Explora: Tackling Corpus Analysis with a Distributed Architecture

Leonel Merino

Software Composition Group, University of Bern, Switzerland
`http://scg.unibe.ch/`

Abstract. When analysing a corpus of software, researchers often ask questions that entail exploration and navigation, such as "what packages contain fat interfaces in open-source systems?", "how consistently is the code being commented?" and "are naming conventions being followed?". The answers to these questions can impact software maintainability and evolution. Software visualisation can be of aid to understanding and exploring the answers to such questions, but corpus visualisations are time-consuming and difficult to achieve since they require large amounts of data to be processed. We tackle this constrain by using a distributed architecture. In this paper we propose an environment where researchers can build queries for their questions and afterwards rapidly visualise them. We elaborate on a proof-of-concept tool named *Explora* and we report early results when visualising Qualitas Corpus [4].

This paper uses colours in the figures. Please read a coloured print-out of this paper for a better understanding.

1 Introduction

A software corpus is a curated catalogue of software systems intended to be used for empirical studies of code artefacts. A corpus can be understood as the set of models of the systems that it contains. Our approach aims to analyse in parallel all systems at once. Loading the entire set of models in a single server can be difficult since it could require more memory than available. Even if memory is not a problem the performance would be compromised. We overcome this constrain by splitting the corpus into subsets which can be computed in parallel in a distributed architecture. In the following section we introduce Explora the proof-of-concept tool that we developed.

2 Explora: a proof-of-concept

Explora uses the Moose [2] platform for running analysis of FAMIX [5] models. By loading models of systems —an abstraction of a system that contains its properties— it can answer questions about the system's properties such as "what is the longest class hierarchy?", "what is the average depth of class hierarchies?", "what dependencies are between package A with package B?", "are there interfaces without implementations?". These questions are about software properties. In Moose, instead of writing questions in natural language users can obtain their answers by rephrasing them as Smalltalk messages. For instance, for the question "what is the average depth of class hierarchies?" we could send the query `allInheritanceDefinitions average:[:e | |n size| n := e next. size:=0. n isNil ifTrue:[0] ifFalse:[ [ n isNil ] whileFalse:[ size := size +1. n := n next ]. size ] ]` to the model. A query such that which quantifies a software property is called a *software metric.*

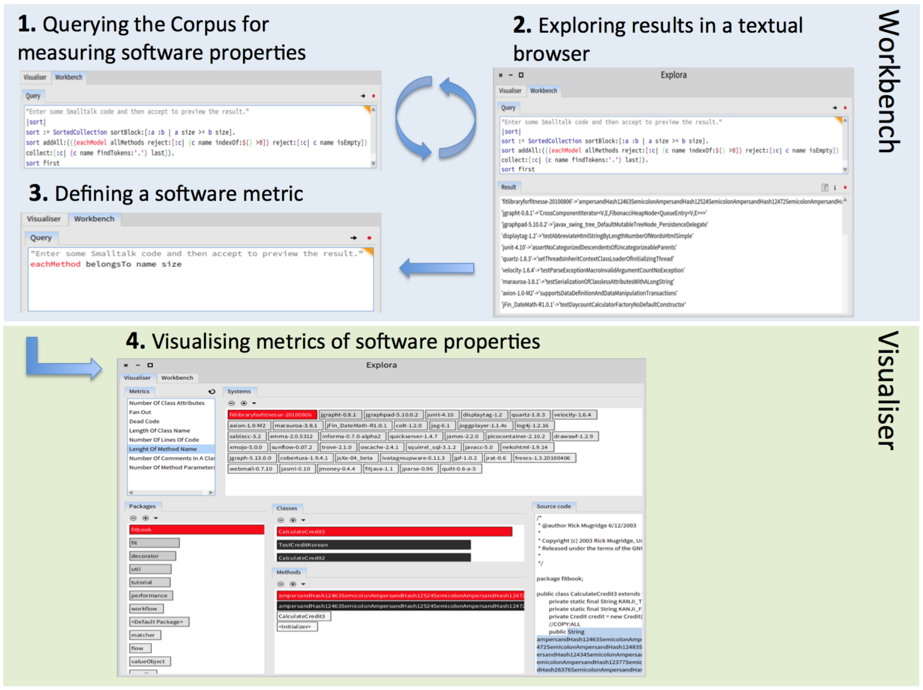
Moose runs on the Pharo Virtual Machine in 32-bit mode. It means that it can manage only a portion of the total available memory of a 64-bit computer. In our experience this threshold is a bit above 1 GB. Moose models tend to be quite large. For instance, a model of JRE (almost 1 MLOC) occupies around 230 MB. It means that it is not feasible to load an entire corpus in a Moose image (a corpus can contain dozens of systems like this and even larger).

