Software Visualisation Conundrums

Department of Computer Science Technical Report 05/01, July 2001

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, Information Visualisation, Future Research

Abstract

Visualisation is compelling because of it has multidimensionality of presentation, data integration, and use. It facilitates understanding and analysis of complex data through opening up the visual perceptive channel as well as relying on the more standard forms of information location, assimilation and insight. The use of visualisation techniques to aid those who work with systems and software is one of the areas of complex and large data sets known to be candidates for successful visualisations. There are also the historical issues of software visualisation being exclusively linked to areas such as algorithm animation and for learning to debug. Software is complex, multi-faceted, large, and contains many relationships between its component parts. Therefore there are many aspects of software that may be appropriate for visualisation.

1. Introduction

Visualisation can support a multitude of presentation styles and integrate various data sources in order to create these presentations. The benefits of visualisation with large and complex data sources, including those of abstract data, are now well documented in the information visualisation field.

The software field has used various forms of diagrammatic support for many years. There are various algorithm animations used to teach the fundamentals of the underlying algorithms. There is the use of semi-visual displays to aid the process of debugging, and for viewing statistics related to dynamic program analysis. There is also the use of various notations for describing software and program code over time; Flow-charts, Jackson Structured Programming diagrams, Booch notation, and now the various diagrams of the Unified Modelling Language.

Unfortunately the concept of software visualisation being similar to information visualisation is an idea that only few have grasped, up until the last couple of years. Software is complex, large, and intangible; a perfect situation for the use of visualisations to make visible the invisible, to highlight the hidden, and the aggregation and abstraction to support varying amounts of data.

The concept of using information visualisation for visualising software is highlighted by the authors in Chapter 9 of Card et al. [2]. It has also been supported by some research that has moved away from the standard graph representations so familiar to those who work in program comprehension. This paper provides a very short history of software visualisation and then looks at some issues that still affect software visualisation. These issues and some questions that remain unanswered about the field are presented as areas of future research and the paper seeks to define software visualisation research over the next few years in conjunction with the recent advances made in information visualisation.

2. Software Visualisation; A Potted History

If the detail of what many consider to be the software visualisation field is reviewed, then this paper would

have to be much larger in size and also consist mainly of variations on the nodes and arcs theme. The other main older focus of software from a visualisation perspective is that of algorithm animation. Obviously there are some exceptions to this work, such as early uses of 3D graphics, or the profiling information that is received from dynamic analysis of code. What would seem a strange view to those who are in visualisation from other domains or applications is that this is still considered the state of the art by some. In fact some still hold the view that software visualisation is equivalent to algorithm animation and that to do not the latter means that it cannot be said to be software visualisation!

Software visualisation can be seen as a specialised subset of information visualisation. This is because information visualisation is the process of creating a graphical representation of abstract, generally non-numerical, data. This is done to amplify cognition [2]. This is exactly what is required when trying to visualise software. The term software visualisation has many meanings depending on the author, and as noted in the previous paragraph some have a restrictive view of what it can involve. For the purposes of this paper, software visualisation can be taken to mean any form of program visualisation that is used after the software has been written as an aid to understanding (i.e. it does not mean visual programming). More formally software visualisation can be defined as [1]:

> *"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."*

The goal of software visualisation is included in the above definition. To create a visualisation for no real purpose would be a pointless exercise. It has long been known that understanding software is a complex and hard task because of the complexity of the software itself. Therefore techniques that aid the programmer in his comprehension of an existing software system deserve research focus. Software visualisation aims to aid the programmer by providing insight and understanding through the graphical displays and views, and to reduce the perceived complexity through the use of suitable abstractions and metaphors.

2.1 Taxonomies

Early work often contained animation, or was thought to have to contain animation. This is reflected in the detail of the various classification and evaluation taxonomies developed in the early 1990s. Whilst such detail is beyond the scope of this article it is worth reviewing the major points of the three main taxonomies. Myers [5] identifies one of the first program visualisation taxonomies. In this taxonomy he makes the distinction that all the included systems use graphics to illustrate some part of the program after it has been written. This distinction is important because it makes clear that the taxonomy covers only program visualisation and not visual programming. Other authors are not so clear and even confuse the two terms.

