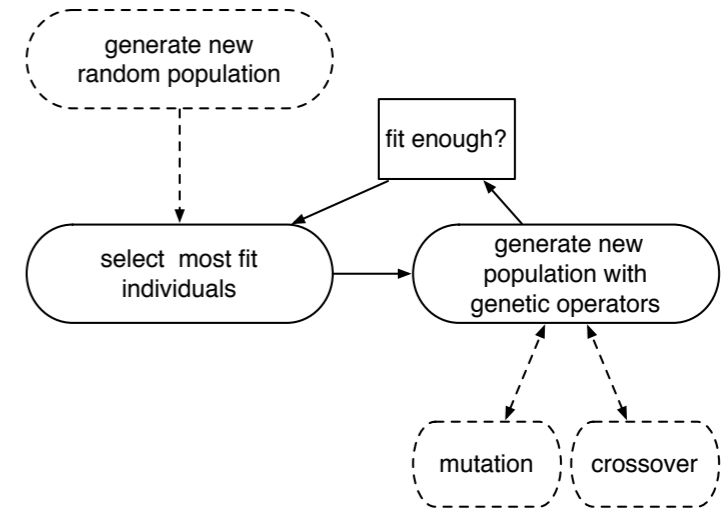
Heterogeneous projects

Developing a parser for a new language is a big challenge. Parsers may be hard to scavenge from existing tools. Not only source code, but other sources of information, like bug reports and emails can be invaluable for model building. Few projects today are built using a single language. Often a GPL is mixed with scripting languages, or even home-brewed DSLs.

Ideas



Grammar Stealing

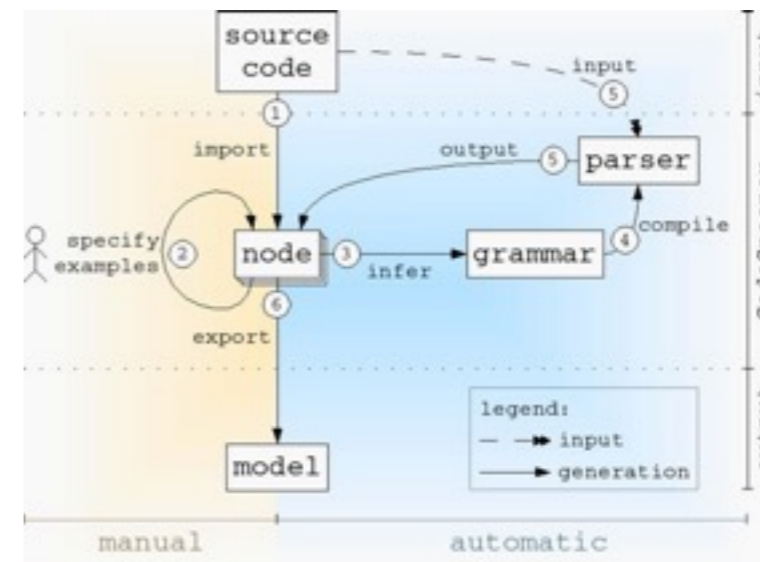


Recycling Trees

Evolutionary Grammar Generation



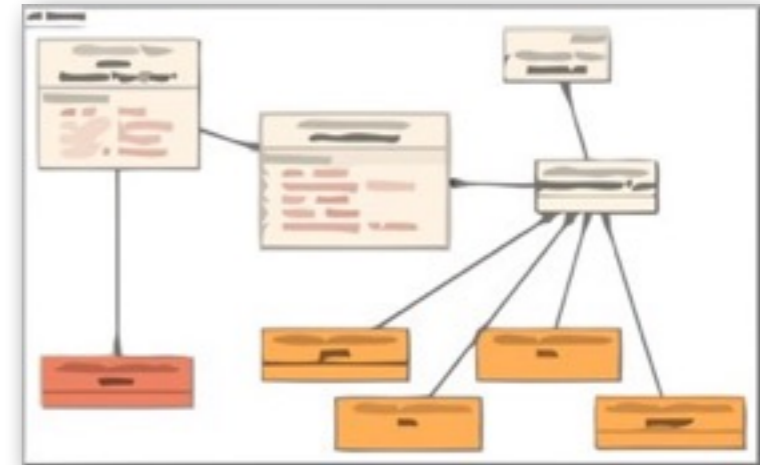
Hooking into an existing tool



Parsing by Example

Agile Modeling Lifecycle

**Build a
coarse model**



**Refine the
model**



**Build a custom
analysis**

Idea: use island grammars to extract coarse models

```
class Shape
  int x;
  int y;

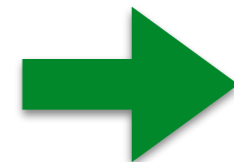
  method draw() ... end
end

method main() ... end
```

```
'class' ID
      (method / . {avoid})*
'end'
```

method?

method



. {avoid}



Problem: island grammars lead to shipwrecks

```
class Shape
```



```
method
```



```
end
```



Tweaking island grammars till they work is not an option ...

```
'class' ID  
    (method / !'end' !method)*  
'end'
```

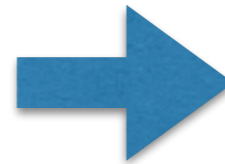
```
method?
```

A Bounded Sea searches for an island in a bounded scope

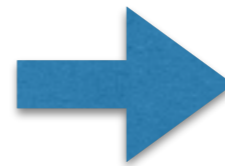
```
'class' ID  
  (~method~)*  
'end'
```

method?

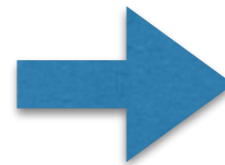
~method~



method



~method~



Architectural Monitoring



Challenge



“What will my code change impact?”

Large software systems are so complex that one can never be sure until integration whether certain changes can have catastrophic effects at a distance.

Ideas: Tracking Software Architecture; exploiting Big Software Data



Problems



SA is not in the code



Diverse views of SA



The IDE focuses on code

Ideas



Uncovering “Software Architecture in the Wild”



**Architecture monitoring
(beyond dependencies)**

What is SA in the Wild?



Andrea Caracciolo, et al. How Do Software Architects Specify and Validate Quality Requirements? Software Architecture 2014.
DOI: 10.1007/978-3-319-09970-5_32

The screenshot shows a web browser window with the address bar containing 'scg.unibe.ch/survey/survey.php'. The page content is as follows:

Survey on Architectural Constraints

In the following survey you will be asked to estimate the relevance of several *architectural constraints*. At each step you will be presented with:

- an **architectural concern**: an interest which pertains to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.
- some **constraint examples** related to the specified architectural concern.
- **questions** related to constraints associated to the specified architectural concern.

The evaluation is subjective and should be based on your personal professional experience. Before expressing your evaluation, please make sure you carefully read and understand the examples given for each constraint. All information obtained from this study will be kept anonymous.

Motivation

The architecture of a software system consists of **design constraints that guarantee non-functional properties**, such as ease of evolution, good run-time performance, and rapid build times. Architecture is rarely explicit in code and alignment between intended and actual architecture is often very difficult to validate and requires consistent human effort. There have been several attempts to address this problem (e.g. component dependencies: JDepend, Sotograph, Structure 101; performance: JMeter) but there are still many aspects that remain ignored. Our goal is to develop new ways for **expressing and verifying architectural constraints**. We envision a set of tools that help identifying implementation solutions that break specified architectural requirements. This questionnaire will help us to identify the most relevant and commonly specified aspects of an architecture. This will provide a solid starting point for our research project.

Thanks for your collaboration.