Explora offers two main features for end-users: *Workbench* and *Visualiser.* The former offers capabilities to explore the corpus in a textual browser. It is meant as a starting point on which the user rephrase his question by building a software metric. Figure 1 shows the workflow of Explora. Firstly, the user defines a query and triggers its computation. Secondly, he analyses the results. He can perform this exploration several times to fine-tune the query. Thirdly, once he finds results that he want to visualise he can write a metric based on the query (something quantifiable). Finally, the corpus can be explored through the lens of this metric in the other main feature —Visualiser. In the next sections we will explain these features in more detail and show examples.

2.1 Workbech

The workbench has two main use cases: (1) a user is exploring the corpus by posing queries and inspecting results; and (2) a user defines a new software metric for visualisation. The first scenario is an exploratory process used when looking for interesting software properties for visualisation. Once the user stops interacting with this feature the results are discarded. In contrast, the definition of new metrics is persistent and shared with all other Explora instances (once a user creates a metric users of other Explora instances can take advantage of it).

The user does not need to know the details of the implementation of workbench to build a metric. Neither he need to be aware of which object stores the system models, nor he needs to iterate through the collection of systems to evaluate a metric. The implementation provides the user with bounded keywords to refer both at the highest level —a system— and at the lowest level —a method. We named the former *eachModel* which allows users to perform some computation on every model of the systems; and *eachMethod*, which does the same for all methods. For example, lets say a user is analysing if naming conventions are being followed. He can evaluate name length of methods by exploring the model with a query —as shown in Figure 2— which associates each system with the name of the method with the longest name. The results show a method with a huge name in the *fitlibraryforfitnesse* system, the rest seem to be related to


1. Querying the Corpus for measuring software properties
2. Exploring results in a textual browser
3. Defining a software metric
4. Visualising metrics of software properties
Workbench
Visualiser
Explora
Visualiser
Workbench
Query
Result
Metrics
Systems
Packages
Classes
Methods
Source code

```
"Enter some Smalltalk code and then accept to preview the result."
|sort|
sort := SortedCollection sortBlock:[:a :b | a size >= b size].
sort addAll:(((eachModel allMethods reject:[:c| (c name indexOf:$() >0]) reject:[:c| c name isEmpty])
collect:[:c| (c name findTokens:'.') last]).
sort first
```

```
'fitlibraryforfitnesse-20100806'->'ampersandHash12463SemicolonAmpersandHash12524SemicolonAmpersandHash12472SemicolonAmpersandHa
'jgrapht-0.8.1'->'CrossComponentIterator<V,E,FibonacciHeapNode<QueueEntry<V,E>>>'
'jgraphpad-5.10.0.2'->'javax_swing_tree_DefaultMutableTreeNode_PersistenceDelegate'
'displaytag-1.2'->'testAbbreviateHtmlStringByLengthNumberOfWordsHtmlSimple'
'junit-4.10'->'assertNoCategorizedDescendentsOfUncategorizeableParents'
'quartz-1.8.3'->'setThreadsInheritContextClassLoaderOfInitializingThread'
'velocity-1.6.4'->'testParseExceptionMacroInvalidArgumentCountNoException'
'marauroa-3.8.1'->'testSerializationOfClasslessAttributesWithALongString'
'axion-1.0-M2'->'supportsDataDefinitionAndDataManipulationTransactions'
'jFin_DateMath-R1.0.1'->'testDaycountCalculatorFactoryNoDefaultConstructor'
```

```
"Enter some Smalltalk code and then accept to preview the result."
eachMethod belongsTo name size
```

