Model-based Mining of Software Repositories

Markus Scheidgen

Agenda

- *Mining Software Repositories* (MSR) and current approaches
- *srcrepo* – a model-based MSR system
  - *srcrepo* components and analysis process
  - a meta-model for *source code repositories*
  - gathering software metrics with an *OCL-like* internal Scala DSL
- **work in progress** - discussion of remaining problems and limitations

Relevant Research Fields

Mining Software Repositories (MSR)

The term *mining software repositories* (MSR) has been coined to describe a broad class of investigations into the examination of *software repositories*.

The premise of MSR is that empirical and systematic investigations of repositories will shed new light on the process of *software evolution*. [1]

Software Metrics

A software metric is a mathematical definition mapping the entities of a software system to numeric metrics values.

[...] to express features of software with numbers in order to facilitate software quality assessment. [2]

Reverse Engineering

Reverse engineering is the process of analyzing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or at a higher level of abstraction [3]

Software Evolution Research (SER)

- (dis-)proving *Lehmann's Laws* of software evolution
- empirical investigations of software repositories through statistical analysis of software and software change metrics over the evolutionary cause of many software systems.

Model-based Mining Software Repositories (with *srcrepo*)

Overcoming **heterogeneity** and **accessibility** by raising the level of abstraction, while ensuring **scalability** and retaining meaningful **information depth**.

1. **H. Kagdi, M.L. Collard, J.I. Maletic:** *A survey and taxonomy of approaches for mining software repositories in the context of software evolution*; Journal of Software Maintenance and Evolution: Research and Practice; Vol.19/Nr.2/2007
2. **R. Lincke, J. Lundberg, W. Löwe**: *Comparing Software Metrics Tools;* 8th International Symposium on Software Testing and Analysis; 2008
3. **E.J. Chikofsky, J.H. Cross**: *Reverse engineering and design recovery: A taxonomy*; IEEE Software; Vol.7/Nr.1/1990

Contemporary Approaches to Large Scale MSR for SER

FLOSS Metrics [1]	Sourcerer [2]	Boa [3]
■ database for over 3000 open source software projects ■ contains data about all revisions ■ *Alitheia*, multiple *version control systems* (VCS), but only text-based metrics ■ not only *source code repositories* (SCR) via VCS, also issue-tracking systems, mailing-lists, etc.	■ database and searchable index of declarations from over 4000 Java software projects ■ tracks only release revisions ■ metrics based on declarations (classes, methods, fields, etc., e.g. CK-metrics), but not based on actual implementations (e.g. McCabe, Halstead)	■ *domain specific language* (DSL) for mining meta-data in ultra-large software repositories ■ only tracks VCS meta-data, e.g. *"How many revisions are there in all Java projects using SVN?"*

	Scalability	Heterogeneity	Accessibility	Information Depth
	■ a project ■ large scale: multiple related projects, e.g. Apache, Eclipse ■ ultra-large scale: 100k+ unrelated projects with varying quality [1,2,3]	■ abstraction from VCS [1,2,3] ■ abstraction from programming language: only meta-data [3] or text [1]	■ programming ■ database with index [1,2] ■ DSL [3]	■ all revisions [1,3], sample revisions [2] ■ meta-data [3] ■ text [1] ■ declarations [2]

1. **G. Gousios, D. Spinellis:** *Alitheia core: An extensible software quality monitoring platform*; Proceedings of the 31st International Conference on Software Engineering; 2009
2. **E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, P. Baldi.**: *Sourcerer: mining and searching internet-scale software repositories;* Data Mining and Knowledge Discovery; Vok.18/Nr.2/2009
3. **R. Dyer, H.A. Nguyen, H. Rajan, T.N. Nguyen**: *Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories;* Proceedings of the 2013 International Conference on Software Engineering; 2013

Goals and Hypothesis

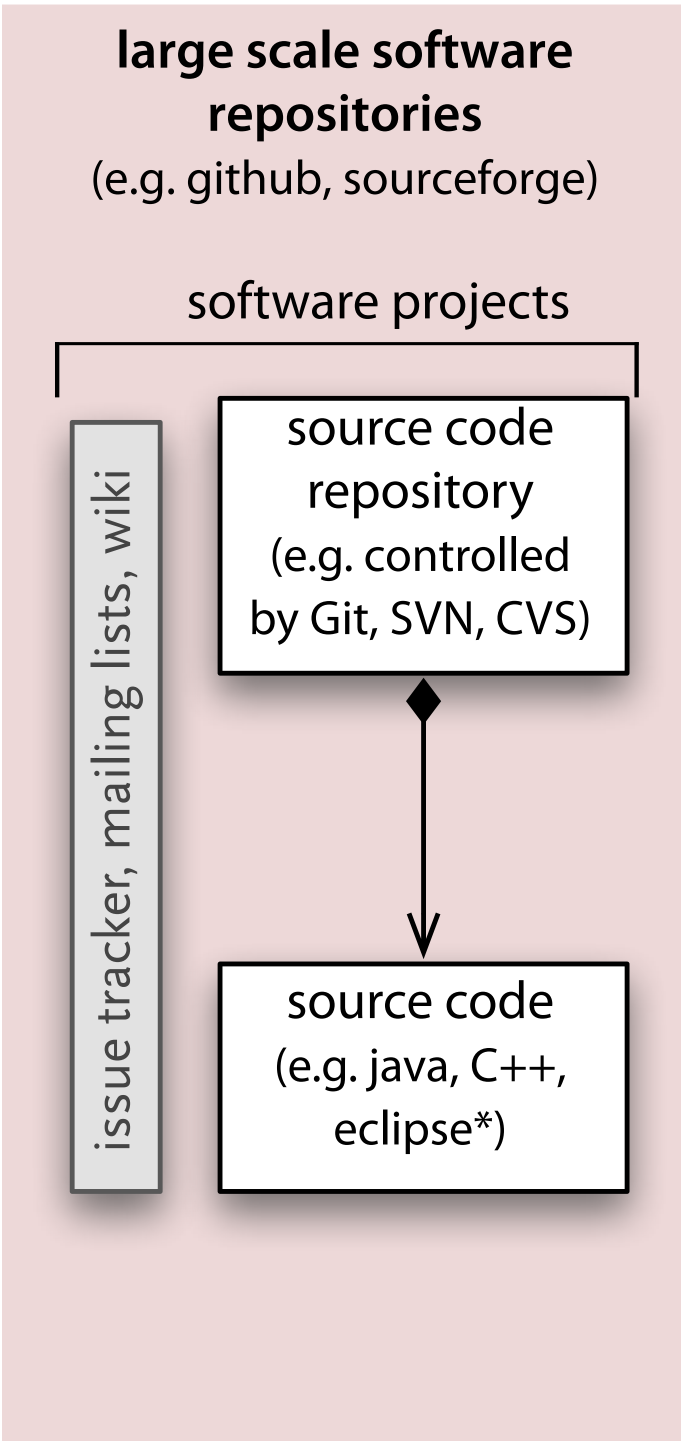
	Scalability	Heterogeneity	Accessibility	Information Depth
	▪ a project ▪ large scale: multiple related projects, e.g. Apache, Eclipse ▪ ultra-large scale: 100k+ unrelated projects with varying quality [1,2,3]	▪ abstraction from VCS [1,2,3] ▪ abstraction from programming language: only meta-data [3] or text [1]	▪ programming ▪ database with index [1,2] ▪ DSL [3]	▪ all revisions [1,3], sample revisions [2] ▪ meta-data [3] ▪ text [1] ▪ declarations [2]

