Explora: Infrastructure for Scaling Up Software Visualisation to Corpora

Leonel Merino, Mircea Lungu, Oscar Nierstrasz

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

Abstract. Visualisation provides good support for software analysis. It copes with the intangible nature of software by providing concrete representations of it. By reducing the complexity of software, visualisations are especially useful when dealing with large amounts of code. One domain that usually deals with large amounts of source code data is empirical analysis. Although there are many tools for analysis and visualisation, they do not cope well software corpora. In this paper we present Explora, an infrastructure that is specifically targeted at visualising corpora. We report on early results when conducting a sample analysis on Smalltalk and Java corpora.

1 Introduction

A software corpus is a curated catalogue of software systems intended to be used for empirical studies of code artefacts. The advantage of doing research on corpora is that it encourages repeatable analyses. Corpus analysis is especially used in the context of empirical software engineering, where results should be repeatable [9, 15, 16]. Visualisation is especially useful for dealing with large amounts of source code, since it provides software a concrete representation making complex data easier to understand. However, most visualisation tools are not designed for corpora.

Imagine Edgar, an empirical software engineering researcher, who wants to assess the prevalence of reuse of software in different languages. To begin, he chooses to study reuse through inheritance and invocation in one of the oldest object-oriented languages, Smalltalk and one of the most popular, Java. To gain an initial insight into the data, he must carry out explorative data analysis, including visualisation.

To set up the environment for the analysis, he needs to overcome several problems: *1*) visualising one system at a time (as most visualisation systems allow) prevents patterns from being recognized at the corpus level; *2*) the lack of means for real-time data manipulation (such as sorting, filtering, searching, inspecting) discourages experimentation; and *3*) two technical issues, memory consumption and performance, complicate fetching and manipulating corpus data.

In this paper we introduce our approach, Explora, which copes with these issues.

2 Analysis Example

Edgar wants to target Smalltalk and Java corpora. He has a fair experience implementing and maintaining systems in both languages. He realises that in his experience Smalltalk systems have deeper hierarchies than Java ones, so he wants to explore whether increased specialisation in Smalltalk systems correlates with less reuse of their classes. Thus, he wants to answer the following research question:

RQ: "*How different is reuse by inheritance and invocation in Smalltalk and Java systems?*"

He believes that metrics are a suitable way for tackling this research question. He chooses four metrics from a catalogue of metrics proposed by Lanza and Marinescu to characterise two types of reuse: reuse via inheritance and reuse via invocation [12].

1) Average Hierarchy Height (AHH): the average depth of the inheritance trees of a system is one of the two metrics that characterise inheritance.
2) Depth of Inheritance Tree (DIT): the maximum length of the path of each class to the root class in a hierarchy; a measure of the system can be obtained by aggregating this metric for each of the classes.
3) Fan-In: this quantifies the dependent classes (access and invocation relationships) of a class.
4) Fan-Out: the outgoing coupling, which characterises communication.

Edgar decides to start his analysis by visualising these metrics on all the systems in both corpora. He decides to use Explora for his analysis. He downloads the models of the systems in the two corpora following the installation instructions of Pangea[1] locally and places them in a dedicated folder called the *model workspace*. To query corpora, Explora uses a so-called interactive Playground implemented on top of the Moldable Inspector infrastructure [3] of Moose [8].

Step 1: Computing the Metrics Figure 1 shows a screenshot of the Playground in which:

1) As the user modifies the query in the *Mapper* pane, the right pane is updated. The Mapper pane shows a query written in plain Smalltalk for collecting AHH (line 3), and the aggregated maximum value of DIT (line 7), Fan-In (line 9) and Fan-Out (line 11) from every system in the two studied corpora. The query defines an inline array of associations, where each association is linked to a different metric. The query uses the bound variable *eachModel* to refer to the FAMIX meta-model of each system. The model enables code such as *"eachModel allModelClasses"* and *"c superclassHierarchy"*.

