MetaVis: Exploring Actionable Visualization

Leonel Merino, Mohammad Ghafari, Oscar Nierstrasz
Software Composition Group, University of Bern
Bern, Switzerland

Alexandre Bergel and Juraj Kubelka
PLEIAD, University of Chile
Santiago, Chile

***Abstract*—Software visualization can be very useful for answering complex questions that arise in the software development process. Although modern visualization engines offer expressive APIs for building such visualizations, developers often have difficulties to (1) identify a suitable visualization technique to answer their particular development question, and to (2) implement that visualization using the existing APIs. Examples that illustrate the usage of an engine to build concrete visualizations offer a good starting point, but developers may have to traverse long lists of categories and analyze examples one-by-one to find a suitable one.**

We propose *MetaVis*, a tool that fills the gap between existing visualization techniques and their practical applications during software development. We classify questions frequently formulated by software developers and for each, based on our expertise, identify suitable visualizations. MetaVis uses tags mined from these questions to offer a tag-iconic cloud-based visualization. Each tag links to suitable visualizations that developers can explore, modify and try out. We present initial results of an implementation of MetaVis in the Pharo programming environment. The tool visualizes 76 developers' questions assigned to 49 visualization examples.

I. Introduction

Software visualization can play an effective role to answer a number of questions that arise during software development. For instance, before *"refactoring a legacy software system"*, developers should know *"what are the dependencies of this code?"*. Obviously, a visualization on which developers can identify entities and trace dependencies would help them to prioritize the tasks that might require more effort.

Though existing visualizations are often characterized by the types of questions that they are well-suited to answer, based on our recent research on 65 design study papers in SOFTVIS/VISSOFT venues, each work introduces a new tool or technique [1]. That is, developers may need to explore a long list of existing visualizations to adopt the one that fits their needs. Consider the case of the Roassal visualization engine [2] available for Smalltalk. Although it provides 363 examples that developers can adapt, the examples belong to 36 different visualization categories that are categorized based on the addressed technique or feature rather than on development concerns.

We conjecture that the low adoption of visualization is a direct result of the difficulties that developers experience in searching for a suitable visualization. We believe that providing visualization support within IDEs and categorizing existing techniques in a way that maps to the certain needs for development tasks is very helpful for developers. We have performed a small experiment that supports our hypothesis. We instrumented the Roassal example browser to monitor the behavior of users who have installed Roassal recently, and thus have demonstrated their interest in adopting visualizations. Over the period of one month we collected the usage behavior of 58 anonymous users. They showed a trend that confirms our intuition. The top 10 users who browsed the highest number of examples had to traverse at least 5 categories on average (with a maximum of 13 categories traversed by a user who tried 60 examples) before they found an example of interest.

Nevertheless, little research has been carried out to fill the gap between existing software visualization techniques and their practical applications. For example, *Hassaine et al.* [3] proposed an approach for generating visualizations specifically for maintenance tasks. *Sfayhi* and *Sahraoui* [4] proposed an approach to derive interactive visualizations from descriptions of code analysis tasks. Their approach, however, required developers to use a domain-specific language to describe the task. *Grammel et al.* [5] studied how information visualization novices construct visualizations. They analyzed the usage of basic visualization techniques such as charts and scatter plots. Although these techniques provide limited support for the analysis of development concerns, they acknowledge the need for tools that suggest a potential visualization.

In this paper, we propose *MetaVis*, a tool for exploring visualization examples suitable to answer frequent development questions. MetaVis offers a tag-iconic cloud-based visualization to connect frequently recurring and meaningful words, called *tags*, retrieved from the collected questions to icons that represent visualization examples. The tool allows users to discover and adapt appropriate visualization examples with the help of tags that are relevant to their needs. We present initial results of integrating MetaVis into the Pharo programming environment [14]. Amongst 173 questions that developers frequently ask during software development, collected from related work, we assigned 76 of them to 49 suitable visualization examples selected from 363 examples in the Roassal engine. To ease the reproducibility of our research MetaVis and our data sets are publicly available [6].

The remainder of the paper is structured as follows: Section II describes our tool; Section III presents examples of analyses; Section IV discusses our findings; and Section V concludes and presents future work.

