Lasaris Seminar

Software Repository Mining

Bruno Rossi

Introduction

- The problems involved in the increased computer power and complexity of software development started with the so-called *Software Crisis*
- Term first used during the first NATO Software Engineering Conference in 1968

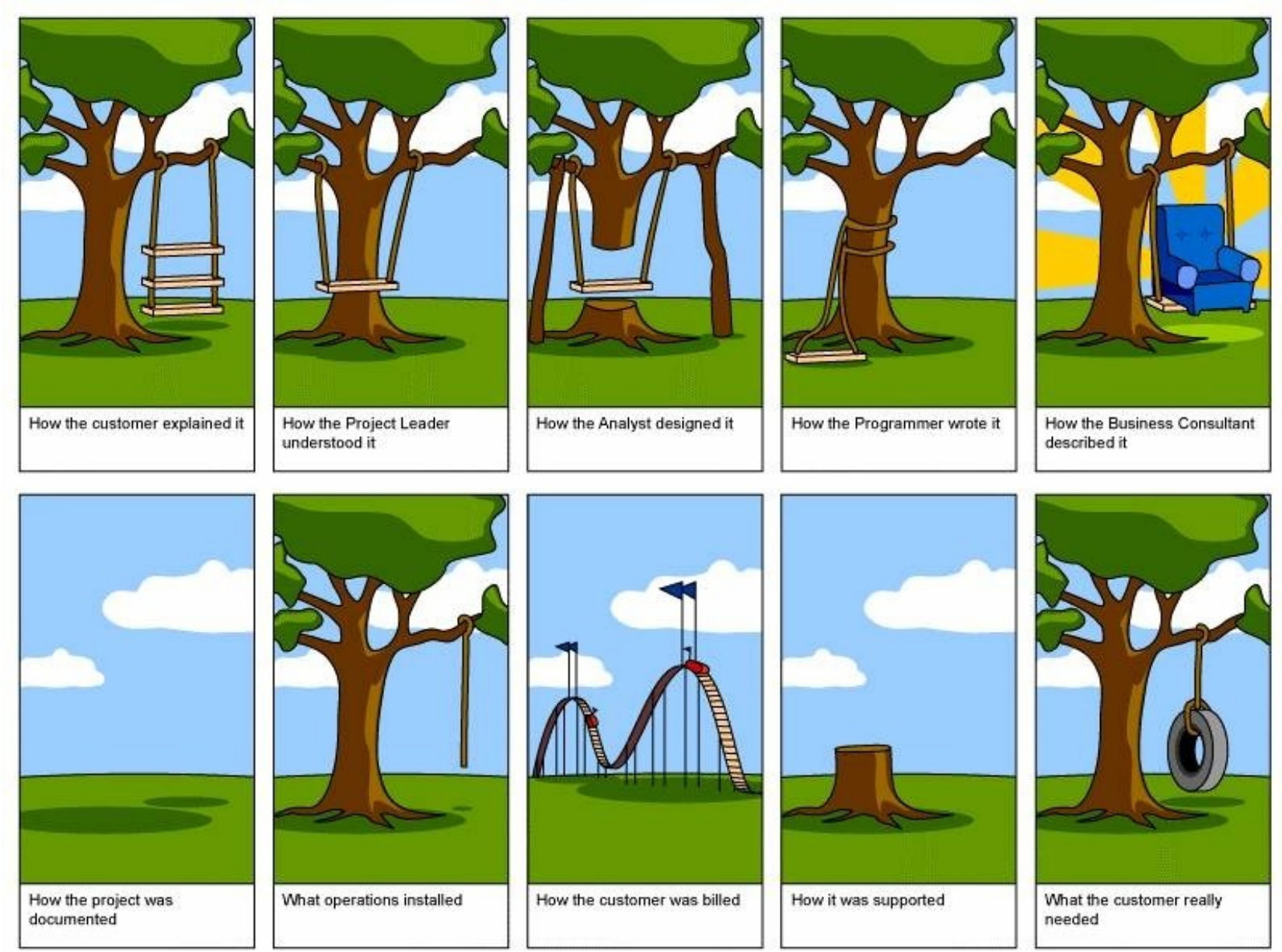
- Outcome: the necessity to develop processes, techniques and models to overcome the increasing complexity of projects

"To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem" - Edsger Dijkstra, The Humble Programmer (1972)

Introduction

- Different approaches to hamper the problem:
  - Tools (e.g. UML Modeling Tools)
  - Methodologies (e.g. Object Oriented programming, Formal Methods)
  - Development processes (e.g. SCRUM)
  - Frameworks (e.g. Spring)
- ..but still no *Silver Bullet*!

Still this is very often the case...



Source: http://www.projectcartoon.com/cartoon/2

Some of the most important software failures in history...

July 28, 1962 - Mariner I Space Probe
A bug in the flight control software causes the rocket to take a wrong trajectory. Mission control had to destroy the rocket.
Cause: *a formula written on paper that was not correctly ported to code*

1985-1986 – Therac-25
Six people were overexposed during radiation treatments for cancer by Canada's Therac-25 radiation therapy machine (3 later died).
Cause(s): *low quality assurance, non-testability of the software, race conditions, bad user interface, etc..*

1996 – Ariane 5
The rocket exploded on the main flight.
Cause: *code was reused from Ariane 4 without proper testing*

Why do we Mine Software Artifact Repositories?

- We mine **software repositories** to gather information about all **the technical** and **social relations** that happen within a software development project
- This can be useful to **predict future** effort based on the past history of a project or to **better understand** the software development process and products

- Typical usage of mining:
  - Detection of **type of changes** and **couplings**;
  - **Defect prediction**;
  - Analysis of **developers networks**;
  - **Visualizations** to help software evolution;

What is a software artifact repository?

It is a repository that is used during the development process: there can be many for managing versions, release history, bug fixes, etc...

*"**Anything that leaves a trail** about **any software development** or **maintenance activity**", Herraiz 2010*

Example, SVN:

Version and release identification

→ Systems assign identifiers automatically when a new

version is submitted to the system

Storage management

→ System stores the differences between versions rather than all the version code

Change history recording

→ Record reasons for version creation

Independent development

→ Only one version at a time may be checked out for change

→ Parallel working on different versions

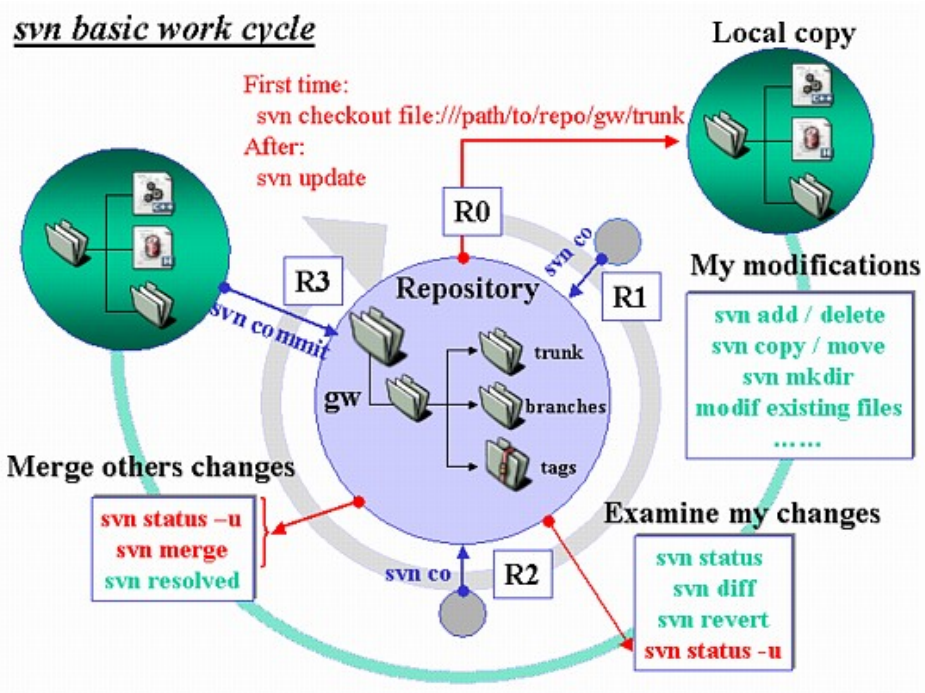


Image source: http://agata.in2p3.fr

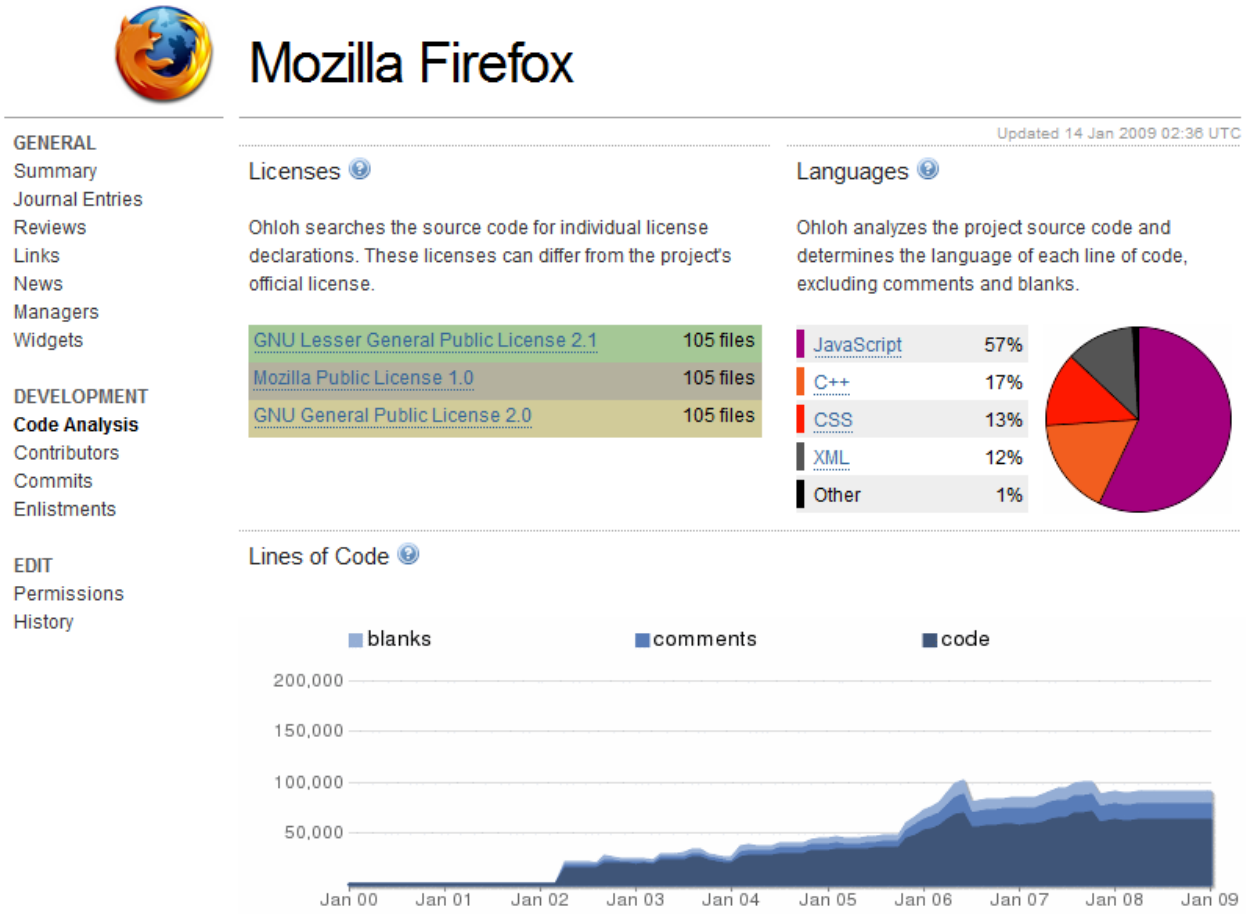
There is actually more...