[1] `http://scg.unibe.ch/research/pangea`

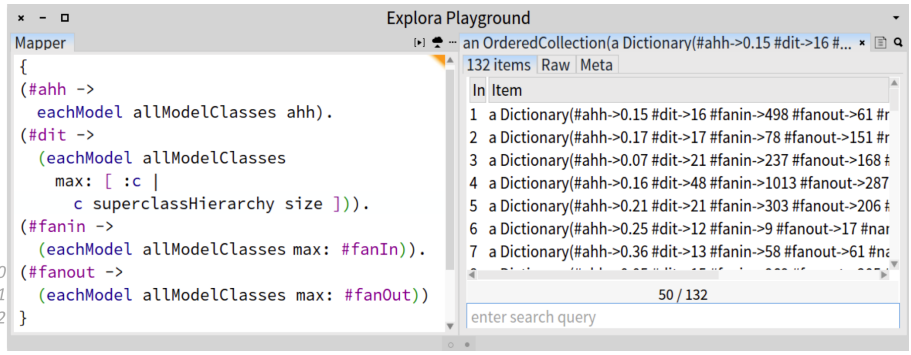


Fig. 1. Collecting metrics from Smalltalk and Java corpora.

The user can evaluate metrics defined in the system model, as is the case of AHH, Fan-In and Fan-Out, or compute custom ones such as the one specified for DIT. The query is sent to all the models of the systems in the model workspace

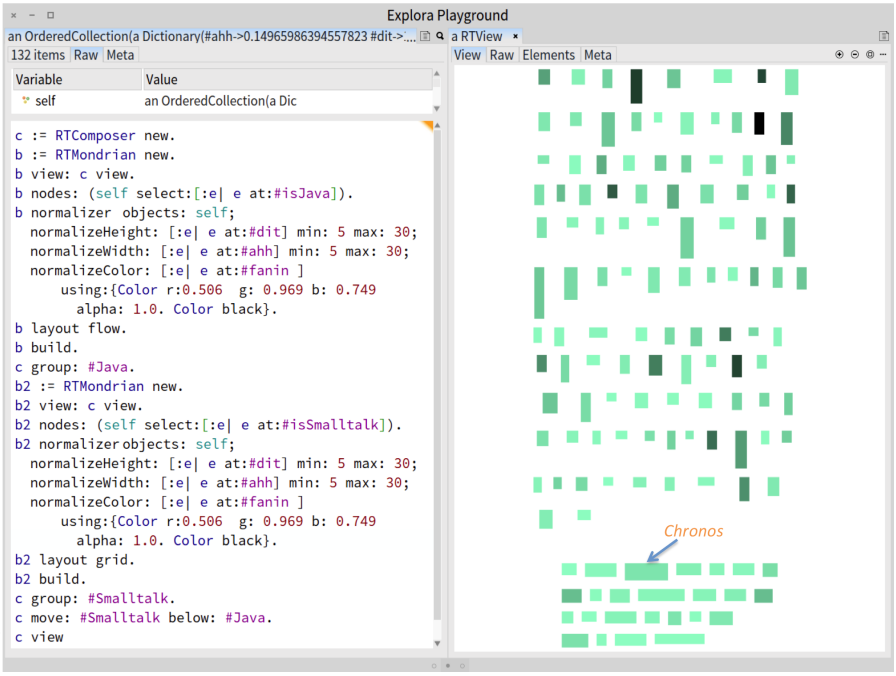
2) The right pane shows the result of the query, which is always a collection of objects retrieved from the queried system models by executing the query in the Mapper pane. The objects in this collection can be further manipulated (*i.e.* sorted, filtered, further queried, visualised).

Each pane of the Playground is linked to an object. By using the bound variable *self* the user can manipulate the object. The Playground supports navigation using the Miller column technique[2] on which the evaluation of a script in a pane adds a new pane to the right that is linked with the returned object. The Mapper is a special pane (since it is the first) that uses a different bound variable (eachModel) for referencing specifically system models.