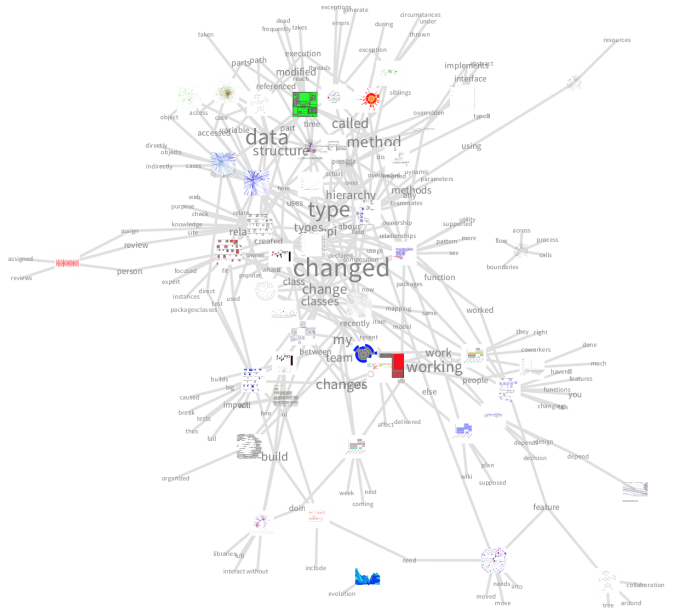

Figure 1. MetaVis visualization depicts *tags*, collected from frequent questions that arise during development, linked to *icons* of suitable visualization examples.

II. MetaVis

Figure 1 shows *MetaVis* visualization that is based on three main components: (1) a set of developer's questions, (2) a set of visualization examples, and (3) the relationships between the two sets. We now explain these components and elaborate on how the visualization supports users for their comprehension.

A. Developer's Questions

Developers often should answer several questions to perform a development task. Indeed, a complex task, such as *"refactoring a legacy software system"*, is broken down into some specific questions like *"what are the dependencies between these two packages?"*, *"who is the owner or expert for this code?"*, etc. Various researchers have mined, analyzed and thoroughly classified such questions. LaToza and Myers [7] surveyed 179 seasoned developers who answered *"what hard-to-answer questions about code have you recently asked?"*, and identified 91 types of such questions. Sillito *et al.* [8] collected 44 types of questions from two observational studies: in one study they interviewed 9 computer science graduate students, and in another, 16 industrial programmers. Fritz and Murphy [9] also interviewed 11 developers with varying expertise in industry, and gathered 46 types of questions.

We could identify 173 distinct questions from the aforementioned studies. Two authors of this paper (Merino and Bergel) studied these questions to identify those for which visualization represents a suitable means to reveal an answer. Each participant studied each question independently. In our experience, questions that aim at analyzing relationships among entities, comparing metrics and classifying entities using a certain criteria can benefit from visualization. At the end, we compared our results and discussed any conflict. We agreed that out of 173 questions visualization significantly helps to answer 76 of them (44%) like *"how big is this code?"*, *"where is this method called or type referenced?"*, and *"what classes have been changed most?"* just to name a few (the complete list of these questions is available online [6]). We excluded the 97 remaining questions (shown in Figure 2) for multiple reasons. We mainly excluded questions (1) already supported by tools part of the standard development environment, and (2) on which visualization is trivial and gathering the data represents most of the answer, labeled as *Trivial Visualization*, or on which the input data is not available (*e.g.*, assumptions, intent, policies), labeled as *Lack of Data*. We thus excluded (1) questions such as *"what are the arguments to this function?"* for which the *debugger* is appropriate, *"who made a particular change?"*, which can be queried in the *versioning control System*, or *"is this code tested?"* for which a *test coverage* tool will provide a more comprehensive analysis, and (2) questions such as *"what parameter values does each situation pass to this method?"*, *"how many recursive calls happen during this operation?"*, and *"why was it done this way?"*. Similar questions from other studies could be incorporated into our approach by expanding the set of related tags that represent a given development concern.