Other authors have produced taxonomies of program visualisation and classified things in a different way to Myers [5]. Price et al. [6] produced a taxonomy of software visualisation systems that is based on six categories. In creating such a taxonomy the authors aimed to create a ``road map" of the research to the point when the taxonomy was created. The taxonomy created by Price is more detailed than that of Myers' and can be arranged into a hierarchical structure. This structuring was a deliberate move by the authors to allow for the taxonomy to be extended and revised as software visualisation (the term they use instead of program visualisation) systems evolved and matured. Another taxonomy is the one defined by Roman and Cox [7]. This is closer to the taxonomy of Price et al. [6] than the one of Myers [5] and some parallels can be drawn between the two.

2.2 Early Representations; Nodes and Arcs

For many years basic visualisation, based around simple boxes and lines, has been done in an attempt to be able to ease some of the cognitive overload caused by program comprehension. The problems with such visualisations is that they can very easily become incomprehensible by trying to force large amounts of information into a small space, relying solely on two-dimensions for the representations. Baker and Eick [9] acknowledge the problems of such approaches:

> ``*When applied to production-sized systems, routines for producing flow charts, function call graphs and structure diagrams often break because the diagram is too complicated. Or they produce displays that contain too much information and are completely illegible."*

Much effort has been spent on visualising programs in two-dimensions, with graph structures such as call-graphs being prominent. It is acknowledged that these forms of visualisation suffer when the number of, and relationships between, information is complex. The representations themselves can even become more

complicated than the code itself. An example of this can be seen in Figure 1. If this is the only thing that can be used to aid the comprehension of a well maintained medium sized commercial system, then it is obvious that something more is needed.

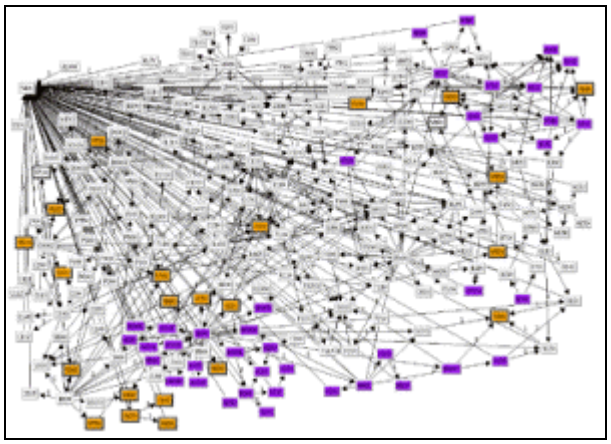

Fig. 1. **Call graph of a medium sized system**

In attempts to address the obvious problems with the existing representations several systems and layout algorithms have been developed. The following sections provide more detailed information on a representative sample of these. Another technique that has been applied, with little success due to the same problems still being prevalent, is the representation of node and arc structures such as call graphs in three-dimensions [3, 4, 14]. Much of this work was done without thought or design being applied to the virtual space in which the graphics were located which destroyed the possible usefulness of such an environment.

2.3 Newer Software Visualisation

Recently more attempts at utilising three dimensions for software visualisation have been carried out by Feijs and De Jong [14] where they visualised node and arc structures using variously coloured lego blocks for the nodes. These visualisations showed architectural structures and relationships present in the code and used various colours to indicate different pieces of information about that structure to the viewer. Whilst this paper acknowledged that they have more research to do the work does not seem to address many of the issues and limitations of the use of three dimensions. The creation of their visualisations relies again on nodes and arcs but using an extra dimension. This does provide a greater degree of flexibility than the two-dimensional form of such structures but again does not scale or evolve well; two very important issues.

A different style of representation that involves two-dimensional display but does not rely on the use of nodes and arcs was developed for the display of system and software information [9, 11, 12, 24]. These visualisations are part of a research effort that produced similar displays for several underlying data types. The visualisation technique used by these systems is based on the idea of decomposition of the information to be visualised into its component form. Colour and interaction are incorporated into the systems, and the displays make much use of colour scales to visualise extra information about the underlying data. The system also uses the overlay of additional information onto the display to provide yet more facts for the user.

SeeSys is a visualisation system for software metrics whilst SeeSoft visualises the program code and the constituent files. The visualisations work on the concept of providing overview but then contextual detail, through allowing individual components to be visible whilst maintaining a view of the whole system. This type of display is described as focus + context in Chapter 9 of [2].

