Explora: A Visualisation Tool for Metric Analysis of Software Corpora

Leonel Merino
Software Composition Group
University of Bern
Bern, Switzerland
merino@iam.unibe.ch

Mircea Lungu
Software Composition Group
University of Bern
Bern, Switzerland
lungu@iam.unibe.ch

Oscar Nierstrasz
Software Composition Group
University of Bern
Bern, Switzerland
oscar@iam.unibe.ch

***Abstract*—When analysing software metrics, users find that visualisation tools lack support for *(1)* the detection of patterns within metrics; and *(2)* enabling analysis of software corpora.**

In this paper we present Explora, a visualisation tool designed for the simultaneous analysis of multiple metrics of systems in software corpora. Explora incorporates a novel lightweight visualisation technique called *PolyGrid* that promotes the detection of graphical patterns. We present an example where we analyse the relation of subtype polymorphism with inheritance and invocation in corpora of Smalltalk and Java systems and find that *(1)* subtype polymorphism is more likely to be found in large hierarchies; *(2)* as class hierarchies grow horizontally, they also do so vertically; and *(3)* in polymorphic hierarchies the length of the name of the classes is orthogonal to the cardinality of the call sites.

I. Introduction

Meet Edgar, an empirical software engineering researcher. In the past he has been interested in assessing the prevalence of reuse by inheritance and invocation of software in different languages [1]. Currently he wants to expand his analysis to subtype polymorphism [2]. He believes that although polymorphism can be an elegant solution for modelling variability, it can make systems more difficult to understand. Therefore, he wants to *(1)* understand how widely spread is the usage of polymorphism in systems written in Java — one of the most popular languages, compared to systems written in Smalltalk — one of the oldest object-oriented languages. He also wants to *(2)* find interesting systems for more detailed analysis. He realises that many research studies have used JHotDraw for their analysis [2], [3]. JHotDraw —a Java port of Smalltalk's HotDraw by Erich Gamma— is a great example of a well-designed system that was developed as an exercise in applying design patterns [4]. However, Edgar asks himself what other systems can also be good candidates for his analysis. The answer could be obtained by building a metric for measuring the occurrence of polymorphism. Instead, the answer is not that straightforward because *(3)* he also would like to know how inheritance and invocation relate to polymorphism. He wonders what characterises polymorphic systems; *(4)* is polymorphism most likely found in systems that have large class hierarchies?; *(5)* are polymorphic call sites more invoked than other entities of the system?.

This could represent an excellent candidate for applying the Goal-Question-Metric paradigm (GQM) [5]. GQM provides a three-step framework in which one must: *(1)* define a *goal*, *(2)* derive *questions* to determine if the goal is met, and *(3)* decide what *metrics* are suitable to answer the questions.

Indeed, Edgar has *(1)* a goal, namely he wants to understand how subtype polymorphism relates to inheritance and invocation in Java and Smalltalk systems; *(2)* initial questions, namely those posed previously; and *(3)* metrics that tackle attributes related to polymorphism, inheritance and invocation. However, since his goal is broad, he realises that his questions are rather incomplete. He wonders what other questions could be formulated that can be answered with those metrics. He therefore wants to measure various attributes of the software and compare their metric values to expose trends. Afterwards he might develop more precise hypotheses to explore.

A software corpus offers to Edgar diverse software systems suitable for his research. The analysis of software corpora of different languages allows researchers to understand not only the potential capabilities of the language but the way in which practitioners actually use it.

Additionally, software visualisation can provide support to Edgar for the analysis of multiple metrics. He could detect graphical patterns that can lead to the detection of relationships among metrics. The visual inspection that can be performed when visualising software can unveil information that could not be detected by statistical analysis, as seen in Anscombe's quartet [6].