- Issue trackers
- Mailing lists
- Building Systems
- Wikis
- Social network tools
- etc...

Data aggregators...

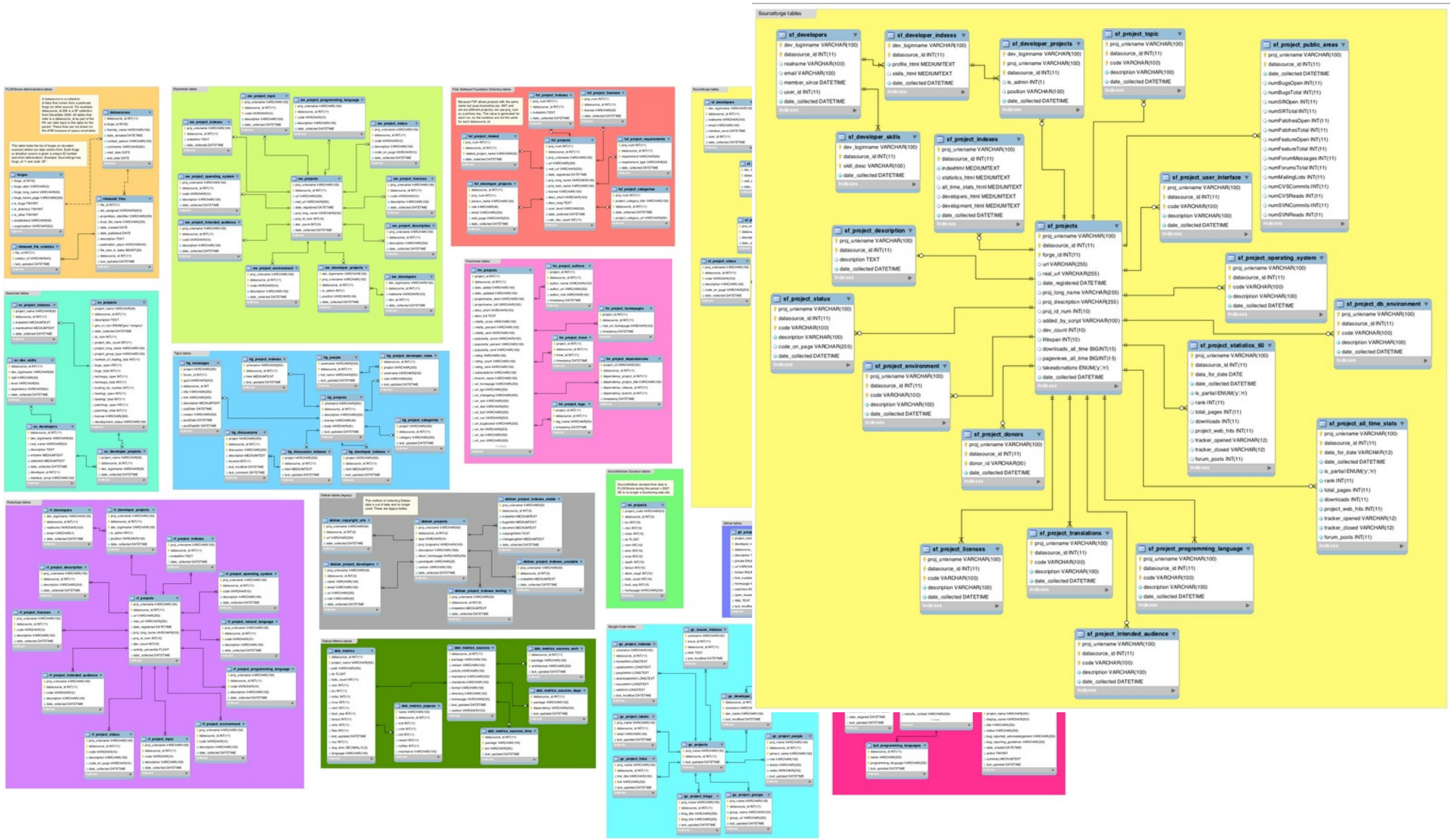
- An example is Ohloh (http://www.ohloh.net/):

You can access programmatically the data by parsing http responses in form of XML files



FLOSSmole

Collaborative collection and analysis of free/libre/open source project data



Mining Software Repositories (1/5)

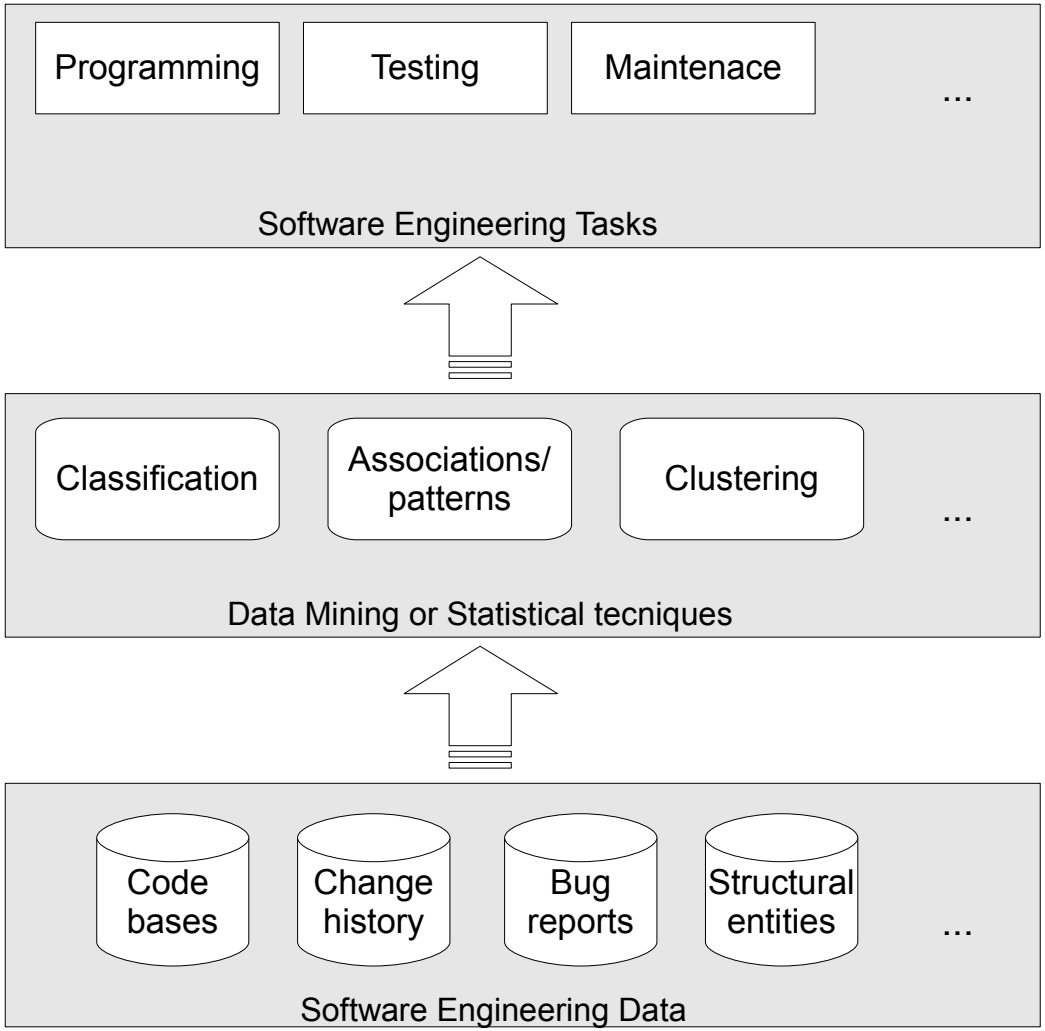
- When mining software repositories, we have typically 2 goals:

    1. **Analysing** & **understanding** the status and evolution...
    2. **Predicting** future evolution...

    ...of **social** and **technical** phenomena

- We can mine several repositories, bug tracking systems, versioning systems, message lists, wikis, etc...
- *Very often the analyses are non-trivial, like heuristics to associate developers across repositories*

Mining Software Repositories (2/5)



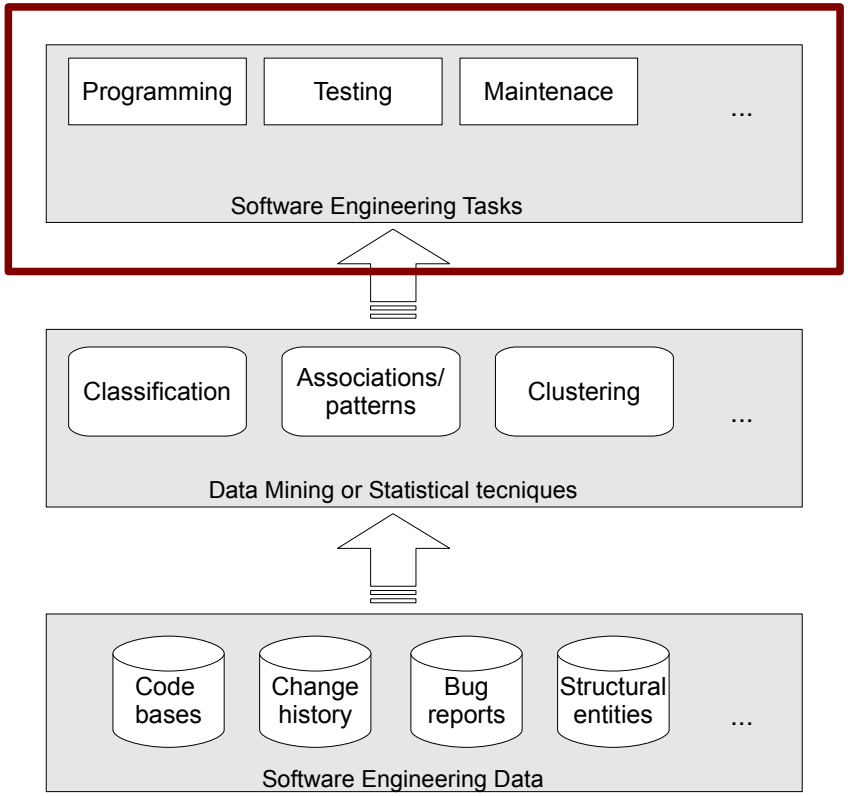
Software Engineering Domain of interest that researchers want to investigate

Datamining or statistical techniques used to analyze the original software engineering data

Sources of data for the analysis: from code bases that can be parsed, to change history, bug reports, etc...