Number Of Class Attributes
Fan Out
Dead Code
Length Of Class Name
Number Of Lines Of Code
Lenght Of Method Name
Number Of Comments In A Clas
Number Of Method Parameters

fitlibraryforfitnesse-20100806 jgrapht-0.8.1 jgraphpad-5.10.0.2 junit-4.10 displaytag-1.2 quartz-1.8.3 velocity-1.6.4
axion-1.0-M2 marauroa-3.8.1 jFin_DateMath-R1.0.1 colt-1.2.0 jag-6.1 joggplayer-1.1.4s log4j-1.2.16
sablecc-3.2 emma-2.0.5312 informa-0.7.0-alpha2 quickserver-1.4.7 james-2.2.0 picocontainer-2.10.2 drawswf-1.2.9
xmojo-5.0.0 sunflow-0.07.2 trove-2.1.0 oscache-2.4.1 squirrel_sql-3.1.2 javacc-5.0 nekohtml-1.9.14
jgraph-5.13.0.0 cobertura-1.9.4.1 jsXe-04_beta ivatagroupware-0.11.3 jpf-1.0.2 jrat-0.6 freecs-1.3.20100406
webmail-0.7.10 jasml-0.10 jmoney-0.4.4 fitjava-1.1 jparse-0.96 quilt-0.6-a-5

fitbook
fit
decorator
util
tutorial
performance
workflow
<Default Package>
matcher
flow
valueObject

CalculateCredit3
TestCreditKorean
CalculateCredit2

ampersandHash12463SemicolonAmpersandHash12524SemicolonAmpersandHash1247
ampersandHash12463SemicolonAmpersandHash12524SemicolonAmpersandHash1247
CalculateCredit3
<Initializer>

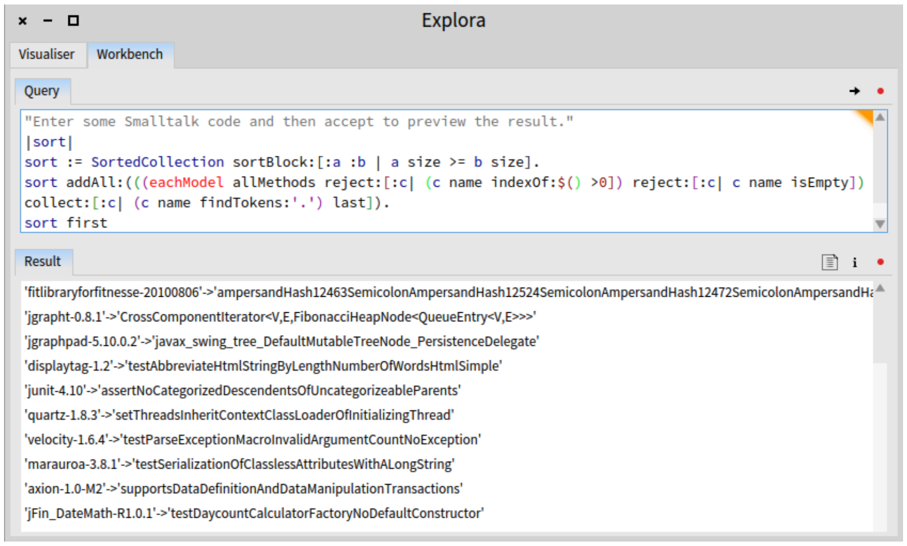
```
/*
 * @author Rick Mugridge 6/12/2003
 *
 * Copyright (c) 2003 Rick Mugridge, Ur
 * Released under the terms of the GNU
 *
 */

package fitbook;

public class CalculateCredit3 extends
    private static final String KANJI_T
    private static final String KANJI_F
    private Credit credit = new Credit(
    //COPY:ALL
    public String
ampersandHash12463SemicolonAmp
472SemicolonAmpersandHash12483S
ersandHash12434SemicolonAmpersa
emicolonAmpersandHash12377Semic
dHash26376SemicolonAmpersandHa
```

