Visualisation for Program Comprehension: Information and Issues

Claire Knight

Visualisation Research Group,
Centre for Software Maintenance.

Department of Computer Science,
University of Durham,
Durham, DH1 3LE, UK.

E-mail: C.R.Knight@durham.ac.uk
Web: http://vrg.dur.ac.uk/

Computer Science Technical Report 12/98 – October 1998

Abstract

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 themselves by trying to force large amounts of information into a small space, relying solely on two dimensions for the representations and because generally the information is presented in a static way. There are many different issues that must be considered for three-dimensional visualisation, especially when those visualisations are of intangible software systems, and a selection of these are presented and discussed.

1. Introduction

Software visualisation is an important tool in the maintainer's armoury. It is a technique that can, when designed and used effectively, aid him in his quest to understand existing program code. This document provides an introduction to software visualisation issues, with program comprehension influencing the areas considered.

Visualisation has come to the forefront of scientific investigation in recent years because it relies on the use of graphical machines. For many years it was not feasible to find, or even expect to find, such equipment on everyone's desks. With the advancements made in hardware and corresponding falling costs, visualisation has the potential to become a very powerful tool.

For many years the focus of software visualisation has been two-dimensional images. Various aspects of the program code and metrics calculated from the code have been displayed in 2D. A common display was, and still is, based around graph structures. There is now a move away from such structures in an attempt to solve the inherent layout problems, but much work remains to be done in this area.

2. Visualisation

An area of computer science that is often confused with visualisation is that of visual programming. These are two distinct areas. The use of a visual programming language/tool is to aid the programmer in the creation of user interfaces (and in some situations, software). Essentially the visual objects are symbols that represent some item of code that would otherwise have had to be written by hand. Visualisation in the other hand is something that uses existing (possibly complex) data sets and tries to make them more understandable.

It could be said that visual programming is just one small area of visualisation. The use of graphics and imagery is as an aid to the programmer albeit in creation rather than understanding the system. But what is being made more accessible through the images is the programming language. Despite this link visual programming and program visualisation are not the same thing and the terms are not interchangeable.

Visualisation can be used with computer science in the same ways it is being used in other disciplines. Because part of computer science research is concerned with how to implement visual systems and how to create the hardware to make virtual reality useful, it does not prevent the field from using its own tools.

2.1 What is Visualisation?

Visualisation has been defined in several ways. One definition is:

> *"Visualization is the use of computer-generated media based on data in the service of human insight/learning."*
>
> *comp.viz.faq – comp.visualization newsgroup FAQ*

There are grounds for dispute in this definition. Human beings are visualising things all the time in thought and action. The term visualisation should not be restricted to those images or displays that a computer can present.

Another definition from the same source follows the same line of thought but it is included because of the other part of the definition.

> *"Visualization: the use of computer imagery to gain insight into complex phenomena"*

The last part of this definition is something that is very true. Many hold the view that one of the great uses of visualisation is to provide help for humans when dealing with complex ideas and concepts.

A better definition from the FAQ file is:

> *"The purpose of visualization is insight, not virtual realities or pictures."*
>
> *Eugene N. Miya, President, Bay Area ACM/SIGGRAPH*

But this does not go far enough.

Friedhoff and Benzon [Frie91], write (p16):

> *"... is that much is to be gained by recognizing that visualization, because of the computer, is emerging as a distinctive new discipline."*

This brings in again the reliance on computers but it does not rule out other forms of visualisation. The point is that the technological advances have brought visualisation to the forefront. Having done this and providing such a rich vein of opportunity for exploring visualisation, computers do not have to be the only form of visualisation. In a later paragraph (p16) the authors write:

> *"If this new discipline of visualization is to realize its full potential, however, it will also have to borrow from those areas traditionally concerned with imagery such as art history and perceptual and cognitive psychology. The field of visualization should not be so absorbed by the miracles that are its technical basis that it ignores a larger interest in the way in which images can be used to enhance the power of our thinking."*

A better definition of visualisation is one that encompasses several of these ideas. One such definition could be:

> *"Visualisation is a discipline that makes use of various forms of imagery to provide insight and understanding and to reduce complexity of the phenomena under consideration."*

This is a general definition that covers all aspects of visualisation as a discipline rather than concentrating on any specific area. In situations where more clarification is needed then qualifying statements can be added to the definition.

2.2 Why 3D?

It is often said that a picture is worth a thousand words. What suitable numerical value can then be attached to 3D images? If a 2D picture is worth a thousand words and can easily and, one would hope, clearly explain the same, then a 3D image can contain many more. The one biggest advantage of using three dimensions is that there is an extra dimension that can be used to encode some knowledge or to aid visualisation of the knowledge shown in the two dimensions.

There are many ways of creating a three dimensional effect, some of which are better suited to visualisation than others. The first, most basic, three-dimensional perception comes from our own eyes in everyday life. The brain is constantly fusing together images from each eye of our surroundings and using this information to aid the creation of what is known as depth perception.

Stereograms (also known as autostereograms) make use of the brain fusing two images together to present a three dimensional image in a 2D printed page. These became very popular in the early 1990s and there are now several books available containing these images. Simplified versions of these are Single Image Random Dot Stereograms (SIRDS) that are simply dots on a page. The images can then be seen by allowing the eye and brain to fuse together the two images hidden in the dots. The only problem with these methods is that the 3D image only has the colours of the 2D image. Depth perception is created but there is no way of encoding colour in the separated images. The principle of stereograms lies in presenting the viewer with two slightly different images, one for each eye. Two image versions of these (using photographs and paintings) have been around for about a century. It is only with the advent of better computer graphics techniques that the single images containing a 3D image have become popular and widely available. One of the problems with this form of 3D imaging is the inability to make use of the third dimension to aid in the visualisation.

Computer systems provide a wide scope for creating 3D images. The earliest were technical drawing packages (such as AutoCad) that allowed things to be drawn in orthographic projections. These packages

advanced with the hardware to allow, at first simple, 3D images to be shown on the screen based on the orthographic projections.

From the 3D images displayed in this form, several authoring tools (and code libraries for programmers) that allowed 3D scenes to be constructed were created. The first of these were rendering programs (one version widely available and used is POV-Ray) that allowed a scene to be described in terms of objects and their properties, the camera view to be used, the lighting angle(s) and sources and certain special effects. The problems with such systems was that whilst they produced stunning 3D graphic scenes they took an excessively long time to process. Rendering programs and computer hardware have improved but to create animated renderings still takes an extremely long time and this method is currently not realistically possible for instant virtual world generation. The systems that are interactive need to be able to process images and render new world views in real time.

The rendering programs led onto programs that took scene scripts and then generated a Window on a World (WOW) type of virtual world. These systems allowed 3D worlds to be defined and then generated them. The advantage was that once a world had been defined the user could move through that world and with each virtual "step" the scene would be updated to reflect the new viewpoint. Within the world, properties for the displayed objects could be defined and, since the end result was interactive, "hot spots" could be defined. This means that if the user carried out a certain action in a certain place an event could be triggered to do many things. Some examples of hot spot actions are loading a different world and requesting a certain World Wide Web page. Simplified versions of the world rendering programs are ones that render a world and then perform what is known as a "fly-through" of the world. This means the user is taken on a tour of the generated world but has no control of the route taken. Often architectural demonstrations use this to provide the clients with a guide to the finished building(s), inside and out.

Advances are now being made in multi-user virtual worlds. These are generally still based on the window on a world VR technology but enable other users to see a representation of the user and for a user to see representations of them. One such game that has been adapted is Quake. Programs known as Quake Servers run on one machine and co-ordinate the multi-player information so that the game running on the player's machine can see the other players and be seen by other players. Multi-player games, especially those that are network compatible, are starting to become popular even if they are not as sophisticated as VR systems.