Mining Software Repositories (3/5)

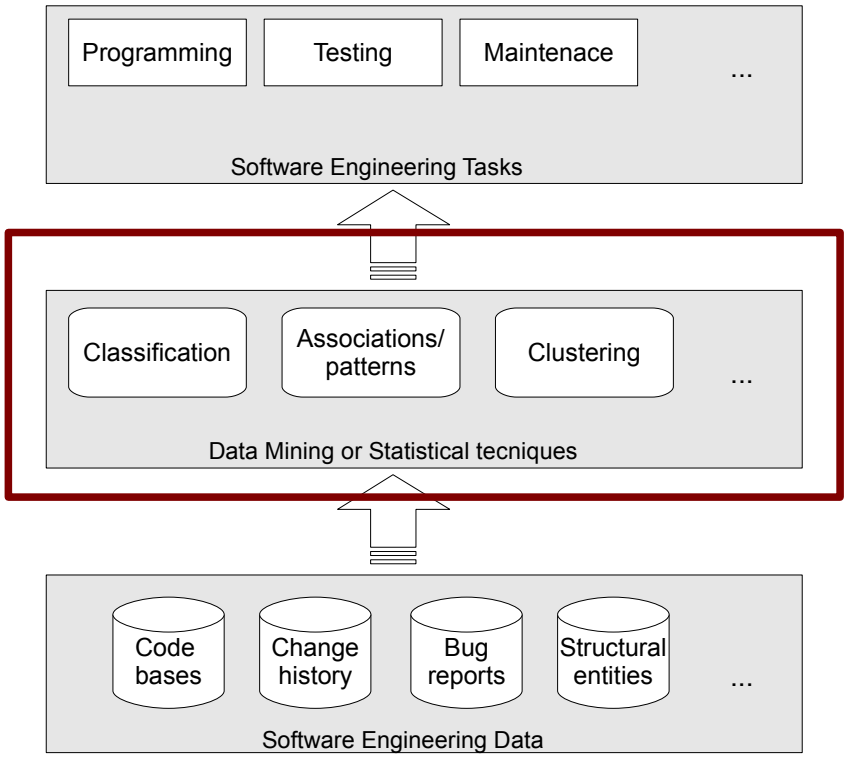
- Many software engineering practices can benefit from mining information from software repositories



- We can see the importance of software maintenance practices or software testing techniques
- We can empirically validate new techniques and tools
- Provide support for understandability of software
- ...

Mining Software Repositories (4/5)

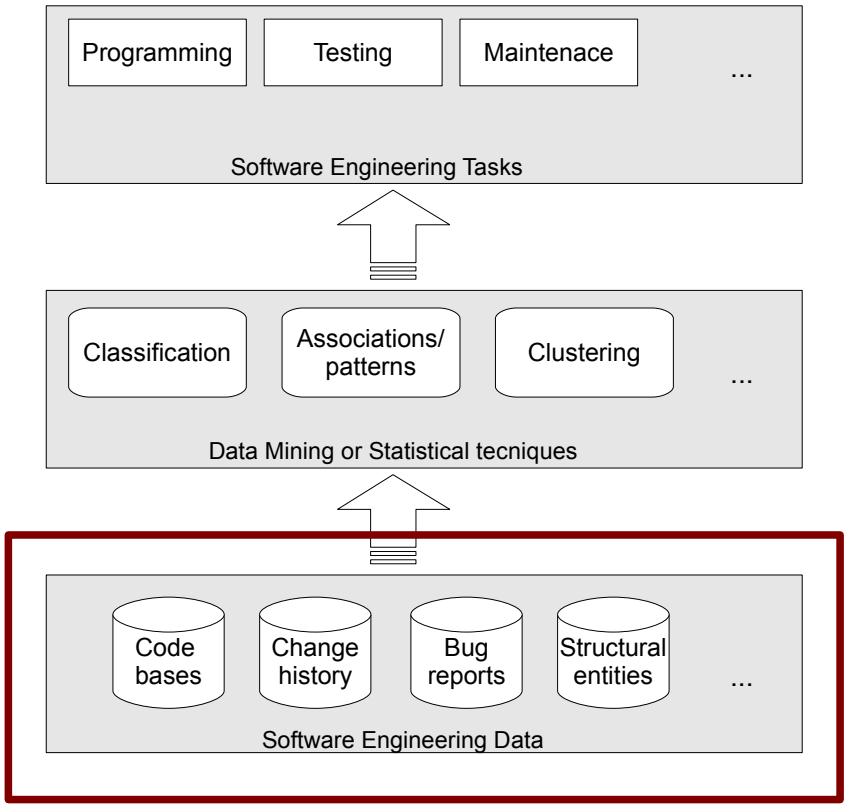
- The techniques are highly dependent on the dataset and on the research aim



- Descriptive statistics is very often used
- Regression and time series analysis are also commonly used
- Many times techniques for data classification and clustering are used

Mining Software Repositories (5/5)

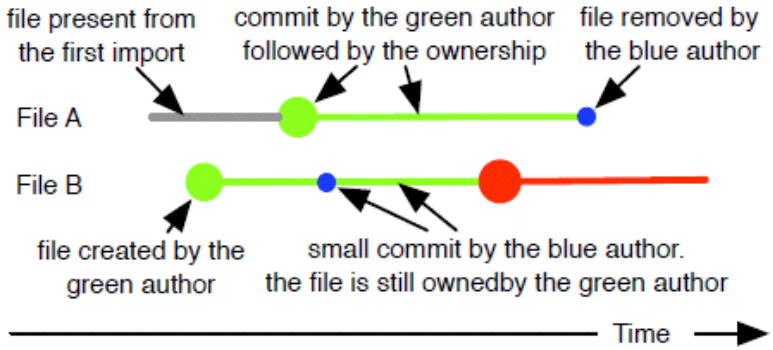
- We focus now in more detail on the data collection phase



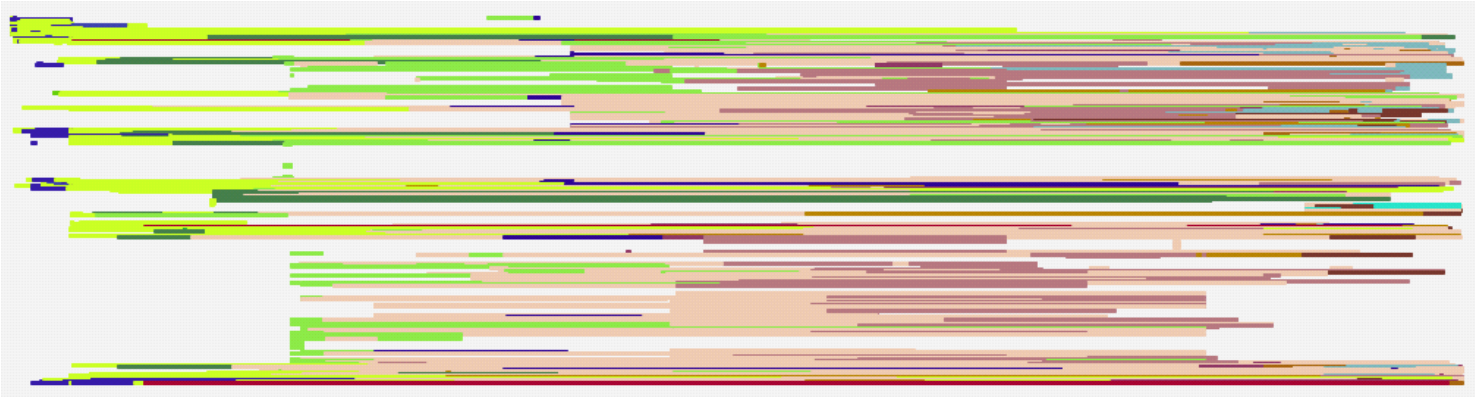
- Can be automated with scripts and tools
- Requires typically long time
- Needs to be carefully planned according to the type of data that needs to be collected
- Need to parse the data and process it
- How to link different repositories?

Examples in the Visualization Area (1/4)

- Determining Source Code Ownership:



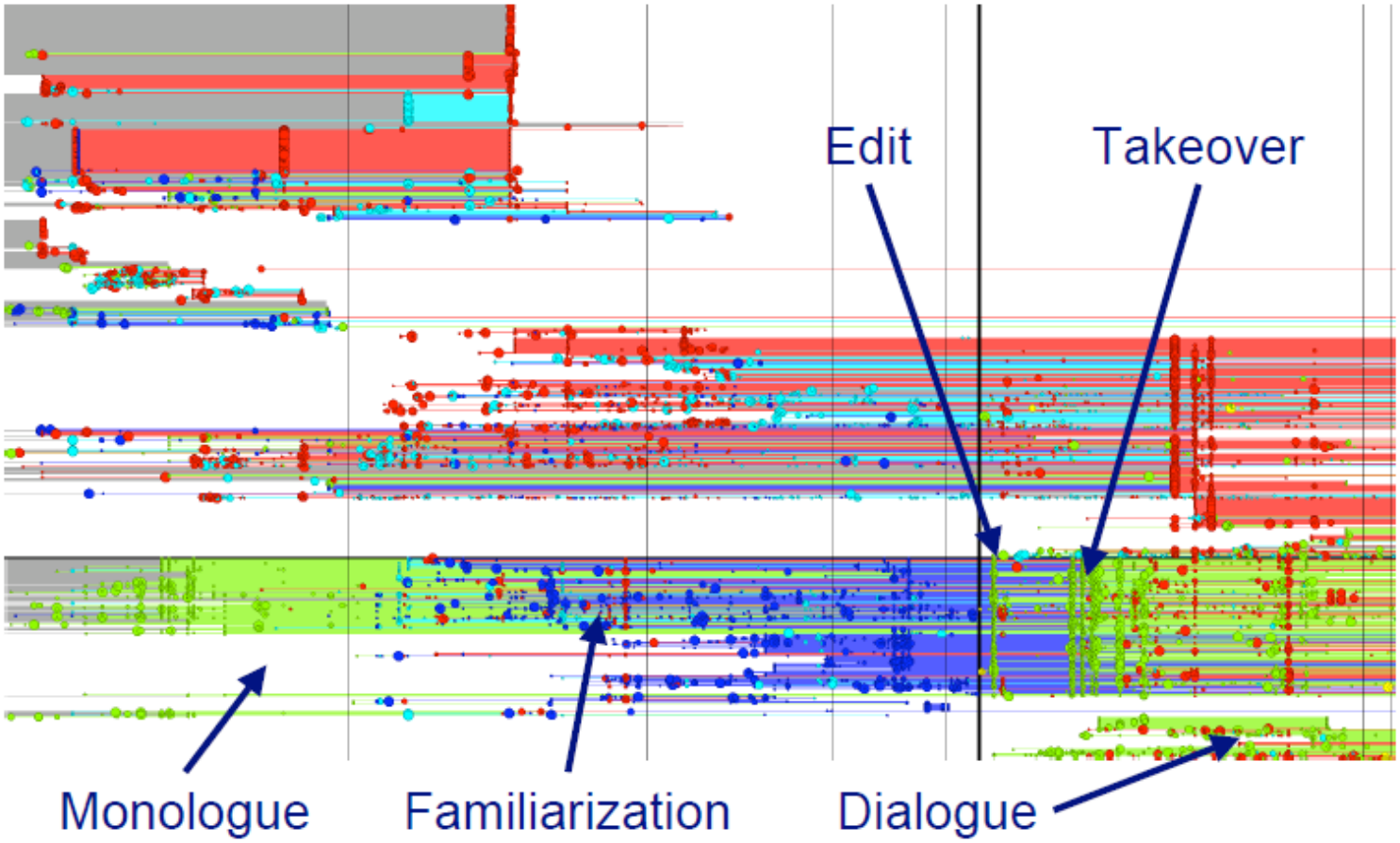
- Analyze SVN logs and provide a visualization of the evolution



M. Seeberger, A. Kuhn, T. Girba, e S. Ducasse, “Chronia: Visualizing How Developers Change Software Systems,” in Software Maintenance and Reengineering, European Conference on, pp. 345-348, 2006..