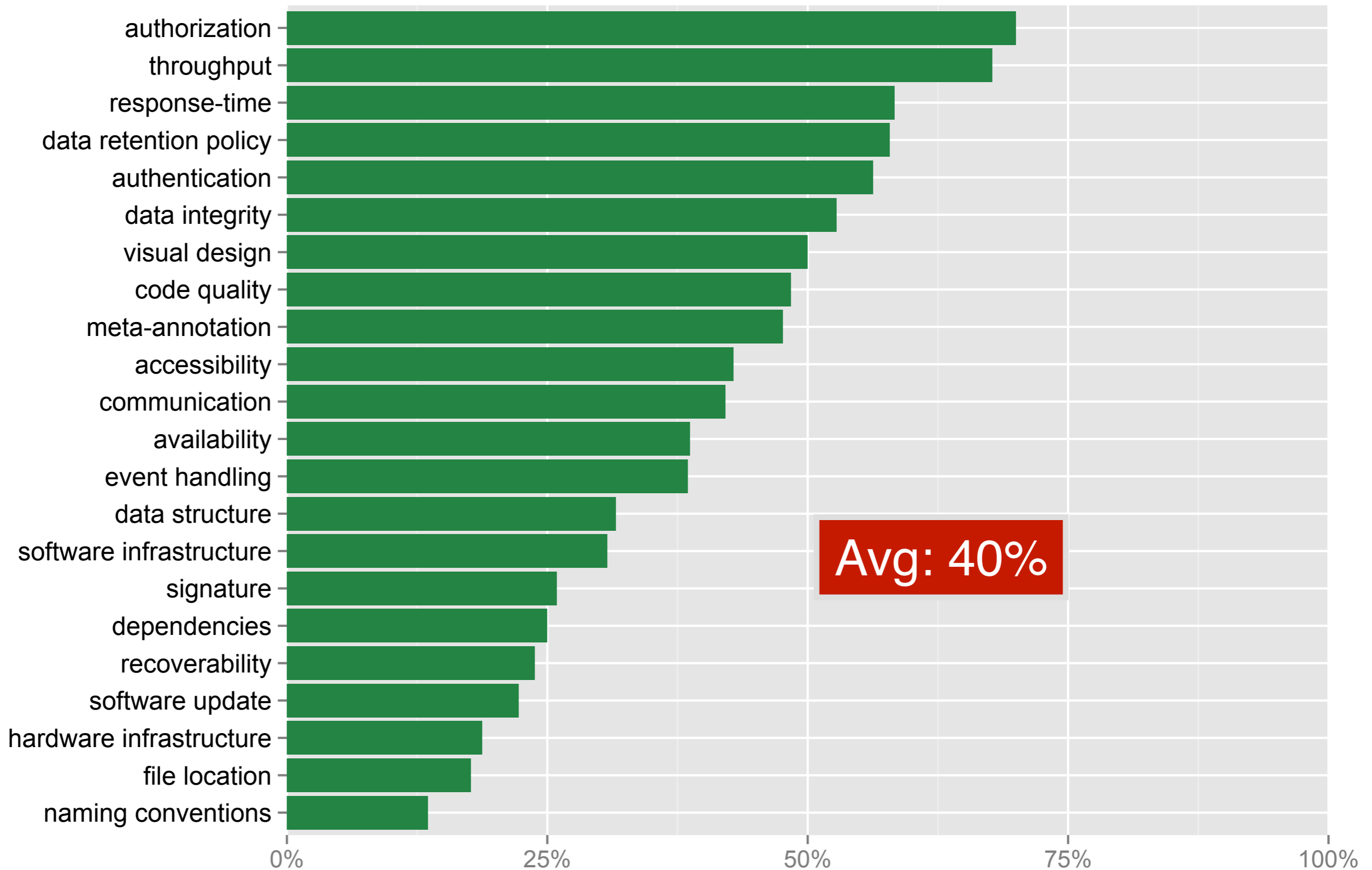
Andrea Caracciolo
caracciolo (at) iam.unibe.ch

[Start Survey](#)

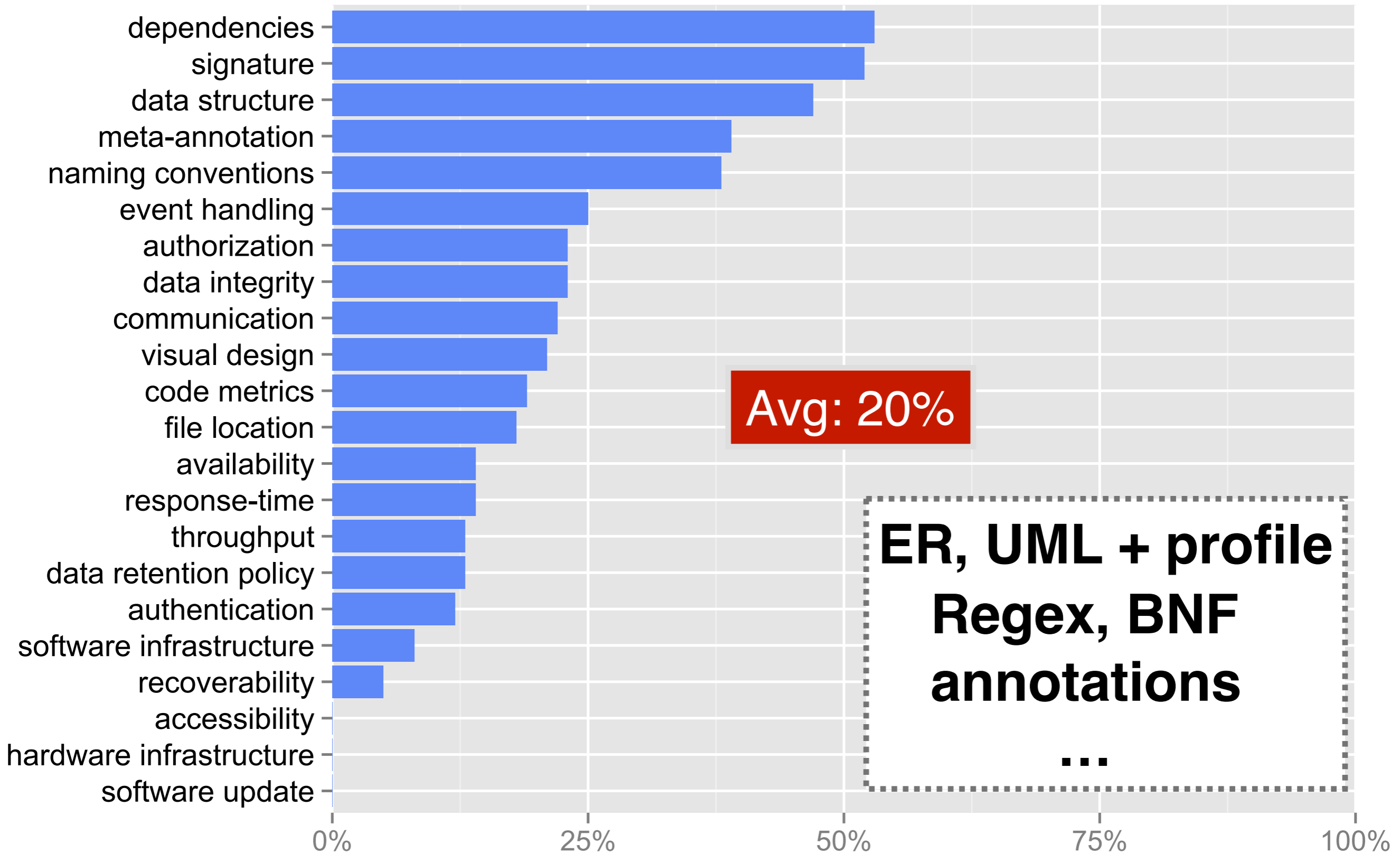
Impact of SA constraints

constraint	Impact (1-5)
availability	4.2
response-time	4.0
authorization	3.9
authentication	3.6
communication	3.4
throughput	3.4
signature	3.4
software infrastructure	3.3
data integrity	3.3
recoverability	3.1
dependencies	3.1
visual design	3.0
data retention policy	3.0
hardware infrastructure	2.9
system behavior	2.9
data structure	2.9
event handling	2.9
code metrics	2.7
meta-annotation	2.6
naming conventions	2.6
file location	2.5
accessibility	2.5
software update	2.2