In an attempt to move away from the obvious connectivity displayed in software visualisations Young and Munro [22] and Young [23] produced visualisations based around abstract three-dimensional geometrical shapes. The first, known as *CallStax*, showed the visualisation of the calling structure of C code (essentially the same information as a call graph) with coloured stacks of blocks showing the routes through the graph.

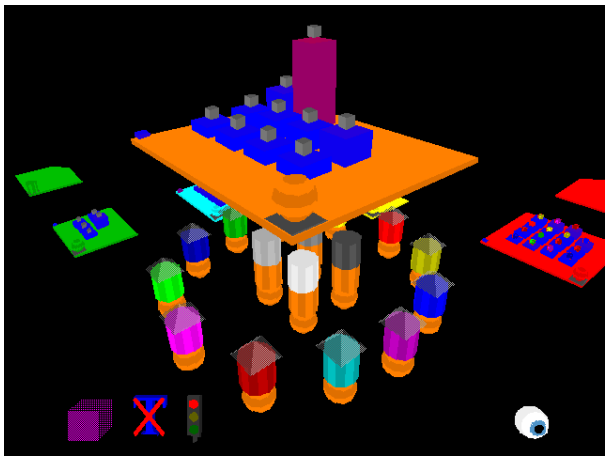

Fig. 2. **CallStax visualisation underneath part of a FileVis display**

The second, *FileVis*, showed a view of the software system with the code files represented individually as floating platforms around a central point which represented the connectivity of the source code files. A view of these visualisations can be seen in Figure 2. This visualisation combines both in order to show two aspects of the C code at the same time.

Fig. 3. **View over a software district showing many, possibly complex, methods**

Further advancing the three-dimensional space aspect of the visualisations, work by Knight and Munro [1, 20, 21] moved to considering the use of virtual reality environments for software visualisation. *Software World* was created to show that three-dimensions (in this case also showing the viability of real world metaphors) could be used to create automatable and scalable software visualisations. Buildings, cities, and also at the highest level atlas views, were used to represent Java source code. An example view of this visualisation can be seen in Figure 3.

In order to show some of the method level views that can be created with this visualisation over 17,000 lines of Java code (which together composed a package) written by others was parsed, and district visualisations automatically generated. The code was split across 70 classes, which provided different district views.

The use of environments allowed extensions of this work to consider the benefits and challenges of creating virtually inhabitable spaces where the visualisation provides a view of the data under consideration and facilitates communication and collaboration within that data by those working on the same tasks or having similar knowledge. Some of these further issues can be found in [25].

2.4 Summary

In his paper *No Silver Bullet* [13], Brooks wrote

> ``*Software is invisible and unvisualizable.*
> …
> *…software is very difficult to visualize. 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."*

At the time Brooks wrote this, visualising software meant displaying some information about (or some aspect of) the software in a graph structure. From what can be seen in Figure 1, he has a point. This need not now be the case with the advances in computer hardware and graphics technology.

For an overview of the software visualisation field Software Visualisation; Programming as a multimedia experience [26] shows how much of the prior focus has been on the use of algorithm animation techniques. The need to visualise the complexities of dynamic execution are covered, as well as some of the cognitive aspects of visualisation of software. There is very little reference made to the future work in the field, and the benefits that visualisation can bring if it is applied to software through the consideration of software as a complex artefact that can be decomposed into other complex artefacts. Visualising program code, and the various static and dyanamic analysis that program comprehension often employs have not yet found any universal solution and therefore these still require research, but there is also a need to consider software as information and therefore the sorts of visualisations of subsets of that information that would be appropriate for which users for which situations.

Two-dimensional techniques have shortcomings, but this is not to trivialise the issues that still remain with the nodes and arcs techniques. Layout, for example, is a hard problem and one that is not appreciated by many. The information visualisation field has shown that there is no longer the need for a reliance on nodes and arcs and solely investigating layout algorithms and clustering is not necessarily the only way forward, even with two-dimensions. This work is a stepping stone to moving onto more mature software visualisations.

3. Information Visualisation

Despite the identified problems with the traditional forms of software visualisation, it is not simply enough to throw the techniques of information visualisation at the source code and expect useful and usable visualisations to emerge. Information visualisation influences have produced many hierarchical and tree oriented visualisations, and some information landscape style visualisations [15, 16, 17, 18, 19] and these could have application for software visualisation. Indeed Hendley and Drew [3, 4] produced springs layout algorithms for nodes and arcs in three-dimensions, which bears a similarity to such research.