Fig. 1. Explora workflow

methods for testing and exceptions. Although this query is not a metric (the results are not a quantification of a property) it allows the user to easily build a metric based on it. In this case the user can define a metric like this `allMethods collect:[:m | m name -> m name size]` . Once the user is happy with the new metric and wants to visualise it, he can save it by clicking the red icon. This triggers the creation of the metric locally and communicates it to other servers. From this point on each server has a persistent definition of the new metric and is able to deploy it over its models.

2.2 Visualiser

The visualiser is a tool for top-down exploration of the corpus. The interface provides the list of metrics previously created. Once the users select a metric, they see boxes representing each system of the corpus. Each box is labelled with the system's name. The box is filled in with different intensities of grey. The darker the box, the higher the value of the metric. The metric is aggregated bottom-up using the maximum value. Systems are sorted from darker to lighter. In this way, not only can users spot the systems with the highest and lowest value, but they can also compare the distribution of the value of the metric across all



```
"Enter some Smalltalk code and then accept to preview the result."
|sort|
sort := SortedCollection sortBlock:[:a :b | a size >= b size].
sort addAll:(((eachModel allMethods reject:[:c| (c name indexOf:$() >0]) reject:[:c| c name isEmpty])
collect:[:c| (c name findTokens:'.') last]).
sort first
```

```
'fitlibraryforfitnesse-20100806'->'ampersandHash12463SemicolonAmpersandHash12524SemicolonAmpersandHash12472SemicolonAmpersandHa
'jgrapht-0.8.1'->'CrossComponentIterator<V,E,FibonacciHeapNode<QueueEntry<V,E>>>'
'jgraphpad-5.10.0.2'->'javax_swing_tree_DefaultMutableTreeNode_PersistenceDelegate'
'displaytag-1.2'->'testAbbreviateHtmlStringByLengthNumberOfWordsHtmlSimple'
'junit-4.10'->'assertNoCategorizedDescendentsOfUncategorizeableParents'
'quartz-1.8.3'->'setThreadsInheritContextClassLoaderOfInitializingThread'
'velocity-1.6.4'->'testParseExceptionMacroInvalidArgumentCountNoException'
'marauroa-3.8.1'->'testSerializationOfClasslessAttributesWithALongString'
'axion-1.0-M2'->'supportsDataDefinitionAndDataManipulationTransactions'
'jFin_DateMath-R1.0.1'->'testDaycountCalculatorFactoryNoDefaultConstructor'
```

Fig. 2. Workbench User Interface

systems. As shown in Figure 3 the result can be a long gradient of grey boxes from white to black, or in some cases it can be something more monotone with very slightly differences between systems —as shown in Figure 4— showing that for all systems the metric produces similar values. This gives an idea of how widely spread are the values of the metric for these system entities. We selected this metaphor for the visualisation because, like a histogram, it aids users to see the distribution of a metric value but at the same time it allows them to identify an entity and to compare them each other. To see the distribution of the values of a metric helps users to identify if the answer to their queries is an abnormal observation or rather common.