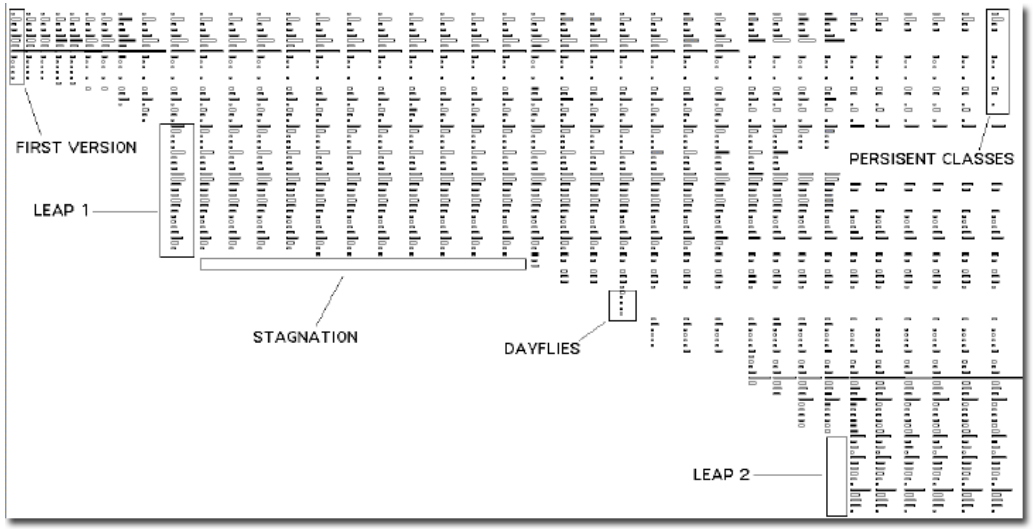
Examples in the Visualization Area (2/4)

- What about detecting patterns in development?



Picture from: http://www.iam.unibe.ch/~akuhn/s/2005-IWPSE-OwnershipMap.pdf
T. Girba, A. Kuhn, M. Seeberger, e S. Ducasse, “How Developers Drive Software Evolution,” in Eighth International Workshop on Principles of Software Evolution (IWPSE'05), pp. 113-122.

Examples in the Visualization Area (3/4)

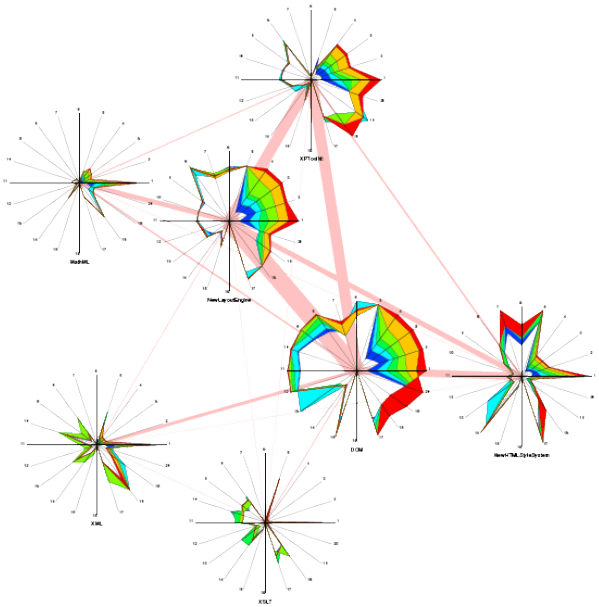


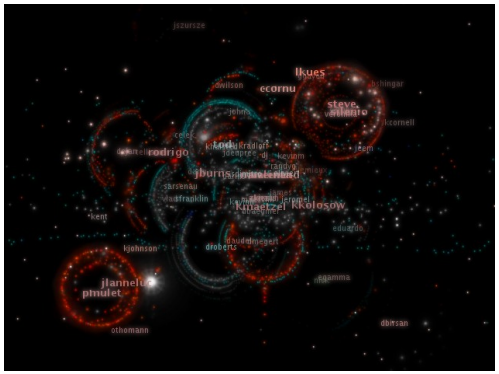
← Codecrawler

CodeCrawler application at the address: http://www.inf.unisi.ch/faculty/lanza/codecrawler.html

M. Pinzger, H. Gall, M. Fischer, and M. Lanza, “Visualizing Multiple Evolution Metrics,” in Proceedings of the 2005 ACM Symposium on Software Visualization, New York, NY, USA, 2005, pp. 67–75

→

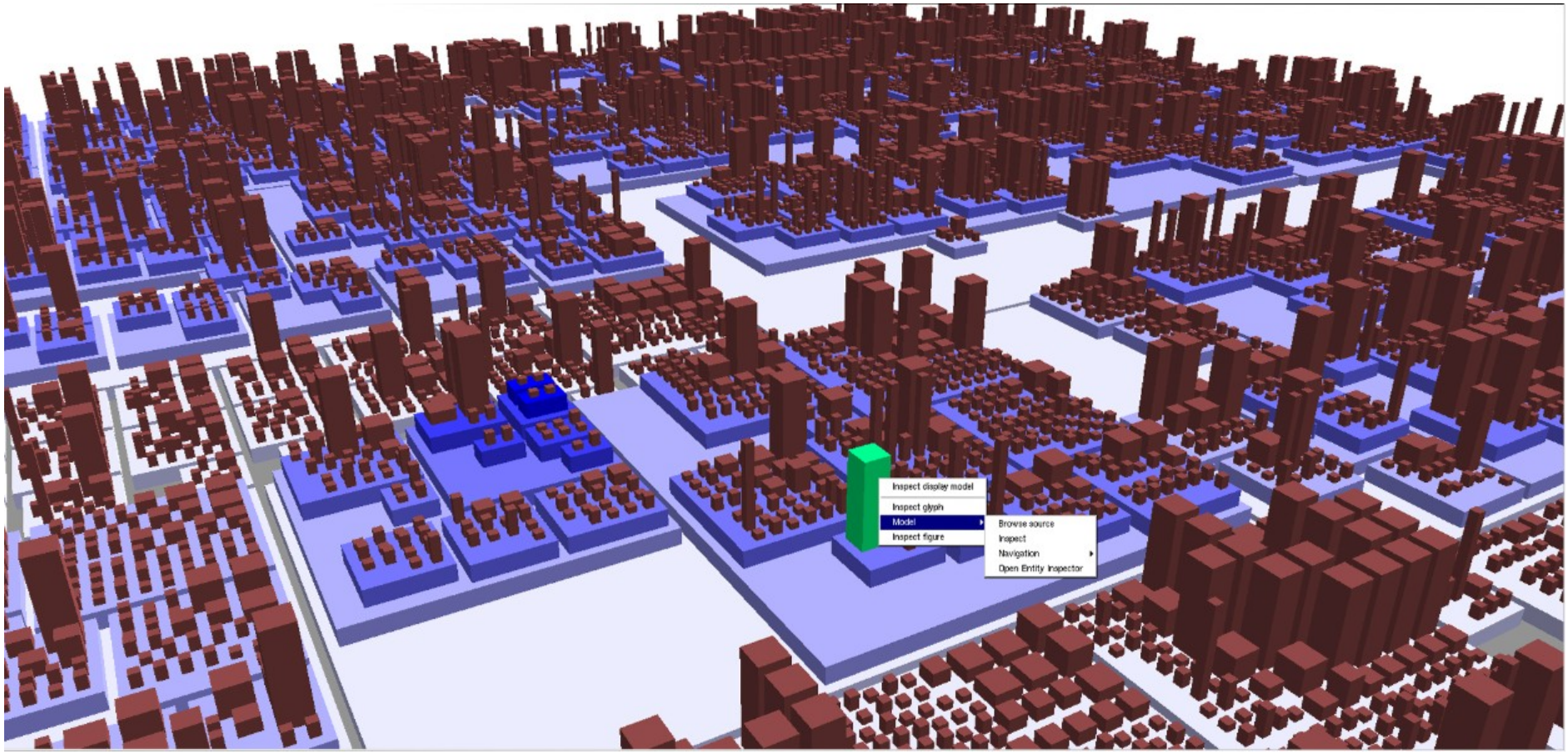




← *http://www.michaelogawa.com/code_swarm*

Examples in the Visualization Area (4/4)

- Software as cities by mining software repositories



R. Wettel, M. Lanza, and R. Robbes, “Software systems as cities: a controlled experiment,” in 2011 33rd International Conference on Software Engineering (ICSE), 2011, pp. 551–560.

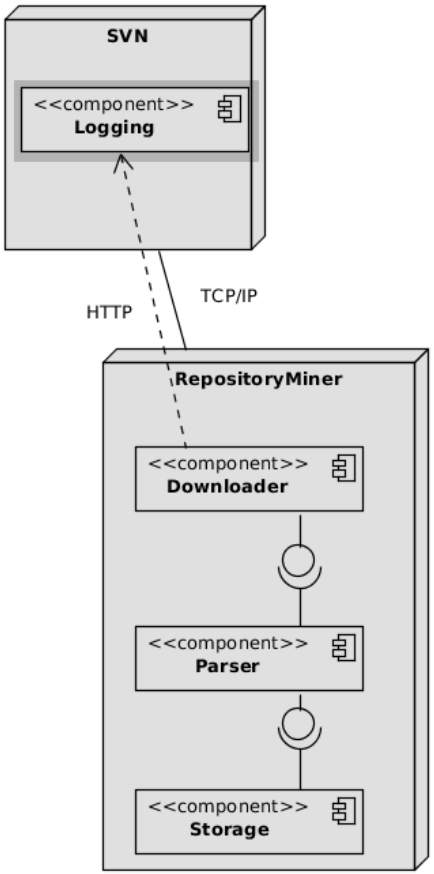
Implementing a Repository Miner

1. **Access repositories that already mine information** from the projects versioning system, bug-tracking systems, etc.. (e.g. *Ohloh*, *FLOSSMole*)
   - You can analyze a larger number of projects;
   - Problems of data granularity;
2. **Build your own "Miner"** and connect directly to the repositories/web pages of the project(s) you want to analyze
   - More fine-grained information;
   - Implementation / time consuming;
   - What about changes in technology? (e.g. introduction of git)

How to start with a simple Repository Miner

Our First Approach

- Quite simply our first approach can be the following



A sort of **pipes & filter** style that we use to download data and store
This is very often the simplest architecture – quite effective as it can be implemented by **chaining several scripts**

Working with SVN logs (1/3)