Step 2: Generating an Initial Visualisation A special type of manipulation supported in right pane of the Playground uses the result object (*i.e.* the collection of dictionaries) as input for a visualisation. To use this feature, Edgar switches from the list view of the result to the Raw tab, which allows him to write a visualisation script. For instance, Figure 2 shows in the left pane an implementation of a lightweight visualisation using the Mondrian DSL. The result object is referred to in this script as "self" (see lines 4, 5, 11 and 12).

The right pane shows the generated visualisation. Each box represents a system, Smalltalk ones being grouped at the bottom while Java systems are at the top. The width of each box is mapped to AHH and the height to DIT. Edgar mapped the metrics in this way so that boxes with a larger area will indicate systems with many deep hierarchies. Furthermore, the darker the green

[2] `http://en.wikipedia.org/wiki/Miller_columns`




```
1  c := RTComposer new.
2  b := RTMondrian new.
3  b view: c view.
4  b nodes: (self select:[:e| e at:#isJava]).
5  b normalizer objects: self;
     normalizeHeight: [:e| e at:#dit] min: 5 max: 30;
     normalizeWidth: [:e| e at:#ahh] min: 5 max: 30;
     normalizeColor: [:e| e at:#fanin ]
         using:{Color r:0.506  g: 0.969 b: 0.749
           alpha: 1.0. Color black}.
6  b layout flow.
7  b build.
8  c group: #Java.
9  b2 := RTMondrian new.
10 b2 view: c view.
11 b2 nodes: (self select:[:e| e at:#isSmalltalk]).
12 b2 normalizer objects: self;
     normalizeHeight: [:e| e at:#dit] min: 5 max: 30;
     normalizeWidth: [:e| e at:#ahh] min: 5 max: 30;
     normalizeColor: [:e| e at:#fanin ]
         using:{Color r:0.506  g: 0.969 b: 0.749
           alpha: 1.0. Color black}.
13 b2 layout grid.
14 b2 build.
15 c group: #Smalltalk.
16 c move: #Smalltalk below: #Java.
17 c view
```

Fig. 2. Visualising AHH and *maximum* values of DIT and Fan-In among Java and Smalltalk systems.

of the box the higher the value of Fan-In. At a first glance, the analysis of the visualisation seems to reveal a pattern. Smalltalk systems are landscape oriented in lighter colour and Java ones have a portrait orientation with a darker colour. Since he took the maximum values of Fan-In and DIT, Edgar realises that in general the Java corpus contains the most invoked class (highest Fan-In), and the deepest hierarchy (highest AHH and DIT). This suggests that Java systems exhibit more reuse than Smalltalk ones. This can be a misleading result due to the decision of aggregating DIT and Fan-In using maximum values, since they do not provide Edgar insight into a general tendency.

Step 3: Exploring alternative Visualisations In consequence, Edgar decides to find out if this pattern still prevails when values are aggregated using the median. He modifies the implementation accordingly, by aggregating the values of DIT and Fan-In using the median. Without leaving the environment, he goes to the left pane shown in Figure 1 and changes lines 9 and 11 accordingly (collecting median instead of maximum values). The Playground recallsl the implementation of the previous visualisation generating a new one automatically (Figure 3). Edgar notes the difference between system *Chronos* in Figure 2 and in Figure 3. The analysis shows that even though it has neither the most