In this paper we present Explora, a visualisation tool that encourages top-down exploration of relationships among multiple software metrics of software corpora. Explora is written in Pharo [7], an open-source Smalltalk system. It uses the Moose platform [8] for software analysis as well as the Roassal engine [9] for building visualisations. The users of Explora can analyse metrics defined on the FAMIX meta-model [10]. Explora shows aggregated results of multiple metrics from the top (systems) to the bottom (methods). It incorporates a novel visualisation technique that enables users to analyse many metrics of a system simultaneously, thus allowing users to define thresholds for understanding when a value is low, high or common.

978-1-4673-7526-9/15 © 2015 IEEE

Accepted for publication by IEEE. © 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

It combines Polymetric Views [11] with the Small Multiples technique [12] that boosts comparison. The main strengths of Explora are:

- *Discoverability*. It encourages exploration for discovering hidden relationships among metrics.
- *Corpora*. It is designed to analyse software corpora, which boosts repeatable analysis.
- *Flexibility*. It allows users to use custom metrics towards achieving their specific needs.

The remainder of the paper is structured as follows: Section II describes the preliminary tasks for the analysis; Section III introduces our lightweight visualisation technique; Section IV shows an analysis example; Section V presents related work; and Section VI concludes and presents future work.

II. Setting Up Explora

Edgar has used Explora in the past. He is aware of the workflow for setting up Explora for the first time.[1] Before Edgar begins the analysis, he defines the set of metrics that he wants to explore. After some minutes inspecting the documentation of the FAMIX meta-model, he realises that FAMIX does not provide specific metrics for the analysis of polymorphism. However, he can build a metric for that purpose based on the data stored in the model. Inspired by Class Hierarchy Analysis [13] he develops a metric called *polymorphismCardinality* (PC) that measures the degree of polymorphism of a call site. Basically for each method the metric counts top-down the number of times that method is overridden. Listing 1 shows the specifics of its implementation.

Listing 1. Smalltalk implementation of *polymorphismCardinality*

```
| class cardinality |
class := self belongsTo.
(class directSubclasses isEmpty or: [ class isStub ])
  ifTrue: [ ↑ 0 ]
  ifFalse: [
    self isOverriden ifTrue: [
      cardinality := 0.
      class allSubclassesDo: [ :sc |
        | submethod |
        submethod := sc methods select:
              [ :sm | sm signature = self signature ].
        submethod isEmpty ifFalse: [
          cardinality := cardinality + 1 ] ].
      ↑ cardinality ]    ifFalse: [ ↑ 0 ] ]
```

Secondly he specifies the pre-existing set of metrics that he wants to analyse. He selects metrics defined at the class level, such as *fanIn (FI), fanOut (FO), hierarchyNestingLevel (HNL), numberOfMethods (NOM), nameLength (NL), totalNumberOfChildren (TNOC)*, as well as metrics defined at the method level, such as *numberOfLinesOfCode (NOLOC), cyclomaticComplexity (CC), numberOfComments (NOC)*.

[1] http://scg.unibe.ch/research/explora

III. The PolyGrid

Explora is designed for top-down exploration. Users start analysing metrics at the highest abstraction level (system) through a visualisation technique called *PolyGrid*, which is designed for the analysis of several metrics simultaneously. A PolyGrid is built as a grid of polymetric views, called *PolyCells*.

A PolyCell, such as the one seen in Figure 1 (a), depicts a bird's-eye view of the corpora. A label on top of the PolyCell describes the metrics that are mapped to the position, height, width and colour intensity respectively. Note that position in this context refers to the order (left-to-right, top-to-bottom) in which rectangles are sorted. In the figure, TNOC (total number of children) determines the position; NL (name length) defines the height; NOM (number of methods) defines the width; and HNL (hierarchy nesting level) determines the colour intensity of rectangles. Each rectangle represents a different system of the corpora. In the figure, the mouse is over the rectangle representing the Chronos system, which is also highlighted in all the other PolyGrids. Systems are sorted (left-to-right, top-to-bottom) by the left-most metric listed in the label. This means that the rectangle on the top-left corner represents the system that has the highest TNOC value, while the rectangle in the right-bottom corner depicts the system that has the lowest value. The corpus to which a system belongs can be distinguished by the colour of the rectangle. Systems of Smalltalk corpus are green while system of Java corpus are blue. In Figure 1 (a), users can notice a colour gradient top-down that exposes a high correlation between the TNOC and HNL (the metrics mapped to the position and the colour intensity respectively). TNOC shows how large a class hierarchy is, while HNL measures its depth. This suggests that as hierarchies become larger they also become deeper, which seems not be a property related to the language of systems (rectangles of the same colour are scattered).

