Visualising Software – A Key Research Area

Claire Knight and Malcolm Munro
Visualisation Research Group,
Research Institute in Software Evolution.
Department of Computer Science,
University of Durham,
Durham, DH1 3LE, UK.
{C.R.Knight, Malcolm.Munro}@durham.ac.uk

Keywords: Software Visualisation, Metaphors, Program Comprehension, Software Maintenance

Abstract

The visualisation of software systems has long been investigated, primarily through the use of graph structures representing at most two aspects of the system under consideration. It has been claimed that such visualisation cannot be successful because of the size and complexity of software; this is true of purely graph displays. There are many auxiliary discussions, including areas such as Intelligence Amplification and Metaphors that are discussed in detail elsewhere. For more information the reader is referred to http://www.dur.ac.uk/~dcs3crk/workfiles/documents/tech-report-5-99.ps.gz. This position paper makes the case for three-dimensional visualisations that try to overcome these problems, from a starting point of Brooks' comments about software as unvisualizable and going via (unsuccessful) 2D attempts to visualise graphs. It then poses the question that if these 3D visualisations are successful is it not then true to say that software visualisation has the ability to be both successful and a useful tool.

1. Introduction

Visualisation has long been used to illustrate facts about programs and software systems graphically, but it is only in recent years that visualisation research has moved to consider the use of three dimensions. The consideration of structures other than graphs within these three dimensional spaces is an even newer concept.

The use of the more traditional graphs (for example call graphs) is not discredited by the 3D visualisation community, but seen more as one facet amongst many useful techniques. The sheer size and complexity of many modern systems (and even legacy systems after many years of evolution and maintenance) very often illustrate the shortcomings of the graph-structured approach because of the cluttered and confusing pictures generated.

Whilst using three-dimensions and/or other display techniques is not going to magically remove these problems, they do provide reason enough to investigate other visualisation styles and methods.

Proponents of the graph-based displays have been known to discredit such research purely on the basis of lack of empirical evidence as to the benefit of three dimensions. Whilst such evidence is in short supply in the software maintenance and program comprehension fields there is much psychological information to be found and reused from studies relating to graphics, colours, navigation and general information visualisation.

2. Software can't be Visualised!

In his paper *No Silver Bullet* [1], F. Brooks wrote

> *"Software is invisible and unvisualizable."*

and

> *"…software is very difficult to visalize. Whether one diagrams control flow, variable-scope nesting, variable cross-references, dataflow, hierarchical data structures or whatever, one*

feels only one dimension of the intricately interlocked software elephant. If one superimposes all the diagrams generated by the many relevant views, it is difficult to extract any global overview."

Some of these points are very valid, but if they can be overcome, or minimised, does that then mean that software visualisation is possible and effective?

Theoretically the answer is yes, but there are many considerations and concessions that must be made to make the visualisation effective at communicating information to the viewer.

In the above excerpt the visualisations referred to by Brooks are all graph based data sets, displayed visually as such. At the time Brooks was writing, visualising software meant displaying some information about (or some aspect of) the software in a graph structure. This now need not be the case with the advances in computer hardware and graphics technology. As can be seen in Figure 1, Brooks has a valid point about the problem of visualising complex pieces of software with graphs.

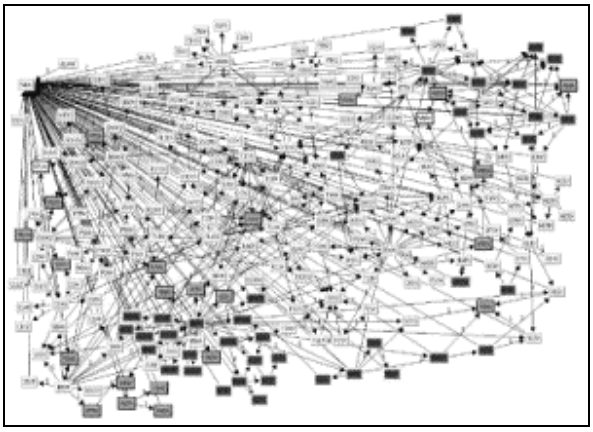

Figure 1 - Call graph of a commercial system

Figure 1 shows a call graph of a small well-maintained commercial system. If there is this much visual overcrowding of information from such a system then it is clear that new techniques for visualisation are needed.