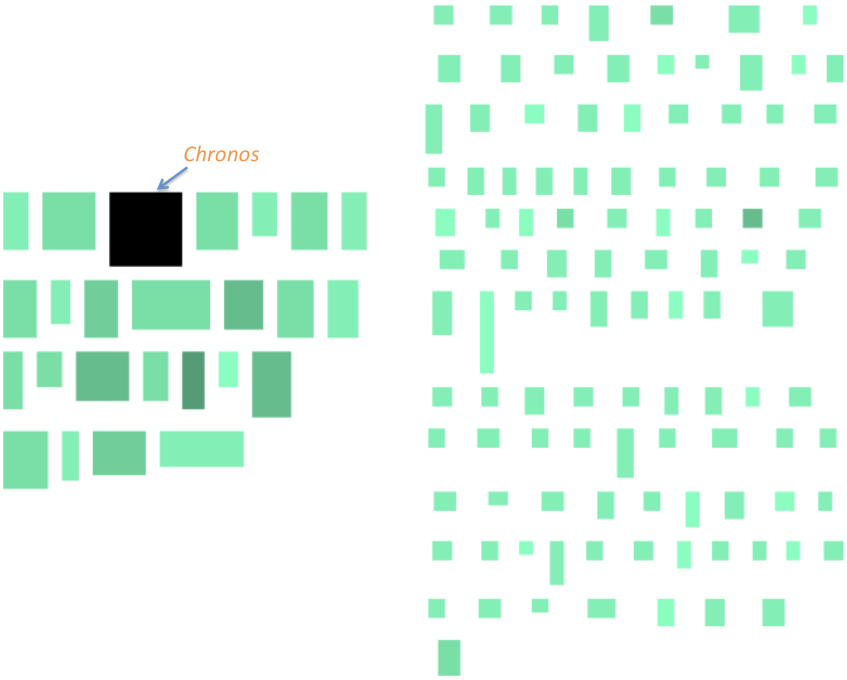


Fig. 3. Visualising AHH and *median* values of DIT and Fan-In among Java and Smalltalk systems.

invoked class in the two corpora (light green in Figure 2), nor the deepest hierarchy (lower height in Figure 2), in general its classes are the most invoked and have deep hierarchies (large and dark box in Figure 3). Figure 3 shows that most systems of the Smalltalk corpus exhibit more reuse by having larger, deeper and more invoked hierarchies.

Diving Into an Individual System Figure 4 shows a detailed visualisation of Chronos which is a library for manipulating dates and times.[3] Classes are represented by circles. The darker the circle, the more invoked the class (higher Fan-In). The size of the circle is mapped to Fan-Out allowing the user to compare classes that behave as clients and providers in invocation relationships. Blue edges between classes show inheritance relationships and grey edges represent invocations. For a better analysis, only invocations of highly invoked classes are shown (Fan-In greater than 90). From the visualisation Edgar can distinguish main provider classes that are highly invoked (dark circles), even though some of them are clients as well (darker and larger circles). Most notably, there is a hierarchy in Figure 4-A that includes many highly invoked classes (*ChronosObject*).

[3] `http://smalltalkhub.com/#!/~Chronos/Chronos`

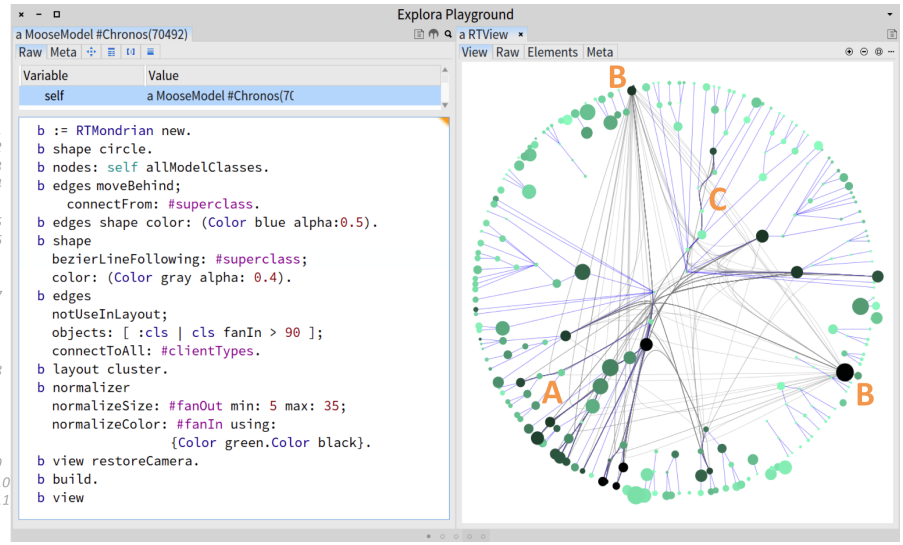


Fig. 4. Drilling-down in the Chronos system. Large and dark nodes represent classes with high fan-out and respectively high fan-in

Besides, there are two classes in Figure 4-B *TimeZoneAnnualTransitionPolicyFactory* and *DateSpec* that without being part of a hierarchy attract many invocations. Finally in Figure 4-C Edgar notes a small hierarchy of chained invocations.