Goals and Hypothesis

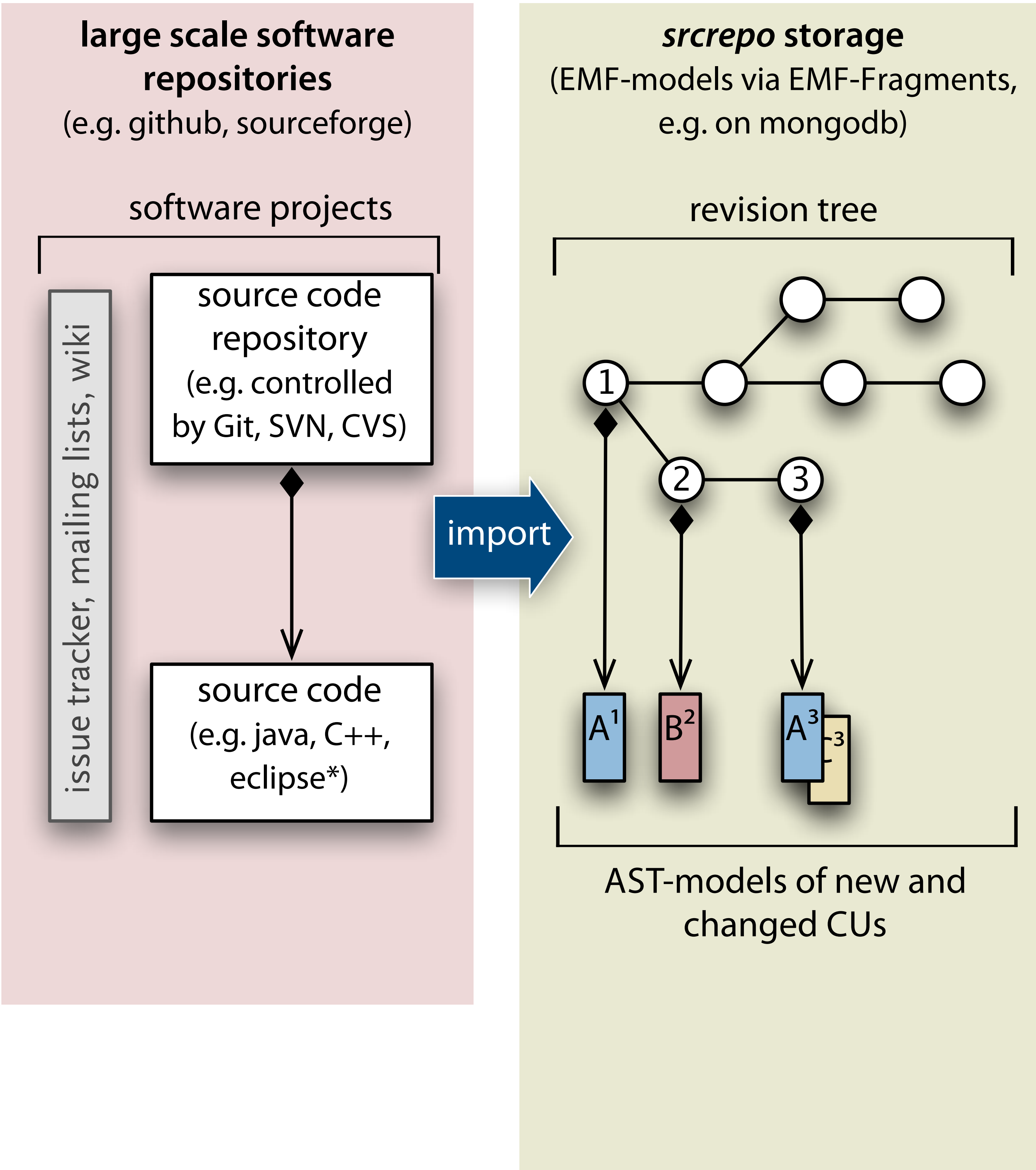
	Scalability	Heterogeneity	Accessibility	Information Depth
approaches	■ a project ■ large scale: multiple related projects, e.g. Apache, Eclipse ■ ultra-large scale: 100k+ unrelated projects with varying quality [1,2,3]	■ abstraction from VCS [1,2,3] ■ abstraction from programming language: only meta-data [3] or text [1]	■ programming ■ database with index [1,2] ■ DSL [3]	■ all revisions [1,3], sample revisions [2] ■ meta-data [3] ■ text [1] ■ declarations [2]
goals	■ cluster- (batching) and cloud- (Map/Reduce)- computing support ■ distributable databases	■ common meta-model for VCSs ■ meta-models for programming languages ■ common meta-model for metrics	■ internal DSL: DSL + programming with models ■ common modeling framework ■ existing tools/ frameworks	■ all revisions ■ *abstract syntax trees* (AST) ■ differences between revisions (e.g. metrics on adaptations and refactorings)
hypothesis	■ distributable model persistence ■ distributed processing of models	■ abstraction for different VCSs exists ■ abstraction regarding metrics for diff. progr. languages exists ■ abstraction for diff. languages exists	■ is there a reasonable programming abstraction for gathering metrics/ change metrics	