Visualisation has the possibility of becoming a very powerful and much used tool by software engineers and maintainers. The following sections of the paper present information on three main areas of visualisation and then considers whether visualisation of software is feasible or whether Brooks' comments still hold true.

3. Visualisation Hypothesis

The premise of this paper is that visualisation is possible and effective. More detailed arguments supporting this premise can be found in [7] in which Knight and Munro consider not only Brooks comments, existing two dimensional and newer three dimensional approaches but also intelligence amplification and metaphors.

The definition of visualisation that will be used throughout the remainder of this paper is:

"Software visualisation is a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the existing software system under consideration."

This definition, with justifications for the various parts of it, as defined by Knight can be found in full in [6]. The basis for this definition is that the term visualisation should not be restricted to only computer displayable images, and that visualisation should have a purpose; i.e. to help users investigate complexity.

3.1 Abstract and Real World Visualisations

Much of the visualisation of programs done to date has been abstract in nature; geometrical shapes in either two or three dimensions, possibly of differing colours, have been used to represent elements of the program code. It has, however, been shown that simply transferring the traditional two-dimensional abstract graphs into three-dimensional space does not generate effective and usable visualisations in most cases [3].

Some more recent work on three-dimensional abstract visualisations moves away from these graph oriented displays and an example of this can be seen in Figure 2. This work is documented in more detail by Young [3], but briefly this shows the modules of the software system laid out around the calling structure of the program. On some of the modules more detailed information can be seen on top of the module blocks.

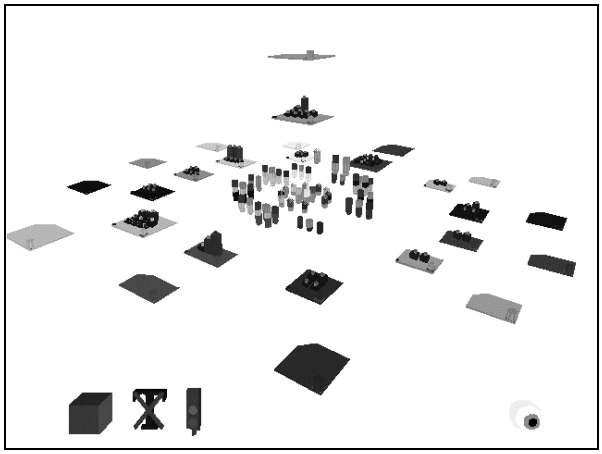

Figure 2 - 3D Abstract Visualisation

Real world visualisations rely on metaphors and mappings that translate between the data items and elements from the everyday human world. These data items are then visualised through the mappings to create a scene that does not, at first glance, appear alien to the human eye. The commonality of the scene then allows the thought and analysis to be directed to the underlying data and, for example, any incompatibilities between the scene and the real world can indicate certain properties about the underlying data.

An example of real world software visualisation can be found documented by Knight and Munro [2] and an example of this can be seen in Figure 3. The *Software World* visualisation uses a city metaphor to represent the software system and here the buildings represent methods, with the doors and windows on the buildings representing the number of local variables and the number of parameters to the method respectively.

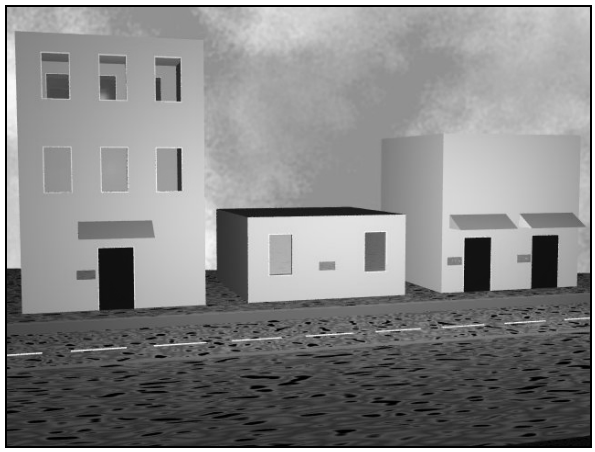

Figure 3 - *Software World* Real World Visualisation

Whilst these abstract and real world approaches differ in the representations used, they can be complimentary systems of visualisations. It is very likely that solutions to any visualisation problem will encompass some elements of both. The underlying data set can also have an effect on the visualisation style utilised. With program code, either style is acceptable because the software is intangible, but for more tangible data, the actual data values may indicate the style to be used. As Feijs and de Jong [5] note in their recent visualisation paper:

> *"But there is no such thing as the one right view;..."*

4. Visualisation is Possible!

Along with the comments presented in Section 2, Brooks [1] also writes

> *"A geometric reality is captured in a geometric abstraction. ... The reality of software is not inherently embedded in space. Hence it has no ready geometric representation in the way that land has maps, silicon chips have diagrams, computers have connectivity schematics."*

This tends to suggest that the use of metaphors, and hence real world mappings, are of no use because of the intangibility of software. But it is precisely this intangibility of software that makes the use of metaphors and abstractions such a powerful technique.

This is not to say that a successful visualisation mapping, and implementation of that mapping, is an easy task. There are many problems associated with visualisations of all forms. A basic one when visualising code is deciding on the best mappings between language elements and the metaphor. Knight and Munro [2] point out that one of the most difficult problems is in dealing with the possible range of data items to be visualised.

> *"The hardest problem in visualising anything is that, theoretically, the visualisation has to be able to deal with the range of items from one to infinity. This massive range means that automation and layout algorithms (both for two and three-dimensional visualisations) are hard problems. It also means that the visualisations need to be defined with this in mind; or to provide a way of dealing with very large numbers and indicating this fact to the user."*

This sizing issue can be a hindrance to any metaphor, but it is no reason to discredit the use, or investigation, of visualisations.

Later in his paper, Brooks [1], writes

> *"In spite of progress in restricting and simplifying the structures of software, they remain inherently unvisualizable, thus depriving the mind of some of it's most powerful conceptual tools. This lack not only impedes the process of design within one mind, it severely hinders communication among minds."*

This makes a link to both intelligence amplification, and also the power that metaphor is able to bring to visualisations.

From the information presented in this paper it is possible to conclude that software visualisation, especially with the newer techniques of three-dimensional non-graph based imagery, is indeed a feasible research area.

There have been a few forays into the use of visualisations of software that (a) make use of three dimensions and (b) do not explicitly draw any form of graph (or other node and arc) structure but it is this sort of research that should be encouraged. It is in doing this that the valid criticisms made by Brooks can be avoided and the complexity of the software system portrayed without hindering comprehension.

Acknowledgements

This work is partly financed by an EPSRC studentship.

References

[1] F. P. Brooks.
No Silver Bullet.
IEEE Computer, April 1987, pp10-19.

[2] C. Knight, M.Munro.
Comprehension with[in] Virtual Environment Visualisations.
Proceedings of the IEEE 7th International Workshop on Program Comprehension, Pittsburgh, PA, May 5-7, 1999, pp4-11.

[3] P. Young, M. Munro.
Visualising Software in Virtual Reality.
Proceedings of the IEEE 6th International Workshop on Program Comprehension, Ischia, Italy, June 24-26, 1998, pp19-26.

[4] K. Lynch.
The Image of the City.
The M.I.T. Press & Harvard University Press, Cambridge Massachusetts 1960.

[5] L. Feijs, R. de Jong.
3D Visualization of Software Architectures.
Communications of the ACM, December 1998, Vol. 41, No. 12, pp72-78.

[6] C. Knight.
Visualisation for Program Comprehension: Information and Issues.
University of Durham, Computer Science Technical Report 12/98.
http://www.dur.ac.uk/~dcs3crk/workfiles/documents/Lit_Survey_Tech_Reports/Tech_Report_12-98.ps.gz

[7] C. Knight, M. Munro.
Visualising Software – A Key Research Area.
University of Durham, Computer Science Technical Report 5/99.
http://www.dur.ac.uk/~dcs3crk/workfiles/documents/tech-report-5-99.ps.gz