Notwithstanding these advances, there are general unsolved problems with three dimensional visualisations, such as scalability and evolution, which are important to all visualisations but paramount to software visualisation due to the complexity and changeability of the underlying data; the source code. There may also be issues that are pertinent to software visualisations that do not have much effect on information visualisations; database visualisation is a subset of information visualisation because of the nature of the data source being visualised. For a more complete review of information visualisation in three-dimensions, the interested reader is referred to the comprehensive survey by Young [10], and for a wider view of information visualisation to the more recent book edited by Card, Mackinlay, and Shneiderman [2].

As the editors note in Chapter 9 [2], the information visualisation field has promise for five reasons:

1. *It brings increased resources to the human in the form of perceptual processing and expanded working memory.*
2. *It can reduce the search for information*
3. *It can enhance the recognition of patterns*
4. *It enables the use of perceptual inference and perceptual monitoring*
5. *The medium itself is manipulable and interactive*

These very intentions are motivation enough for the analysis of any (abstract) data source; hence these provide a very apt definition of information visualisation. They also highlight why software and systems are suitable for candidates for visualisation at a level beyond that of the source code. Software consists of many objects; the relationships between and the context of which are complex. The use of anything to ease the problem of understanding and analysing this data can only be beneficial to all those who have a need to do such activities.

4. Software Visualisation Issues

Not all authors assume that software visualisation is a closed, solved, problem domain. Unlike those who believe software visualisation is equivalent to algorithm animation! Petre et al. [27] highlight this when they write about some of the cognitive aspects and aims of software visualisation.

> *"Are all aspects of software amenable to visualisation? Software visualisation is trying to find simplicity in a complex artefact (e.g. 1000 lines of code) to produce a selective representation of a complex abstraction."*

As with any piece of engineering, then the solution may not be to mimic the current state of the real world, and visualisation (both general information and software) need to be aware of this. Many software visualisations have tried to replace the diagrams that programmers have used on paper for years. By just producing electronic versions of them is not really augmenting the process. Indeed much of the benefit of creating such diagrams by hand is in the discovery of the information necessary to finish the diagram. It is not the answer so much that is important (although this can be significant) but what is learnt in going about finding that answer. There is also the issue that the full power of the visualisations is being lost if the only focus is only replicating what can already be achieved, even if the process has been speeded up or automated to some degree. The idea is to leverage human abilities with the processing that computers are best suited to. Therefore to support such intelligence amplifying activities it is necessary to consider what extra views, different dimensions, and so on, that any visualisation can provide.

Some of the issues that have been a concern for program comprehension over the years are still a concern for software visualisation. Indeed they may also be a concern for some information visualisations. These are evolution, scalability, navigation and interaction, automation, correlation, visual complexity, and finally metaphor.

4.1 Evolution

The only effective judge as to whether a visualisation can deal with evolution is to evolve the underlying data and then update the visualisation to reflect those changes. This is not the fast evolution that can be illustrated well with techniques such as animation, apart from an obvious speed-up using prior data. It is also not about being able to regenerate a visualisation from data. It is about being able to keep parts of the visualisation consistent whilst reflecting the data changes. This use of visual constants makes it easier to highlight changing areas of data, and the amount of change that data goes through.

Evolution is an important issue with visualisations of systems and software, since they are known to change in a variety of ways and for a variety of reasons. Implementation issues of this aside, it is important for the visualisation representations and metaphor to be able to support this; if it cannot happen logically within the constrained framework that the metaphor provides, it might as well not happen at all. This is because the changes involved visually would cause too great a cognitive effort on the part of the user to be beneficial. In these cases (a) it would be better to generate the visualisation from scratch and (b) question whether the visualisation tool is of use for the work tasks of the user. It may be that the answer to (b) is that yes it is useful, even with a complete regeneration, since there is adequate relearning time but this cannot be assumed.

4.2 Scalability

Scalability of visualisations is related to the ability of a visualisation to evolve. Again, the only way to answer the question of how scalable a visualisation is requires it to be tested with varying amounts of source data. Scaling could be considered to be an evolution of the visualisation, but since it depends each time on the base data, it is more of an issue with whether an initial development algorithm can handle a wide range of data sizes. A hard problem for designers of visualisations is that, on the whole, visualisations must be created to accommodate a very wide range of data. Essentially the visualisation has to be able to deal with one to an infinite number of items. Keeping this in mind during the development of the visualisations should enable them to scale better. It may be that some smaller visualisations developed for a very specific need, where the data is known to be limited, do not have to consider such scaling issues and this is quite acceptable. Just as long as when designing visualisations that can be applied to data that is known to vary in size and content this fact is borne in mind.