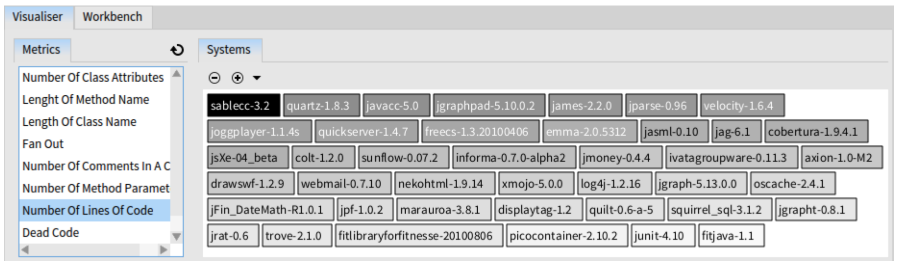


Fig. 3. Highly spread

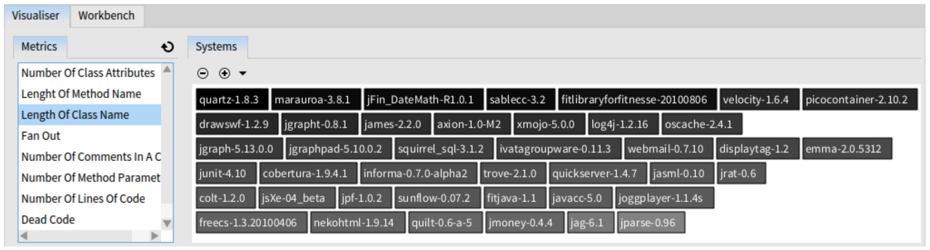


Fig. 4. Monotone

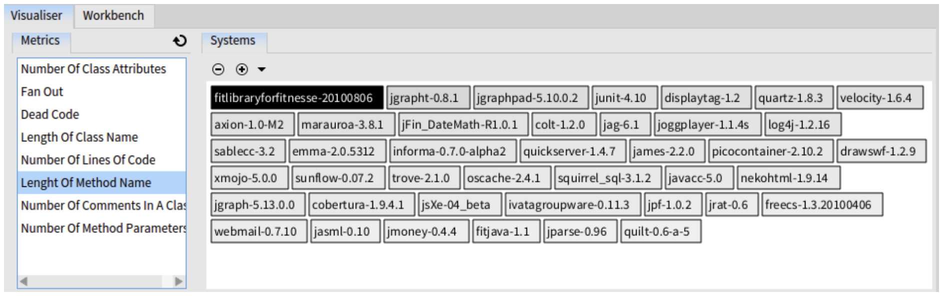


Fig. 5. Outlier

Following the example of method name length—as shown in Figure 5— the user can see that there is an outlier system containing a method with a name much longer than the ones in the rest of the systems (its colour is much more darker than the rest). Figure 6 shows the sequence of actions made by the user when visualising this metric: 1) the user request "details-on-demand" about a system by clicking on its box representation. The result is a list of packages depicted with the same metaphor. Each package is depicted as a box labelled with its name and coloured with an intensity of grey which represents the value of the metric for that package. Once again users can compare packages and easily spot the package with the highest and lowest value. In our example, the method with the longest name is in the package *fitbook*. It also shows that the rest of the packages contain methods with name's length in a similar range to the other systems (the intensity of the grey is similar); 2) users can go deeper in the structure hierarchy. By clicking a package users see the classes that belong to that package. Each class is represented by a box with its name and filled in with an intensity of grey representing the metric; 3) by clicking a class users can see both the source code and the list of methods of that class. Methods are depicted using the same metaphor; 4) by clicking on a method its source code is selected. Notice that entities which are selected are highlighted in red.