With the popularity of computer supported collaborative working, research is being carried out into the effectiveness of using such systems instead of group databases and e-mail systems. The limiting factor in this, and much of networked simple VR and 3D images, is the hardware. The networking and even the desktop computers still lack the power to create such worlds realistically in real time.

Using three dimensions for visualisation adds an element of familiarity and realism into systems. The world is a three-dimensional experience and by making the visualisation more like that world means there is less cognitive strain on the user. This in turn makes the system easier and more comfortable to use because of all the experience and knowledge the user has built up elsewhere. In using three dimensions the depth cues that make the word, and the visualisation, appear 3D can be used as part of the visualisation. This means that the aim of the visualisation to aid the comprehension of complex phenomena can be achieved without adding unnecessary complications because of the visualisation used.

Stasko, [Stas92], when writing about an Information Visualizer System from Xerox Parc and writes that the designers noted that:

> *"…the three-dimensional displays help shift the viewing process from being a cognitive task to being a perception task. This transfer helps to enable humans' pattern mactching skills."*

This is only true of well designed displays, but supports the views that images that are known to the user aid them in understanding the images presented.

Chalmers [Chal95] writes

> *"In designing an information display, we should support movement and exploration through the space so as to let people build up their own models of the information. By moving and searching through a complex environment, looking in detail as some parts, and in overview at others, we make sense of it and make our our decisions about how to use it and work with it.*

Note that it is not enough to have an information space through which people can move. One has to give thought to what people will see from different positions and angles."

This, whilst containing basic information, is very often ignored or not implemented properly. This forces the user into patterns of working they are not familiar with and can decrease the effectiveness of that work. Later in the paper the following is written:

"For such uses of an information space, a naturalistic 3D view seems a powerful but familiar way of controlling information detail. Perspective lets us gain an overview of distant regions and detail of what is close."

From the simple scale above, and the information provided by Chalmers ([Chal95]), it would seem appropriate to choose interactive 3D worlds as one of the best form of information visualisation, and therefore for software (program) visualisation.

2.3 Information Visualisation

Information visualisation is the process of graphically displaying data that has no inherent form, although for some data sets the actual values (or subject) may provide a suitable structure. The use of information visualisation, rather than using the raw data, is done for several reasons, some of which are

- Being able to display a large amount of information in one view and thus providing an overview
- Being able to see correlations or patterns that may have otherwise been missed had only the figures been used.
- Trying to display structural relationships and context that may be more difficult to detect by individual retrieval requests (provided by Card et al. [Card91]).
- Providing an effective way of going between overview abstractions and the detail of the data.

Young [Youn96] has written a comprehensive overview of the techniques applied in three-dimensional visualisation. In this he surveys the graphical techniques used for such visualisations and some applications of these techniques in research visualisation systems. The main techniques covered can be summarised as:

- Surface plots
- Cityscapes
- Fish-eye views
- Benediktine space
- Perspective walls
- Cone tree and cam trees
- Sphere visualisation
- Rooms
- Emotional icons
- Self organising graphs
- Spatial arrangements of data
    - Benediktine Cyberspace
    - Statistical clustering and proximity measures
    - Hyper-structures
    - Human centred approaches
- The information cube

The cityscapes mentioned above are not urban but are more to do with 3D bar charts and surface plots. The rooms techniques is similar to the method used by Dieberger [Dieb93a, Dieb93b, Dieb93c, Dieb94, and Dieb97] in his Information City. He used the rooms to represent particular hypertext nodes in the city. The human centred approach to spatial arrangements of data is related to urban environments. These tend to use (possibly abstracted) real world metaphors. Examples of these sorts of metaphors are cities, buildings and rooms.

In his conclusions Young identifies the problem of generating abstract representations which are capable of both hiding the complexity of the data under investigation whilst displaying a simplistic graphical

representation of that data. This is still very much a problem and there is no hard and fast way of generating suitable abstractions and metaphors.

3. Software Visualisation

Software visualisation can be seen as a specialised subset of information visualisation. This is because information visualisation in the process of creating a graphical representation of abstract, generally non-numerical, data. This is exactly what is required when trying to visualise software. The term software visualisation has many meanings depending on the author. For the purposes of this document software visualisation can be taken to mean any form of program visualisation that is used after the software has been written (i.e. it does not mean visual programming) as an aid to understanding. More formally, based on the definition of visualisation written in *2.1 What is Visualisation?* software visualisation can be defined as

> *"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 also 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.

Myers [Myer90] effectively sums up the benefits of using graphics in the presentation of program information when he writes

> *"The human visual system and human visual information processing are clearly optimized for multi-dimensional data. Computer programs, however, are conventionally presented in a one-dimensional textual form, not utilizing the full power of the brain."*

In the same paper he also identifies a program visualisation taxonomy. 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.

Myers classifies program visualisation into the following areas:

- Static code visualisation
- Dynamic code visualisation
- Static data visualisation
- Dynamic data visualisation
- Static algorithm visualisation
- Dynamic algorithm visualisation

This can be expressed nicely with the use of two axes, and divides neatly into six regions as Figure 1 shows. That is not to say that all program visualisation systems can be classified as easily as assigning them to one of the six regions!

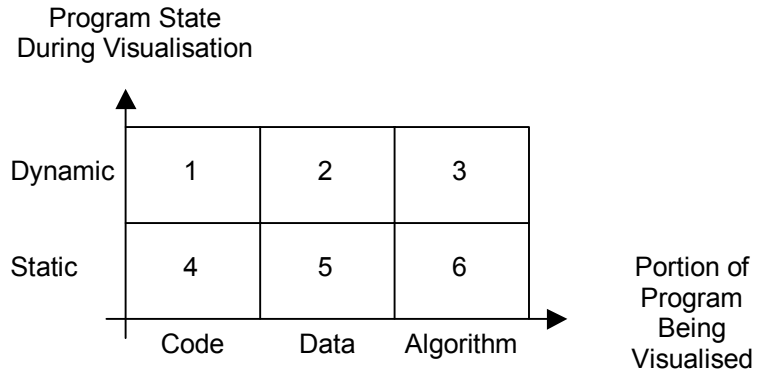

Figure 1 - Visual Representation of Myers' Taxonomy

Other authors have produced taxonomies of program visualisation and classified things in a different way to Myers. Price at al. [Pric92] 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.

The six main categories of this second taxonomy are

- Scope
- Content
- Form
- Method
- Interaction
- Effectiveness

These are based on a general model of software. This was again done to allow for the revision and expansion of the taxonomy should it be necessary. Each of the areas listed can be broken down at least one level, with several having more than just the one level of refinement.

The entire taxonomy can be best expressed in the form of several diagrams. Figure 2 shows the top level of the tree, with Figure 3 to Figure 8 inclusive showing the detail for each of the six categories.

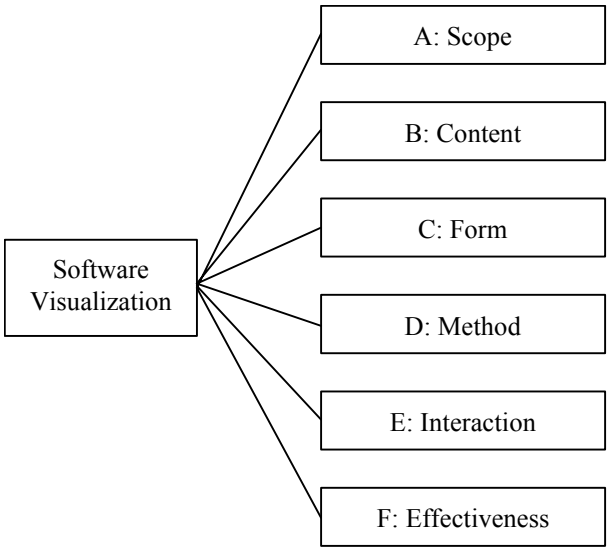

Figure 2 - Top Level of the Price et al. Software Visualisation Taxonomy

The *scope* category can be summarised as the range of programs that the software visualisation system can take as input and then visualise. The sub divisions of this category can be seen in Figure 3.