- The commands to work with SVN logs are the same used when managing a project
- e.g. to checkout a project

```
svn co https://notepad-plus.svn.sourceforge.net/svnroot/notepad-plus notepad-plus
```

- Information about logs

`svn log FILE` → shows you the log messages of every change made to a certain file or path (FILE can be either the URL to the repository or a local checked out path)

`svn log -v FILE` → will also show all the files that were changed by a certain commit

`svn log -v -q FILE` → will just show files that have been changed

`svn diff -r REV FILE` → will show you what changes were made to a file in a certain revision

`svn diff -r REV1:REV2 FILE` → will show you changes between different revisions in a range

`svn help log` & `svn help diff` → get additional options

Working with SVN logs (2/3)

- Parsing the history

```
------------------------------------------------------------------------
r593 | donho | 2009-12-23 20:23:20 +0100(mer, 23 dic 2009) | 2 lines

[BUG_FIXED] Fix status bar display bug that xml/html utf8 indicator makes status
 bar display ANSI instead of ANSI as UTF-8.
[UPDATE] Convert process thread module from ANSI to Unicode.
------------------------------------------------------------------------
r588 | donho | 2009-12-09 00:43:24 +0100(mer, 09 dic 2009) | 2 lines

[BUG_FIXED] Fix the Unicode localization file display incident.
[UPDATE] Update 3 localization files.
------------------------------------------------------------------------
r580 | donho | 2009-12-03 03:11:05 +0100(gio, 03 dic 2009) | 2 lines

[NEW] Add find result commands in the menu.
[NEW] Add DOS CodePage : CP437, CP737, CP850, CP852, CP855, CP857, CP858, CP860,
 CP861, CP863, CP865, CP866 and CP869.
------------------------------------------------------------------------
r575 | donho | 2009-11-30 02:31:55 +0100(lun, 30 nov 2009) | 4 lines

[RELEASE] Notepad++ v5.6 release.
[NEW_FEATURE] Add ISO_8859-1 to ISO_8859-16 encodings.
[BUG_FIXED] Fix last recent file list menu items localization encoding bug.
[BUG_FIXED] Fix last recent file number goes to zero issue.

------------------------------------------------------------------------
```

Release

User performing the commit

Comments about commit.
You can see that in this project they use a convention of defining a) *bug fixes*, b) *refactoring*, c) *new features*

Working with SVN logs (3/3)

- Get the changes for a file in a revision compared to the local one:

```
svn diff -r r580 EncodingMapper.cpp
```

```
Index: EncodingMapper.cpp
===================================================================
--- EncodingMapper.cpp  (revisione 580)
+++ EncodingMapper.cpp  (copia locale)
@@ -17,6 +17,7 @@

 #include "precompiledHeaders.h"
 #include "EncodingMapper.h"
+#include "Scintilla.h"

 // Don't change the order
 EncodingUnit encodings[] = {
@@ -128,6 +129,9 @@

 int EncodingMapper::getEncodingFromString(const char *encodingAlias) const
 {
+    if (isInListA(encodingAlias, "utf-8 utf8"))
+        return SC_CP_UTF8;
+
         size_t nbItem = sizeof(encodings)/sizeof(EncodingUnit);
         int enc = -1;

         for (size_t i = 0 ; i < nbItem ; i++)
```

Revision to compare with

Compared with a local copy

Added(+) or removed (-) lines of code

Using SVNKit (1/2)

- Very likely, it can make sense to use a library to parse SVN data, e.g.

Language	Repository	Library/binding
Java	SVN	WebKit http://svnkit.com/
Java	CVS	JCVS http://www.jcvs.org/
Java	Git	JavaGit http://javagit.sourceforge.net/
Python	SVN	Pysvn http://pysvn.tigris.org/docs/pysvn.html
Python	CVS	Pycvs http://rhaptos.org/downloads/python/pycvs/
Python	Git	GitPython http://gitorious.org/git-python

Using SVNKit (2/2)

- This simplifies the development of the parser:
- The SVNRepository class will represent your repository
- It will be typically created with

```
Repository repository =
SVNRepositoryFactory.create(SVNURL.parseURIEncoded(url));
```

- To load all revisions in a collection

```
logEntries = repository.log(new String[] {""}, null, startRevision, endRevision,
true, true);
```

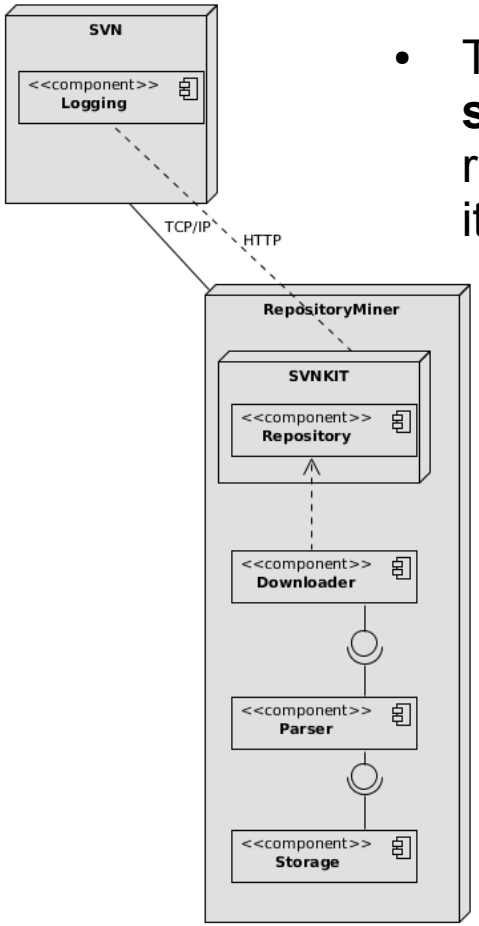
- You can then iterate the collection to get all the information about the SVN logs

- You should also check

`SVNRepository.diff(....)` for taking diffs

Two Main Problems – 1. Different Types of Repositories

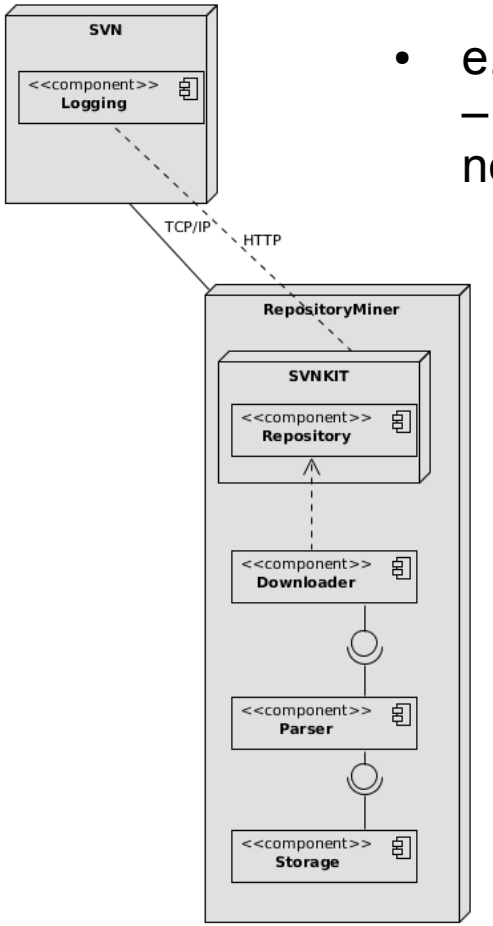
- So our initial Miner becomes more according to the following– but **how do we deal with the different types of repositories and data sources?**



- The approach of researchers has been to write **single-purpose** applications for each mining necessity – replicating the same for each repository (or at least part of it), just when needed

Two Main Problems – 2. Data Granularity

- **Which type of data should we store**? Every different type of analysis will have different needs



  - e.g. should we store metrics computed from the repository – but then we will need to query the repository again if needed

Current Problems in the Mining Software Repositories Area

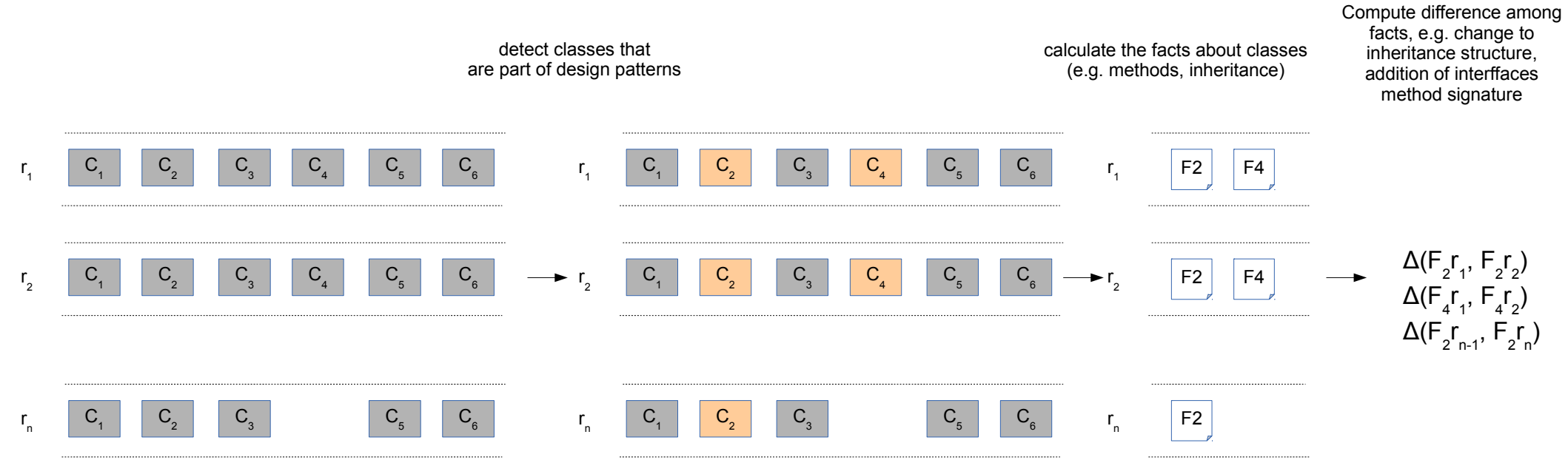
- Recently, we were working on a paper[1] for the **analysis of the evolution** of Design Patterns, to see whether the **project maturity stage** was impacting on the way **design patterns** are **modified** and introduced within projects
- We essentially repeated the approach from a paper from 2007[2], introducing the **maturity factor** as a variable
- We found the following problems:
  - Every researcher uses **different tools**
  - Very often **"reinventing the wheel"**
  - Difficult to have **replicability** of the results
  - Analyses can take huge amount of time – that is in the order of **days** to analyze **one version** of a large software project
  - What about **software archaeology**? As well as researchers not conserving the data

1 Rossi, B., Russo, B. *"Evolution of Design Patterns: a Replication Study", currently under review*

2 L. Aversano, G. Canfora, L. Cerulo, C. Del Grosso, and M. Di Penta, \An empirical study on the evolution of design patterns," in Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ser. ESEC-FSE '07. New York, NY, USA: ACM, 2007, pp. 385-394.