srcrepo – Components and Process

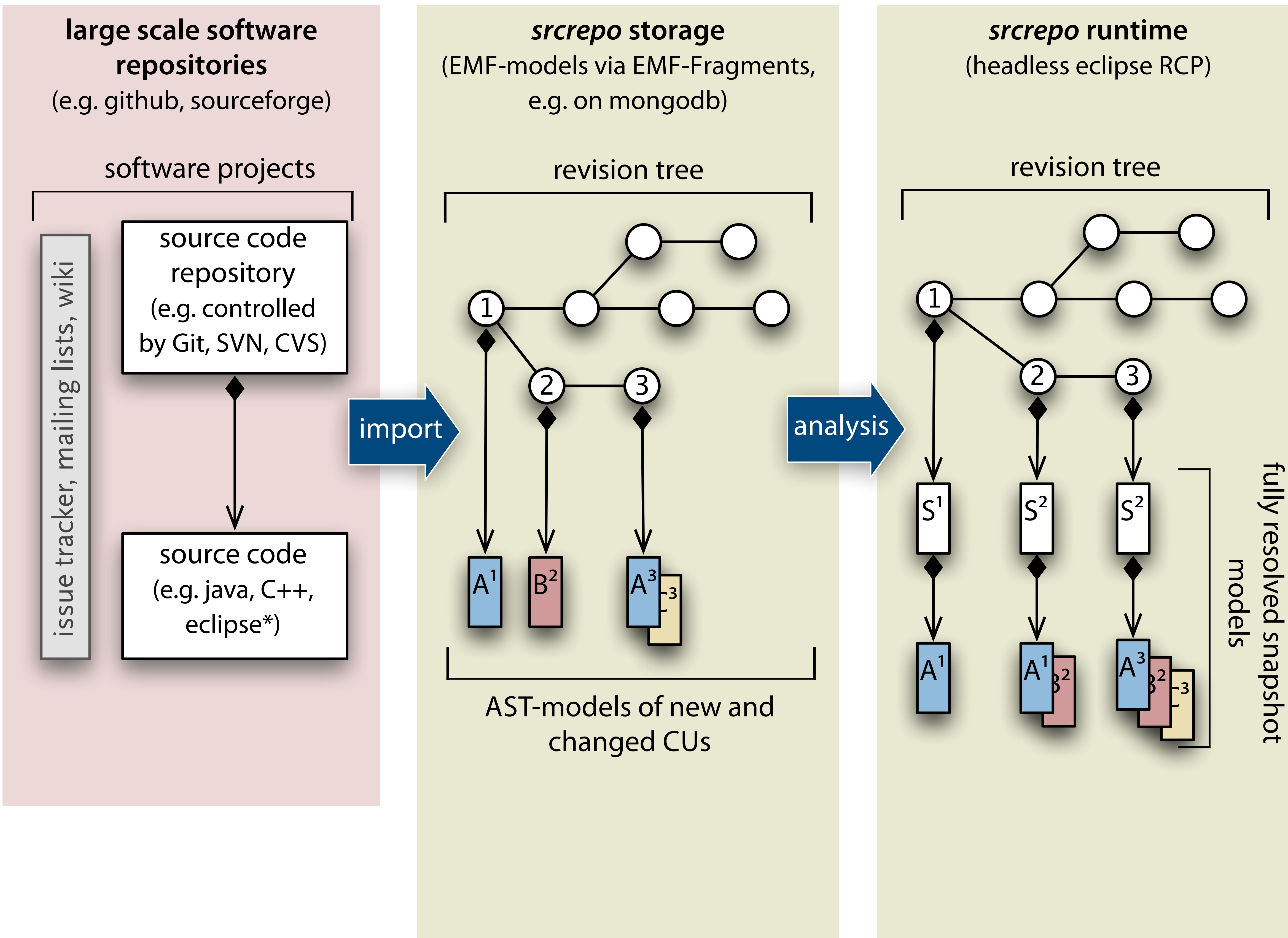
srcrepo – Components and Process



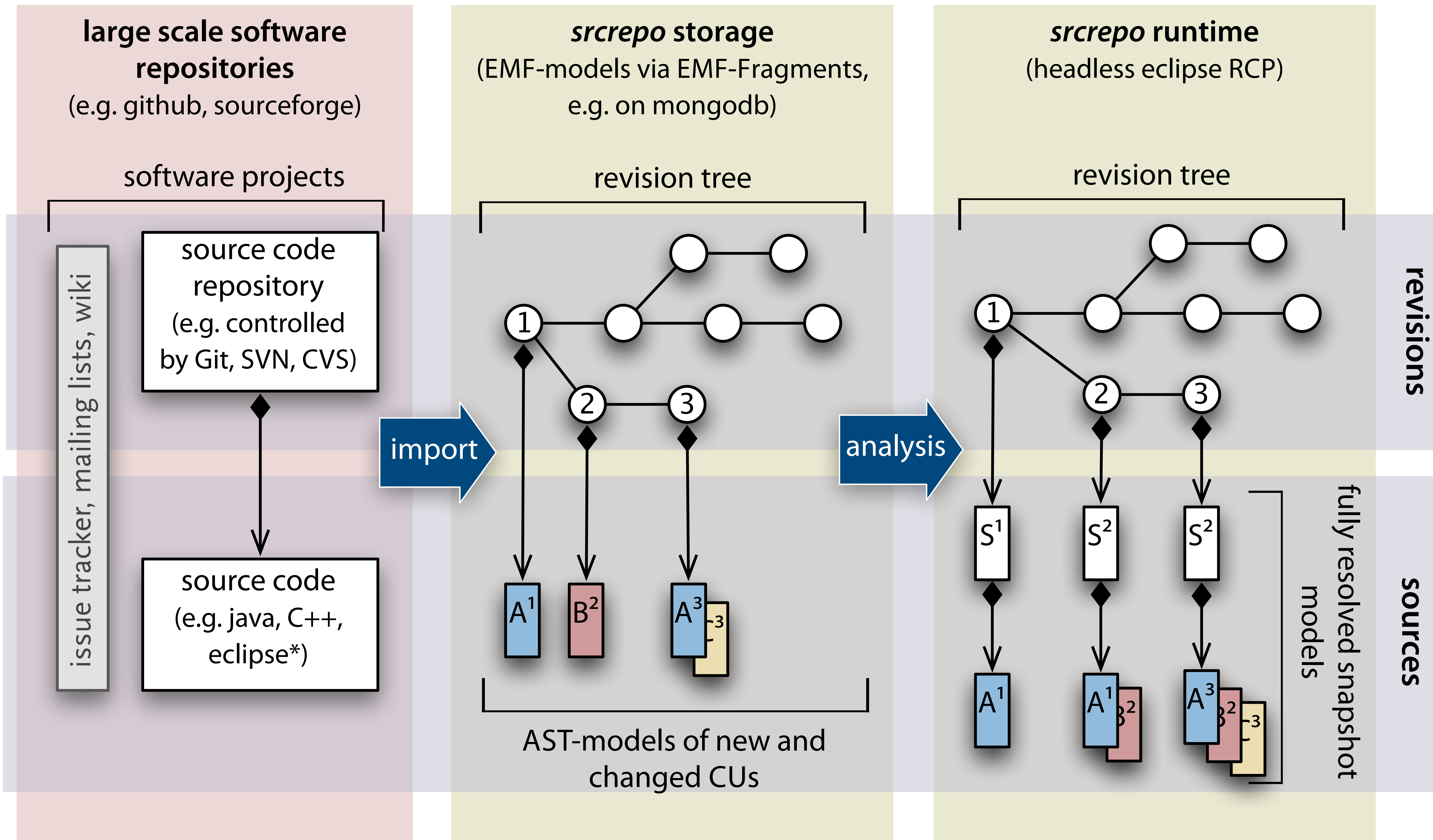
srcrepo – Components and Process

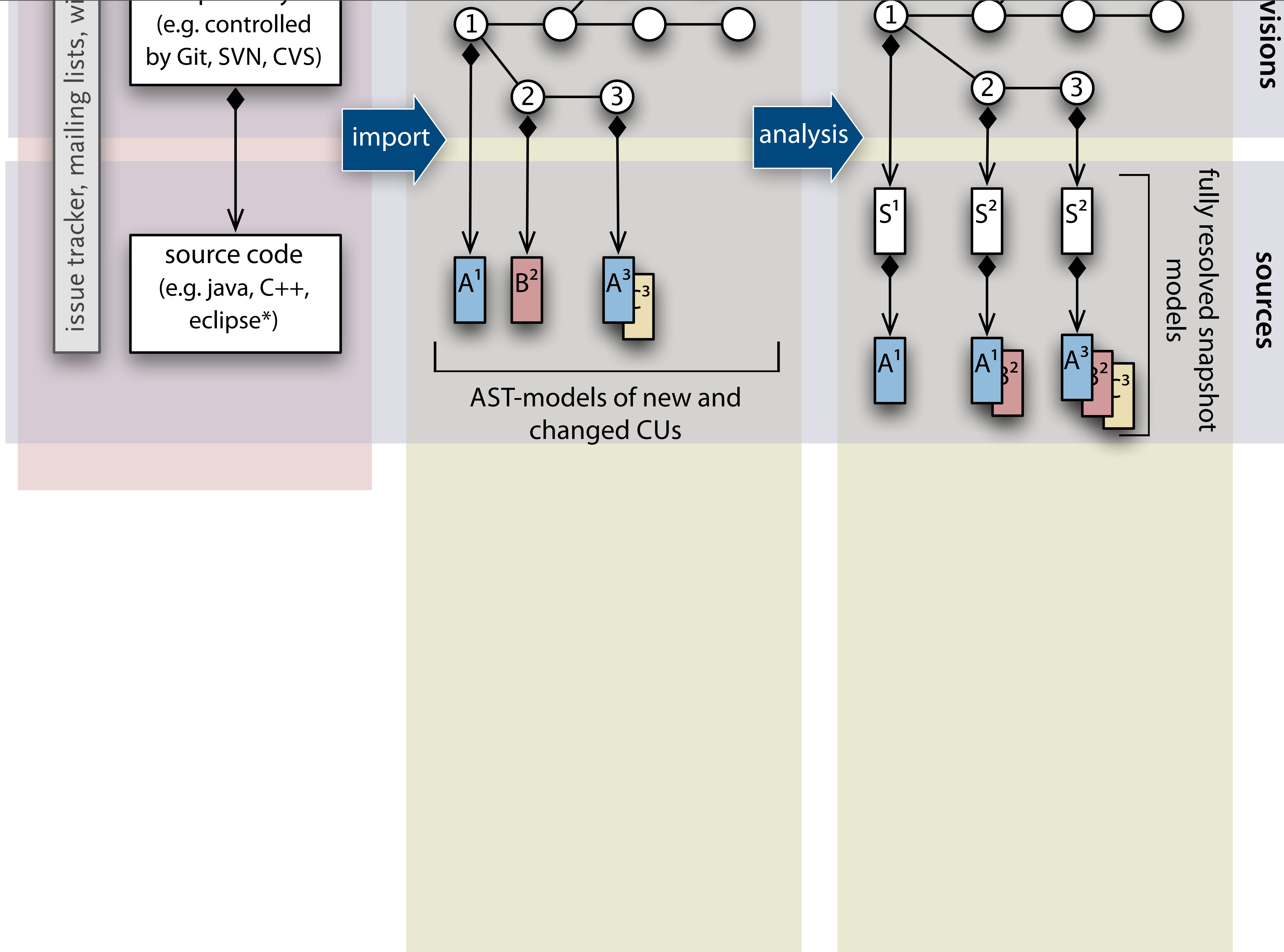


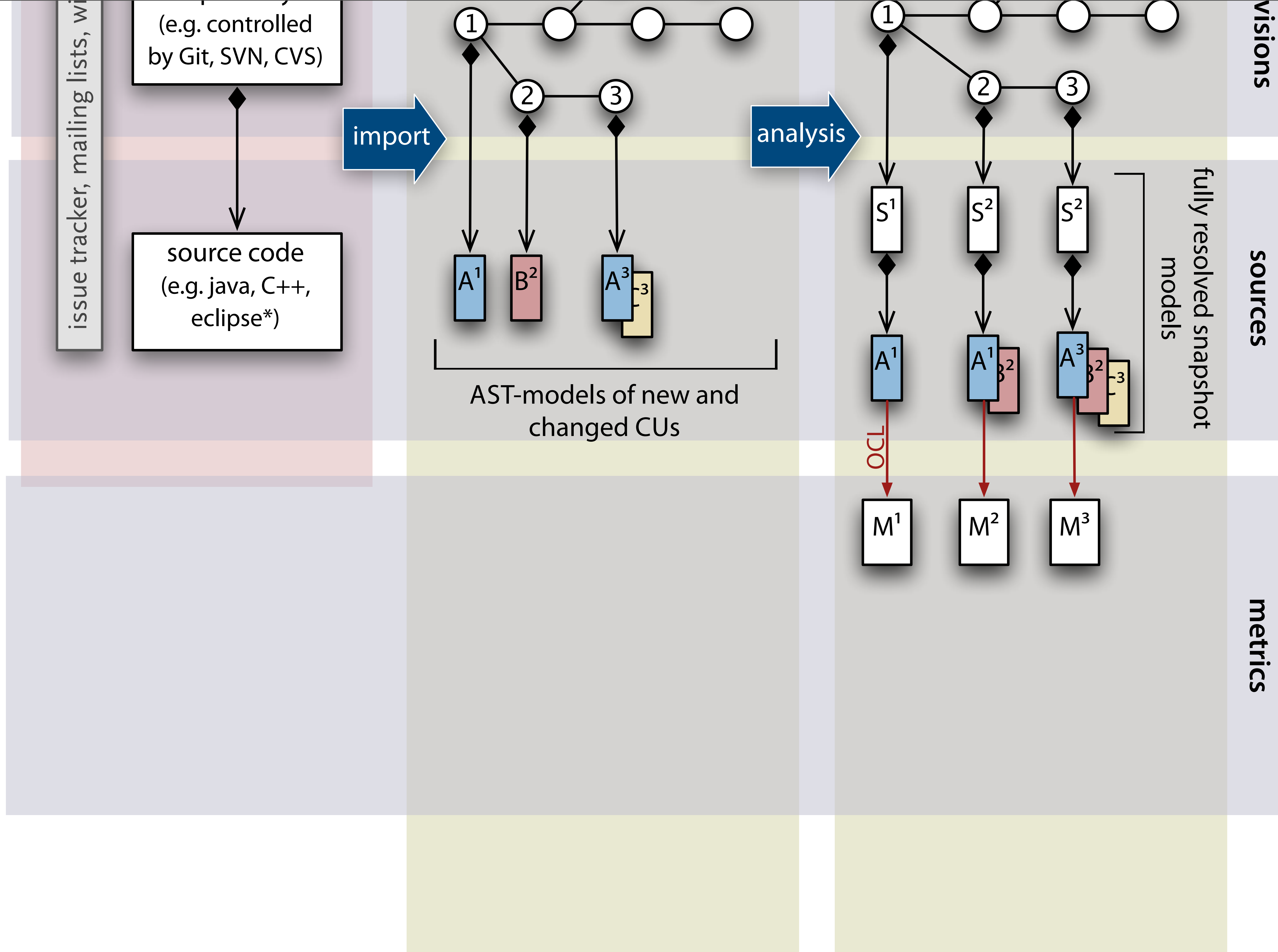
srcrepo – Components and Process