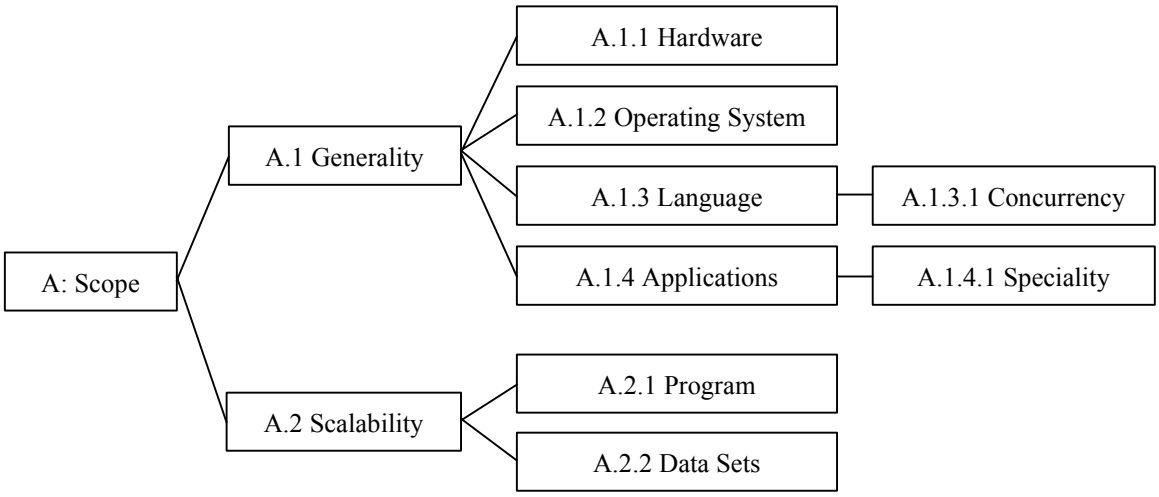

Figure 3 - Detail of the *Scope* Category of the Taxonomy

The *content* category can be summarised as the subset of information (of the original software) that the software visualisation system visualises. The sub divisions of this category can be seen in Figure 4.

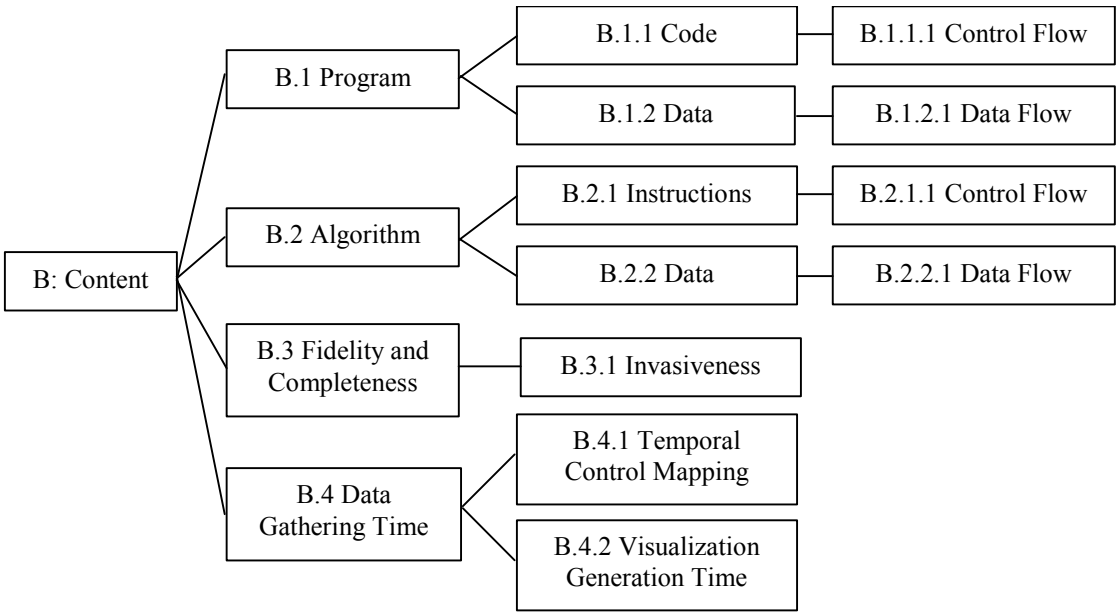

Figure 4 - Detail of the *Content* Category of the Taxonomy

The *form* category can be summarised as the characteristics of the visualisation output by the system. The sub divisions of this category can be seen in Figure 5.

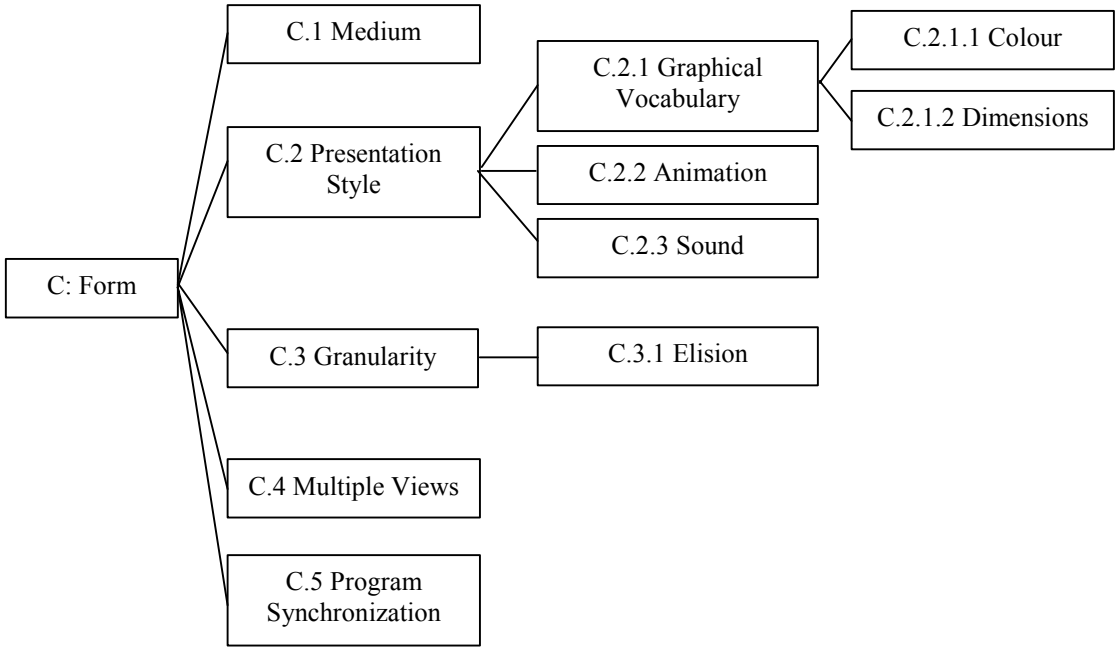

Figure 5 - Detail of the *Form* Category of the Taxonomy

The *method* category can be summarised as how the visualisation is specified. The sub divisions of this category can be seen in Figure 6.