Exemplification

- Suppose now that we need to parse data from files to review **revision changes** to detect **structural changes** in source code for all the files that belong to a design pattern
- We need to check-out all the revisions and process for the presence of patterns



Exemplification

Using a similarity scoring algorithm N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S. T. Halkidis, "Design Pattern Detection Using Similarity Scoring", IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 896-909, November, 2006.

```
<?xml version="1.0" encoding="UTF-8"?>
<system>
	<pattern name="Factory Method">
		<instance>
			<role name="Creator" element="org.eclipse.jdt.core.dom.ASTNode" />
			<role name="FactoryMethod()"
			element="org.eclipse.jdt.core.dom.ASTNode::clone0(org.eclipse.jdt.core.dom.AST)
			se.jdt.core.dom.ASTNode" />
		</instance>
		<instance>
			<role name="Creator" element="org.eclipse.jdt.core.dom.ITypeBinding" />
			<role name="FactoryMethod()"
			element="org.eclipse.jdt.core.dom.ITypeBinding::getElementType():org.eclipse.jd
			.ITypeBinding" />
		</instance>
		<instance>
			<role name="Creator" element="org.eclipse.jdt.core.dom.VariableDeclaration" />
			<role name="FactoryMethod()"
			element="org.eclipse.jdt.core.dom.VariableDeclaration::getName():org.eclipse.jd
			.SimpleName" />
		</instance>
```

detect classes that are part of design patterns

Using the Javex fact extractor www.swag.uwaterloo.ca/javex/

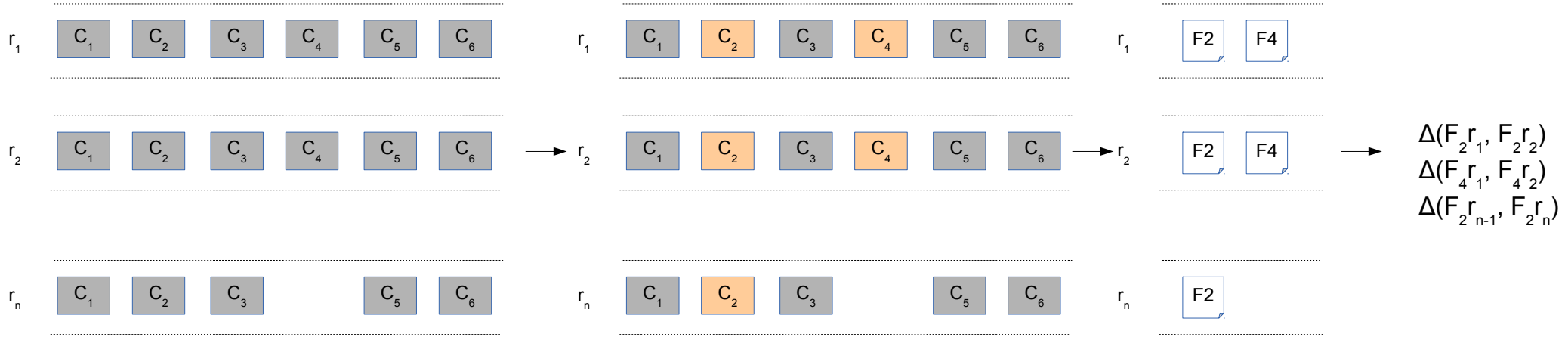
```
$INSTANCE C0x801c70 C
$INSTANCE C0x8015a0 C
$INSTANCE M0x801fd0 M
$INSTANCE M0x802160 M
$INSTANCE M0x801d20 M
$INSTANCE M0x801f00 M
$INSTANCE L0x801db0 P
contain D0x801b90 D0x801bf0
contain D0x801bf0 S0x801680
contain S0x801680 C0x8015a0
E258 C0x8015a0 C0x801c70
contain C0x8015a0 M0x801d20
contain M0x801d20 L0x801db0
E263 L0x801db0 C0x8015a0
E25 L0x801db0 M0x801d20
E183 M0x801d20 M0x801f00
contain C0x8015a0 M0x801fd0
E184 M0x801fd0 M0x802160
contain D0x801b90 D0x802250
contain D0x802250 C0x802080
contain D0x8022b0 D0x802310
contain D0x802310 C0x801c70
E256 M0x801d20 M0x801f00
contain C0x802080 M0x802160
contain C0x801c70 M0x801f00

FACT ATTRIBUTE :

D0x801b90 {label="/media/Data/Dropbox/Data/Paper_ReplicationFSE/Projects/Eclipse/packagesUI/ui_3.5.0/org/eclipse/jdt/internal"
D0x801bf0 {label="corext" file="./corext"}
S0x801680 {label="Corext.java" file="./Corext.java"}
C0x8015a0 {label="org.eclipse.jdt.internal.corext.Corext" file=. lineno=18 major=46 access=public synchronized=y}
M0x801d20 {label="<init>" file=. lineno=18 descriptor="() returns void" access=public maxstack=1 maxlocals=1}
L0x801db0 {label="this" no=0 file=. descriptor="org.eclipse.jdt.internal.corext.Corext" access=default}
(E263 L0x801db0 C0x8015a0) {file=. freq=1}
(E25 L0x801db0 M0x801d20) {file=. lineno=18 freq=1}
(E183 M0x801d20 M0x801f00) {file=. lineno=18 freq=1}
M0x801fd0 {label="getPluginId" file=. lineno=21 descriptor="() returns java.lang.String" access=public static=y maxstack=1}
(E184 M0x801fd0 M0x802160) {file=. lineno=21 freq=1}
D0x802250 {label="ui" file="./ui"}
C0x802080 {label="org.eclipse.jdt.internal.ui.JavaPlugin" file=. access=default}
D0x8022b0 {label="java" file="./java"}
D0x802310 {label="lang" file="./lang"}
C0x801c70 {label="java.lang.Object" file=. access=default}
(E256 M0x801d20 M0x801f00) {file=. lineno=18 freq=1}
M0x802160 {label="getPluginId" file=. descriptor="() returns java.lang.String" access=default}
M0x801f00 {label="<init>" file=. descriptor="() returns void" access=default}
```

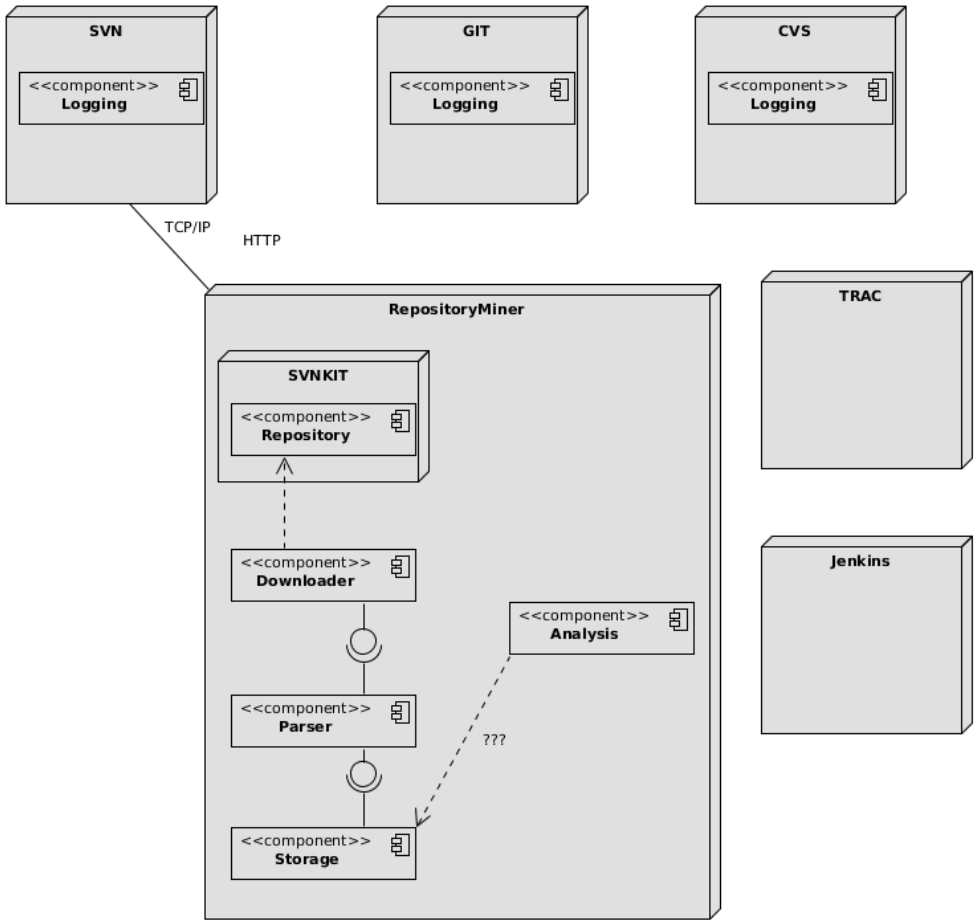
calculate the facts about classes (e.g. methods, inheritance)

Compute difference among facts, e.g. change to inheritance structure, addition of interffaces method signature



The Current Status

- This translates to the following (*incomplete*) architectural representation



Should we store all the **temporary information**? How? And **how to exchange data** between the different analyses?

A Concrete Platform

One RESTful platform

- Surprisingly, there is **only one platform** that actually provides an overall solution
- Some of the concerns have been address by a research project called **SOFAS (SOFtware Analysis Services)** that has been developed to have a better replicability of research results
- It is a RESTful platform that has been designed based on three layers/main components:
  - Software Analysis Web Services – provides the main analyses;
  - Software Analysis Ontologies – represents the data consumed and produced by different services;
  - Software Analysis Broker – the main entry point for the users, that is the main interface to the services;

1 Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

One RESTful platform

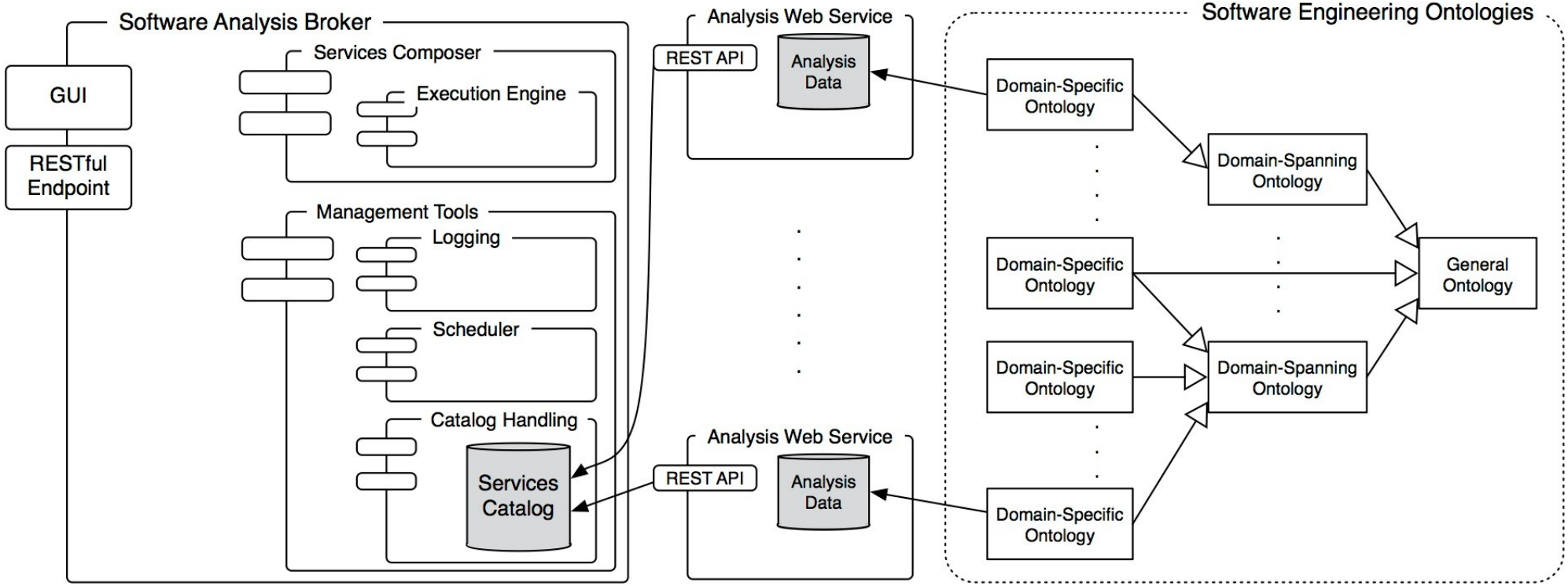


Diagram from: Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

One RESTful platform

A **domain ontology** is used to given semantic meaning to all the services
The ontology is managed with the JENA framework in RDF/OWL and analysis can be queried with SPARQL