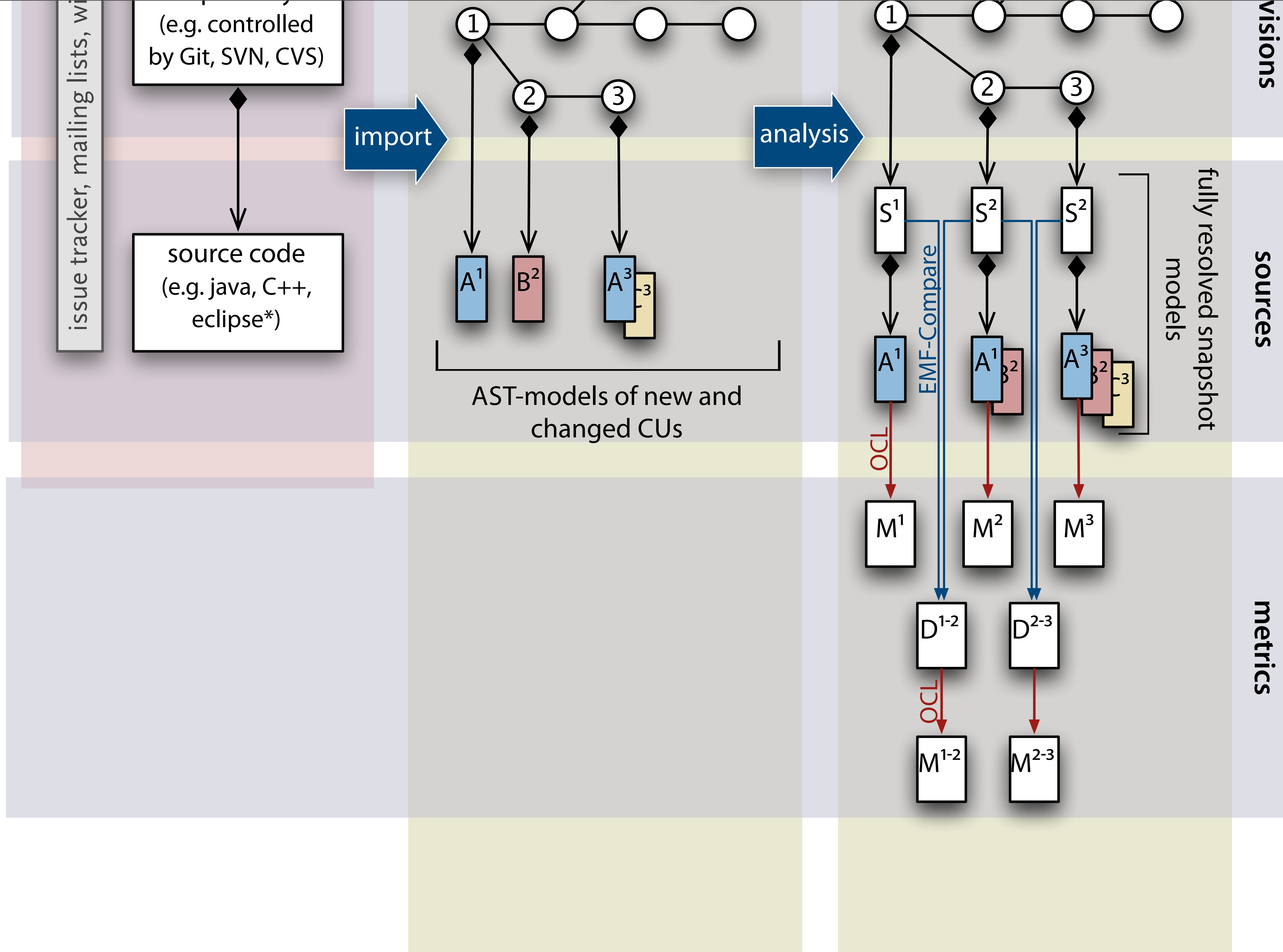
srcrepo – Components and Process

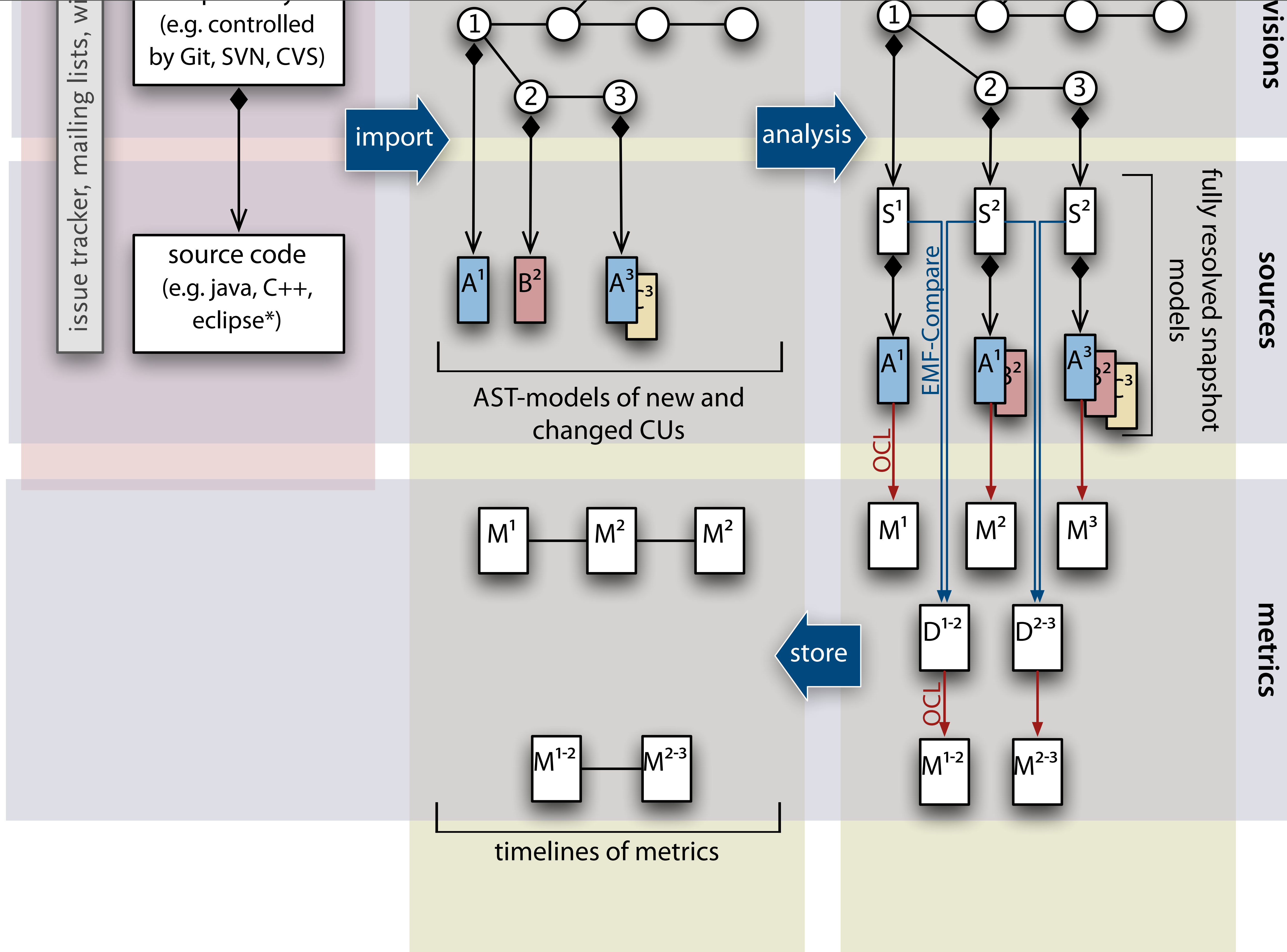


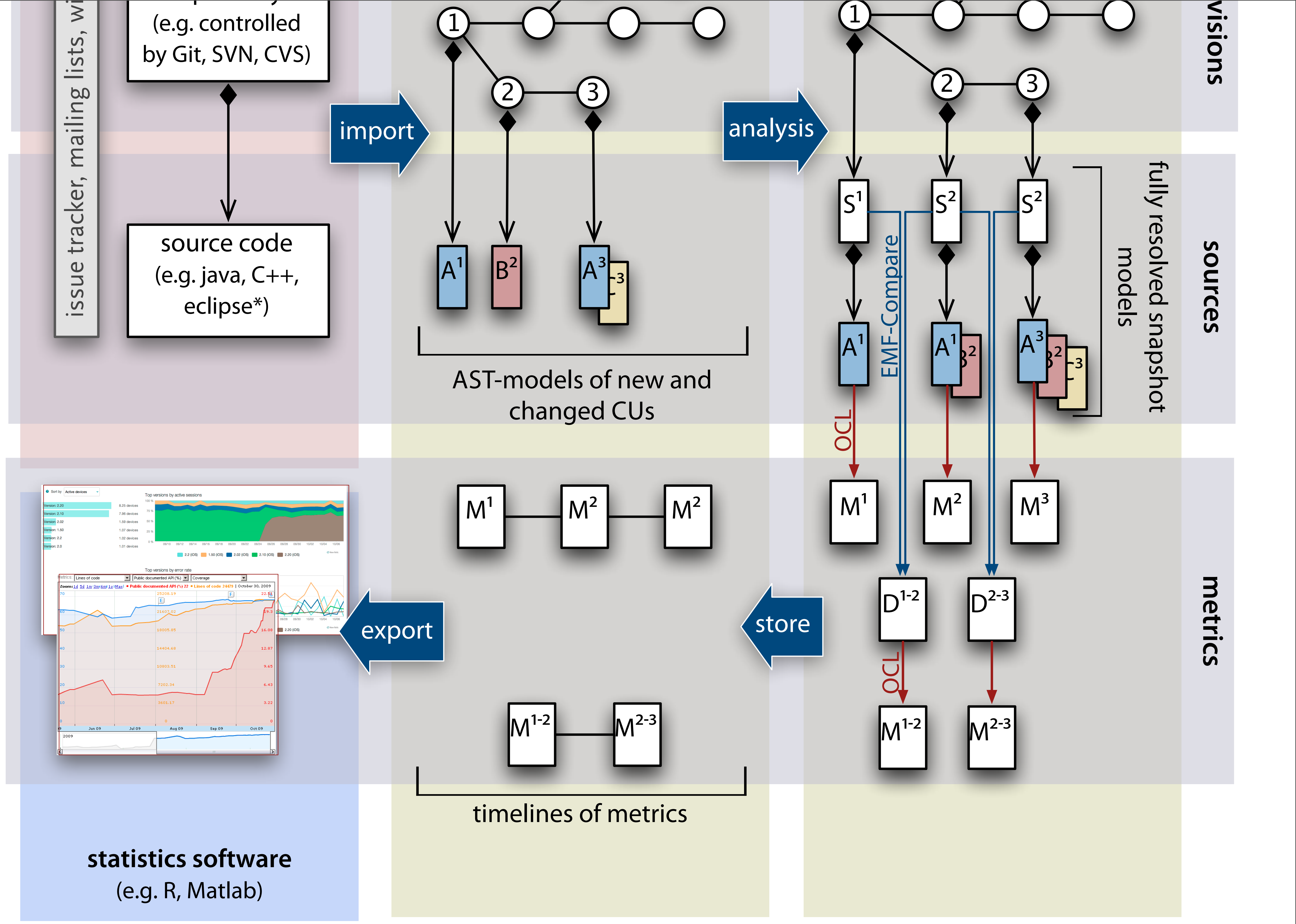

issue tracker, mailing lists, wi
(e.g. controlled
by Git, SVN, CVS)
source code
(e.g. java, C++,
eclipse*)
import
analysis
AST-models of new and
changed CUs
fully resolved snapshot