Notice that Edgar wants to analyse ten metrics but each PolyCell can only handle four. This means that there are $\binom{10}{4}$, or *5040* different PolyCells that can be generated potentially. Explora simplifies the analysis by setting the number of PolyCells in a PolyGrid to the number of metrics under analysis, as seen in Figure 1. In it, each PolyCell maps a different metric to the position of rectangles. The remaining three metrics are arbitrarily selected for promoting exploration. Edgar can generate a new PolyGrid, for exploring another part of the solution space, as many times as he wants. Explora provides support for traversing back and forth the history of views. Users can also save/load the history in/from a file encouraging users to share their findings.

IV. Analysis Example

Edgar analyses the visualisation shown in Figure 1. He detects some graphical patterns.

- *Cluster-based*: In PolyCells (c), (f) and (g) rectangles of the same colour are mostly clustered together. This

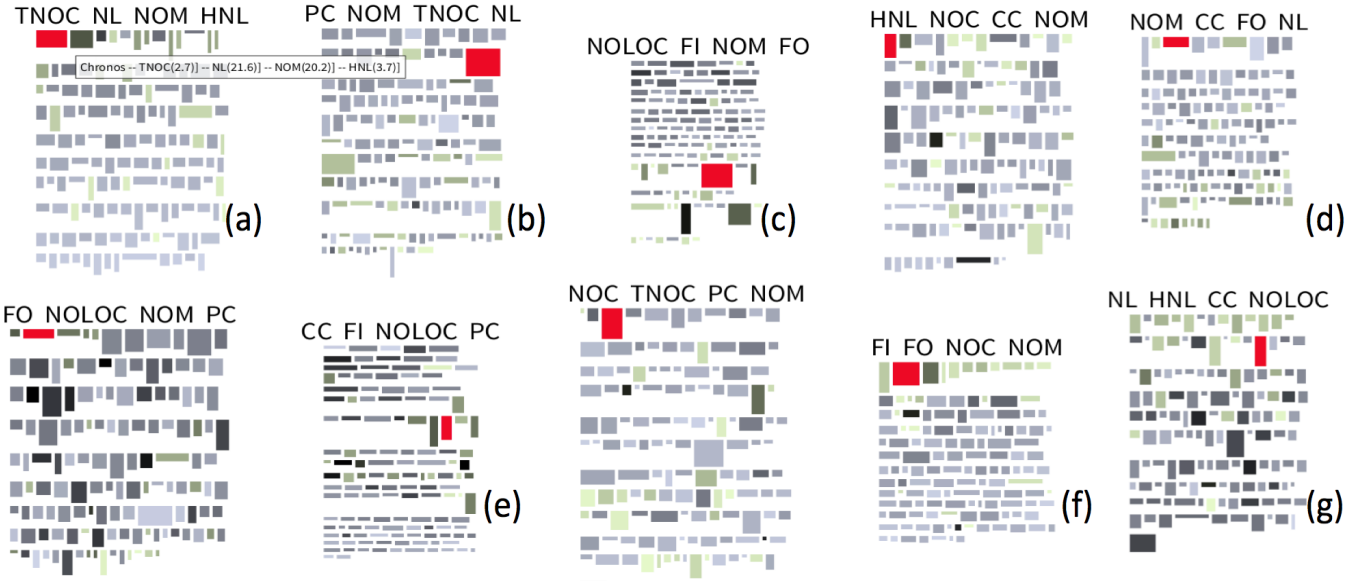

Figure 1. A PolyGrid for the analysis of 10 metrics on Java and Smalltalk corpora. A label on top of each PolyCell lists the metrics that are mapped to the position, height, width and colour intensity of a rectangle respectively. Note that each PolyCell maps a different metric to the position. Here the user has moused over the "Chronos" system in the first PolyCell, causing it to be highlighted in all the others as well.