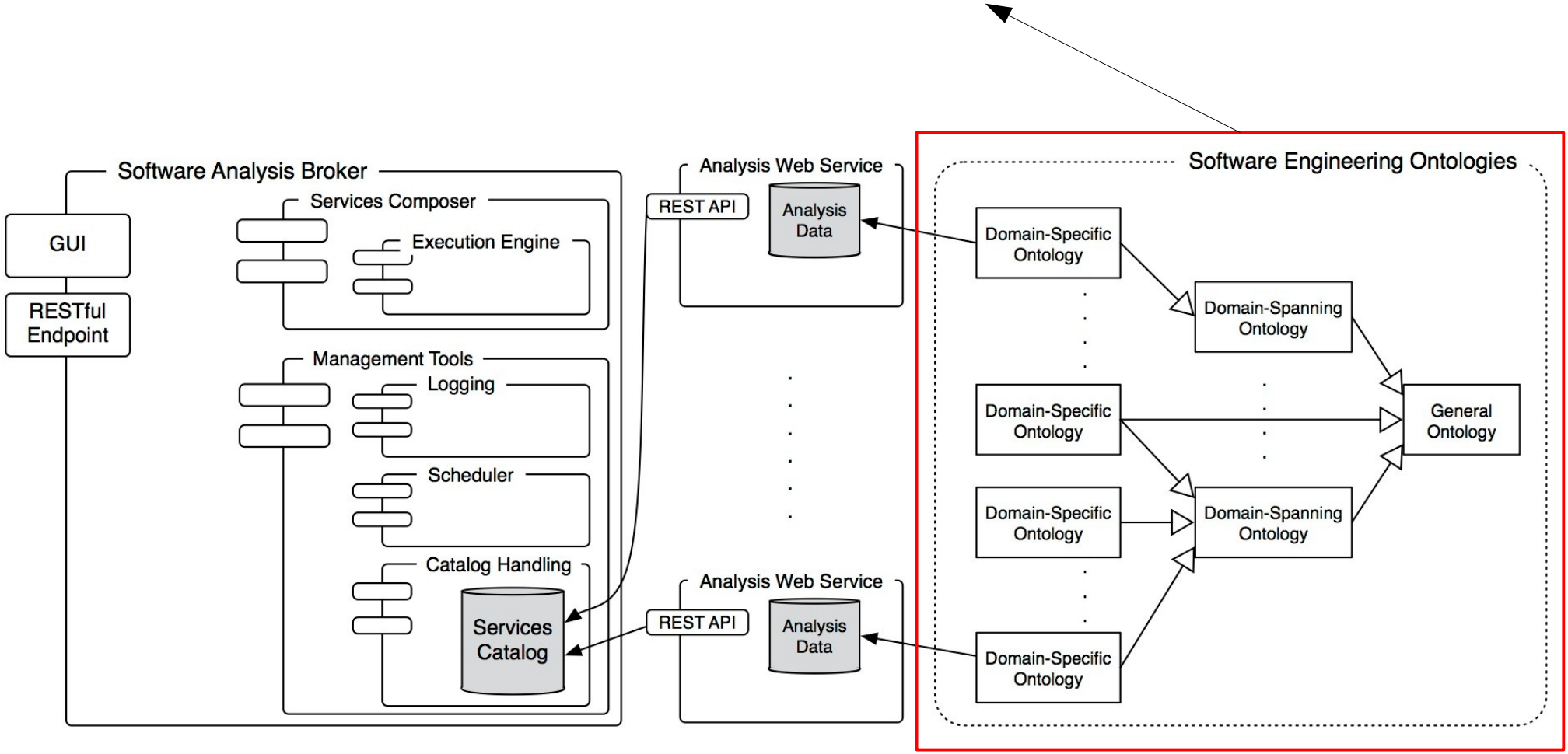


Diagram from: Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

One RESTful platform

An example of part of the ontology that deals with the **structure of source code**

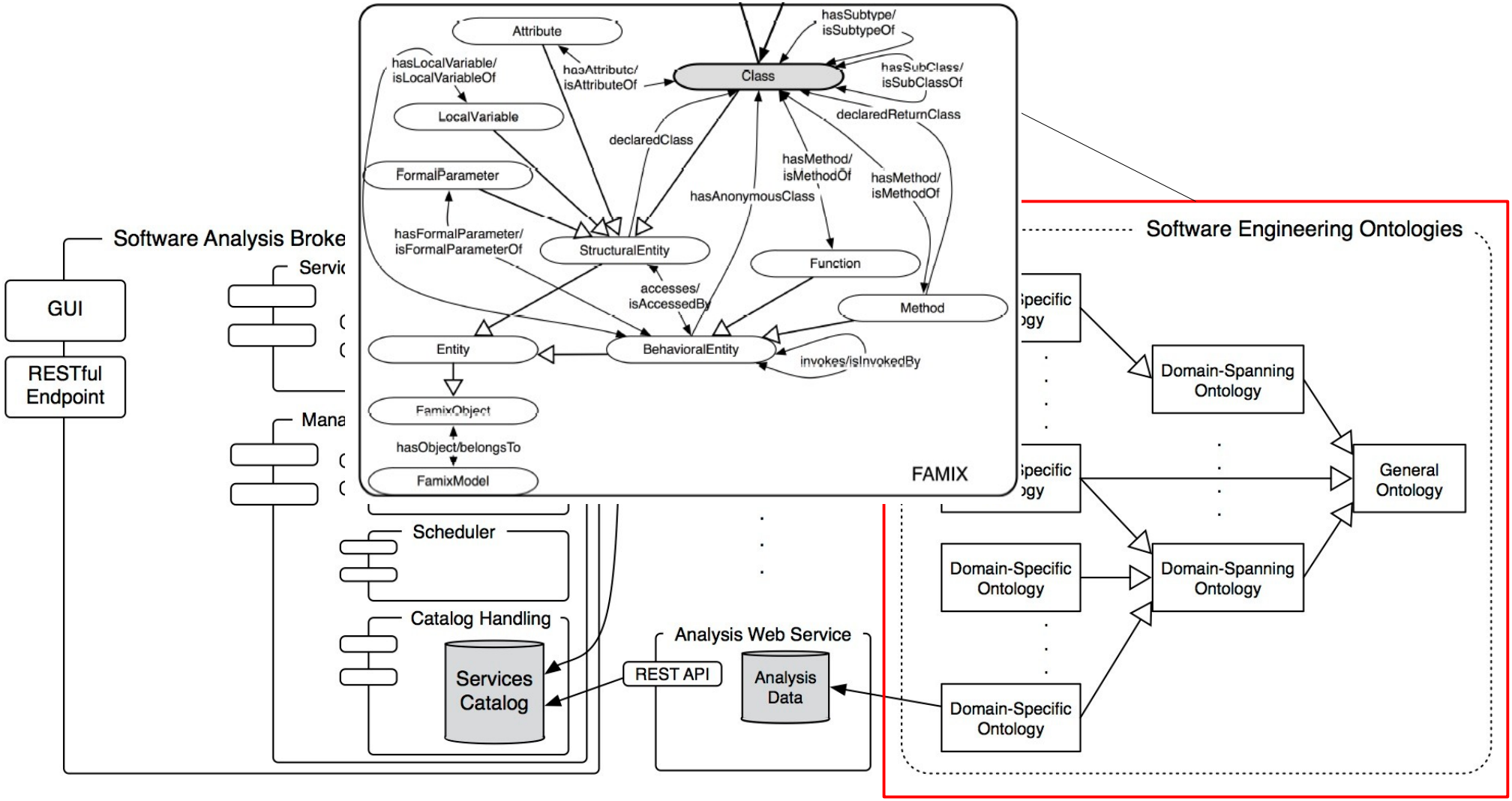


Diagram from: Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

One RESTful platform

The **analysis webservices** extract from the **repositories** (Git, SVN, CVS):
- historical information about commits;
- the **static structure of the source code** (in FAMIX format);
- different set of metrics;

Data from **issue trackers** (like Bugzilla, but also Sourceforge) with a **linkage with the revisions** (by means of heuristics)