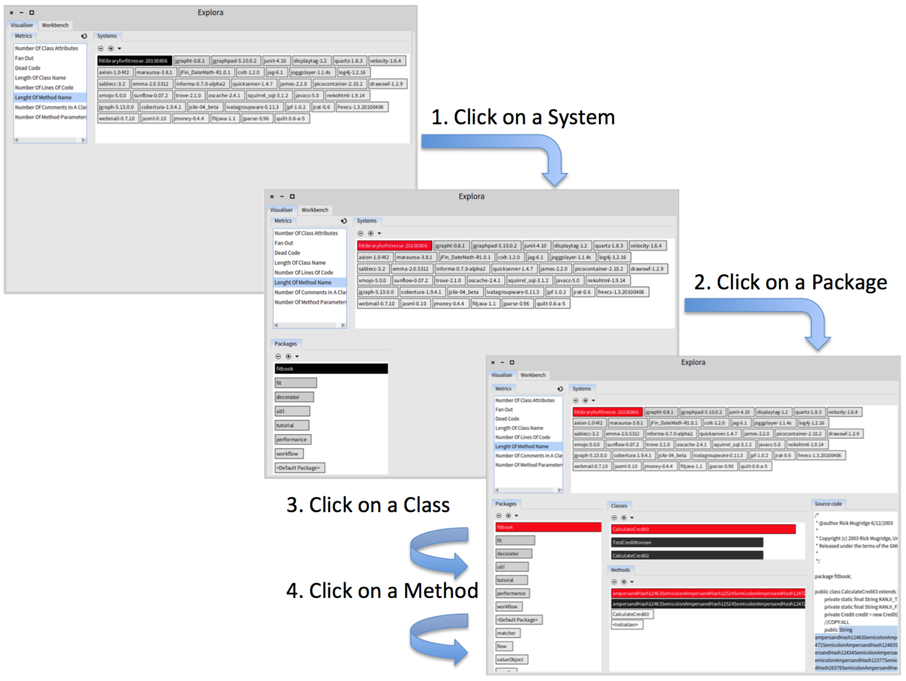


Fig. 6. Top-down visualisation of method name length

2.3 Architecture

Explora instances can run in one of three modes: Broker, Server and Client. The purpose of a Broker is to provide Clients with the information needed to communicate with Servers. They do it by holding a list of triples with the IP address, port and systems that Servers offer. Consequently, when a Server starts, it sends to the Broker the list of system models that it holds. If a Server stops its records are automatically removed from the Broker. When a Client starts it retrieves the list from the Broker, so afterwards it can send queries to Servers directly. When a Server receives a message it creates an independent process to respond to it. This process receives a String containing the Smalltalk message sent by the Client. Servers execute it locally and send a message back with the result. In theory this result can be any Smalltalk object, but in practice we are constrained by the amount of bytes needed by the object to be transmitted over the network. This means that we privilege to transmit little objects to have a better performance. In this way, in the visualiser instead of returning a FAMIX object which size can be several megabytes, we return an Array object with only the information needed to produce the visualisation which typically take up few kilobytes. This result (also an object) is serialised and transmitted to the Client. Finally, the Client is able to materialise the answer. This architecture supports

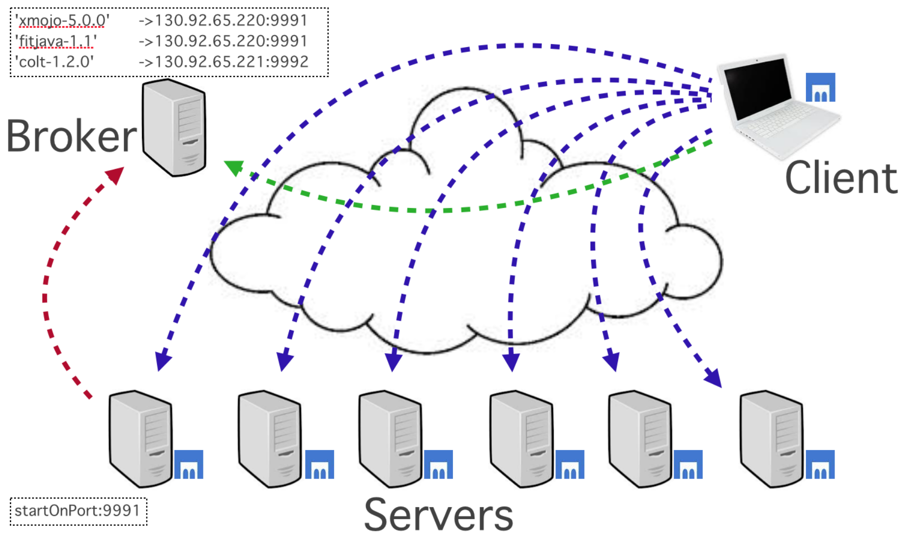

'xmojo-5.0.0' ->130.92.65.220:9991
'fitjava-1.1' ->130.92.65.220:9991
'colt-1.2.0' ->130.92.65.221:9992
Broker
Client
startOnPort:9991
Servers

Fig. 7. Explora architecture (arrows show TCP/IP communication)

both features Workbench and Visualiser. This means that actions performed in these features are computed and solved remotely by other Explora instances.