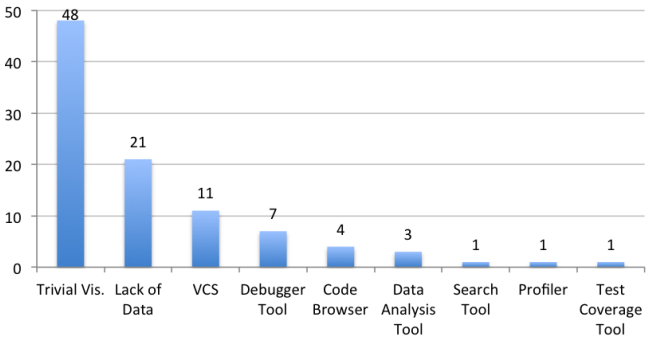

Figure 2. Classification of the 97 excluded questions.

B. Visualization Examples

We take the specific case of the examples that are shipped with the Roassal visualization engine. Roassal is a general-purpose visualization engine, which means that it is not limited to visualization of software concerns. It provides 363 examples that show novice users how various APIs can be used to obtain a certain visualization. The examples are organized into 36 categories (*e.g.*, *Color Palettes*, *Interaction*, *Tree map*). Users browse a category and see small screenshots of its visualization examples. Users can select an example, inspect its implementation and shape it to their needs.

We analyzed the 363 examples one by one. Although examples are not designed specifically for visualization of software development concerns, we found 49 that provide a useful starting point on which users can build visualizations to answer some of the questions identified in II-A.

Identifying which of dozens of questions relate to the actual need of a developer is a hard task. Consequently, MetaVis automatically split questions into frequently occurring and meaningful words (*e.g.*, verbs, nouns), called *tags*, that we manually relate to suitable visualization examples. In the following we elaborate on the visualization that we designed for their exploration.

C. TIC: Tag-Iconic Cloud-Based Visualization

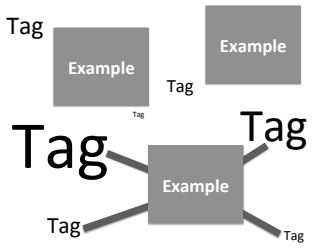

Figure 3. TIC wireframe composed of (1) tags from questions, (2) visualization examples, and (3) on-demand edges that connect tags and examples.

The TIC visualization follows Shneidermann's visualization mantra [10]: first users explore an *overview* of the cloud of development concerns to identify tags of interest, then they *zoom* into details of surrounding visualization examples, and finally they obtain *details-on-demand* by selecting an example that they can modify to fit their needs. Figure 3 shows the basic components of the TIC visualization: (1) tags that encode in their size how frequently they arise in the set of questions, (2) icons that represent visualization examples, and (3) on-demand edges that connect tags to their suitable examples. We use a force-directed algorithm [11] to lay out the bigraph of tags and icons. As a consequence, related elements are clustered together, thus revealing types of visualization techniques that are suitable to tackle the development concerns represented by the tags in the neighborhood. Edges are transparent to avoid cluttering. They are revealed on demand when users hover over a tag or an icon.

We chose the tag cloud technique to ease the comprehension of our visualization. Its popularity makes it self-explanatory. However, we reflected that in a tag cloud typically the positions of tags do not encode data. We decided then to group tags by development concerns. We expect that this will encourage users to discover suitable visualizations proposed for other needs within the concern.

The TIC visualization can also be used to tackle problems in other domains. We consequently classify it using the five dimensions proposed by Maletic *et al.* [12] to ease its reuse. The *task* tackled by our visualization is the exploration of appropriate visualization examples to answer development questions; the *audience* of this visualization are software developers who want to adopt visualization techniques for software analysis; the *target* data consists of a set of questions, a set of visualization examples, and a relation between questions and suitable examples for answering them; the *representation* is a tag-iconic cloud-based visualization that can be classified as *iconic-based* according to Keim's taxonomy [13]; and the *medium* used to display the visualization is a high-resolution monitor with at least 2560 x 1440 pixels.

D. Implementation

We realized a prototype tool implementation of MetaVis in Pharo. [14] The tool is based on the Roassal visualization engine and builds upon the GTInspector tool [15], which provides users with navigation and basic interactions (*e.g.*, zoom-in/out, pop-up, view center), and GTSpotter [16], which is used to search less frequent tags that can be difficult to find visually. *MetaVis* supports the following workflow: (1) users explore the cloud and select a visualization of their interest, (2) they inspect the associated code example and adapt it for their needs, and finally (3) they are able to put it into *action* and view the outcome visualization.