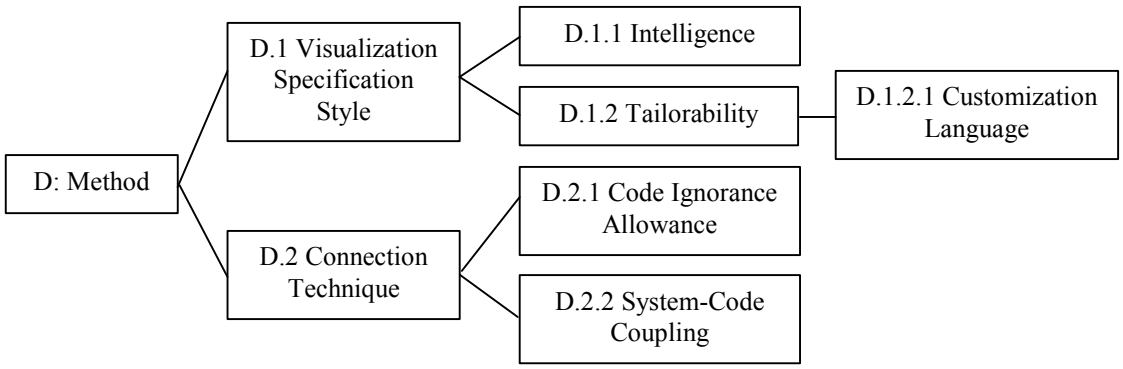

Figure 6 - Detail of the *Method* Category of the Taxonomy

The *interaction* category can be summarised as how does the user of the software visualisation system interact and control it? The sub divisions of this category can be seen in Figure 7

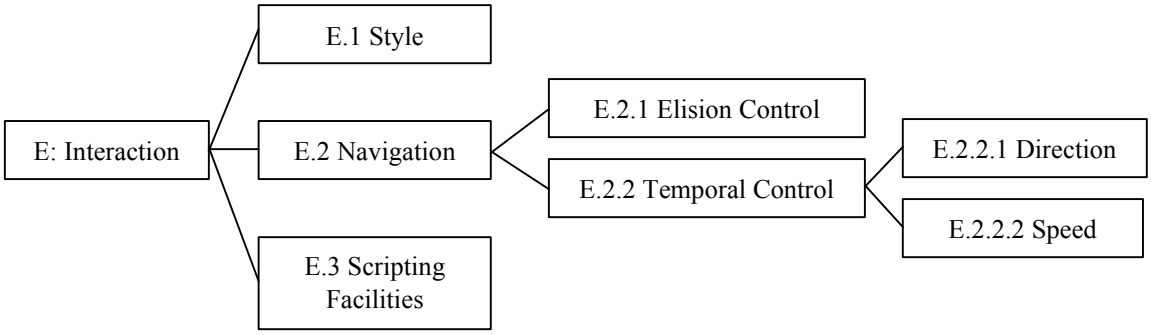

Figure 7 - Detail of the *Interaction* Category of the Taxonomy

The *effectiveness* category can be summarised as does the software visualisation system provide facilities for managing the recording and playback of interactions with specific visualisations? The sub divisions of this category can be seen in Figure 8.

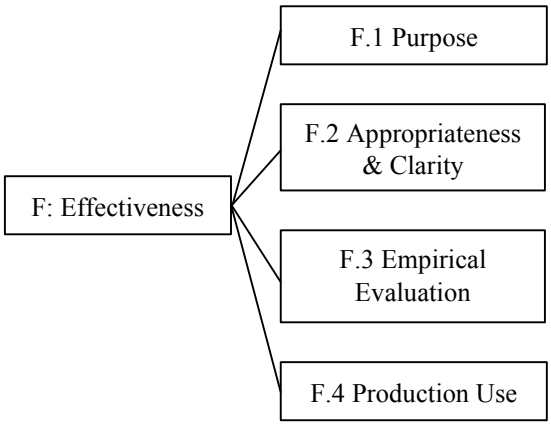

Figure 8 - Detail of the *Effectiveness* Category of the Taxonomy

Another taxonomy is the one defined by Roman and Cox [Roma93]. This is closer to the taxonomy of Price et al. than the one of Myers and some parallels can be drawn between the two. Roman and Cox define five main criteria for the classification of program visualisation systems. These are

- Scope
- Abstractions
- Specification method
- Interface
- Presentation

As with the previous taxonomy these areas all have sub divisions against which the systems can be classified. These sub divisions will be listed, and any parallels with the Price et al. taxonomy will be identified explicitly.

The four sub divisions in the **scope** criteria are *code, data state, control state* and *behavior*. These can be seen to tally with aspects of the **content** category in the preceding taxonomy. It is the B.1 and B.2 aspects of the content category (program and algorithm) that most closely correspond with the definitions and distinctions made in this taxonomy.

Abstraction breaks down into three sub divisions; *direct representation, structural representation* and *synthesized representation.* To an extent these are part of the **form** category of Price et al.'s taxonomy. This is not a direct parallel because abstraction is only a small part of the form category, and is part of the

representation type created. Roman and Cox differ from other taxonomies in that they make the level of abstraction explicit in their taxonomy.

The **specification method** has been broken down in four smaller criteria. These are *predefinition, annotation, declaration* and *manipulation*. The specification of the visualisations is dealt with by Price et al. in their **method** category, and whilst Roman and Cox specify more detail as to how the visualisation is explicitly specified Price et al. manage to incorporate more classification criteria.

The fourth of Roman and Cox's criteria is **interface**. They have only split this into two categories; *graphical vocabulary* and *interaction.* Price et al. cover interaction in their category of the same name, **interaction**, whilst graphical vocabulary is dealt with in their **form** category.

The final criteria, **presentation**, has four sub divisions; *interpretation of graphics, analytical presentation, explanatory presentation* and *orchestration.* These can all be classed under the **effectiveness** category defined by Price et al.

Whilst there are differences between the taxonomies of Roman and Cox and Price et al. there are also very many similarities. The main differences are generally down at a very low level of detail, and can be compensated for in the most part by a different part of the opposite taxonomy. Myers' taxonomy does not bear such a close relation to either of these other two taxonomies, although a criticism levelled at that taxonomy by Price et al. is that it fails to go into enough depth.

3.1 Creating Tangible from Intangible

One of the main problems for software visualisation (and other forms of information visualisation) is of trying to create a tangible representation of something that has no inherent form. Therefore the aim is to visualise the intangible in an effective and useful way. Effective and useful here refer to the visualisation being able to increase the understanding of the user whilst reducing the perceived complexity.

Ball and Eick [Ball96] recognise this problem when they write

> *"Software is intangible, having no physical shape or size. After it is written, code "disappears" into files kept on disks."*

and

> *"The invisible nature of software hides system complexity, … ."*

Walker [Walk95] comments on the software being the intangible part of information systems when he writes

> *"Some aspects of an information system are tangible, but a major component is the software which is an abstract and invisible collation of computer instructions."*

Chapin and Lau [Chap96] also recognise the intangible nature of software

> *"Furthermore, software is intangible, and it is only the representation of the software which can be communicated between people and between people and computers."*

An important point to be drawn from this is the communication aspect. Since software is intangible and each programmer has his own mental representation then an effective visualisation can also act as a common frame of reference. In discussing pieces of the software either informally between colleagues or formally in meetings, if the participants do not concur over the code being discussed the discussion may as well not take place. Visualisation of the software can provide not only a graphical representation of the piece of code under discussion (for clarity over the section being discussed), but also allow the discussion to take place in the realm of that visualisation. This means that the discussion can be based around the visualisation and the code it represents rather than the piece of code. In doing this, the visualisation has provided a starting point for common understanding.

3.2 The Benefits and Challenges of Software Visualisation

Walker [Walk95] sums up the benefits of visualisation when he writes (emphasis added)

> *"The traditional interface of mouse, keyboard and screens of text allows us to work* on *computers, while techniques such as visualisation will truly enable us to work* with *computers."*

This comment only really holds if the visualisations are well designed and implemented. Just using visualisation with no real thought or planning, i.e. not effectively using it, will not lead to the situation described by Walker. If the visualisations are usable and effective then the above description is perfect for showing the benefits that visualisation can bring.