Automated Validation is not Prevalent



Formalization is not Prevalent



Architectural Rules

Naming Conventions

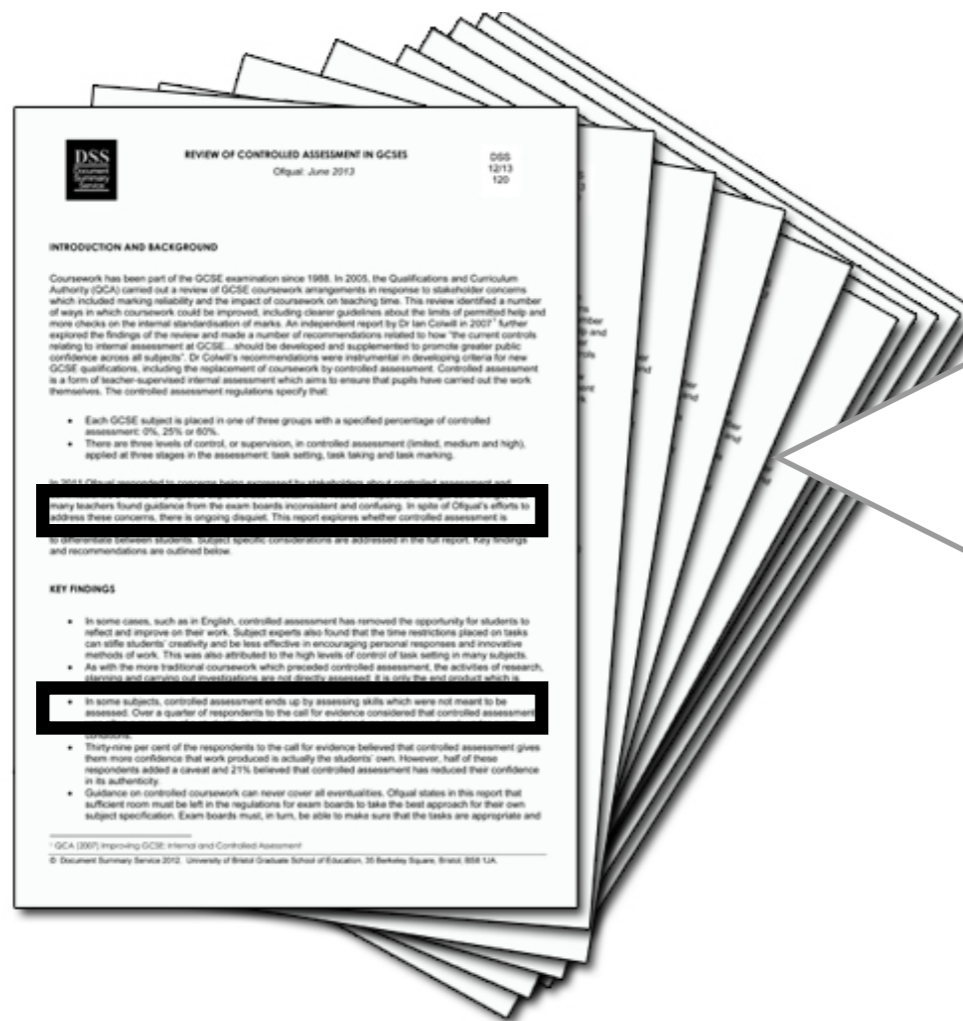
“Repository interfaces can only declare methods named find..()”

Dependencies

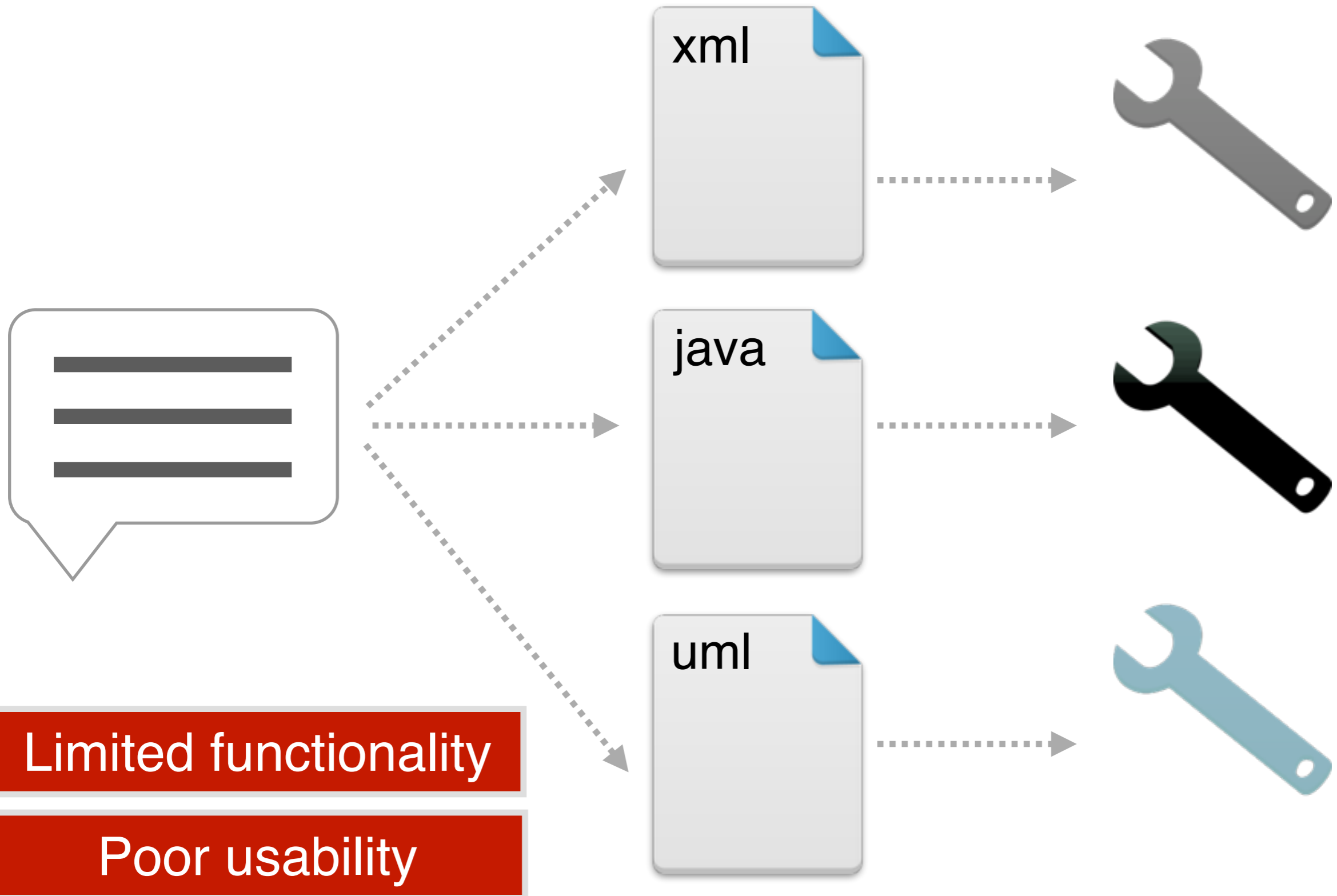
“Only Service classes are allowed to throw ApplicationException”