III. Analysis Example

In this section we present some sample questions from the literature, and show how MetaVis helps us to identify suitable visualizations to answer these questions.

A. Who is the owner or expert for this code? [7]

We observe that *owner* and *expert* are not frequent tags in our data set, hence their corresponding tags are difficult to find at first sight and require us to search for them. When we search for *owner*, two results *owner* and *ownership* are returned. Once we select the first tag, the visualization centers and highlights it. We then follow three steps shown in Figure 4 (top): *1*) we select one of the visualization examples that is linked to the selected tag (left pane); *2*) the code example of the selected visualization appears in the center pane. We modify the source code towards the analysis of code authorship. In particular, we add line 4 to collect all distinct authors of the set of classes, add lines 5-6 to create an object that returns a different color for each author, and modify line 7 to assign those colors to methods based on their author; *3*) we obtain a visualization (right pane) that shows classes with their methods colored according to their authors.

B. Where is this method called or type referenced? [8]

We identify two potential tags in this question: *method* and *called*. In Figure 4 (bottom) we show the sequence of steps performed. The visualization pane (left) shows the tags that we spot at first glance since they are quite common. We select one depicting a node-base diagram of the linked visualization examples and inspect its source code. Although the example already includes the main elements required in the analysis (classes, dependent classes, relationships), the number of edges depicted obstruct the