4.3 Navigation and Interaction

Navigation is important because it affects the usability of the visualisation. The visualisation should be designed and structured with navigation in mind. If navigational features are added as an afterthought it will then be hard to add the necessary paths and beacons. As Young and Munro [Youn98] write

> *"Well structured data terrain should also result in a more understandable layout and easier navigation".*

There are also guidelines for navigation and orientation that can be taken from city planning textbooks which indicate ways in which humans orient themselves in three-dimensional space.

Tied into navigation issues is the way in which any user of the visualisation is able to interact with it; to move around the landscape and to find the information they require must be as intuitive as possible to make people view visualisations as useful tools. Unfortunately for designers of visualisations all users have different wants and needs where interaction with computers is concerned. For this reason the more flexibility the system offers, the better. The ability for the user to have a degree of configuration is also likely to lead to the acceptance and use of the visualisation system.

4.4 Automation

A visualisation should be able to be generated from the data with minimal intervention. Configuration and preferences are acceptable because the graphics are still created in a fully automatic manner. User generation of visualisations may allow tweaking for that user but the resulting visualisation is then only really suitable for that person. The visualisation is then not really applicable to any visualisation aiming for consistent appearances between versions (releases of the code). It also prevents a visualisation system being used as a common frame of reference for discussion. In the creation of multi-user visualisation environments the freedom for users to create their own landscapes would also completely destroy the notion of having a shared workspace – all users would have their own environments and each one (apart from their own) would be unfamiliar to everyone else. There are some contexts where the aggregation of information within a visualisation (user generated) would want to be saved as *new information* but this can

be saved as an addition to the main data source and therefore available for all.

4.5 Correlation

Correlation of information can, in part, be done by the visualisation. Cross-referencing and relating visualisation components to the original data or documentation file is only a data processing task. This correlation of information within the visualisation to the various external data sources (which may have been used in the construction of the visualisation) is important because it allows the user to access the required data at a level of abstraction suited to their current needs. Being able to view the actual source code of a method (for example) will ultimately be necessary for implementing changes, and being able to correlate the visual cues with the underlying code is important.

4.6 Visual Complexity

Visual complexity is an issue that can be considered to be heavily subjective, although it is often discussed. An example is the often quoted 7±2 boxes in a nodes and arcs style diagram. The situation in a three-dimensional virtual environment is a slightly different issue, especially if the representations used are based on a real world metaphor of some description. In "natural" scenes the brain and visual processing of humans is very good at filtering extraneous information, whilst providing overall scene information and the detail of the item being focused on.

The issue of low visual complexity (in three-dimensions) being a trade off against the complexity of the representations used is a case of simplification where it is hard, if not impossible, to make such a simplification. There are several questions that remain unanswered in such discussions. It is important to ask at what level the trade off is made. It can be suggested that it depends on the scoping, level of abstraction and the "zoom" level of the visualisation. It is also important to consider what is meant by low visual complexity in three-dimensions. This can be very user dependent, but certainly all humans have the power of visual filtering. Without it they would be immediately overwhelmed with information the moment they opened their eyes. If the data demands (apparently) complex representations for ease of long range and also close identification and for comparison then surely this must be considered more important. Or it could suggest that the metaphor used to guide the representations is either not powerful enough or not adequate for the task in hand.

There are also several factors that uniquely relate to how a user or evaluator judges whether a scene is of low visual complexity.

1. Whether or not three-dimensional visualisations are considered acceptable forms of visualisation.
2. Visual tolerance.
3. Experience – knowledge base built up through every experience that has happened to that user.
4. Possible prejudices (in this case relating to colour, imagery and metaphor) instilled by parents, friends and experiences. Prejudices can help shape the "knowledge base".

It may actually be the case that the scene requires visual complexity (from a design and construction viewpoint) in order to appear ordinary and non-complex to the user. Since humans filter everyday information then a scene missing this information becomes unusual, more visually complex and hence requires a greater effort to understand and navigate in.