Performance

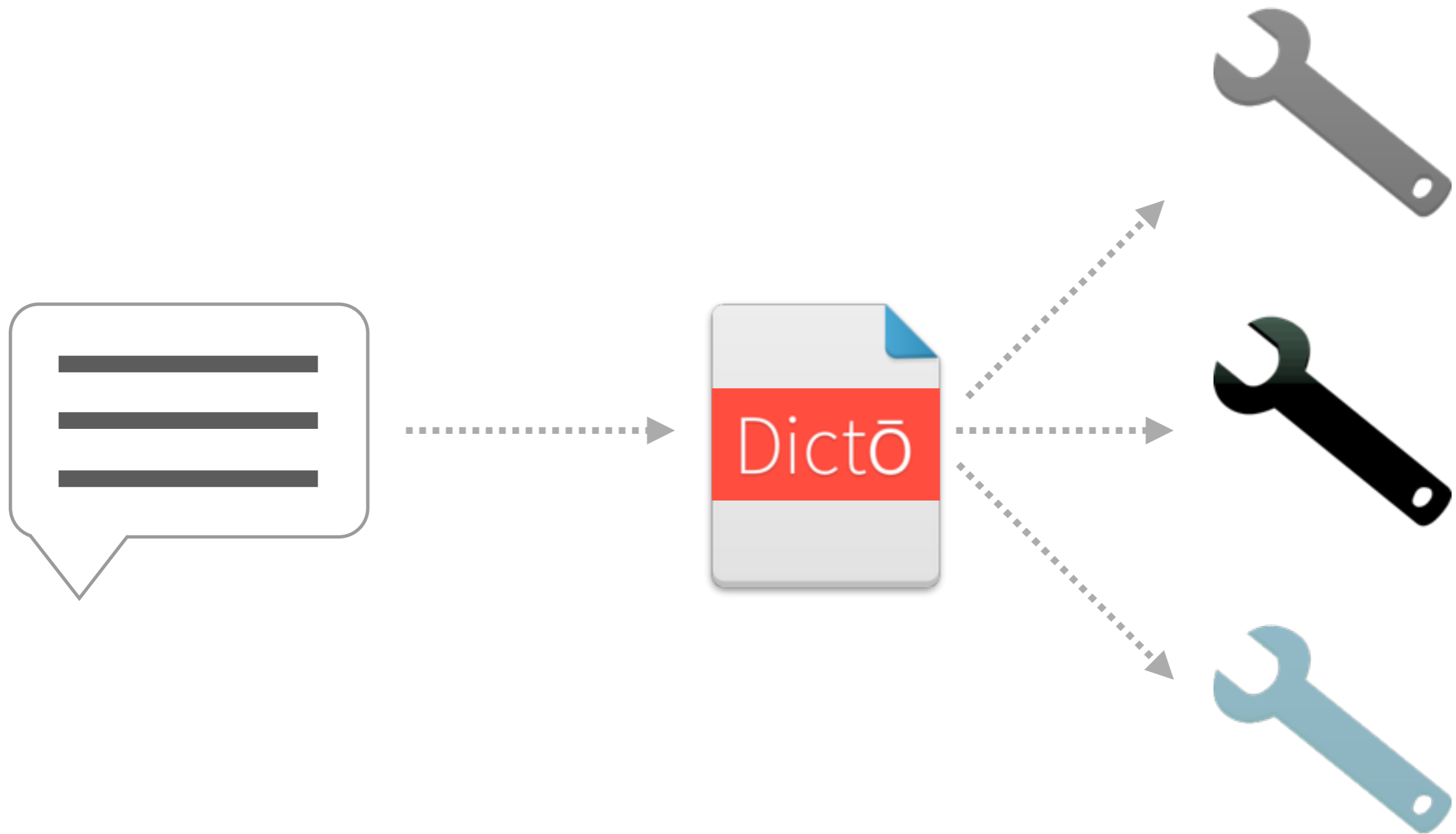
“The rendering operation has to be completed in less than 4ms”



Rule Validation



Dicto — a unified ADSL



Dicto Rules



...

MyService : Website with url="http://www.abc.com/api"

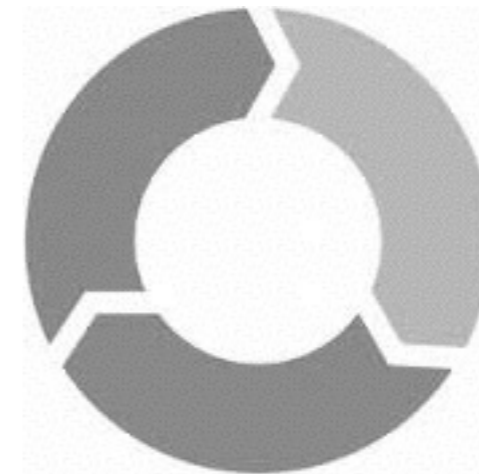
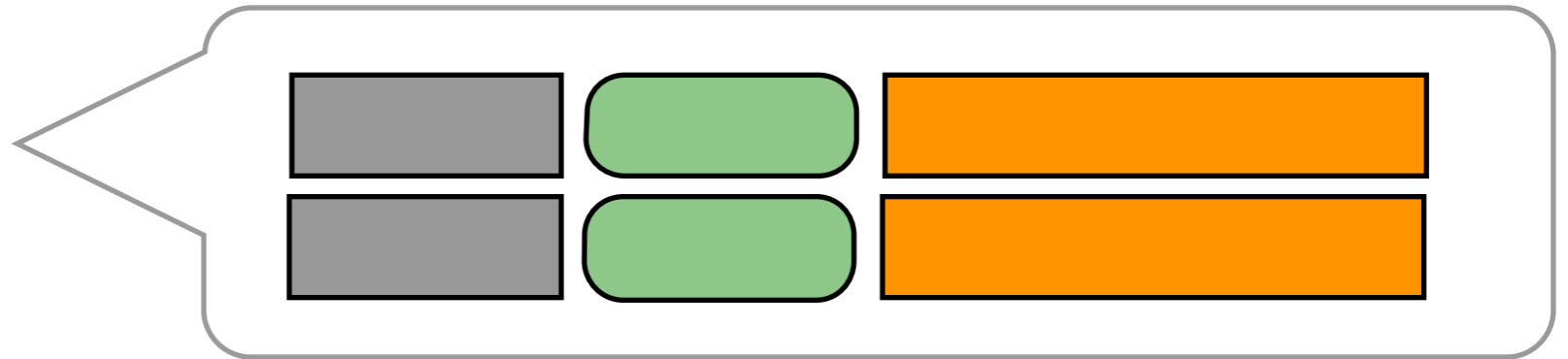
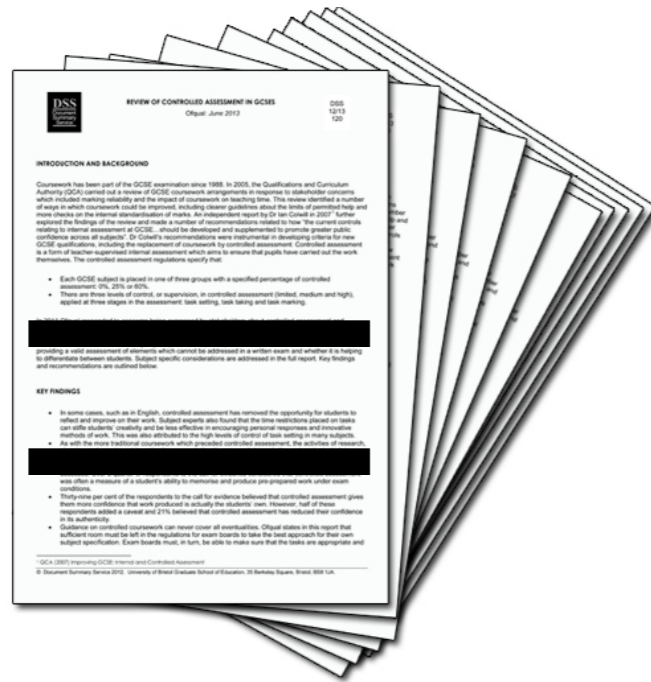
MyService **must** HandleLoadFrom("10 users")

MyService **cannot** HaveResponseTimeLessThan("1000 ms")

MyService **can only** HandleSOAPMessages()

...