Figure 7 shows a diagram with the distributed architecture of Explora. In the picture are shown the Broker, 6 Servers and a Client. This deployment supports 85 systems from Qualitas Corpus distributed across 6 Explora Servers each one using less than 500 MB of memory.

2.4 Technical Infrastructure

Explora is developed in Pharo —a Smalltalk inspired language. Explora's user interface uses Glamour [3] which is a framework for building browsers. It offers means to define browser properties such as the number of panes, their sizes, layout and the flow that connects the information among panes.

The visualisations offered by Explora in the Visualiser are based on Roassal [1]—a visualisation engine— which provides the means to define graphical representations of objects. Although we used very basic capabilities of Roassal we left as future work to provide more advanced visualisations. Advanced visualisations could use visual features such as other shapes, colours, edges and layouts to represent more properties of an entity.

3 Future work

Customised representations The visualiser provides a basic representation of model entities, just depicting them by labelled-boxes and using grey for representing the metric. We think that more advanced users can profit from means to customise these graphical representations. We envision a way in which users can freely customise some parts of the visualisation.

Results in Workbench When sending a query, in the Workbench, the results are shown in the lower pane —as shown in Figure 2. This pane shows a list of elements, each of which is a string composed of the name of a system followed by a list with the result of the message which was sent to that system (it can be a list of one element). We would like to present this data as a table offering features to sort data by each column.

Tackling with malicious queries Explora is meant for accessing user's own corpus, it does not have features for tackling with malicious code —i.e. code injection. This means that Explora servers can execute harmful queries. In the future Explora can be used as a distributed environment where users from different places (using Explora clients) can access a shared corpus (cluster of Explora servers) which they do not own. In that case, we think would be mandatory to provide Explora with mechanisms to prevent code injection.

Covering the entire catalogue In Qualitas Corpus there are large systems which require more than 500 MB. Although we wanted each server to use no more than this threshold so they would have enough space for developing memory-intensive metrics, we would like to cover the entire catalogue. For that purpose we propose to provide dedicated servers —each one holding only one model— with more memory. A complementary strategy would be to load less information when creating the model of the system.

4 Conclusion

In conclusion, although software visualisation is of great aid for analysis when trying to apply it on large pieces of software —such as a corpus— managing large data in memory turns to be complex. A distributed architecture for parallel computing offers a good fit to software analysis and visualisations. In this paper we presented Explora as a proof-of-concept tool for corpus analysis and visualisation based on a distributed architecture. The tool provides means to build software metrics and to visualise them. We described the specifics of the implementation and discussed some future work to improve metrics creation and to provide custom visualisations for advanced users.

Acknowledgements

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Assessment" (SNSF project No. 200020-144126/1, Jan 1, 2013 - Dec. 30, 2015). This work has been partially funded by CONICYT BCH/Doctorado Extranjero 72140330.

References

1. Vanessa Peña Araya, Alexandre Bergel, Damien Cassou, Stéphane Ducasse, and Jannik Laval. Agile visualization with Roassal. In *Deep Into Pharo*, pages 209–239. Square Bracket Associates, September 2013.
2. Stéphane Ducasse, Tudor Gîrba, and Oscar Nierstrasz. Moose: an agile reengineering environment. In *Proceedings of ESEC/FSE 2005*, pages 99–102, September 2005. Tool demo.
3. Tudor Gîrba, Alexandre Bergel, Damien Cassou, Stéphane Ducasse, and Jannik Laval. Glamour. In *Deep Into Pharo*, pages 191–207. Square Bracket Associates, September 2013.
4. E. Tempero, C. Anslow, J. Dietrich, T. Han, Jing Li, M. Lumpe, H. Melton, and J. Noble. The Qualitas Corpus: A curated collection of Java code for empirical studies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 336 –345, December 2010.
5. Sander Tichelaar. *Modeling Object-Oriented Software for Reverse Engineering and Refactoring*. PhD thesis, University of Bern, December 2001.