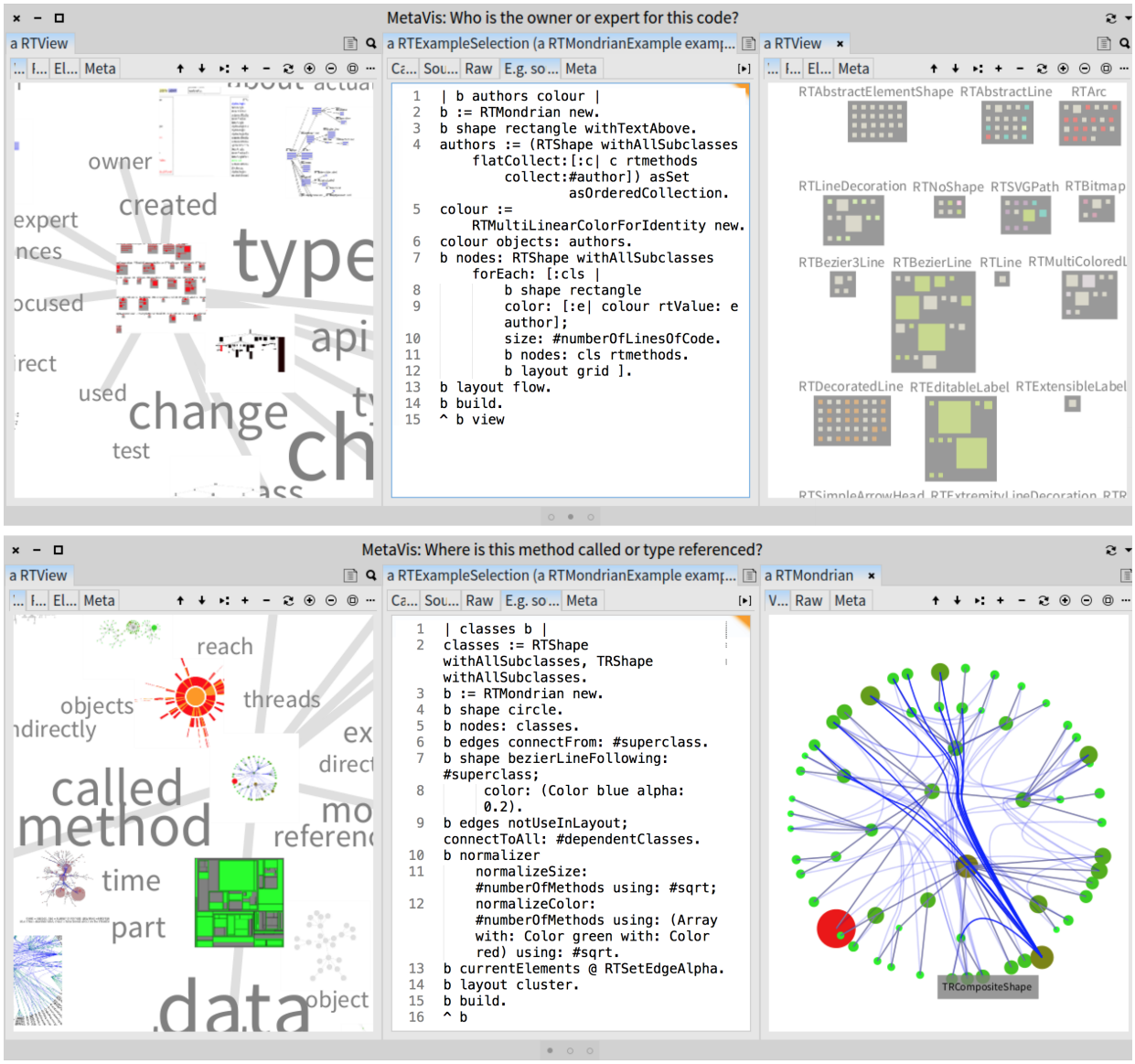

Figure 4. Two examples of the usage of MetaVis. On the top, we use it to answer *"who is the owner or expert of this code?"*. The left pane shows the exploratory visualization that links a visualization to tags retrieved from questions. In the example, we look for *owner*, select a visualization example and start modifying its source code (center pane) to identify the authors of the various methods of classes. The resulting visualization is shown in the right pane. At the bottom, we aimed at answering *"where is this method called or type referenced?"*. For this example we just needed to add interaction to nodes to highlight the outgoing edges representing dependencies.

analysis of dependencies of a particular class. We add interaction to the class nodes to highlight their dependencies when we hover over one of the classes.

IV. Discussion

During the analysis of questions that were good candidates for visualization, we identified three key groups of questions:

1) Relating Some questions sought to analyze *relationships* among software artifacts such as types, methods, objects, exceptions, and libraries. For example *"what depends on this code?"*, *"how are these types related?"*. We found that suitable visualizations for this group are based on node-link diagrams, parallel coordinates [17], and Sunburst [18].

2) Weighting Certain questions tried to *weigh* entities for comparison. Examples are *"how big is this code?"*, *"which part of this code takes the most time?"*. The visualizations that we found suitable for them were mostly based on simple charts, TreeMap [19], and Polymetric Views [20].

3) Identifying Other questions aim to *identify* entities such as software artifacts, or people involved in development tasks. Examples are *"who is using that API?"*, *"who implements this interface?"*. We recognize multiple visualization techniques suitable to tackle such questions, therefore we do not identify a particular preferred technique.

We observe that detecting what visualization techniques are frequently proposed to answer a particular group of questions (*e.g.,* relate, weigh, identify) suggests a future work direction on automating the process of visualization.

A. Limitations

A general limitation of *MetaVis* is bias in the choice and size of the set of development questions, in the set of visualization examples, and in the relationships between them. We mitigated these limitations by building the set of questions from relevant research in the field, collecting examples from a visualization engine developed by a highly active community, and discussing the relationships (manually assigned) between two authors of this paper. Regarding the *TIC* visualization technique, the size of the tags across multiple development concerns makes less frequent ones difficult to find visually. We observe that this issue can be mitigated by providing users with independent clouds for each development concern. Also the choice of words used to formulate the selected questions can affect the discoverability of development concerns; normalizing words and unifying synonyms could alleviate that issue.

V. Conclusion and Future Work

Although large numbers of visualization techniques have been proposed, and much research has investigated their effective use, little support is available for developers seeking a suitable visualization for their task at hand.

We have studied related work and have collected questions that programmers frequently ask during software development. We manually mapped these questions to suitable visualization examples. We designed a tag-iconic cloud-based visualization that relates frequent tags retrieved from questions and links them to appropriate visualization examples. Developers explore the cloud, identify important tags for their particular needs, and find suitable examples that they can customize.

We plan to (1) evaluate the tool with developers using a larger set of questions and enriched visualizations, and (2) investigate classifications of development concerns and suggested visualizations from the field towards automating the construction of visualization.

