Adaptable Visualisation Based On User Needs

Leonel Merino

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

Abstract. Software developers often ask questions about software systems and software ecosystems that entail exploration and navigation, such as who uses this component?, and where is this feature implemented? Software visualisation can be a great aid to understanding and exploring the answers to such questions, but visualisations require expertise to implement effectively, and they do not always scale well to large systems. We propose to automatically generate software visualisations based on software models derived from open source software corpora and from an analysis of the properties of typical developers queries and commonly used visualisations. The key challenges we see are (1) understanding how to match queries to suitable visualisations, and (2) scaling visualisations effectively to very large software systems and corpora. In the paper we motivate the idea of automatic software visualisation, we enumerate the challenges and our proposals to address them, and we describe some very initial results in our attempts to develop scalable visualisations of open source software corpora.

1 Introduction

A visualisation has the advantage that it can present in a comprehensive manner a great deal of information, which makes it a good candidate for software analysis. Analyzing many systems together enables the exploration of trends and comparisons in software evolution. The users of a visualisation, whether researchers or developers, may differ in their needs. Some of them are going to be interested in visualising software quality metrics, while others will want to explore software evolution or maybe to detect code smells. To understand their specific needs we will build a taxonomy of the user needs. The taxonomy should define the means that will be provided to the user for interacting with the visualisation (e.g. to inspect entities, to launch new visualisations based on the selected entity), and the type of information he is going to receive (e.g. method attributes, invocation sender). The taxonomy should produce a classification of user needs, such as most complex methods, extent of polymorphism, method invocations or class hierarchy length.

1.1 Automatic Software Visualisation

We want to provide a query mechanism to adapt automatically the visualisation to the specific user needs. The mechanism should provide means to specify the kind of attributes to be explored (e.g. the kind of entity, the associated properties, and the corresponding relations). For instance, we could specify that we want to visualise at a package level a view with the method invocations to others packages in the system. The mechanism should automatically select the most appropriate visualisation that satisfies our needs by choosing the right shapes, colours, layout and navigation. The visualisation should allow the user to explore and to interact with entities, to inspect them and to launch new visualisations based on the entity selected. For instance, in a visualisation that shows methods, classes, packages and systems, if we select a system it could provide means to launch a new visualisation with the evolution history of the versions of that system. Although related work supports this kind of interactions, they have to be specified manually.

1.2 Visualisations for Large Systems and Corpora

From a scalability point of view, one of the most difficult challenges in software visualisation is representing corpora of software systems. A corpus can contain hundreds of software systems, so the requirements on memory and processing power can be much higher than those for visualising individual software systems. Also the effort involved in generating the visualisation is higher: once a corpus is downloaded, we have to extract models from all the included systems. By querying attributes of the model we can extract facts about the systems. The visualisation should also provide means to navigate and to interact with those entities and relations.

2 Early results

As a first approach we wanted to cope with the visualisation of large systems and corpora. Open-source software fits very well this kind of analysis since it gives us access the source code of complete systems. Qualitas Corpus [4] and SqueakSource [2] are two available corpora. Pangea [3] is a tool that enables running language independent analyses on corpora of object-oriented software. It provides Moose [1] models for Qualitas Corpus and SqueakSource systems allowing us to create Moose images for every system and interact with them by running scripts. Qualitas Corpus comprises 112 open source Java systems, 58,557 classes, almost 15 MLOC and 754 versions, while SqueakSource contains 28 Smalltalk systems.

For our visualisation we needed a tool to depict many components in a comprehensive manner, in the sense that fine and coarse grained entities can be distinguished at a sight. Although there are many tools for creating visualisations of software, not many are suitable for large systems. Even fewer tools can support visualising many systems at the same time or more than one corpus at the time.

Figure 1 presents five systems of Qualitas Corpus: *AspectJ, ArgoUML, ANTLR, AOI, Axion* comprising more than 1.5MLOC. Since in this example we wanted to visualise the use of polymorphism at a fine-grained level, a TreeMap is an appropiate layout. We used red color to indicate the presence of polymorphism, this is done by an heuristic that marks as polymorphic the methods of interfaces that have more than one implementation. We colored every level of the hierarchy such as method, class and package. The intensity of the color is related to the number of lines of code involved in their polymorphic methods.