models
sources
visions



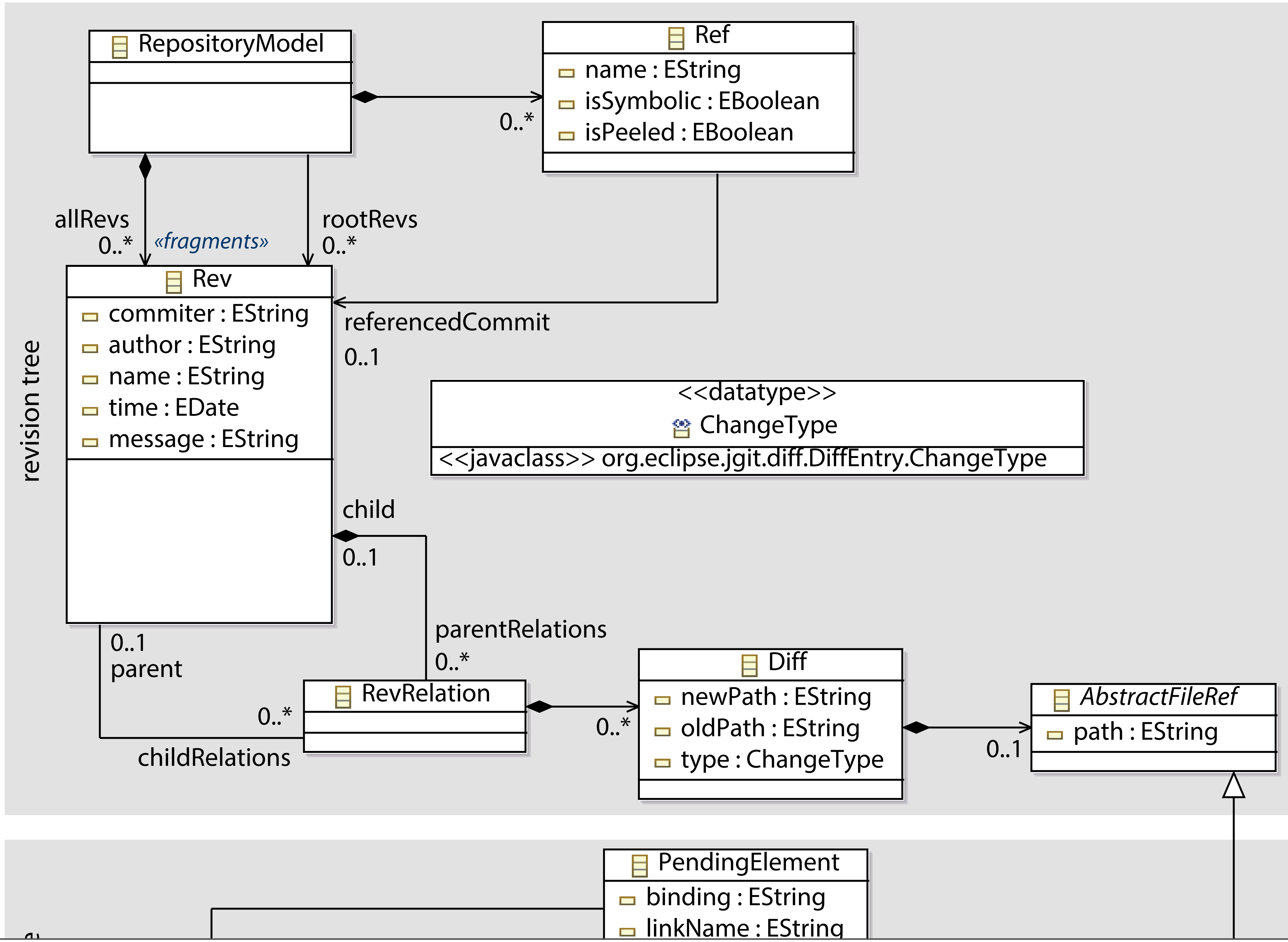
issue tracker, mailing lists, wi
(e.g. controlled
by Git, SVN, CVS)
source code
(e.g. java, C++,
eclipse*)
import
analysis
1
2
3
A1
B2
A3
AST-models of new and
changed CUs
S1
S2
S2
A1
A1
A3
OCL
M1
M2
M3
fully resolved snapshot
models
visions
sources
metrics