Walker does admit that there are challenges to be overcome in visualisation (information and software). Walker describes the first of these challenges as

> *"We therefore conclude that in response to the first* Challenge *of growing data volume combined with declining information content, there is a key role for visualisation in enabling human exploration, navigation and browsing of complex information environments."*

He also comments on the need to have a natural and intuitive user interface so that the natural perceptual skills of humans can be employed when they are using the visualisations. The second challenge is identified as

> *"... the conversion of* appropriate data *to* relevant information*."*

This relates to the metaphors and abstractions used as a basis for the graphics drawn in the visualisations. Another challenge is connected to the intangible nature of software discussed in the preceding section. Walker describes this as

> *"... the problems in managing increasingly abstract problems against ever decreasing timescales."*

The final challenge can be simply described as

> *"... effectively communicating a vision, such that it is accurately shared and understood by the audience."*

In trying to achieve the aim of increased understanding of the software under investigation, Walker identifies the benefits of visualisation, but also shows the explorative powers that visualisation can afford to the user.

> *"Although the visualisation adds nothing to the capability of the underlying models, the increased user involvement offers the potential for faster understanding and more rapid, exhaustive exploration of the model's predictions and limitations."*

Visualisations can be of indeterminable value in trying understanding complex information. This is summed up by Ware et al. [Ware93] as

> *"There is increasing evidence that it is possible to perceive and understand increasingly complex information systems if they are displayed as graphical objects in a three dimensional space."*

They also suggest that the comprehension of complex software systems is one of the most challenging issues facing the software engineering field. Their research concentrates on the use of three dimensions in the display of object-oriented software in an attempt to ease the complexity problem.

Walker [Walk95] concludes his paper with this sentence

> *"Creating a world in which humans and computers are equally at ease in the analysis and interpretation of data will not be easy, but the rewards will be immense."*

Visualisation is just one of the many possible ways of trying to achieve this.

3.3 IA Not AI

Intelligence amplification (IA) is the use of computers to aid and enhance human intelligence rather than the artificial intelligence (AI) aim of trying to substitute humans with computers. Intelligence amplification builds on the skills that humans already have, and tries to augment the areas that are lacking in some way. Frederick Brooks (documented in by Rheingold [Rhei92]) describes his beliefs about intelligence amplification in the following way

> *"I believe the use of computer systems for intelligence amplification is much more powerful today, and will be at any given point in the future, than the use of computers for artificial intelligence (AI). In the AI community, the objective is to replace the human mind by the machine and its program and its data base. In the IA community, the objective is to build systems that amplify the human mind by providing it with computer-based auxiliaries that do the things that the mind has trouble doing."*

Brooks identifies three areas in which humans are more skilled than computers. The first is *pattern recognition* (aural or visual). The second is in performing *evaluations*, and the third is the *overall sense of context* that allows previously unrelated pieces of information to become related and useful in a new situation.

Walker [Walk95] also touches on the subject of intelligence amplification in his discussion on the challenges of visualisation.

> *"A natural and intuitive visual interface can retain the critical contribution from human perceptual skills, ensuring that opportunities for lateral thinking or perhaps an unexpected leap of imagination are not lost. Programming a computer to "look for something interesting" in a database is a major undertaking, but given appropriate tools, it is a task for which humans are well equipped."*

The first sentence can be seen to the similar to the third skill identified by Brooks, that of a sense of context. The second sentence by Walker is essentially talking about the pattern recognition skill specified by Brooks.

Intelligence amplification is of importance to software visualisation (and any other form of visualisation) because in representing large and complex data sets graphically the aim is to help the user to get a better understanding of content of the data sets. By aiding the user in this way visualisation tools are acting also as intelligence amplification tools. Reading through many thousands of pieces of information and then summarising them in a finite graphical space would be an immense, complex and possibly tedious task. For a computer with the right "instructions", it is a simple data processing exercise.

Hubbold et al. [Hubb93] make a similar connection with the field of virtual reality (and therefore visualisations that make use of virtual reality as an enabling technology). They also identify the pattern recognition and contextual abilities of humans.

> *"In our everyday existence we cope with, and filter out, tremendous amounts of information almost effortlessly and with very little conscious thought. Indeed, if the same information, in all its detail, were to be presented in a form that we had to think about consciously, then we would be overwhelmed quite easily. Spatial awareness, pattern recognition, information filtering, coordination of multiple information streams are things we take for granted. Rather than look for a solution in AI, part of the VR thesis is that information presented in a suitable way can be processed far more effectively and directly by people."*

As R. W. Hamming [Hamming] said

> *"The purpose of computing is insight not numbers."*

It is a very simple, but important, comment and unfortunately one that is ignored by many (computer) scientists.

3.4 The Use of Colour and Text

Text and colour are both simple and common elements of display but used in certain ways can be very effective. Without a doubt, both can be very useful in visualisation systems, as discussed by Dieberger [Dieb94] in his thesis. He covers the concept of *writing on the world* for virtual environments, a term (according to Dieberger) coined by Jay Bolter. This "writing" does not have to be text, although in many situations a few words can convey an enormous amount of information to the user of the system.

The normal view of program code is in textual form. Because of this "pretty printers" that format source code have been around for almost as long as programming via terminals. The concept is still actively used today in development environments and often makes use of coloured syntax highlighting.

Pretty printers were developed to automatically lay the source code out with, amongst other things, indentation to show the block structures of the programming language used. Whilst this is a simple thing to do, it has been empirically shown that the use of indentation makes code reading easier. Arab [Arab92] and Miara et al. [Miar83] document two such empirical studies of the use of indentation on program code.

Tapp and Kazman [Tapp94] also carried out empirical studies, but to test the use of colour and font in aiding understanding (of program code). They found no conclusive evidence that either helped, although the subjects expressed a preference for displays using colour, and would use such displays again.

The visualisation systems developed by Ball and Eick [Ball96] made use of both colour and text for displaying program code in a two dimensional space. Through this system they achieved both overview and detailed information, and applied the systems to real software engineering problems. Baker and Eick [Bake95] also discuss these visualisation systems. An example screenshot of their *SeeSoft* visualisation system can be seen in Figure 9.

