SOFTWARE VISUALISATION FOR OBJECT-ORIENTED PROGRAM COMPREHENSION

Michael J. Pacione
Department of Computer and Information Sciences, University of Strathclyde, Glasgow

Key words to describe the work: Software abstraction, program comprehension, software visualisation
Key results: Current software visualisation tools cannot address the full range of software comprehension tasks when used individually. The level of abstraction at which information is presented and the choice of statically or dynamically extracted information determine the tasks for which a tool is suitable.
How does the work advance the state of the art?: The aim of this research is to improve the effectiveness of visualisation techniques for large-scale software understanding based on a model with diagrams arranged in abstraction hierarchies, interrelated facets, and the integration of statically and dynamically extracted information.
Motivation (problems addressed): Previous research in software visualisation was often ad hoc in nature and focused on specific sub-problems of comprehension while failing to consider the goals of visualisation. Most tools are relatively tightly focussed, lack the capability to integrate statically and dynamically extracted information, and fail to address the difficulties in comprehension caused by inherent features of the object-oriented paradigm.

1. Context, problem, solution

Software visualisation is the process of modelling software systems for comprehension [3]. The comprehension of software systems both during and after development is a crucial component of the software process [4]. The complex interactions inherent in the object-oriented paradigm make visualisation a particularly attractive comprehension technique, and the large volume of information typically generated during visualisation necessitates tool support.

A recent study by the author [2] revealed that current visualisation tools address only specific software comprehension and reverse engineering issues. Most are relatively tightly focussed, lack the capability to integrate statically and dynamically extracted information, and fail to address the difficulties in comprehension caused by inherent features of the object-oriented paradigm.

In order to address the disadvantages with current visualisation techniques, an approach is proposed that integrates a formal model of abstraction hierarchies, structural and behavioural perspectives, and statically and dynamically extracted information.

2. Prior research

Prior research by the author [2] evaluated the performance of dynamic visualisation tools in a realistic software comprehension scenario. A range of available tools was evaluated by assessing their performance in a number of dynamic visualisation tasks. The tasks took the form of questions that an analyst would find it useful to be able to answer about a software system, such as ‘How do the high-level components of the software system interact?’ and ‘How does the state of an object change during an interaction?’.

The distinguishing properties of the tools in the study were the extraction, analysis, and presentation techniques of the tools. These properties help define the level of abstraction of a visualisation tool. Abstraction is the process of producing a simplified representation that emphasises the important information while suppressing details that are (currently) uninteresting, with the goal of reducing complexity and increasing comprehensibility. In this earlier work, an ordinal scale with which the level of abstraction of such tools (and also other tools, diagrams, and documentation) can be categorised was proposed. At the microscopic end of the scale, debuggers are representative of the lowest level of abstraction that a dynamic analysis tool can produce. At the opposite, macroscopic, end are tools that provide a broad overview of an entire software system at a high level of abstraction, such as aggregate information about object population, memory usage, load distribution, or deployment. The middle portion of the scale ascends from tools that illustrate method calls and returns, through tools giving an object- or class-level representation of the system, to tools that provide an architectural-level view of the system.

The study found that tools of similar abstraction levels using different extraction and presentation techniques were able to answer different questions. It also showed that the level of abstraction of a tool’s output was important in determining which questions the tool could answer. The results revealed that an abstraction level slightly below the midpoint of the scale was optimal in terms of answering the most questions.

On average, a tool could address only a third of the questions, and the most successful tool addressed just

over half. However, if all five of the tools in the study were used in combination, it should be possible to address almost all of the 15 tasks. This may imply that a combination of both statically and dynamically extracted information and a range of abstraction levels is required in order for a tool to perform well in all tasks.

3. Proposed solution

In order to combine the benefits of these alternative approaches, this research will investigate a multi-faceted, three-dimensional abstraction model for software visualisation. Similar to the abstraction scale proposed in our earlier work, the first dimension of the model will consist of an abstraction scale with a number of levels from microscopic to macroscopic. This arrangement allows the analyst to explore the software system at the level(s) of abstraction appropriate to the comprehension task they are undertaking.

The second dimension of the new model will consist of a number of *facets* [1], each representing an 'interesting' aspect of the system, e.g. behaviour, structure, or data. The use of interrelated facets allows the analyst to examine the structure, behaviour, or data of the software system individually or in combination, allowing them to focus the visualisation on the information appropriate to their query. Each abstraction level of each facet is a *view* and consists of a name, a description, a set of entities, a set of relationships between those entities, and a set of diagrams that illustrate software at that level of that facet. This arrangement will provide the analyst with a clearer view of the software under analysis.

The third dimension of the model will consist of static and/or dynamic analyses of the software. Static analyses examine the program code, while dynamic analyses monitor the system's runtime behaviour. Consequently, static analyses consider the entire software system at a less precise level of detail than dynamic analyses, which consider only a subset of the system in more detail. This third dimension allows the broad coverage of static analysis to be combined with the detail of dynamic analysis without their attendant disadvantages.

There are a number of key research challenges associated with this solution. One such challenge is the way in which the visualisation information will be stored as a model, and how this will be used to generate view hierarchies. Another challenge is the definition of the inter- and intra-hierarchy relationships between the views. Identifying which views are appropriate and useful for which comprehension tasks is a further challenge. The way in which statically and dynamically extracted information is combined and presented will also require investigation.

4. Contributions

It is proposed that this work will increase the utility of visualisation for software comprehension. The contributions of this research to date include: the five-level abstraction scale for software comprehension; the proposal and specification of a range of practical comprehension questions as a basis for tool evaluation; a critique of dynamic visualisation tools; and an initial model based on abstraction, facets, and the integration of statically and dynamically extracted information.

5. Future work

We are currently investigating the use of abstraction in software visualisation and in visualisation in general, techniques for integrating statically and dynamically extracted information, modelling techniques applicable to software, and effective methods for exploring and querying visualisations. A fully specified abstraction model to underpin visualisation across a full range of software comprehension activities will then be proposed, along with a formalism to describe the model, relate the various views, and allow the combination of information from different views. This model will be evaluated in a similar way to our previous work using typical software comprehension questions, possibly with multiple subjects and studies. The feasibility of practical tool support based on this model will be investigated.

References

[1] J.H. Jahnke, H.A. Müller, A. Walenstein, N. Mansurov, and K. Wong, "Fused Data-Centric Visualizations for Software Evolution Environments", *Proceedings of the 10th International Workshop on Program Comprehension*, IEEE CS Press, Los Alamitos, CA, 2002, pp. 187-196.

[2] M.J. Pacione, M. Roper, and M. Wood, "A Comparative Evaluation of Dynamic Visualisation Tools", *Proceedings of the 10th Working Conference on Reverse Engineering*, IEEE CS Press, Los Alamitos, CA, 2003, pp. 80-89.

[3] B.A. Price, R.M. Baecker, and I.S. Small, "A Principled Taxonomy of Software Visualization", *Journal of Visual Languages and Computing 4(3)*, Elsevier, Amsterdam, 1993, pp. 211-266.

[4] A. von Mayrhauser and A.M. Vans, "Program Comprehension During Software Maintenance and Evolution", *IEEE Computer 28(8)*, IEEE CS Press, Los Alamitos, CA, 1995, pp. 44-55.