Periodic Validation



Rule Examples

Website response time
Website load testing



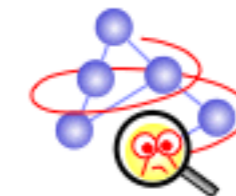
Dependencies



Code clones



Deadlock freeness



File Content

grep

Moldable Tools



Challenge

Build a new assessment tool in ten minutes

Custom analyses require custom tools. Building a tool should be as easy as writing a query in SQL or a form-based interface.



Problems

What tools do developers really need?

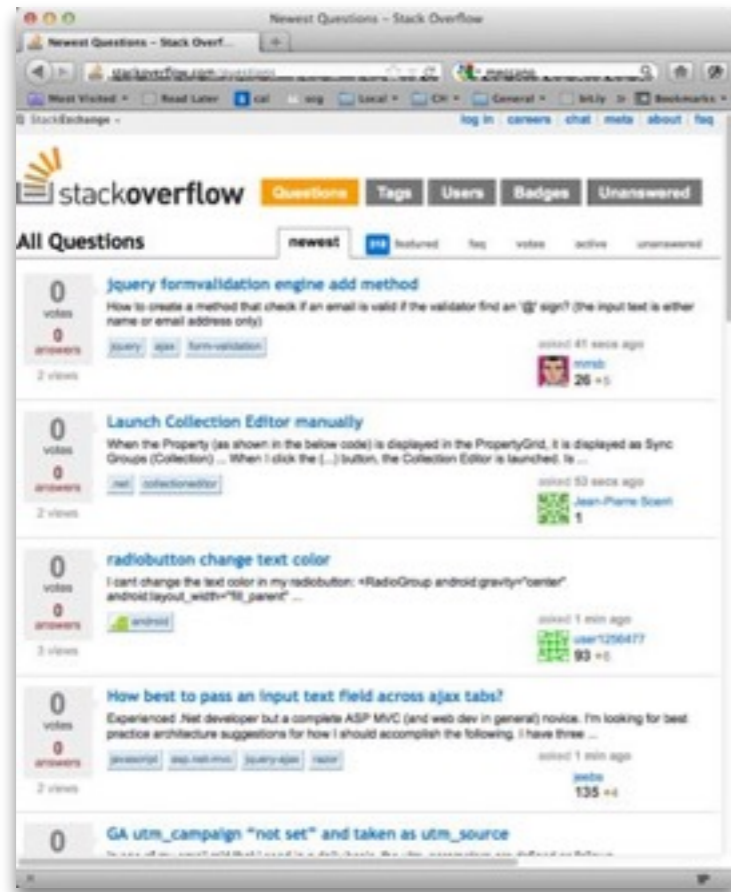


What is a unifying meta-model for tool construction?



What are appropriate meta-tools?

Ideas



Analyze developer needs (!)



“Moldable” Tools
(not just plug-ins)

The screenshot shows a debugger window titled "Halt". It is divided into three main sections:

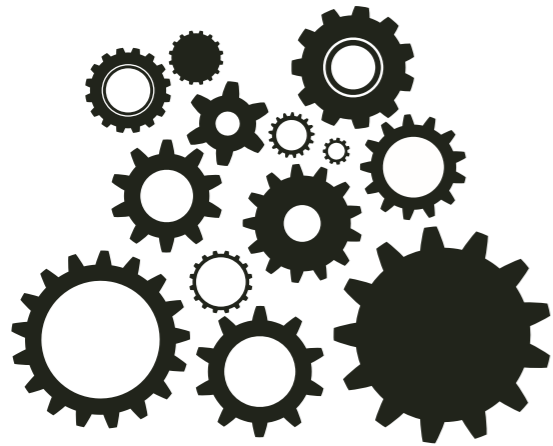
- Stack:** A list of stack frames. The top frame is highlighted: `RubScrolledTextMorph>>whenTextAcceptRequest:`. Below it are several `MessageSend>>` frames and other method calls like `AnnouncementSubscription>>deliver:`.
- Source:** A window showing the source code of the selected frame. The code is:


```
whenTextAcceptRequest: anAnnouncement
  self halt.
  self announcer announce: anAnnouncement.
  self acceptContents
```
- Inspector:** A table showing the current object's variables and their values.

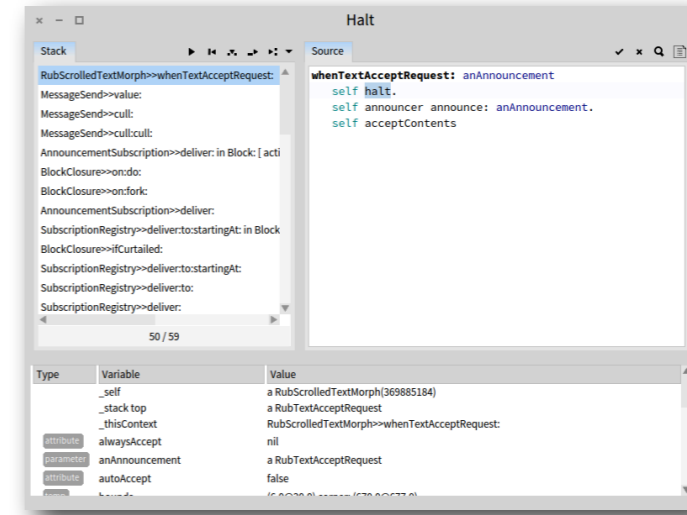
Type	Variable	Value
	<code>_self</code>	a RubScrolledTextMorph(369885184)
	<code>_stack top</code>	a RubTextAcceptRequest
	<code>_thisContext</code>	RubScrolledTextMorph>>whenTextAcceptRequest:
attribute	<code>alwaysAccept</code>	nil
parameter	<code>anAnnouncement</code>	a RubTextAcceptRequest
attribute	<code>autoAccept</code>	false
temp	<code>found</code>	(6 0020 0) ... (670 00677 0)

Conventional debuggers just offer an interface to the run-time stack.