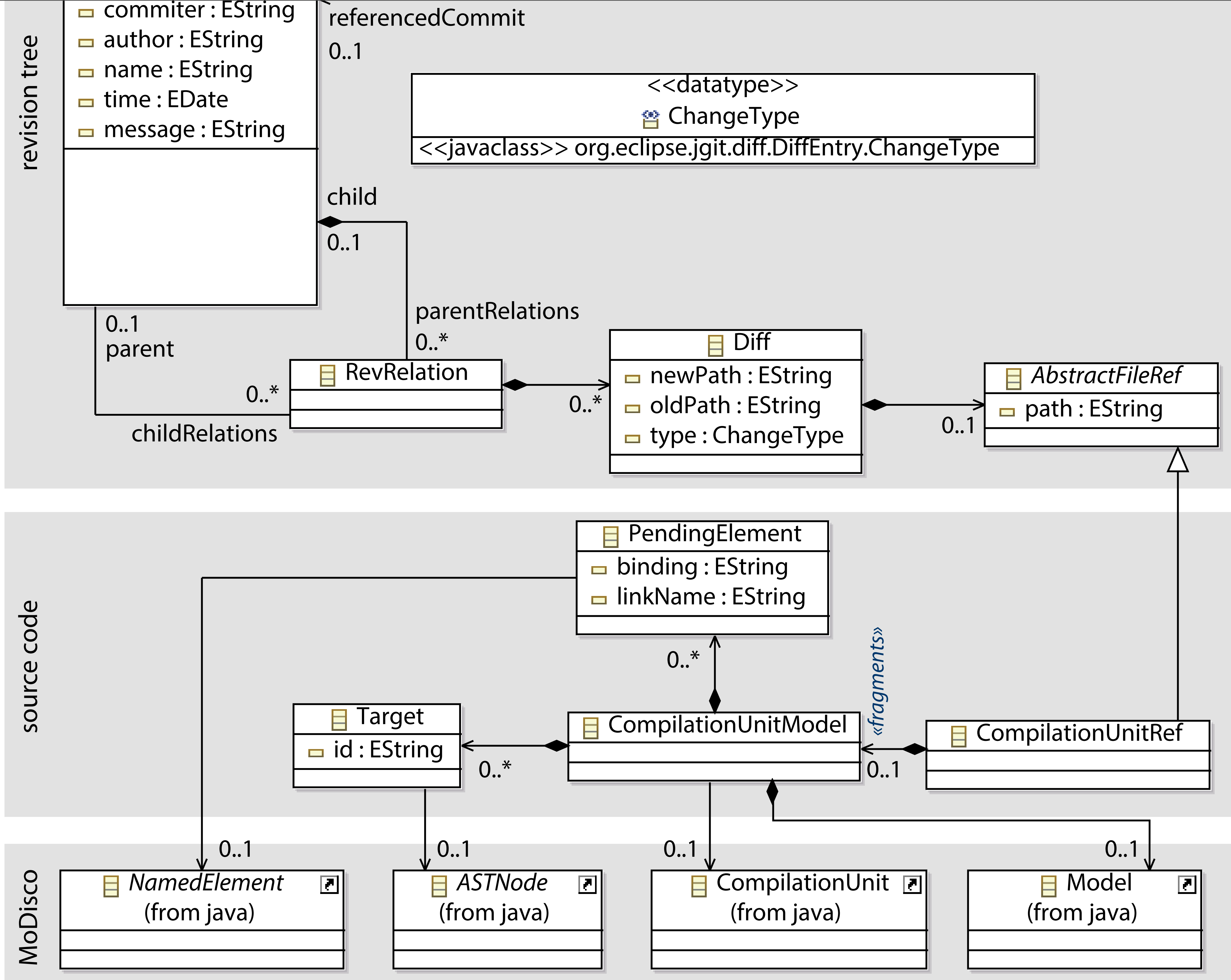

(e.g. controlled
by Git, SVN, CVS)
source code
(e.g. java, C++,
eclipse*)
import
analysis
AST-models of new and
changed CUs
fully resolved snapshot
models
EMF-Compare
OCL
OCL
sources
metrics


issue tracker, mailing lists, wi
(e.g. controlled
by Git, SVN, CVS)
source code
(e.g. java, C++,
eclipse*)
import
analysis
store
AST-models of new and
changed CUs
timelines of metrics
EMF-Compare
OCL
fully resolved snapshot
models
visions
sources
metrics


issue tracker, mailing lists, wi
(e.g. controlled
by Git, SVN, CVS)
source code
(e.g. java, C++,
eclipse*)
import
analysis
export
store
1
2
3
A1
B2
A3
AST-models of new and
changed CUs
S1
S2
S2
EMF-Compare
OCL
fully resolved snapshot
models
visions
sources
metrics
M1
M2
M2
M3
D1-2
D2-3
M1-2
M2-3
timelines of metrics
statistics software
(e.g. R, Matlab)

A Meta-Model for Source Code Repositories




revision tree
commiter : EString
author : EString
name : EString
time : EDate
message : EString
referencedCommit
0..1
<<datatype>>
ChangeType
<<javaclass>> org.eclipse.jgit.diff.DiffEntry.ChangeType
child
0..1
parentRelations
0..*
0..1
parent
0..*
childRelations
RevRelation
0..*
Diff
newPath : EString
oldPath : EString
type : ChangeType
0..1
AbstractFileRef
path : EString
source code
PendingElement
binding : EString
linkName : EString
0..*
«fragments»
Target
id : EString
0..*
CompilationUnitModel
0..1
CompilationUnitRef
MoDisco
0..1
0..1
0..1
0..1
NamedElement
(from java)
ASTNode
(from java)
CompilationUnit
(from java)
Model
(from java)

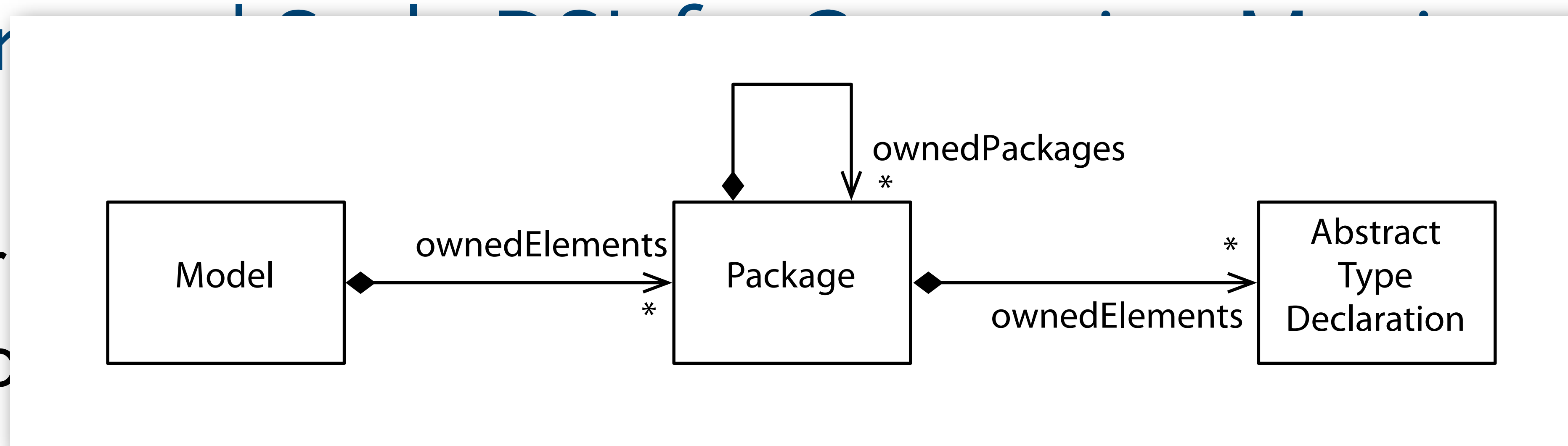
“Demo”

A OCL-like internal Scala DSL for Computing Metrics

- *OCL-like* internal Scala DSL analog to our internal Scala model transformation language [1]
- OCL collection operations mapped to Scala's higher-order fuctions [2]:

1. **L. George, A. Wider, M. Scheidgen:** *Type-Safe Model Transformation Languages as Internal DSLs in Scala*; Theory and Practice of Model Transformations - 5th International Conference, ICMT; 2012
2. **Filip Krikava**: *Enrichting EMF Models with Scala;* Slideshare

A OCL-like i



- *OCL-like* inter
  model transfo
- OCL collection operations mapped to Scala's higher-order fuctions [2]:

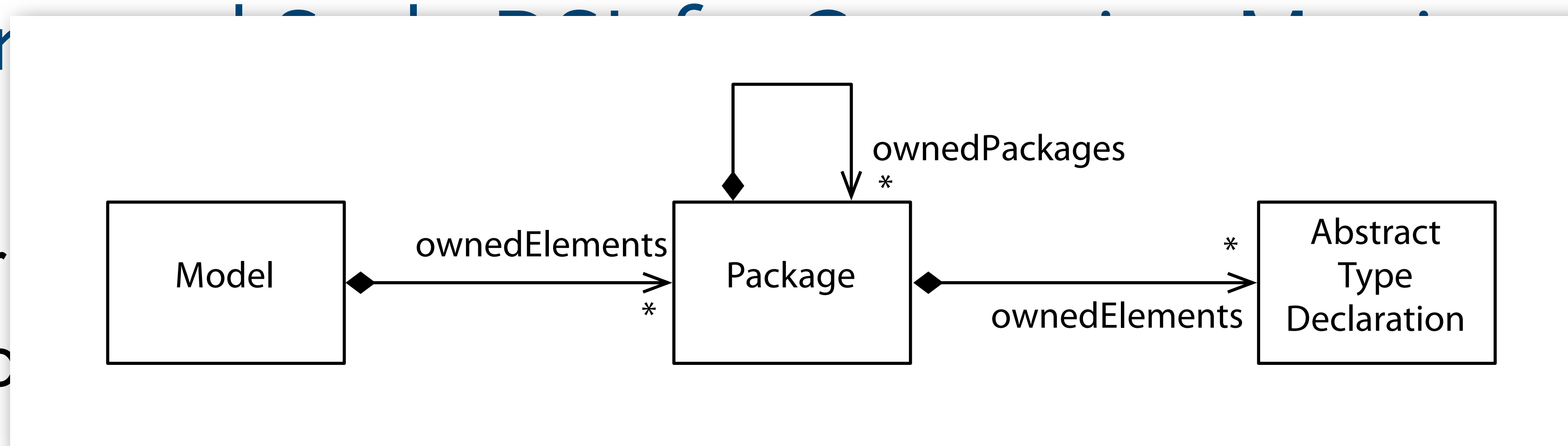
pure OCL

```
context Model:
    self.ownedElements->collect(p|p.ownedElements)->size
```