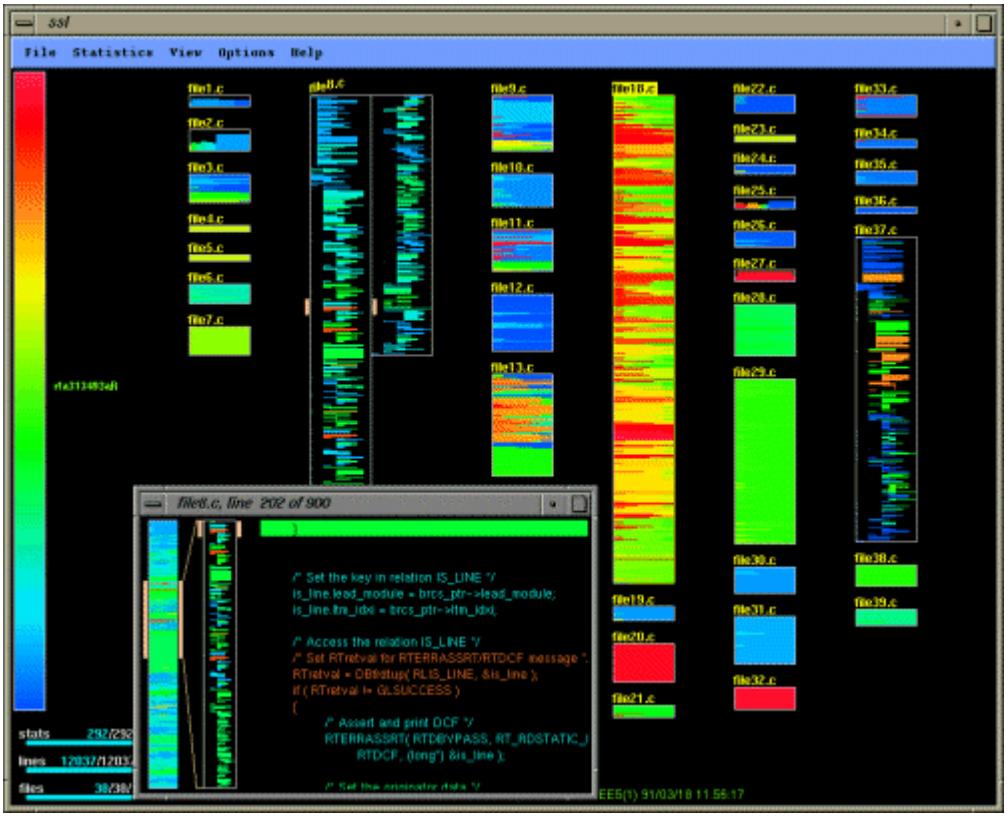

Figure 9 - Screenshot from the *SeeSoft* Visualisation System

Available from http://www.bell-labs.com/~eick/figs/seesoft/seesoft.gif.

Jerding and Stasko [Jerd96] also created a visualisation system that used colour and text in two dimensions. Again the system was capable of showing both overview and detailed information. They applied this to program code and other forms of data, including textual documents.

Petre [Petr95] makes the argument for the combined usage of graphics and texts, rather than graphics alone. She writes

> *"Both graphics and text have their uses – and their limitations. Pictorial and graphic media can carry considerable information in what may be a convenient and attractive form, but incorporating graphics into programming notions requires us to understand the precise contribution that graphical representations might make to the job at hand."*

Much of the discussion in the paper is to do with visual programming rather than visualisation but the point applies to both areas.

It is clear that regardless of the type of visualisations used, colour and text have an important part to play in conveying information to the users of the visualisations.

3.5 Abstractions

Abstractions need to be considered when focusing on visualisation for two reasons. The first is that to represent any form of data (software or otherwise) by visualisation involves abstracting away from the original data. The second reason is that within the visualisation different levels of abstraction can be important.

If program code is considered, it can be seen that it goes through many transformations, and thus levels of abstraction before any visualisation system displays it graphically. Programs are ways of instructing the computer to do something. The problem is that computers understand ones and zeros whilst humans find this very difficult. Third and fourth generation programming languages are now used to create program code, but this then has to be converted in some way to a form that the computer can act on. It is this reverse series of transformations that is creating a higher level of abstraction at each point (from the point of view of humans). By creating a graphical representation of the data the level of abstraction is only increased by one. This is something that many opponents of software visualisation forget. They suggest that by abstracting, the original code is lost and hence the visualisations are useless.

It is true that the abstraction process (at any level) hides some detail, but as long as that detail is accessible should it be required then there is no cause for deriding whatever process creates that abstraction.

Jerding and Stasko [Jerd96] created a visualisation system that reduced a data set into what they termed *an information mural*. This display was at an overview level, but from the abstracted view it was still possible to read the detailed information.

Storey et al. [Stor96] also made use of abstractions in their work on the Rigi system using SHriMP views. The SHriMP visualisation uses nested graphs with fisheye techniques. It is the nested graphs that provide the different levels of abstraction, but the visualisation is designed to allow the user to have a sense of overall context when browsing.

Card et al. [Card91] consider the process of abstraction in their work on the Information Visualizer and write

> *"The abstractions produced by the lower-level processing predetermine, to a considerable extent, the patterned structure that the higher-level processing can detect. The higher-level processing, in turn, reduces still further the quantity of information by processing it into yet more abstract and universal forms."*

This is true of very many abstractions and abstraction mechanisms.

Abstractions of a visual nature are an effective way of providing an overview. The abstractions allow more information to be displayed at once, and for information to be displayed in a different form. These two "extra" insights on the data set can be very useful and can even aid in discovering new information about the data under consideration.

3.6 Traditional Visualisation Directions

For many years the focus of software visualisation has been two-dimensional images. Various aspects of the program code and metrics calculated from the code have been displayed in 2D. A common display was, and still is, based around graph structures. There are problems with this approach to software visualisation, not least because of the complexity of the software. A call graph where the nodes are the procedures or methods and the arcs represent the calls made between them can easily become cluttered and unclear. An example of this is shown in Figure 10.

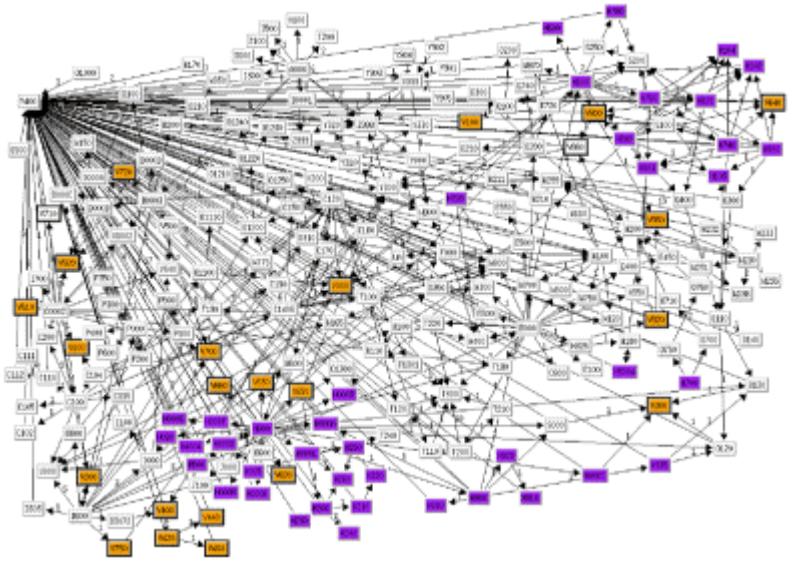

Figure 10 - Call Graph of a Commercial System

This call graph (Figure 10) actually represents the calling information from a highly maintained commercial system. If there is this much overcrowding of information from a reasonable sized, but well maintained, piece of software then it is clear that new techniques for the investigation and visualisation of code are needed.

The use of graphs for the display of program information and relations has been well documented, due to the popularity of this sort of display. A selection of this work can be found documented in the following papers; Burd et al. [Burd96], Vilela et al. [Vile97], Jerding and Stasko [Jerd94], Ware et al. [Ware93] and Storey et al. [Stor96].

3.7 Visualisation in the Future?

These examples are generally of 3D visualisation and avoid the *lines and boxes* approach so long favoured in the program comprehension community. To try to provide examples of every type of software visualisation created would be unfeasible but these are shown because they are different to the established two-dimensional visualisations. With technology increasing in power and lowering in cost, and a human's ability to interact with virtual reality better understood (cognitively), then there is every reason to believe that software visualisation will continue along this path and enable more advanced and useful visualisations to be created.

Young [YoungOnline] has tried several approaches to visualising software without recourse to using the nodes and arcs favoured by practitioners in the field. In particular his representation of call graphs in columns is in stark contrast the traditional representation of such information. This work is documented in [Youn97]. An example of a call graph shown in his *CallStax* visualisation can be seen in Figure 11.