The top-down exploration starting from the visualisation at the corpora level enabled Edgar to decide what systems to focus on. It gave him an overall assessment of reuse between the two corpora as well as a detailed vision of the reuse in a system, when he drilled down into Chronos. He learned that the greater specialisation of Smalltalk systems seems not to affect reuse. Indeed he found deep hierarchies in the Chronos system but he also found that those hierarchies are heavily reused.

3 Technical Infrastructure

When designing Explora we combined several tools for accomplishing the analysis task. Explora is inspired by Pangea [2], and uses the Moose [8] platform for analysing FAMIX [20] models of software systems. It reuses Object Model Snapshots (OMS) from Pangea's data model. An OMS is a custom Moose image containing a single FAMIX model of a system. The model currently includes OMSs of two corpora: 1) Qualitas Corpus [19] and 2) SqueakSource-100 [2].

Explora is written in Pharo, an open-source Smalltalk dialect. The Pharo live programming environment allows users to explore and navigate data in a

dynamic fashion. Explora uses a Playground built on top of the Moldable inspector [3] of Pharo for querying the model, and manipulating results. The Roassal visualisation engine [1] provides a comprehensive API for visualising data in an agile fashion. Roassal provides several Domain-Specific Languages including Mondrian and Grapher.

Workflow

1) The user defines and triggers a query for collecting data from the corpora.
2) A main process looks for OMSs available in a local folder called the source workspace, and evaluates the query in each of them (there are sequential and parallel modes). An OMS is used as a cache holding a live version of the system model that can be awakened, queried, and put back to sleep again.
3) The independent result returned by each OMS is serialised using Fuel [7].
4) These partial results are aggregated to be returned to the user.
5) The user decides whether to go back to 1) or continue with the following step.
6) The user can manipulate the results by filtering, sorting, inspecting or using them as input for a visualisation.

Although the architecture scales up by adding more OMSs horizontally, it is still constrained to the memory available in the Pharo 32 bit environment for materialising objects. A workaround for this issue is to collect less data by including into each OMS only essential information.

Example's Benchmark Table 1 shows a benchmark with the performance results and memory consumption at each step when we run the example analysis. The data were collected by: 1) calling the garbage collector; 2) measuring the memory used (average among 10000 times); 3) executing the step; 4) measuring the memory used (average among 10000 times); and 5) calculating the difference between 4) and 2).

Note that during the execution of the step 1 (Computing the Metrics) new processes are created (in parallel or sequentially) using more memory, however this is released after the execution.

Step	Description	Performance (Secs.)	Memory (MB)
1	Computing the Metrics	39.108	4.3
2	Generating an Initial Visualisation	0.238	2.1
3	Exploring Alternative Visualisation	0.238	2.3
4	Diving into an Individual System	0.212	6.3

Table 1. Performance and memory consumption

4 Future Work

Automatic Visualisation Although Roassal provides a number of expressive DSLs for different tasks, it requires expertise to generate useful visualisations. We envision an approach that exploits expertise of proven well-designed visualisations automatically visualise results. Users without expertise should be able to profit from such visualisations. Lately we are studying how experts visualise software. We have been classifying their visualisations into several dimensions such as goal, domain, granularity. We are working to develop an approach that makes use of this classification to provide automatic visualisation.

Automatic Visualisation Assessment Knowledgeable users who can implement data visualisations would benefit from *Automatic Visualisation Assessment* AVA. While users are implementing visualisations, AVA would give them feedback about visual design guidelines that are violated, and suggest how to resolve them. Our idea is to develop a model of visual design constraints covering the main pitfalls that developers encounter when visualising data, such as visual cluttering, layout selection, and colour conflict.