pattern emerges when the metric mapped to the position of rectangles is related to the criteria used for choosing the systems of the corpora. In the example, the criterion is the language. This means that the values of the metric are similar for systems written in the same language. Examples of this pattern are detected in PolyCell (c) which suggests that methods in Smalltalk are normally written with fewer lines of code (NOLOC) than in Java; (f) which shows that most Smalltalk systems have higher values of FI than Java systems; and (g) which suggests that classes in Smalltalk systems have longer names (NL) than classes in Java.

- *Weight-based*: In PolyCells (c) and (f) larger rectangles are mostly at the bottom and top respectively. This pattern emerges when the metrics mapped to height and width of the rectangle are related to the metric mapped to position. An example of this pattern is PolyCell (c) which shows that systems where methods are shorter (lower values of NOLOC), classes usually have higher FI and/or have more methods (higher values of FI and NOM). Indeed Smalltalk systems have the highest FI values (green rectangles are at the top in PolyCell (f)). However, PolyCell (e) shows that Smalltalk systems are scattered and even some concentrate at the bottom, which means they have lower values of NOM.
- *High-Density*: In PolyCell (f) there is little space left between rectangles. This pattern occurs when the metric mapped to the height of rectangles is highly correlated to the metric mapped to the position. In (f) as the values of FI decrease the same occurs with the values of FO. Notice that this pattern would also occur in PolyCell (e) if Smalltalk systems (green rectangles) were removed. This suggests that in Java systems, as CC increases so does FI (median values).
- *Smooth-Gradient*: In PolyCell (a) a smooth colour gradient is perceived when traversing the rectangles top-down or bottom-up. This pattern reveals a high correlation between the metric mapped to the position with the one mapped to the colour intensity. In PolyCell (a) it suggests that TNOC and HNL might be highly correlated. This means that as class hierarchies become larger, they also become deeper.

Notice that there are PolyCells that reveal several patterns, such as the case of PolyCell (f) that exposes Cluster-based, Weight-based, and High-Density patterns.

A. Research Questions

Edgar revisits his research questions posed in Section I.

RQ1 Do highly polymorphic systems have large class hierarchies?

Figure 2 shows a collection of views of the top 5 systems with the highest median values of PC, plus systems ranked 16 and 17 (the Smalltalk system with the highest value and JHotDraw respectively). The views were collected by drilling down into the systems shown in Figure 1 (b). In the visualisation classes are depicted by rectangles which map four metrics PC, NOM, TNOC and NL to the attributes of the rectangle position, height, width and colour intensity respectively. The analysis suggests that classes that belong to large hierarchies tend to present more polymorphism (wider rectangles are found frequently at the top).

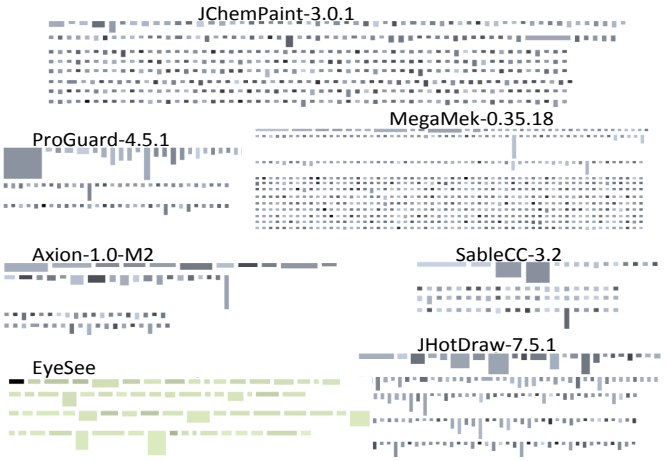

Figure 2. Drilling down into the top systems regarding PC. Rectangles depict classes of systems. The position, height, width and colour intensity of rectangles are mapped to PC, NOM, TNOC and NL respectively. Notice that wider rectangles are systematically at the top, which suggests a high correlation between PC and TNOC

RQ2 What other systems besides *JHotDraw* can be good candidates for polymorphism analysis?