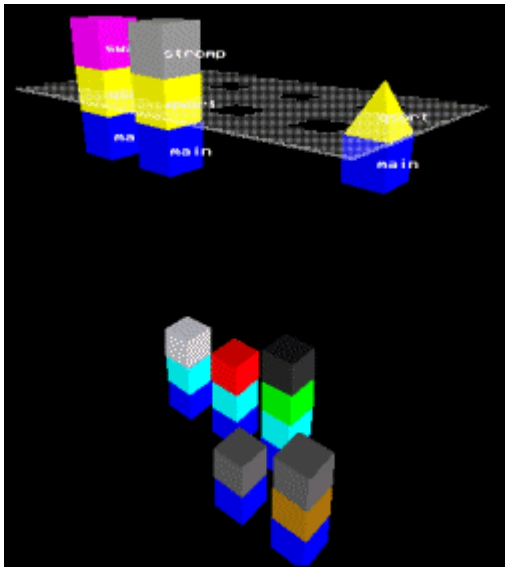

Figure 11 - CallStax Representation of Call Graph Information

Young writes the following of his CallStax representation:

> *"CallStax makes full use of the extra dimension afforded by VR to maximise the amount of information available and the flexibility for displaying and interacting with that information."*

The visualisation does not try and show the information as a network but as paths through the network. The picture shows all the calls stacks for a specific piece of code, with the qsort item selected as the focus of interest (shown as yellow in the image, at the level of the grid towards the top of the image). The user is able to interact with the representation by selecting the focus. The display then updates itself appropriately, in an animated fashion.

Another representation of Young's that has had some success is FileVis. This again uses abstract three-dimensional space to display information about program code. More information can be found in [Youn98], whilst Figure 12 shows one of the views available with this visualisation.

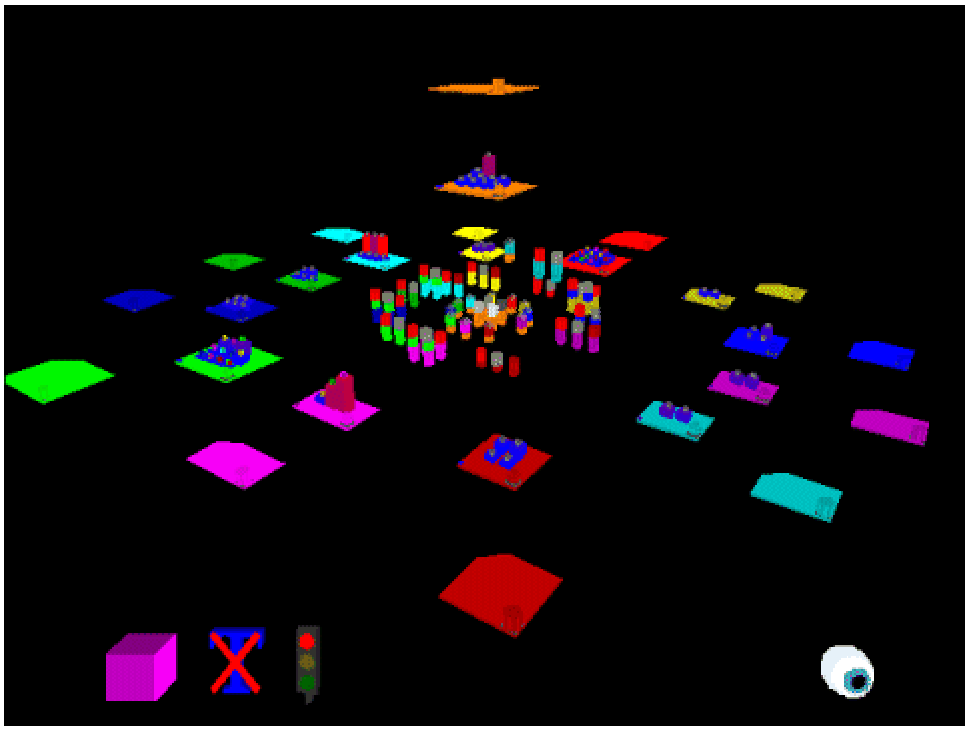

Figure 12 - FileVis Visualisation of Source Code

Knight and Munro [Knig98] took a novel approach to the representation of software in three-dimensional virtual environments by using the Quake 2 graphics engine to visualise Java code. The work done was only at a prototype level but it showed the benefit of such graphical environments for representing software. A screenshot of this work (from an extension to the documented work) can be seen in Figure 13.

Figure 13 - View of Two Alternative Paths through a Call Graph

Whilst the authors document that the call graph representation (a map in three-dimensional space) is not adequate for large systems, the concept of a multi-user virtual environment has been shown to be beneficial.

There are many more examples of information visualisation and hypertext visualisations than there are of software visualisation if a restriction on not using standard two-dimensional "lines and boxes" is made. This is not to say it is not a valid program comprehension research area, but more than ever, it is an area where there is an extensive amount of research still to be done. Early results have been promising, and the successes of information, database, hypertext and web visualisations show that it could still prove to be a very fruitful way of representing software.

3.8 Further Issues

There are several other important issues connected with software visualisation that have so far been ignored. The first of these is to do with the scalability of the visualisation. It is all very well for a visualisation system to work with a small source code sample but unless it works with large systems, with little human intervention then it is of no earthly use to programmers and maintainers. The visualisation needs to scale well, and because of this the metaphors and abstractions used should be carefully considered. There is also a need for automation. If each facet of the visualisation needs to be created by hand there is little point in using the visualisation with real systems. Some human intervention is fine, but on the whole the visualisation needs to be created with as little as possible. Price et al. [Pric92] identify the scalability research issues still outstanding.

Another important issue is that of interaction. Not only does the environment need to adequately support interaction and navigation, but so does the visualisation. If a user cannot select a focus of their choosing, and then manipulate the visualisation in any way it becomes a pointless exercise having the visualisation system.

A metaphor is where a word or phrase (or in terms of visualisation, a graphical representation of that word or phrase) is used in place of another. This tends to suggest some form of analogy between the two concepts, although this may be at a higher level of abstraction than individual words or phrases. From a visualisation perspective the metaphors act as a mapping from the concepts required in the virtual world to their graphical representation. This need was identified by Levialdi et al. [Levi95] in the construction of their database visualisation system.

> *"Using VR visualization techniques to represent the results of queries implies the definition of a mapping, or* metaphor*, among the objects of the database and the objects of some virtual world."*

According to Benford et al. [Benf96] the use of natural metaphors can aid the usability of virtual environments.

> *"… an attempt to exploit people's natural understanding of the physical world, including spatial factors in perception and navigation, as well as general familiarity with common spatial environments…"*

Pettifer and West [Pett97a] suggest that the potential power of virtual reality comes from the strength of its metaphor, and the fact that it is closer to natural interaction than many other forms of computer system. They also identify the benefits of natural metaphors, and making use of perceptual and spatial skills learnt and used in the real world in the virtual environment.

> *"A three-dimensional world metaphor has much more scope for direct human/computer interaction than the two-dimensional desktop because it engages in us those perceptual and spatial faculties that allow us to comprehend our surroundings and to process effortlessly the vast amounts of information that are presented to our senses second by second. It is the potential to directly engage these faculties that is the defining characteristic of virtual reality. As the immersive environment is far richer than the desktop, the metaphors for interaction assume a far greater significance. … The role and management of metaphors for the virtual environment therefore assumes key significance."*

It is obvious from the above that the design of the metaphor used in the virtual environment can play a large part in the usability of that system, both in terms of human computer interaction, and in terms of enabling the user to carry out the required tasks. What is also of benefit is that in using three-dimensional environments some of the cognitive processing needed for navigation and visual interpretation can be shifted to the sub-conscious as these are activities that are carried out daily with no real thought. Metaphors not only provide a basis for the environment, but for the mapping from the data to the visualisation. Similar principles apply when visualising with metaphors, such as consistency, interaction and usability.