Expanding the Corpora There are a few systems in Qualitas Corpus that do not fit into an OMS. This will be solved with a 64-bit version of Pharo. We also realise that SqueakSource-100 should be expanded to more systems to provide more interesting results. Finally, we think that adding more corpora from other languages would be an advantage for experimental analysis.

5 Related Work

Explora expands related work by scaling up software visualisation to corpora. Explora's design is based on three pillars: 1) *liveness* of the Pharo environment which enables interactive exploration; 2) ready-to-use software *corpora* models which encourage repeatable analyses; and 3) agile *visualisation* to provide support for data analysis. Figure 5 shows how related work and Explora cover these concepts.

5.1 Visualisation Tools

To the best of our knowledge there is no visualisation tool that provides support for software corpora. Some of them allow users to load into memory several models of systems, but they cannot visualise systems together. Only one of these systems offers liveness.

CodeCrawler [6] is a visualisation tool based on Moose and FAMIX models. It includes many built-in views covering several common software analysis tasks. Views can be partially customised by assigning a specific mapping between the built-in metrics with the visual properties of the representation. CodeCrawler

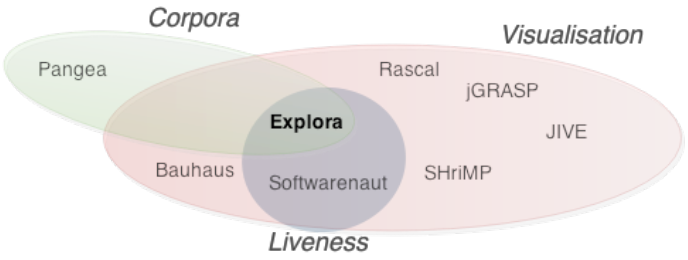


Fig. 5. Explora and related work

was superseded by *Mondrian* [14], a high-level DSL for specifying visualisations. Both of them are meant for analysis of single model systems.

SHriMP [18] visualises software using nested graph views for structural entities such as packages, classes and methods. Edges between artefacts represent dependencies such as inheritance, composition and association relationships. The tool is meant to explore software structure and to navigate source code. Hyperlinks are used ease navigation through source code. SHriMP targets developers analysing their own code or legacy one but always coping with single systems.

jGrasp [4] is a lightweight development environment implemented in Java. Traditional data structures, such as stacks, queues and linked lists, can be easily identified in the visualisation. It is intended to support Java teaching through program visualisation. It only allows the user to visualise an isolated project.

JIVE [10] stands for Java Interactive Visualisation Environment and is mainly used for debugging, maintenance and learning. It provides interactive visualisations of the runtime state and call history of a program. It is integrated in the Eclipse IDE, allowing users to visualise a single project.

Softwarenaut [13] is an analysis tool written in Smalltalk and which profits from its liveness. It visualises software using hierarchical views. It includes three complementary perspectives which allow the user to explore and navigate data. The tool includes pre-packaged metrics that can be mapped to visual properties. Although Softwarenaut allows users to load several model systems into memory, it can only visualise one at a time.

5.2 Data Analysis Tools

Rascal [11] is a Domain Specific Language for source code analysis and manipulation. It is implemented as a plug-in for Eclipse, and consequently benefits from other tools installed in the environment, and exploits Eclipse to obtain software models cheaply. It can only visualise the systems currently loaded in the Eclipse workspace.

Bauhaus [17] is a tool suite written in Ada that supports multi-language program understanding and reverse engineering for maintenance and evolution.

It provides tools to extract, analyse, query and visualise software artefacts. It provides support for analysis and visualisation of single systems.

Pangea [2] is an environment for static analysis of multi-language software corpora. Based on Moose it provides an expressive scripting language. However, since it is implemented as a bash script, it offers neither liveness, nor visualisation. Pangea's output is normally a text file while in Explora it is a live object.

Large-Scale Data Analysis *MapReduce* [5] is a programming model and implementation for processing large datasets. The model is based on the disaggregation of a large dataset into smaller pieces that can be handled by different servers in parallel. The query sent by a client is computed independently by each server. Afterwards, the result of the computation of a server is aggregated. Explora is inspired by MapReduce. In Explora, the corpora are disaggregated into system models which compute queries independently. Explora is not distributed over a network but runs locally.