4.7 Metaphor

All software visualisation systems make use, in some way or another, of a metaphor that acts as a mapping between the visual components used in the realisation of the visualisation and the underlying code. The design of this metaphor can greatly influence the usability of the visualisation. It is also true that the code being represented (or the data for information visualisation) can influence the metaphor; it may be that only certain metaphors are appropriate for certain data sets.

Not only does the data set being visualised influence the visualisation based on whether or not there are already tangible representations of this data in the real world, but the form and classification of that data can also affect it. With information visualisation, the graphics are trying to represent something that is inherently intangible so that many metaphors are appropriate. Whilst this free reign with representations exists, the underlying structure of the data, and the purpose to which the visualisation will be used, both affect what could be considered as suitable solutions to the visualisation of that data.

4.8 Unanswered Questions

Based on the software and information visualisation literature and from experience of software visualisations the following questions are deemed to be important, and still unanswered. The questions are not based around "there is a need for a visualisation to do x" but at a more thought provoking level, although it may be that answers

are best formulated during visualisation creation and refinement.

1. What models are being represented?
2. What data is being represented?
3. What kind of tasks are being supported?
4. Will the nature of the tasks change with the introduction of the visualisation?
5. Is there likely to be a "universal" representation and/or metaphor?
6. Should the "universal" visualisation actually be an aim, or is the acknowledgement of visualisations being suited to specific tasks and/or data styles more important?
7. Are all pieces of information associated with software acquiescent to visualisation?
8. Is this the same as saying that all aspects of software are acquiescent to visualisation?
9. How should evaluations of (software) visualisations develop; the evaluation issues of graphics, cognition, and usability are currently disparate areas [8].
10. What new visualisations are waiting to be discovered for systems and software?

This section provides much of the rationale for the next section on future research. The questions and issues posed form the basis of much of the research still to be done.

5. Key Future Research Areas

The focus of future software visualisation research ties with the above issues and that for information visualisation proposed by Card et al. [2] in Chapter 9 where unsolved problems of information visualisation are presented. Indeed Question 9 of the previous section draws directly on the first point of those authors. The benefit of cross over of information and software visualisation is that in trying to solve the problem of *bringing science to the craft* [2] the solutions most likely apply to both fields.

Obviously the questions presented in the previous section are seen as being important aspects of software visualisation research over the next few years. The issues presented may be solved for specific cases, but more general acceptance that these are issues and that they can have a profound effect on a visualisation and its use is necessary. The software visualisation field is slowing starting to mature as the information visualisation one has done and it is necessary to continue this process to ensure that software visualisation can offer the understanding and analysis benefits they have so long promised.

There are new visualisations to be found. These will come from answering the ten questions (of which new visualisations is one), through addressing the highlighted issues, and through better understanding the data and tasks that are important for those working with systems and software.

The issues and questions are important, but so are the systems and software being produced. These will influence what information is available to be visualised, and also what is considered desirable and/or necessary in a visualisation. The speed with which the software development community has moved to enterprise computing has meant that there is now a whole host of data previously un-represented that has become some of the most important to any organisation. This provides great potential for exploring and expanding the software visualisation field.

6. Conclusions

Software visualisation has the unique feature of being cyclic. The techniques utilised in a software visualisation can be used on the visualisation itself. It is only program code that enables the visualisations; just the source data that is being used for the visualisation. After all, how many techniques can be said to be applicable to themselves?!

There is, in many ways, a need to make more explicit some of the challenges that remain for software visualisation and this paper has tried to do just that. It is also necessary to be able to highlight where cross over of ideas and results between information and software are likely to lead to congruent results. Questions that aim to direct the future research of software visualisation have been presented to illustrate where it is perceived that challenges remain. These are related to the previous software visualisation research and results, and also some pertinent issues that visualisations need to consider.

The future for visualisation is strong given the amount of data currently flowing around daily life. This information is growing, as is the complexity within it. The same is true of software. Trying to remove the complexity from software is likely to mean that the systems created are unable to carry out the complex tasks expected of them. Instead it is better to acknowledge that complexity and size and work with it to try and overcome the understanding and analysis problems these cause. Visualisation can provide this amplification and should therefore be considered to be

an important part of the future of software systems understanding, analysis, and management.

Acknowledgements

This work is financed by an EPSRC ROPA grant: VVSRE; **V**isualising **S**oftware in **V**irtual **R**eality **E**nvironments.