Table I summarises the top five systems that have a higher aggregated value (using the median) of PC, plus systems ranked 16 and 17. The systems were collected from Figure 1 (b). Although these systems do not have huge hierarchies of overridden methods (with the exception of the *Weapon* class in *MegaMek-0.35.18*), they present a much higher frequency of highly overridden methods than *JHotDraw*, which for some analyses can represent more interesting systems.

Table I
SYSTEMS THAT HAVE THE HIGHEST VALUES OF POLYMORPHISMCARDINALITY (AGGREGATED USING THE MEDIAN) OF WHICH JHOTDRAW IS RANKED 17. FOR EACH SYSTEM IS DISPLAYED THE NAME OF THE ROOT CLASS OF THE LARGEST HIERARCHY OF POTENTIAL POLYMORPHIC METHODS.

Rank	Name	Median	Class	Max
1	JChemPaint-3.0.1	0.6	ChemObject	28.6
2	ProGuard-4.5.1	0.5	SimplifiedVisitor	8.6
3	MegaMek-0.35.18	0.5	Weapon	241.5
4	Axion-1.0-M2	0.5	BaseDataType	4.9
5	SableCC-3.2	0.4	Node	18.1
16	EyeSee	0.4	ESAbstractAxisPS	6
17	JHotDraw-7.5.1	0.3	AbstractBean	7.9

RQ3 Do polymorphic methods have longer names?

Neither Figure 1 (b), nor Figure 2 suggests a *Smooth-Gradient* pattern which would be expected if PC would relate to NL. Therefore, it does not seems to be a characteristic of highly polymorphic systems to consistently have longer method names. However, when analysing these metrics in the classes within a system there seems to be a relationship. In Figure 3 (left) classes of the system are depicted by rectangles. The position, height, width and colour intensity of rectangles are mapped to NL, PC, HNL and FI respectively. The analysis suggests that classes that expose higher values of PC (taller rectangles) tend to have shorter names (tall rectangles are mostly at the bottom). This could be explained by the fact that highly extended classes need longer names for describing more specialised entities. Notice that the tallest rectangles correspond to the root classes in the polymorphic hierarchies, which explains why there are only few of them.

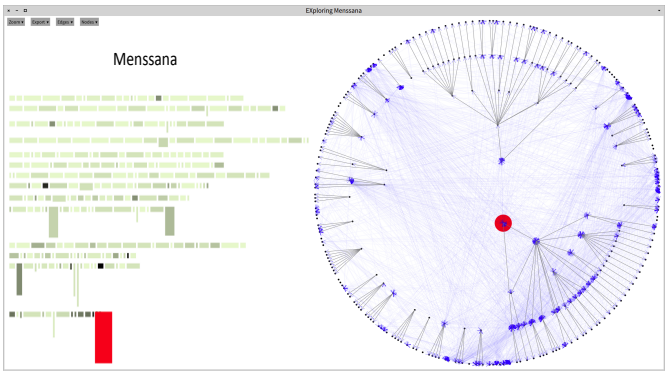

Figure 3. Details-on-demand of *Menssana* system. On the left classes of the system are depicted by rectangles. The position, height, width and colour intensity are mapped to NL, PC, HNL and FI respectively. On the right a dependency graph shows the same classes using a cluster layout following the class hierarchy (grey edges). Blue edges depict class dependencies. An inspection shows that classes in a higher level of the hierarchy are the tallest rectangles in the diagram of the left, such as *MenssanaConcern* the class highlighted in red.