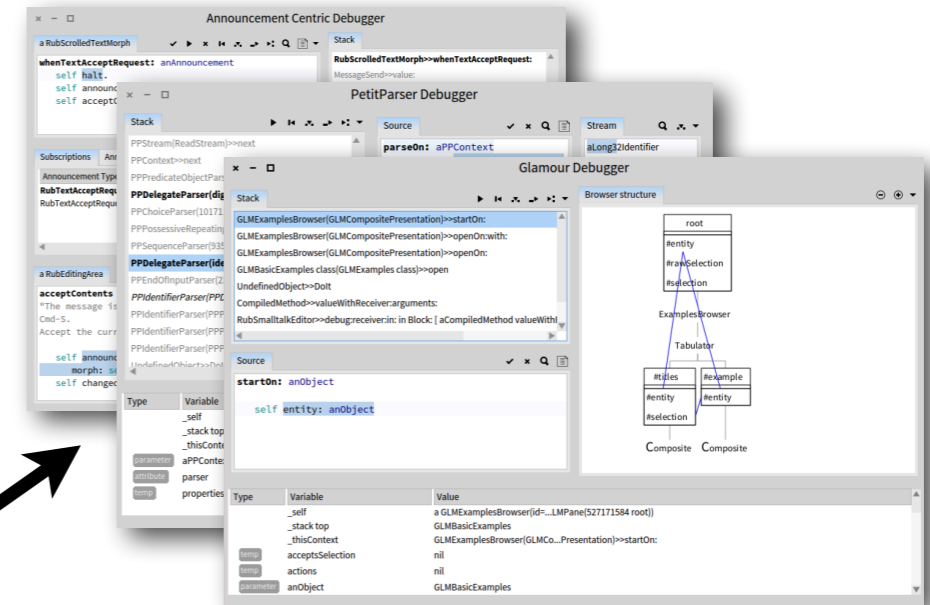
Specific Models



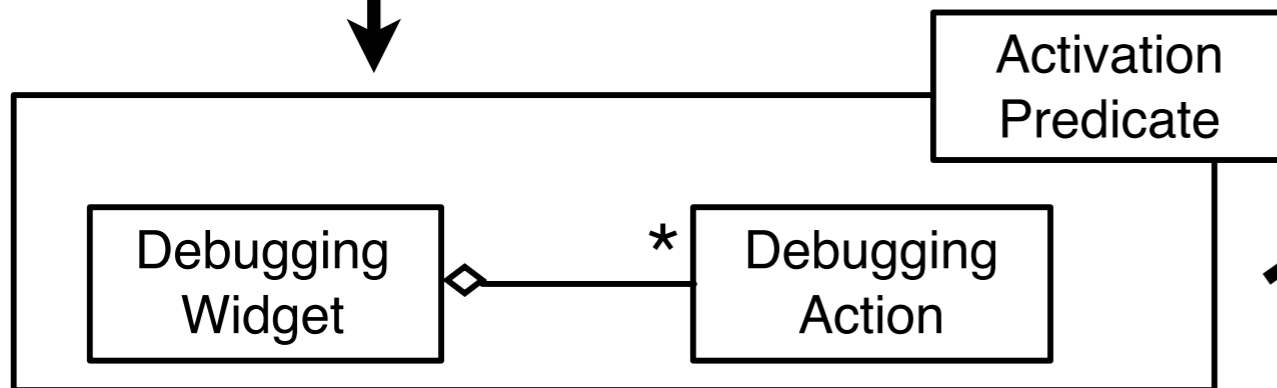
Generic Debugger



Domain-specific Debuggers



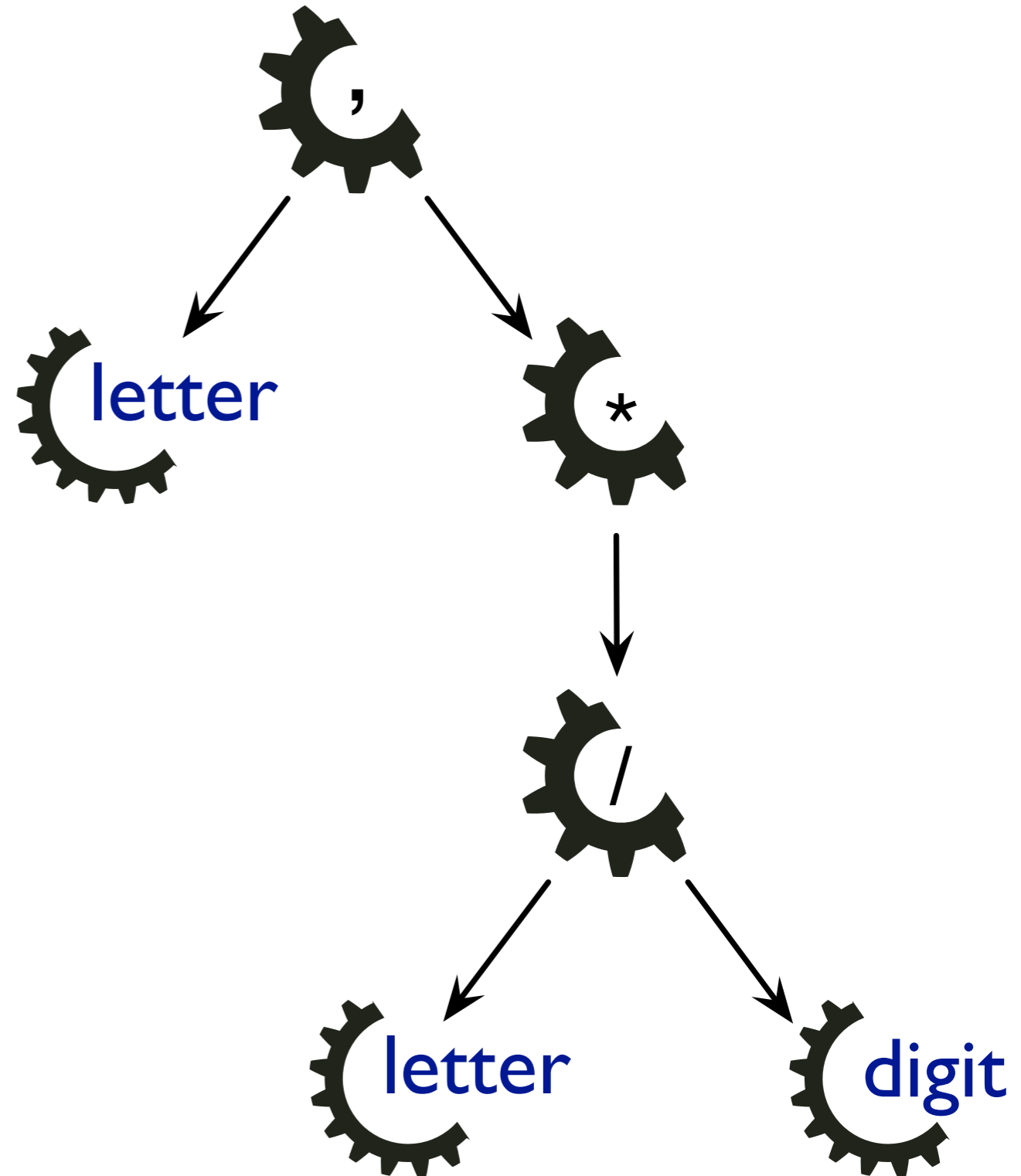
The Moldable Debugger



Andrei Chis et al. The Moldable Debugger: A Framework for Developing Domain-Specific Debuggers. SLE 2014. DOI: 10.1007/978-3-319-11245-9_6



PetitParser



identifier

letter , (letter / digit) *

IdentifierParser new

parse: 'aLong32Identifier'

Default Debugger

Stack



```
PPStream(ReadStream)>>next
PPContext>>next
PPPredicateObjectParser>>parseOn:
PPDelegateParser>>parseOn:
PPChoiceParser>>parseOn:
PPPossessiveRepeatingParser>>parseOn:
PPSequenceParser>>parseOn:
PPDelegateParser>>parseOn:
PPEndOfInputParser>>parseOn:
PPIdentifierParser(PPDelegateParser)>>parseOn:
PPIdentifierParser(PPParser)>>parseWithContext:
PPIdentifierParser(PPParser)>>parse:withContext:
PPIdentifierParser(PPParser)>>parse:
```

Source



```
parseOn: aPPContext
  ^ parser parseOn: aPPContext
```

Type	Variable	Value
	_self	a PPDelegateParser(identifier)
	_stack top	a PPContext
	_thisContext	PPDelegateParser>>parseOn:
parameter	aPPContext	a PPContext
attribute	parser	a PPSequenceParser(273678336)
temp	properties	a Dictionary(#name->#identifier)