The main issues that we foresee are memory use and processing time. This visualisation took a bit more than 8 minutes to be rendered and required to load almost 300MB of Moose models. We implemented this as a proof-of-concept and we did not do any optimisation yet. Since our intent is to provide a visualisation of different corpora at the same time we will need to overcome these contraints.

Acknowledgments

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. Stéphane Ducasse, Michele Lanza, and Sander Tichelaar. Moose: an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems. In *Proceedings of CoSET '00 (2nd International Symposium on Constructing Software Engineering Tools)*, June 2000. URL: `http://scg.unibe.ch/archive/papers/Duca00bMooseCoset.pdf`.
2. Adrian Lienhard and Lukas Renggli. Squeaksource — smart monticello repository. European Smalltalk User Group Innovation Technology Award, August 2005. Won the 2nd prize. URL: `http://scg.unibe.ch/archive/reports/Lien05b.pdf`.
3. SCG: Pangea 2.0. URL: `http://scg.unibe.ch/research/pangea`.
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. `doi:10.1109/APSEC.2010.46`.

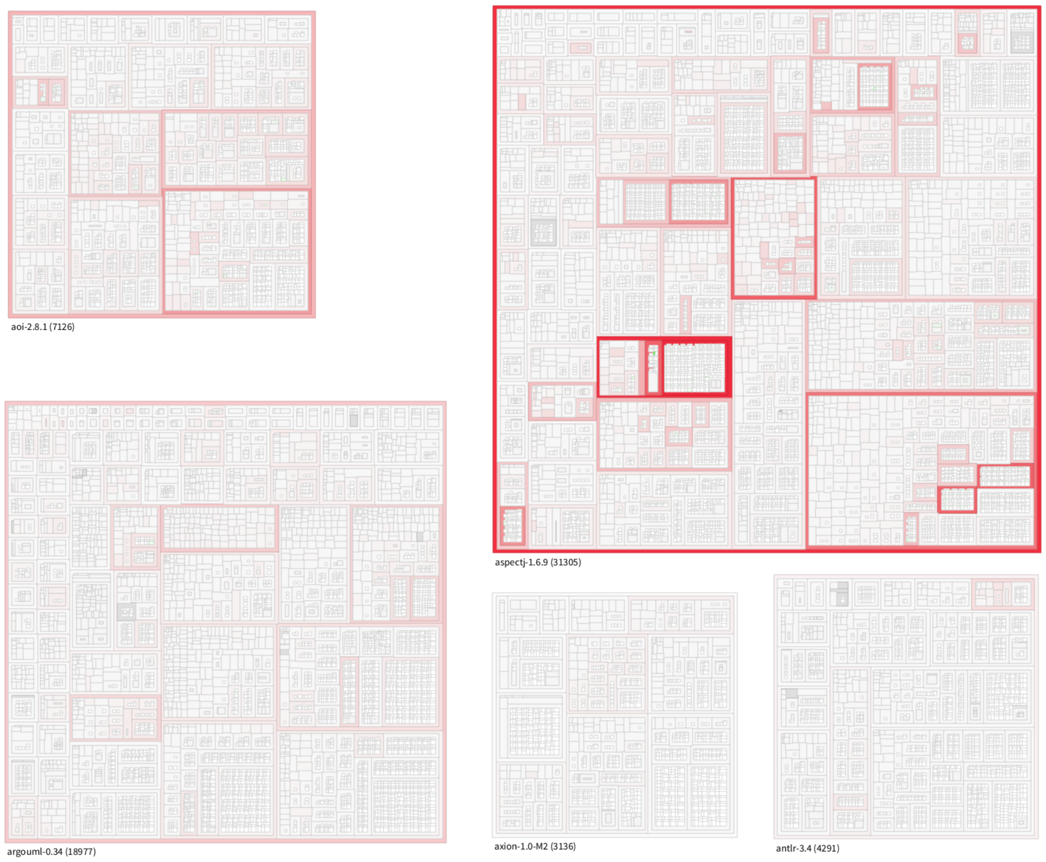

Fig. 1. Visualisation of an heuristic of polymorphism in five systems: *AspectJ, ArgoUML, ANTLR, AOI, Axion*. More than 1.5MLOC at a glance.