V. RELATED WORK

Lange *et al.* developed *MetricViewEvolution* [14], a visualisation tool that allows users to analyse software metrics for managing model quality during development and evolution through an enhanced UML-like diagram. It implements a visualisation called *MetricViews*, which can map up to three software metrics to the colour, size and shape of the marks. MetricViewEvolution allows users to analyse only single systems. The analysis of metrics —constrained at the class level— is meant for improving the understanding of a system, instead of the discovering of relationships among metrics.

Garcia *et al.* [15] developed a tool for understanding software project structure, class relationships, class coupling, class level metrics and source code. It was designed for the analysis of one system at a time, which hampers repeating the analysis in other systems. Although it allows users to analyse metrics, it only does so at the granularity of methods and classes. The tool displays metric values as numbers as well as bar charts which hinders the ability to compare them. Moreover, it does not provide support for users to interpret the values of metrics.

Risi *et al.* [16] proposed a tool for the identification of fault-prone classes through the analysis of software metrics. They developed a visualisation technique that uses a mark composed of an ellipse and a rectangle that can map up to ten metrics to its graphical attributes. However, their tool can only cope with the analysis of one system at a time. Since

their focus is to detect fault-prone classes, users cannot analyse metrics at other level of granularity. Moreover, the tool does not provide flexibility for mapping a different set of metrics.

The Small Project Observatory developed by Lungu *et al.* [17] is a visualisation tool for the analysis of software ecosystems. A central view provides support for users to analyse several projects. In it, for each project a set of predefined metrics is shown (as numbers) such as *number of developers, number of commits*. This does not encourage comparison, and actually hinders the interpretation of metric values and detection of patterns.

In summary, existing tools lack support for reasoning about software source code through the construction and analysis of multiple metrics of software corpora. To the best of our knowledge there is no existing tool that *(1)* allows users to define custom software metrics; *(2)* supports the analysis of corpora; and *(3)* encourages the detection of patterns among metrics.

VI. Conclusion and Future Work

In this paper we introduced Explora, a visualisation tool for the simultaneous analysis of multiple metrics of software corpora. We elaborated on our novel lightweight visualisation technique called PolyGrid, which provides a bird's eye view of several corpora for discovering hidden relationships among metrics. We demonstrated the tool by analysing corpora composed of Java and Smalltalk systems. In the example we showed the flexibility of the tool by implementing a custom metric for the analysis of the cardinality of polymorphic call sites. We performed a top-down analysis showing the tool support at different levels of granularity. The analysis exposed that polymorphic call sites that have a higher cardinality (PC) are more likely found in large hierarchies (TNOC). Additionally, we found that systems that contains larger hierarchies (higher TNOC) also have deeper ones (higher HNL). Moreover, the analysis showed that in polymorphic hierarchies the length of the name of the classes (NL) is orthogonal to the cardinality of the call sites (PC).

We realise that the analysis of metrics in a software visualisation is very sensitive to their values (range and distribution). In the future we want to improve the visualisations by fine-tuning the normalisation of values. The distribution of the values of metrics can differ among languages and domain, therefore we believe that using a polynomial regression for normalising the values could provide more accurate analyses. Additionally, we want to improve Explora by adding features for performing statistical analysis that can complement the findings detected through the visual exploration. Finally, we believe that instead of choosing arbitrary metrics for complementing the one used to map the position, the PolyGrid could improve the exploration by mapping first metrics that have higher correlation.

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). Merino has been partially funded by CONICYT BCH/Doctorado Extranjero 72140330.