1. **L. George, A. Wider, M. Scheidgen:** *Type-Safe Model Transformation Languages as Internal DSLs in Scala*; Theory and Practice of Model Transformations - 5th International Conference, ICMT; 2012
2. **Filip Krikava**: *Enrichting EMF Models with Scala;* Slideshare

A OCL-like i

- *OCL-like* inter
  model transfo



- OCL collection operations mapped to Scala's higher-order fuctions [2]:

pure OCL

```
context Model:
    self.ownedElements->collect(p|p.ownedElements)->size
```

OCL-like expression in Scala

```
def numberOfFirstPackageLevelTypes(self: Model): Int =
    self.getOwnedElements().collect(p=>p.getOwnedElements()).size()
```


1. **L. George, A. Wider, M. Scheidgen:** *Type-Safe Model Transformation Languages as Internal DSLs in Scala*; Theory and Practice of Model Transformations - 5th International Conference, ICMT; 2012
2. **Filip Krikava**: *Enrichting EMF Models with Scala;* Slideshare

A OCL-like internal Scala DSL for Computing Metrics

- Extending OCL's collection operations:
  - convenience operations
  - closure
  - aggregation
  - execution

```
trait OclCollection[E] extends java.lang.Iterable[E]
{
  def size(): Int
  def first(): E
  def exists(predicate: (E) => Boolean): Boolean
  def forAll(predicate: (E) => Boolean): Boolean
  def select(predicate: (E) => Boolean): OclCollection[E]
  def reject(predicate: (E) => Boolean): OclCollection[E]
  def collect[R](expr: (E) => R): OclCollection[R]

  def selectOfType[T]:OclCollection[T]
  def collectNotNull[R](expr: (E) => R): OclCollection[R]
  def collectAll[R](expr: (E) => OclCollection[R]): OclCollection[R]

  def closure(expr: (E) => OclCollection[E]): OclCollection[E]
```

```scala
trait OclCollection[E] extends java.lang.Iterable[E]
{
  def size(): Int
  def first(): E
  def exists(predicate: (E) => Boolean): Boolean
  def forAll(predicate: (E) => Boolean): Boolean
  def select(predicate: (E) => Boolean): OclCollection[E]
  def reject(predicate: (E) => Boolean): OclCollection[E]
  def collect[R](expr: (E) => R): OclCollection[R]

  def selectOfType[T]:OclCollection[T]
  def collectNotNull[R](expr: (E) => R): OclCollection[R]
  def collectAll[R](expr: (E) => OclCollection[R]): OclCollection[R]

  def closure(expr: (E) => OclCollection[E]): OclCollection[E]

  def aggregate[R,I](expr: (E) => I, start: () => R, aggr: (R, I) => R): R
  def sum(expr: (E) => Double): Double
  def product(expr: (E) => Double): Double
  def max(expr: (E) => Double): Double
  def min(expr: (E) => Double): Double
  def stats(expr: (E) => Double): Stats

  def run(runnable: (E) => Unit): Unit
}
```

Complex Example: Average Weighted Methods per Class (WMC)

- WMC is the first CK-metric [1]. There different commonly used weights; here we use cyclomatic complexity.

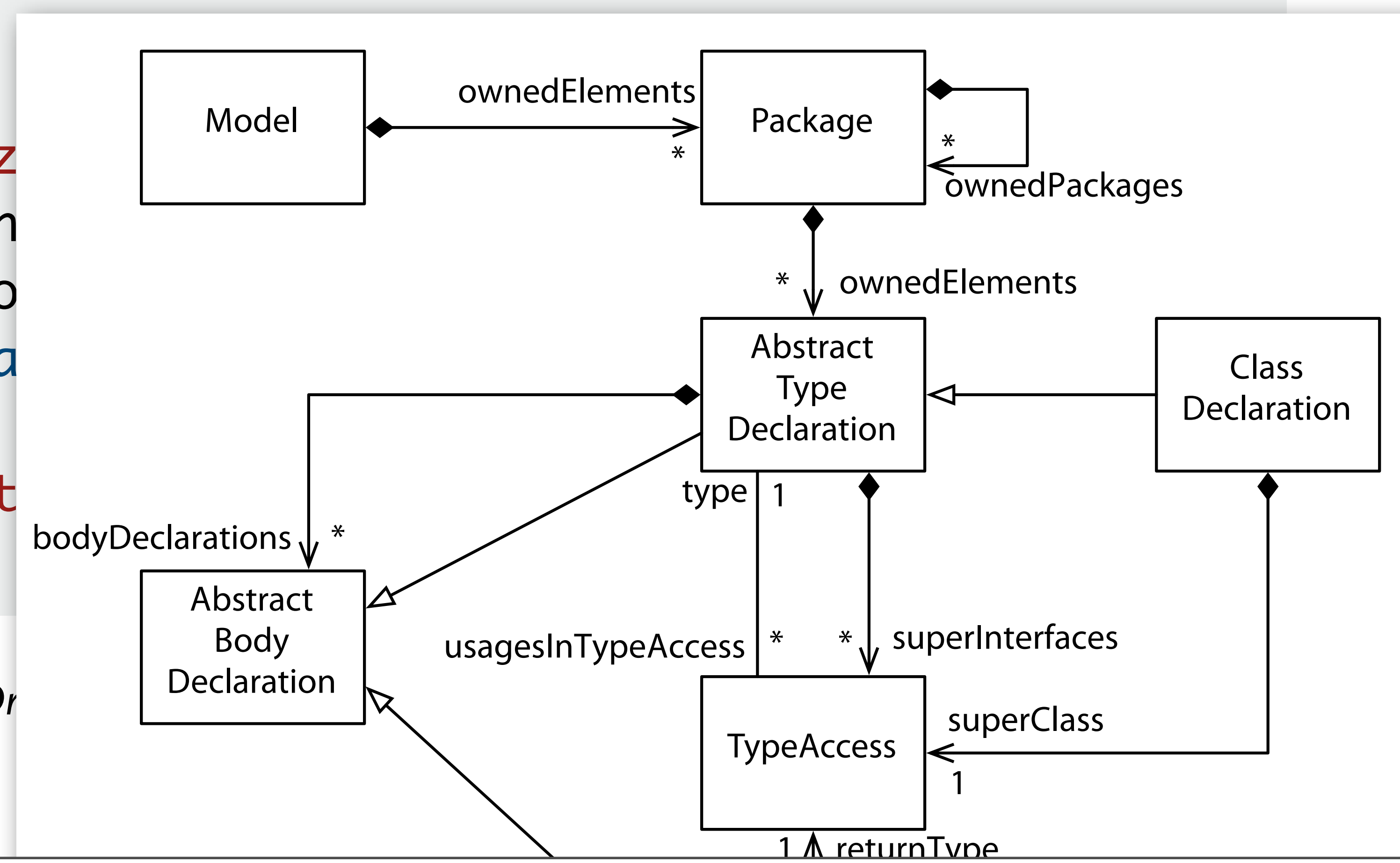
```
def classes(model:Model):OclCollection[ClassDeclaration] =
  model.getOwnedElements()
       .collectClosure(pkg=>pkg.getOwnedPackages())
       .collectAll(pkg=>pkg.getOwnedElements())
       .collectClosure(typeDcl=>
          typeDcl.getBodyDeclarations()
                 .selectOfType[ClassDeclaration])

def WMC(model:Model):Double =
  classes(model).stats(clazz=>
    clazz.getBodyDeclarations()
         .selectOfType[MethodDeclaration]()
         .sum(method=>cyclmaticComplexity(method))).average

def cyclomaticComplexity(method:MethodDeclaration):Int =
  ...
```

1. **S.R. Chidamber, C.F. Kemerer:** *A Metrics Suite for Object Oriented Design*; IEEE Transactions on Software Eng.; Vol.20/Nr.6/1994

Complex Example: Average Weighted Methods per Class (WMC)

- WMC is the first CK-metric [1]. There different commonly used weights; here we use cyclomatic complexity.