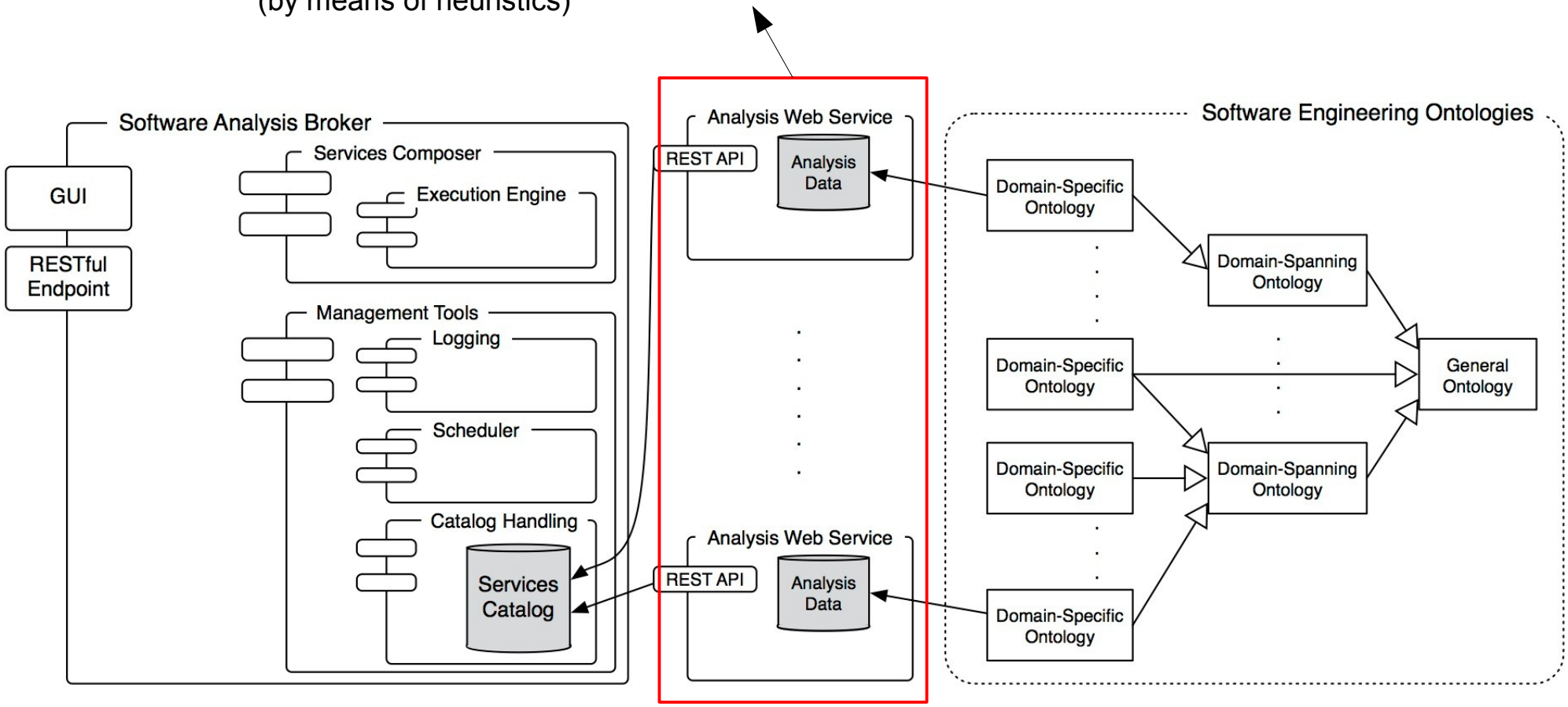


Diagram from: Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

One RESTful platform

An example of the FAMIX format

```
( (FAMIX.Namespace (id: 1)
      (name 'aNamespace'))
  (FAMIX.Package (id: 201)
      (name 'aPackage'))
  (FAMIX.Package (id: 202)
      (name 'anotherPackage')
      (parentPackage (ref: 201)))
  (FAMIX.Class (id: 2)
      (name 'ClassA')
      (container (ref: 1))
      (parentPackage (ref: 201)))
  (FAMIX.Method
      (name 'methodA1')
      (signature 'methodA1()')
      (parentType (ref: 2))
      (LOC 2))
  (FAMIX.Attribute
      (name 'attributeA1')
      (parentType (ref: 2)))
  (FAMIX.Class (id: 3)
      (name 'ClassB')
      (container (ref: 1))
      (parentPackage (ref: 202)))
  (FAMIX.Inheritance
      (subclass (ref: 3))
      (superclass (ref: 2))))
```

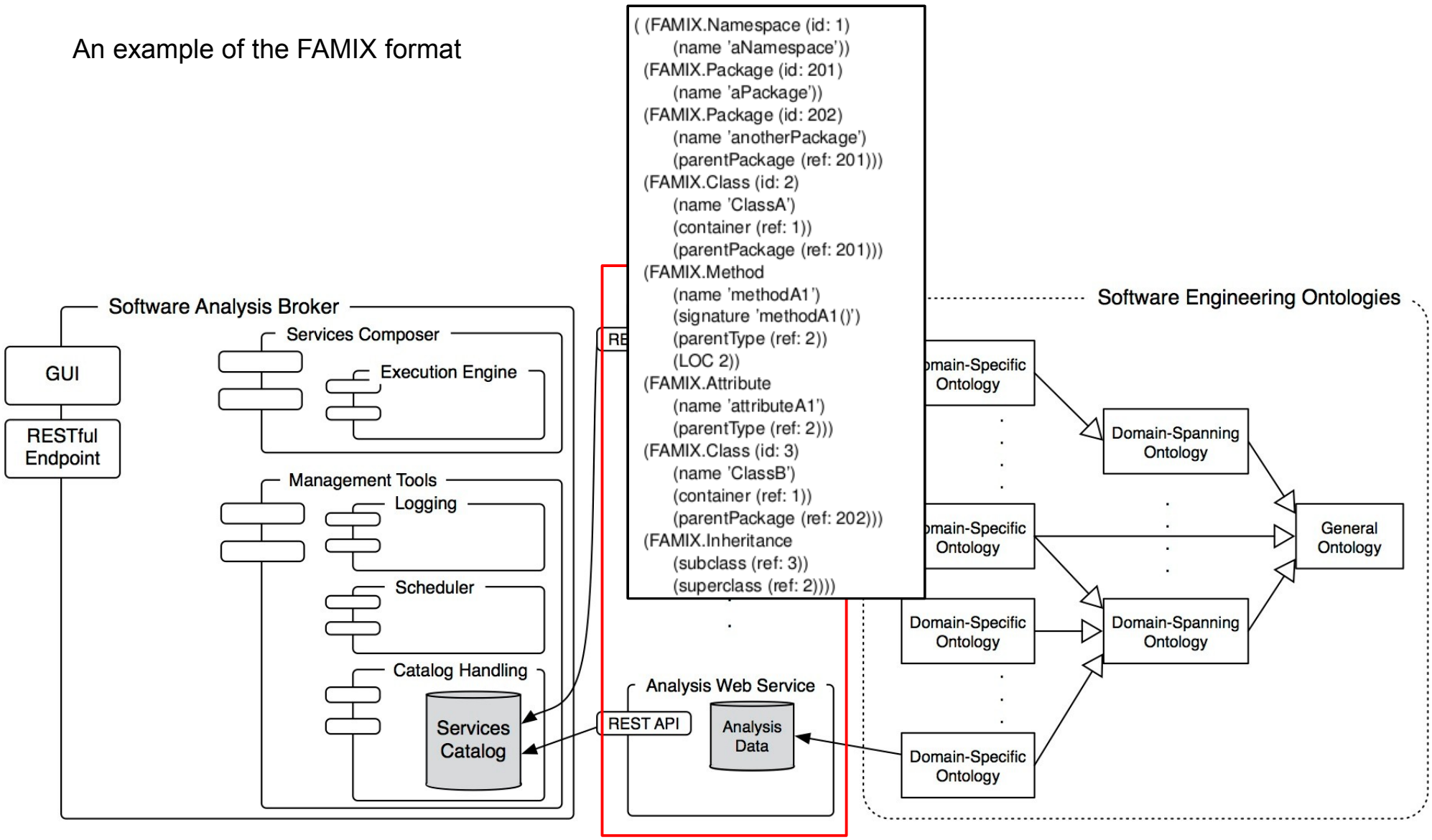


Diagram from: Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

One RESTful platform

The **Software Analysis Broker** is the main interface to the system as well the coordinator of all the services
It has also a **Service Composer** to run the composition of many services (using a similar implementation to WS-BPEL)
There is a service catalogue to refer to all the services

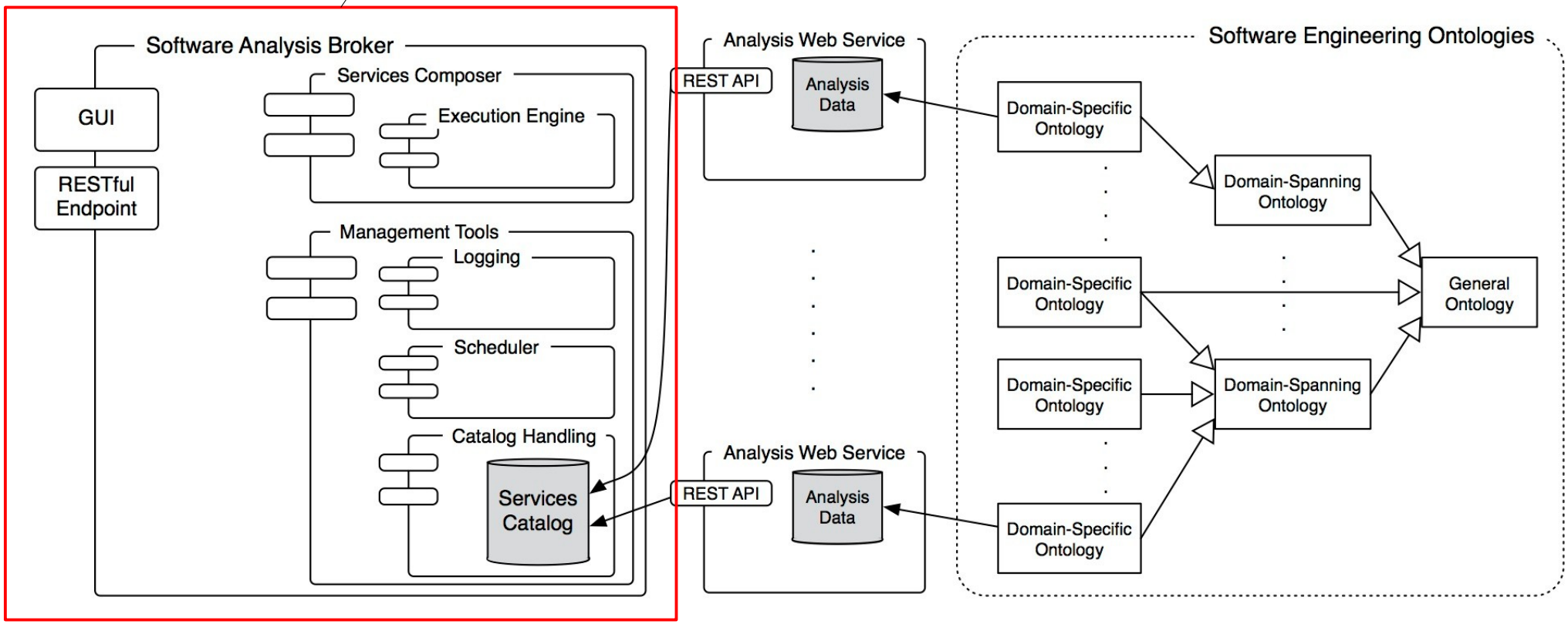


Diagram from: Ghezzi, Giacomo; Gall, Harald (2011). SOFAS: A lightweight architecture for software analysis as a service. In: 9th Working IEEE/IFIP Conference on Software Architecture, Boulder, Colorado, USA, 20 June 2011 – 24 June 2011, 93-102.

Final Remarks

- There are some technical/general considerations about the current platform:
  - To use it, one **needs to contact the authors**, what about making an **"open" solution** that any researcher can use
  - **The implementation of services in on the side of the authors**, no way you can implement your own service
  - As well, the **type of repositories to be mined is fixed**, no way to add more
  - The focus seems more on the **structural analysis** rather than **social analysis**
  - What about **software archaeology** and keeping revisions of historical data

- An open platform will make replicability of studies essentially near to "cost-zero" - and really useful for the research community
- Can be used to support any visualization study based on software engineering data with minimal effort

Note, there is the proposal for a PhD thesis that deals with this topic

Conclusions

- **Mining Software Repositories** deals with the analysis of software repositories to give evidence about past development facts to possibly suggest for improvement
- In this seminar, we saw problems related to the implementation of a *"universal"* miner reducing the problem of the constant rewriting of miners
- We tried to answer to the question on how to build a miner that could be kept neutral with respect to goals and techniques to be applied

Thank you!

Time for questions/discussion