References

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

[2] S. Card, J. Mackinlay, and B. Shneiderman (Editors), *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann, February 1999.

[3] B. Hendley and N. Drew, Visualisation of complex systems, University of Birmingham (UK) School of Computer Science, 1995.

[4] R. J. Hendley, N. S. Drew, A. M. Wood, and R. Beale, *Narcissus: Visualising Information*, University of Birmingham, Advanced Interaction Group Technical Report, 1995.

[5] B. A. Myers, *Taxonomies of Visual Programming and Program Visualization*, Journal of Visual Languages and Computing, Vol. 1, pp97-123, 1990.

[6] B. A. Price, R. M. Baecker, and I. S. Small, *A Principled Taxonomy of Software Visualization*, Journal of Visual Languages and Computing, Vol. 4, No. 3, pp211-266, 1992.

[7] G. C. Roman and K. C. Cox, *A Taxonomy of Program Visualization Systems*, IEEE Computer, pp11-24, December 1993.

[8] C. Knight, *Visualisation Effectiveness*, Workshop on Fundamental Issues of Visualisation, Proceedings of CISST, 2001.

[9] M. J. Baker and S. G. Eick, *Space Filling Software Visualization*, Journal of Visual Languages and Computing, Vol. 6, pp 119-133, 1995.

[10] P. Young, *Three Dimensional Information Visualisation*, University of Durham, Computer Science Technical Report 12/96, 1996.

[11] T. Ball and S. G. Eick, *Software Visualization in the Large*, IEEE Computer, pp33-43, April 1996.

[12] S. G. Eick, *Maintenance of Large Systems*, Chapter 21 of Software Visualization : Programming as a Multimedia Experience by John Stasko (Editor), Blaine A. Price (Editor), Marc H. Brown (Editor), MIT Press, February 1998.

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

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

[15] J.D. Mackinlay, S. Card and G.G. Robertson, *Perspective Wall: Detail and Context Smoothly Integrated*, Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems, pp. 173-179, April 1991.

[16] G.G. Robertson, J.D. Mackinlay and S. Card, *Cone Trees: Animated 3D Visualizations of Hierarchical Information*, Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems, pp. 189 - 194, April 1991.

[17] K.M. Fairchild, *Information Management Using Virtual Reality-Based Visualizations*, in ``Virtual Reality: Applications and Explorations'', Alan Wexelblat (ed.), Academic Press Professional, Cambridge, MA, pp. 45-74, 1993.

[18] M. Hemmje, *A 3D Based User Interface for Information Retrieval Systems*, Proceedings of IEEE Visualization '93, Workshop on Database Issues for Data Visualization, October 25-29, 1993.

[19] M. Hemmje, C. Kunkel and A. Willet, *LyberWorld - A Visualization User Interface Supporting Fulltext Retrieval*, Proceedings of ACM SIGIR '94, 1994.

[20] C. Knight and M. Munro, *Visualising Software – A Key Research Area*, Proceedings of the IEEE International Conference on Software Maintenance, August 30 – September 3, 1999. (Short paper.)

[21] C. Knight and M. Munro, *Virtual but Visible Software*, Proceedings of the IEEE International Conference on Information Visualisation, July 2000.

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

[23] P. Young, *Visualising Software in Cyberspace*, PhD Thesis, University of Durham, October 1999.

[24] S. G. Eick, *Engineering Perceptually Effective Visualizations for Abstract Data*, In Scientific Visualization Overviews, Methodologies and Techniques, IEEE Computer Science Press, pp191-210, February 1997.

[25] C. Knight and M. Munro, *Should Users Inhabit Visualisations?* in Proceedings of Knowledge Media Networking workshop of IEEE WETICE, June 2000.

[26] J. Stasko, B. A. Price, and M. H. Brown (Editors), *Software Visualization: Programming as a Multimedia Experience*, MIT Press, February 1998.

[27] M. Petre, A. Blackwell, and T. Green, *Cognitive Questions in Software Visualization*, Chapter 30 of Software Visualization : Programming as a Multimedia Experience by John Stasko (Editor), Blaine A. Price (Editor), Marc H. Brown (Editor), MIT Press, February 1998.

[28] C. Knight, *Virtual Software in Reality*, PhD Thesis, University of Durham, June 2000.