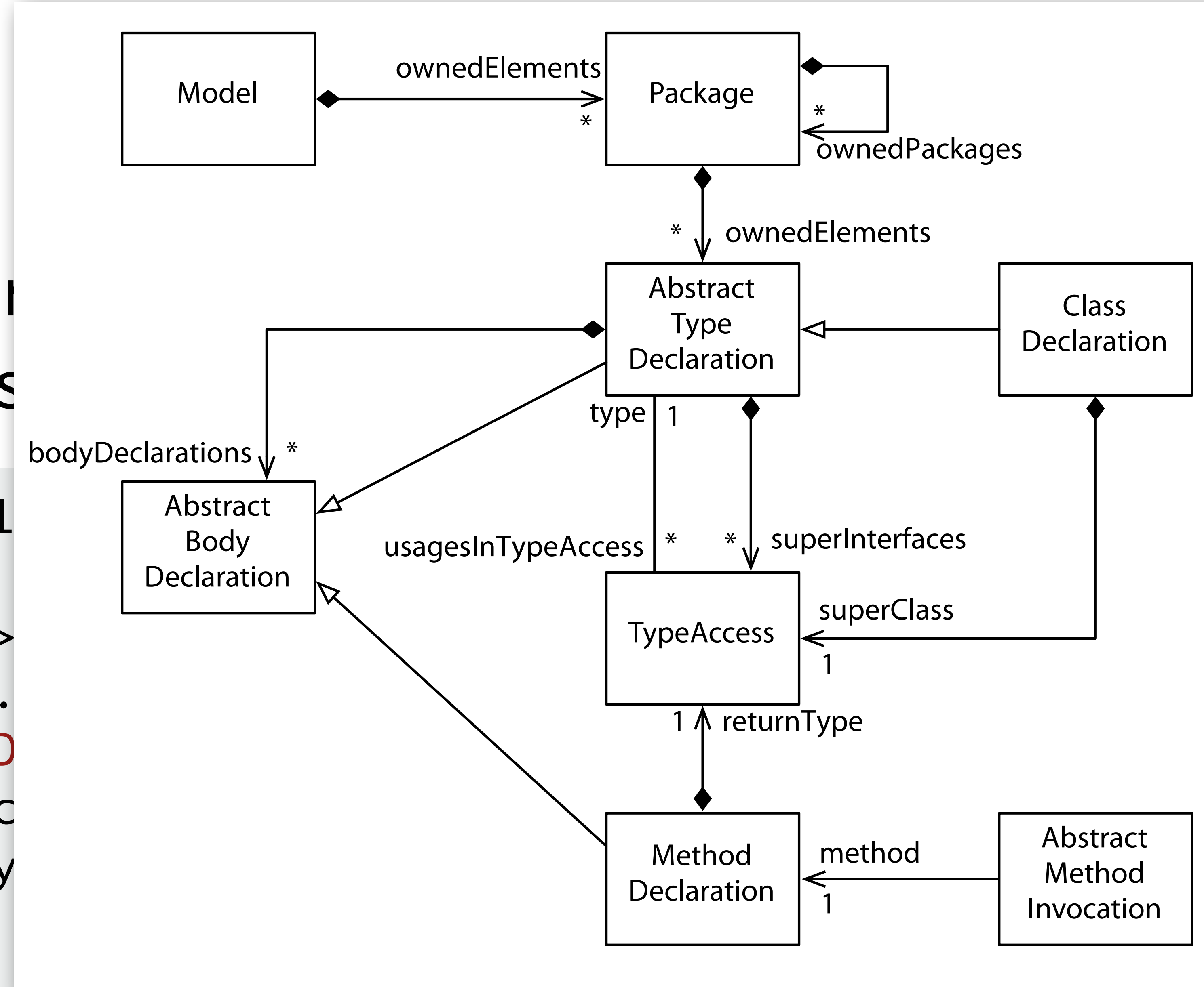
```
def classes(model:Model):OclCollection[ClassDeclaration] =
  model.getOwnedElements()
       .collectClosure(pkg=>pkg.getOwnedPackages())
       .collectAll(pkg=>pkg.getOwnedElements())
       .collectClosure(typeDcl=>
          typeDcl.getBodyDeclarations()
                 .selectOfType[ClassDeclaration])

def WMC(model:Model):Double
  classes(model).stats(clazz
    clazz.getBodyDeclaration
         .selectOfType[Metho
         .sum(method=>cyclma

def cyclomaticComplexity(met
  ...
```



1. **S.R. Chidamber, C.F. Kemerer:** *A Metrics Suite for Object Or*

Complex Example: Average

- WMC is the first CK-metr
used weights; here we us

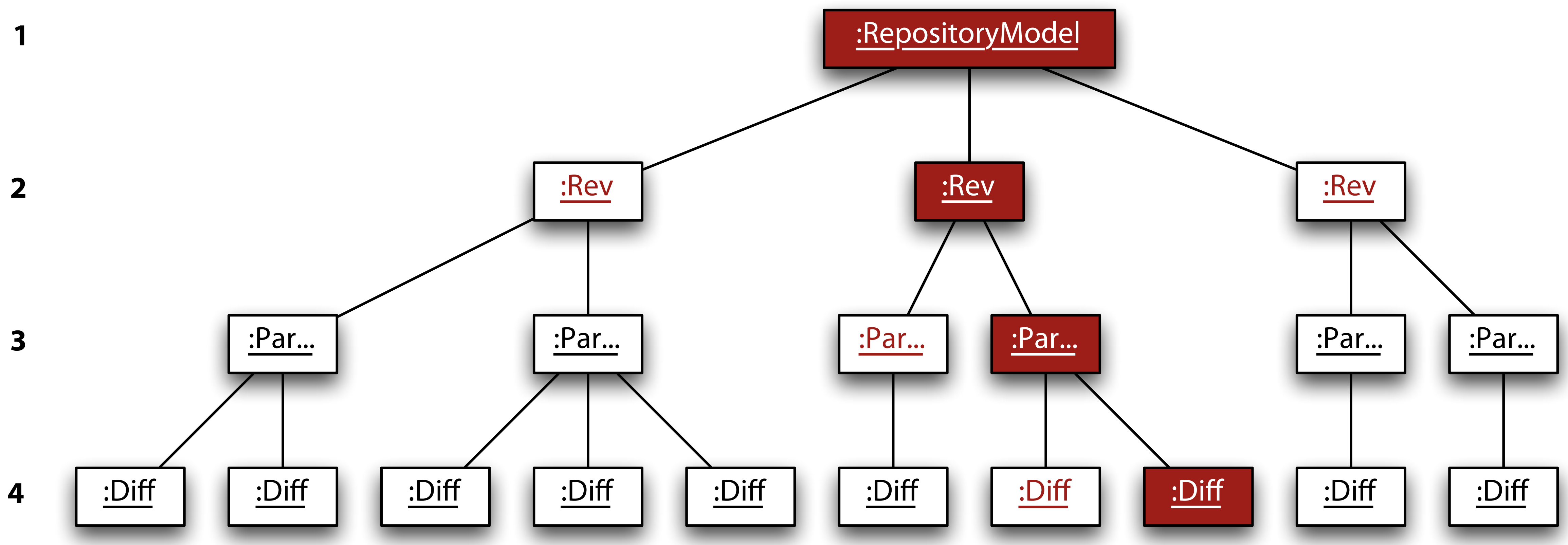



```
def classes(model:Model):Ocl
  model.getOwnedElements()
       .collectClosure(pkg=>
       .collectAll(pkg=>pkg.
       .collectClosure(typeD
          typeDcl.getBodyDec
                 .selectOfTy

def WMC(model:Model):Double
  classes(model).stats(clazz=>
    clazz.getBodyDeclarations()
         .selectOfType[MethodDeclaration]()
         .sum(method=>cyclmaticComplexity(method))).average

def cyclomaticComplexity(method:MethodDeclaration):Int =
  ...
```

1. **S.R. Chidamber, C.F. Kemerer:** *A Metrics Suite for Object Oriented Design*; IEEE Transactions on Software Eng.; Vol.20/Nr.6/1994

Implementation of the OCL-Collection Operations

- *Just in time* iterator-based implementation rather than straight forward aggregation of result collections.



Future Work, Remaining Problems, and Limitations

Scalability	Heterogeneity	Accessibility	Information Depth
■ very large compilation units ■ incremental snapshot creation ■ batching OCL execution ■ experiments with large scale repository (e.g. *git.eclipse.org*)	■ MoDisco for different programming languages ■ common metrics meta-model (e.g. OMG, KDM) ■ VCS abstraction and support for different VCS	■ relating results to software repository entities ■ persisting and exporting results	■ *diff-models* from comparison of compilation units

- Very large *compilation units* (CU): e.g. a 3 MB, 600 kLOC CU in *org.eclipse.emf*
  - tends to have lots of dependencies → changes often → makes problem even bigger
  - CUs are smallest common denominator between text-based VCS view and syntax-based AST view
  - smaller units require model-comparison or text-to-AST mappings
- Support for different programming languages: either abstraction, parallel meta-models, or mixed approach
  - MoDisco is *extendable*, but only Java support exists; other languages need to be implemented → parallel meta-models
  - A reasonable abstraction for multiple (or all) programming language probably does not exist.
  - A shared abstract meta-model that all language meta-models extends could be an sensible compromise.

Summary

- Overall model-based MSR with *srcrepo* works, but it still needs work.
- 80/20: Uncommonly large CUs are problematic and require complex additions to *srcrepo*. Ignored for now.
- Main goal *heterogeneity* is theoretically plausible, but requires lots of efforts to show practically. Not a matter of *if*, but of *how much*.
- Large experiments are still unfeasible due to lots of small issues rooted in the engineering complexity of the subject matter.