References

[1] L. Merino, M. Lungu, and O. Nierstrasz, "Explora: Infrastructure for scaling up software visualisation to corpora," in *SATToSE'14: Post-Proceedings of the 7th International Seminar Series on Advanced Techniques & Tools for Software Evolution*, vol. 1354. CEUR Workshop Proceedings (CEUR-WS.org), 2015, http://ceur-ws.org/Vol-1354/. [Online]. Available: http://scg.unibe.ch/archive/papers/Meri15a.pdf

[2] N. Milojković, A. Caracciolo, M. Lungu, O. Nierstrasz, D. Röthlisberger, and R. Robbes, "Polymorphism in the spotlight: Studying its prevalence in Java and Smalltalk," in *Proceedings of International Conference on Program Comprehension (ICPC 2015)*, 2015, pp. 1–10, to appear. [Online]. Available: http://scg.unibe.ch/archive/papers/Milo15a.pdf

[3] A. van Deursen, M. Marin, and L. Moonen, "A systematic aspect-oriented refactoring and testing strategy, and its application to JHotDraw," *arXiv preprint cs/0503015*, 2005.

[4] "JHotDraw: a Java GUI framework for technical and structured graphics," www.jhotdraw.org. [Online]. Available: http://www.jhotdraw.org

[5] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. Boston, MA, USA: PWS Publishing Co., 1998.

[6] F. J. Anscombe, "Graphs in statistical analysis," *The American Statistician*, vol. 27, no. 1, pp. 17–21, 1973.

[7] A. Black, S. Ducasse, O. Nierstrasz, D. Pollet, D. Cassou, and M. Denker, *Pharo by Example*. Square Bracket Associates, 2009. [Online]. Available: http://pharobyexample.org

[8] O. Nierstrasz, S. Ducasse, and T. Gîrba, "The story of Moose: an agile reengineering environment," in *Proceedings of the European Software Engineering Conference (ESEC/FSE'05)*. New York, NY, USA: ACM Press, Sep. 2005, pp. 1–10, invited paper. [Online]. Available: http://scg.unibe.ch/archive/papers/Nier05cStoryOfMoose.pdf

[9] V. P. Araya, A. Bergel, D. Cassou, S. Ducasse, and J. Laval, "Agile visualization with Roassal," in *Deep Into Pharo*. Square Bracket Associates, Sep. 2013, pp. 209–239.

[10] S. Demeyer, S. Tichelaar, and S. Ducasse, "FAMIX 2.1 — The FAMOOS Information Exchange Model," University of Bern, Tech. Rep., 2001.

[11] M. Lanza and S. Ducasse, "Polymetric views—a lightweight visual approach to reverse engineering," *Transactions on Software Engineering (TSE)*, vol. 29, no. 9, pp. 782–795, Sep. 2003. [Online]. Available: http://scg.unibe.ch/archive/papers/Lanz03dTSEPolymetric.pdf

[12] E. R. Tufte, *Envisioning Information*. Graphics Press, 1990.

[13] J. Dean, D. Grove, and C. Chambers, "Optimization of object-oriented programs using static class hierarchy analysis," in *Proceedings ECOOP '95*, ser. LNCS, W. Olthoff, Ed., vol. 952. Aarhus, Denmark: Springer-Verlag, Aug. 1995, pp. 77–101.

[14] C. F. Lange, M. A. Wijns, and M. R. Chaudron, "MetricViewEvolution: UML-based views for monitoring model evolution and quality," in *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on*. IEEE, 2007, pp. 327–328.

[15] J. García, A. G. Torres, D. A. G. Aguilar, R. Therón, and F. J. G. Peñalvo, "A visual analytics tool for software project structure and relationships among classes," in *Smart Graphics*. Springer, 2009, pp. 203–212.

[16] M. Risi and G. Scanniello, "MetricAttitude: a visualization tool for the reverse engineering of object oriented software," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*. ACM, 2012, pp. 449–456.

[17] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes, "The Small Project Observatory: Visualizing software ecosystems," *Science of Computer Programming, Elsevier*, vol. 75, no. 4, pp. 264–275, Apr. 2010. [Online]. Available: http://scg.unibe.ch/archive/papers/Lung09aSPO.pdf