6 Conclusion

In conclusion, although there are many tools for software analysis and visualisation most of them do not scale to software corpora. Data analysis tools that do scale to corpora are not live. On the other hand, visualisation tools that do offer liveness do not scale to corpora. In this paper we presented Explora, an infrastructure for scaling up software visualisation to corpora. We presented an example of analysis stressing its strengths, showing how visualisation can help one to explore and understand software. However, we acknowledge that useful visualisations are difficult to achieve. In consequence, in the future we want to tackle this issue by automatically visualising software by mapping queries to suitable, proven visualisations. We also think that users with the knowledge for visualising software can profit from automatic visualisation assessment, a dynamic evaluation of the visualisation that provides feedback concerning violations of visual design rules and guidelines.

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. Andrea Caracciolo, Andrei Chis, Boris Spasojević, and Mircea Lungu. Pangea: A workbench for statically analyzing multi-language software corpora. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, pages 71–76. IEEE, September 2014.
3. Andrei Chiş, Oscar Nierstrasz, and Tudor Gîrba. The Moldable Inspector: a framework for domain-specific object inspection. In *Proceedings of International Workshop on Smalltalk Technologies (IWST 2014)*, 2014.
4. James H Cross, Dean Hendrix, and David A Umphress. jGRASP: an integrated development environment with visualizations for teaching java in cs1, cs2, and beyond. In *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages 1466–1467. IEEE, 2004.
5. Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.
6. Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, *Proceedings of 6th Working Conference on Reverse Engineering (WCRE '99)*. IEEE Computer Society, October 1999.
7. Martín Dias, Mariano Martinez Peck, Stéphane Ducasse, and Gabriela Arévalo. Fuel: a fast general purpose object graph serializer. *Software: Practice and Experience*, 44(4):433–453, 2014.
8. 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.
9. Jean-Marie Favre, Dragan Gasevic, Ralf Lämmel, and Ekaterina Pek. Empirical language analysis in software linguistics. In *Software Language Engineering*, pages 316–326. Springer, 2011.
10. Paul V Gestwicki and Bharat Jayaraman. Jive: Java interactive visualization environment. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 226–228. ACM, 2004.
11. Paul Klint, Tijs van der Storm, and Jurgen Vinju. RASCAL: A domain specific language for source code analysis and manipulation. In *Source Code Analysis and Manipulation, 2009. SCAM '09. Ninth IEEE International Working Conference on*, pages 168–177, 2009.
12. Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.
13. Mircea Lungu, Adrian Kuhn, Tudor Gîrba, and Michele Lanza. Interactive exploration of semantic clusters. In *3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005)*, pages 95–100, 2005.
14. Michael Meyer, Tudor Gîrba, and Mircea Lungu. Mondrian: An agile visualization framework. In *ACM Symposium on Software Visualization (SoftVis'06)*, pages 135–144, New York, NY, USA, 2006. ACM Press.
15. Paloma Oliveira, Marco Tulio Valente, and Fernando Paim Lima. Extracting relative thresholds for source code metrics. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 254–263. IEEE, 2014.
16. Ekaterina Pek. *Corpus-based empirical research in software engineering*. PhD thesis, Universitaet Koblenz-Landau, 2013.
17. Aoun Raza, Gunther Vogel, and Erhard Plödereder. Bauhaus — a tool suite for program analysis and reverse engineering. In *Reliable Software Technologies - Ada-Europe 2006*, pages 71–82. LNCS (4006), June 2006.

18. Margaret-Anne Storey, Casey Best, and Jeff Michaud. SHriMP Views: An interactive and customizable environment for software exploration. In *Proceedings of International Workshop on Program Comprehension (IWPC '2001)*, 2001.
19. 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.
20. Sander Tichelaar. *Modeling Object-Oriented Software for Reverse Engineering and Refactoring*. PhD thesis, University of Bern, December 2001.