Acknowledgments

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Analysis" (SNSF project No. 200020-162352, Jan 1, 2016 - Dec. 30, 2018). Leonel Merino has been partially funded by CONICYT-BCH/Doctorado Extranjero/2013-72140330. Juraj Kubelka is supported by a Ph.D. scholarship from CONICYT, Chile. CONICYT-PCHA/Doctorado Nacional/2013-63130188.

References

[1] L. Merino, M. Ghafari, and O. Nierstrasz, "Towards actionable visualisation in software development," in *VISSOFT'16: Proceedings of the 4th IEEE Working Conference on Software Visualization*. IEEE, 2016. [Online]. Available: http://scg.unibe.ch/archive/papers/Meri16a.pdf

[2] 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.

[3] S. Hassaine, K. Dhambri, H. Sahraoui, and P. Poulin, "Generating visualization-based analysis scenarios from maintenance task descriptions," in *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*. IEEE, 2009, pp. 41–44.

[4] A. Sfayhi and H. Sahraoui, "What you see is what you asked for: An effort-based transformation of code analysis tasks into interactive visualization scenarios," in *Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on*. IEEE, 2011, pp. 195–203.

[5] L. Grammel, M. Tory, and M.-A. Storey, "How information visualization novices construct visualizations," *IEEE transactions on visualization and computer graphics*, vol. 16, no. 6, pp. 943–952, 2010.

[6] L. Merino. (2016) MetaVis. [Online]. Available: http://scg.unibe.ch/research/meta-vis

[7] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*, ser. PLATEAU '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:6. [Online]. Available: http://doi.acm.org/10.1145/1937117.1937125

[8] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. SIGSOFT '06/FSE-14. New York, NY, USA: ACM, 2006, pp. 23–34. [Online]. Available: http://people.cs.ubc.ca/~murphy/papers/other/asking-answering-fse06.pdf

[9] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 175–184. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806828

[10] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *IEEE Visual Languages*, College Park, Maryland 20742, U.S.A., 1996, pp. 336–343.

[11] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Softw. Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, Nov. 1991. [Online]. Available: http://dx.doi.org/10.1002/spe.4380211102

[12] J. I. Maletic, A. Marcus, and M. Collard, "A task oriented view of software visualization," in *Proceedings of the 1st Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002)*. IEEE, Jun. 2002, pp. 32–40.

[13] D. A. Keim and H.-P. Kriegel, "Visualization techniques for mining large databases: A comparison," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 8, no. 6, pp. 923–938, 1996.

[14] (2016) Pharo. [Online]. Available: http://www.pharo.org

[15] A. Chiş, T. Gîrba, O. Nierstrasz, and A. Syrel, "GTInspector: A moldable domain-aware object inspector," in *Proceedings of the Companion Publication of the 2015 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity*, ser. SPLASH Companion 2015. New York, NY, USA: ACM, 2015, pp. 15–16. [Online]. Available: http://scg.unibe.ch/archive/papers/Chis15b-GTInspector.pdf

[16] A. Syrel, A. Chiş, T. Gîrba, J. Kubelka, O. Nierstrasz, and S. Reichhart, "Spotter: towards a unified search interface in IDEs," in *Proceedings of the Companion Publication of the 2015 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity*, ser. SPLASH Companion 2015. New York, NY, USA: ACM, 2015, pp. 54–55. [Online]. Available: http://scg.unibe.ch/archive/papers/Syre15a-SpotterPosterAbstract.pdf

[17] A. Inselberg and B. Dimsdale, "Parallel coordinates," in *Human-Machine Interactive Systems*. Springer, 1991, pp. 199–233.

[18] J. T. Stasko, R. Catrambone, M. Guzdial, and K. Mcdonald, "An evaluation of space-filling information visualizations for depicting hier-

archical structures," *International Journal Humain-Computer Studies*, vol. 53, no. 5, pp. 663–694, 2000.

[19] B. Johnson and B. Shneiderman, "Tree-maps: a space-filling approach to the visualization of hierarchical information structures," in *VIS '91: Proceedings of the 2nd conference on Visualization '91*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 284–291.

[20] 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