PetitParser Debugger

Stack

▶ ◀ ⏪ ⏩ ⏴ ⏵

```

PPStream(ReadStream)>>next
PPContext>>next
PPPredicateObjectParser(129761280, 'digit expected'):
PPDelegateParser(digit)>>parseOn:
PPChoiceParser(1017118720)>>parseOn:
PPPossessiveRepeatingParser(214958080)>>parseOn:
PPSequenceParser(935854080)>>parseOn:
PPDelegateParser(identifier)>>parseOn:
PPEndOfInputParser(239861760)>>parseOn:
PPIdentifierParser(PPDelegateParser)(471334912)>>pa
PPIdentifierParser(PPPParser)(471334912)>>parseWithC
PPIdentifierParser(PPPParser)(471334912)>>parse:with
PPIdentifierParser(PPPParser)(471334912)>>parse:
UndefinedObject>>DoIt
    
```

Source

✓ ✕ 🔍 📄

```

parseOn: aPPContext
  ^ parser parseOn: aPPContext
    
```

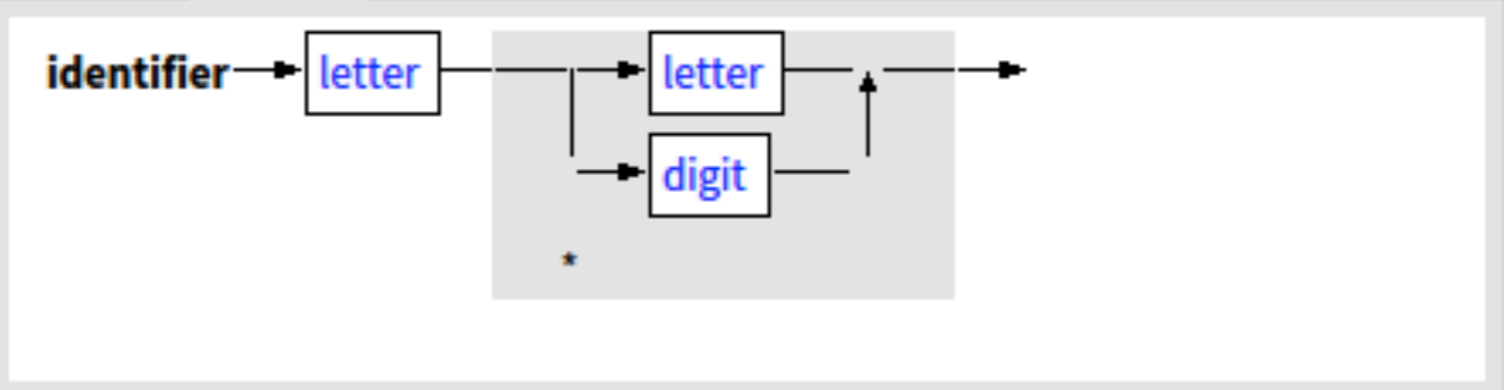
Stream

🔍 ⏪ ⏩

```

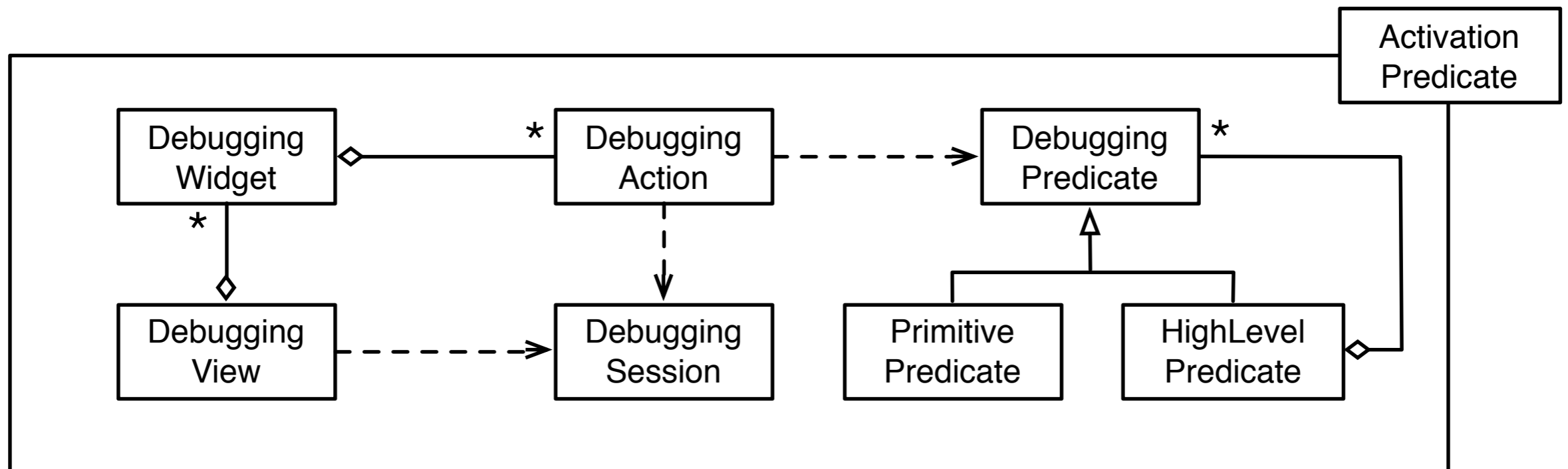
aLong32Identifier
    
```

Source Graph Map Example First Follow



Type	Variable	Value
	_self	a PPDelegateParser(identifier)
	_stack top	a PPContext
	_thisContext	PPDelegateParser>>parseOn:
parameter	aPPContext	a PPContext
attribute	parser	a PPSequenceParser(935854080)
temp	properties	a Dictionary(#name->#identifier)

Domain specific-extensions



Debugging widgets

Debugging actions

Next parser

Next production

Production(a production)

Next failure

Stream position(an Integer)

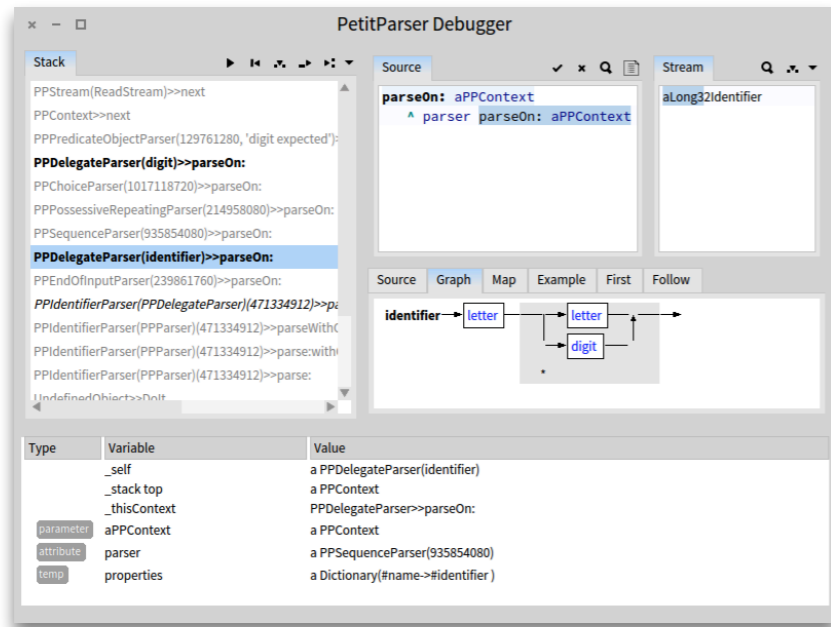
Stream position changed

The screenshot shows a debugger interface with four main panels:

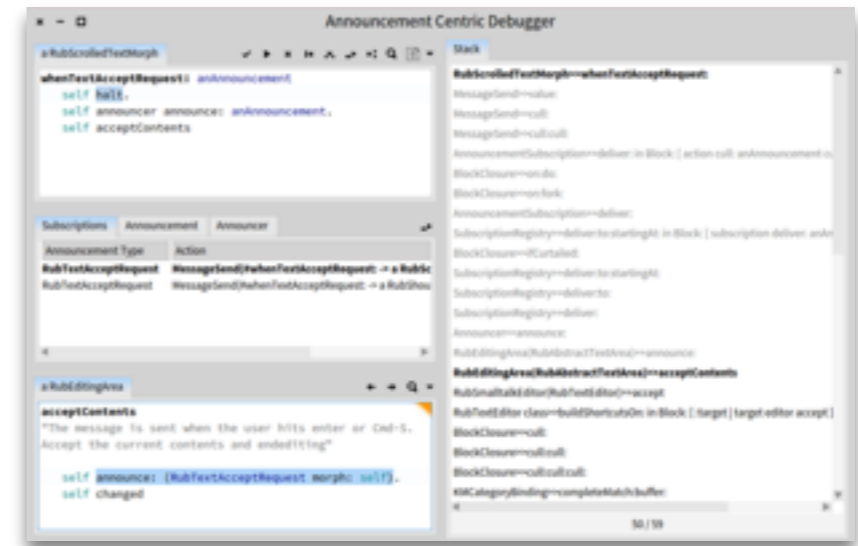
- Stack:** A list of stack frames. The current frame is `PPDelegateParser(identifier)>>parseOn:`. Other frames include `PPStream(ReadStream)>>next`, `PPContext>>next`, `PPPredicateObjectParser(129761280, 'digit expected')`, `PPChoiceParser(1017118720)>>parseOn:`, `PPPossessiveRepeatingParser(214958080)>>parseOn:`, `PPSequenceParser(935854080)>>parseOn:`, `PPEndOfInputParser(239861760)>>parseOn:`, `PPIdentifierParser(PPDelegateParser)(471334912)>>pa`, `PPIdentifierParser(PPParser)(471334912)>>parseWithC`, `PPIdentifierParser(PPParser)(471334912)>>parse:with`, `PPIdentifierParser(PPParser)(471334912)>>parse:`, and `UndefinedObject>>Dolt`.
- Source:** Shows the source code for the current frame: `parseOn: aPPContext` and `^ parser parseOn: aPPContext`.
- Stream:** Shows the current stream content: `aLong32Identifier`.
- Graph:** A parse graph showing the derivation of the current token. It starts with `identifier` which derives `letter`. This `letter` then derives `letter` and `digit`. There is a loop back to `letter` and a `*` symbol indicating repetition.

Type	Variable	Value
	<code>_self</code>	<code>a PPDelegateParser(identifier)</code>
	<code>_stack top</code>	<code>a PPContext</code>
	<code>_thisContext</code>	<code>PPDelegateParser>>parseOn:</code>
parameter	<code>aPPContext</code>	<code>a PPContext</code>
attribute	<code>parser</code>	<code>a PPSequenceParser(935854080)</code>
temp	<code>properties</code>	<code>a Dictionary(#name->#identifier)</code>

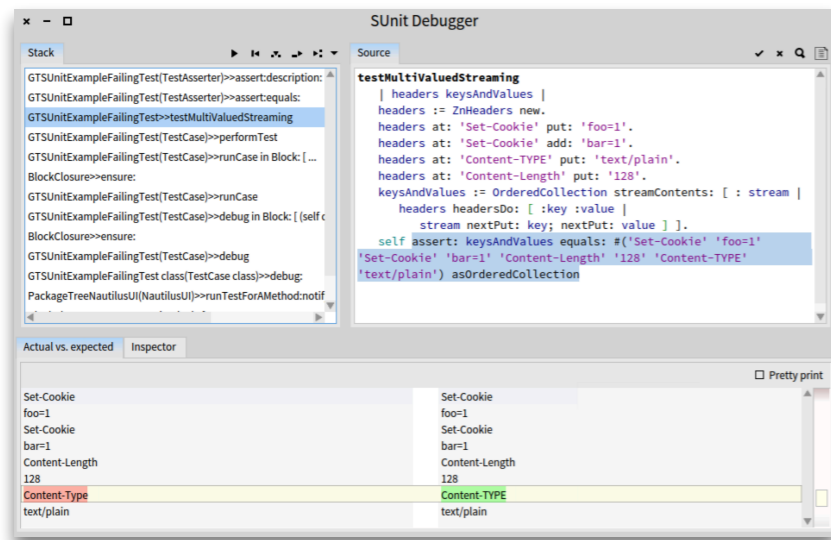
Petit Parser



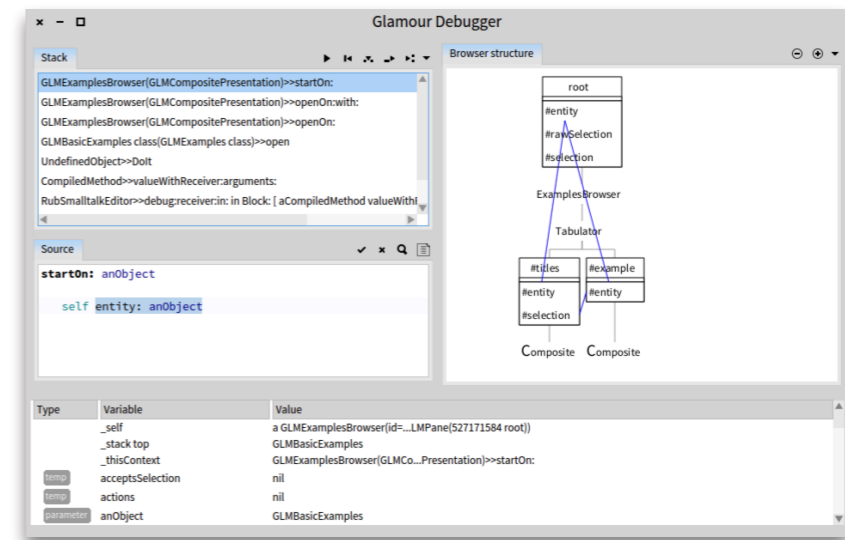
Events



SUnit



Glamour



New debuggers are cheap

	Session	Operations	View	Total
Base model	800	700	-	1500
Default Debugger	-	100	400	500
Announcements	200	50	200	450
Petit Parser	100	300	200	600
Glamour	150	100	50	300
SUnit	100	-	50	150

The Moldable Inspector

The screenshot displays the 'Inspector on a PGConnection (a PGConnection)' application interface, which is divided into three main panes:

- SQL Pane:** Contains a SQL query:

```
select city.countrycode, city.name as cityname, city.population, country.lifeexpectancy as life, country.continent from city left join country on city.countrycode=country.code
```
- Table Pane:** Displays a table with 5 columns: countrycode, cityname, population, life, and continent. It shows 20 rows of data, with a status bar at the bottom indicating '50 / 4079' rows.
- Bar Chart Pane:** A horizontal bar chart titled 'a GET2DiagramBuilder (a GET2DiagramBuilder)'. The y-axis lists 20 cities, and the x-axis represents a numerical value. The bars are color-coded: Mumbai (Bombay) is red, Seoul is red, São Paulo is blue, Shanghai is red, Jakarta is red, Karachi is red, Istanbul is red, Ciudad de México is orange, Moscow is cyan, New York is orange, Tokyo is red, Peking is red, London is cyan, Delhi is red, Cairo is green, Teheran is red, Lima is blue, Chongqing is red, and Bangkok is red.

Conclusion

Current IDEs offer developers primitive support for software assessment



Developers need support for agile modeling, architectural monitoring and moldable tools