The previous section provided a glimpse of one of the futures that software visualisation may have. From this point there are many alternate routes the software visualisation field could go, and its long-term effectiveness in aiding programmers and maintainers depends on the route it takes.

4. Conclusions

The driving force behind software visualisation is to ease the cognitive burden of trying to comprehend existing programs. Myers [Myer90] includes areas for future research at the end of his program visualisation taxonomy. The first of these relates to the scalability and abstraction issues previously identified. The large and complex programs used and written today are the ones that can most benefit from visualisation (from the point of view of the programmer). Unfortunately these programs are the ones that receive the least support! Another previously identified area that Myers suggests is that there needs to be more research in the automation of the visualisation generation from the source code. He also suggests that many representations used are not very good. This can be alleviated, at least partly, through the use of suitable metaphors and abstractions.

Von Mayrhauser et al. [Mayr97] identify several research areas that still need investigation. The first of these is using some form of history of browsed location that is programmer defined, i.e. they decide what and where items are bookmarked, possibly with some annotation as to the reason that the item has been marked. They also make the point that cross-referencing related information, even across document types (such as between code and traditional textual documents) and across abstraction levels is a helpful thing. It has long been shown that cross-referencing of code is useful, so extending this can be seen as the next logical step. Another important item that they note is the provision of some form of "road map" of the software under consideration.

These areas of research previously identified are still areas of research. These problems have not yet been solved, although software visualisation systems have the potential to do so. To be able to use graphics requires some form of mapping metaphor from the software elements to the graphical elements. There is no obvious mapping because software has no defined shape or colour. In creating visualisations as virtual environments there seems little point in restricting that world to one user, especially with the level of system and domain knowledge held by programmers rather than formally documented. Hypertext is also an important facet for both moving around between distant areas of the virtual world and enabling users to follow lines of enquiry (during program comprehension) in a natural way rather than being forced to retrieve that information in predefined ways.

As Myers [Myer90] concludes so succinctly:

> *"The success of spreadsheets demonstrates that if we find the appropriate paradigms, graphical techniques can* revolutionize *the way people interact with computers."*

5. References

[Arab92] M. Arab.
Enhancing Program Comprehension: FORMATTING and DOCUMENTING.
ACM SIGPLAN Notices, Vol. 27, No. 2, pp37-46, February 1992.

[Bake95] M. J. Baker, S. G. Eick.
Space Filling Software Visualization.
Journal of Visual Languages and Computing, Vol. 6, pp 119-133, 1995.

[Ball96] T. Ball, S. G. Eick.
Software Visualization in the Large.
IEEE Computer, April 1996, pp33-43.

[Benf96] S. Benford, C. Brown, G. Reynard, C. Greenhalgh.
Shared Spaces: Transportation, Artificiality and Spatiality.
Proceedings of the 1996 ACM Conference on CSCW, Boston, Masachusetts, USA, November 16-20, 1996, pp77-85. ISBN 0-89791-765-0.

[Burd96] E. L. Burd, P. S. Chan, I. M. M. Duncan, M. Munro, P. Young.
Improving Visual Representations of Code.
University of Durham, Computer Science Technical Report 10/96, 1996.

[Card91] S. K. Card, G. G. Robertson, J. D. Mackinlay.
The Information Visualizer: An Information Workspace.
Xerox Palo Alto Research Centre, Palo Alto, California 94304, 1991.

[Chal95] M. Chalmers.
Design Perspectives in Visualising Complex Information.
Proceedings IFIP 3rd Visual Databases Conference, Lausanne, seizerland, March 1995.
http://www.ubs.com/info/ubilab/print_versions/ps/cha95.ps.gz.

[Chap96] N. Chapin, T. S. Lau.
Effective Size: An Example of Use from Legacy Systems.
Journal of Software Maintenance: Research and Practice, Vol. 8, pp101-116, 1996.

[Dieb93a] A. Dieberger.
The Information City - A Step Towards Merging of Hypertext and Virtual Reality.
Poster at Hypertext 1993.

[Dieb93b] A. Dieberger.
The Information City - A Metaphor for Navigating Hypertexts.
Presented at BCS-HCI '93, Loughborough, England, September 1993.

[Dieb93c] A. Dieberger, J. G. Tromp.
The Information City Project - A Virtual Reality User Interface for Navigation in Information Spaces.
Was to appear in Proceedings of the Symposium Virtual Reality Vienna, Vienna, December 1-3, 1993 but was never printed.

[Dieb94] A. Dieberger.
Navigation in Textual Virtual Environments using a City Metaphor.
PhD thesis, Vienna University of Technology, Faculty of Technology and Sciences, November 1994.

[Dieb97] A. Dieberger.
Navigation Metaphors and Social Navigation in Information Spaces.
Position Paper for the CHI '97 workshop on Navigation in Information Spaces, 1997.

[Frie91] R. M. Friedhoff, W. Benzon.
Visualization: The Second Computer Revolution. Published 1991, W. H. Freeman and Company. ISBN 0-7167-2231-3.

[Hamming] The online biography of Richard Wesley Hamming can be found at http://www-history.mcs.st-andrews.ac.uk/history/Mathematicians/Hamming.html.

[Hubb93] R. Hubbold, A. Murta, A. West, T. Howard.
Design Issues for Virtual Reality Systems.
Presented at the First Eurographics Workshop on Virtual Environments, Barcelona, 7th September 1993.

[Jerd94] D. F. Jerding, J. T. Stasko.
Using Visualization to Foster Object-Oriented Program Understanding.
Georgia Institute of Technology Technical Report GIT-GVU-94-33, July 1994.

[Jerd96] D. F. Jerding, J. T. Stasko.
The Information Mural (DRAFT).
Available online from
http://www.cc.gatech.edu/gvu/softviz/infoviz/information_mural.html.

[Knig98] C.Knight, M. Munro.
Using an Existing Game Engine to Facilitate Multi-User Software Visualisation.
University of Durham, Computer Science Technical Report 8/98.
http://www.dur.ac.uk/~dcs3crk/workfiles/documents/Multi-User_Software_Vis/Multi-User_Software_Vis.html.

[Levi95] S. Levialdi, A. Massari, L. Saladini.
Visual Metaphors for Database Exploration.
A position paper relating the Virgilio system, available on-line from
http://www.darmstadt.gmd.de/~hemmje/Activities/Virgilio/Publications/virgilio.ps.

[Mayr97] A. Von Mayrhauser, A. M. Vans, A. E. Howe.
Program Understanding Behaviour during Enhancement of Large-scale Software.
Journal of Software Maintenance: Research and Practice, Vol. 9, pp299-327, 1997.

[Miar83] R. J. Miara, J. A. Musselman, J. A. Navarro, B. Shneiderman.
Program Indentation and Comprehensibility.
Communications of the ACM, Vol. 26, No. 11, pp 861-867, November 1983.

[Myer90] B. A. Myers.
Taxonomies of Visual Programming and Program Visualization.
Journal of Visual Languages and Computing, Vol. 1, pp97-123, 1990.

[Petr95] M. Petre.
Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming.
Communications of the ACM, Vol. 38, No. 6, pp 33-44, June 1995.

[Pett97a] S. Pettifer, A. West.
Deva: A coherent operating environment for large scale VR applications.
Presented at the first Virtual Reality Universe conference in Santa Clara, California, April 1997.

[Pric92] B. A. Price, R. M. Baecker, I. S. Small.
A Principled Taxonomy of Software Visualization.
Journal of Visual Languages and Computing, Vol. 4, No. 3, pp211-266, 1992.

[Rhei92] H. Rheingold.
Virtual Reality
Mandarin Science, ISBN : 0-7493-0889-3, 1992.

[Roma93] G. C. Roman, K. C. Cox.
A Taxonomy of Program Visualization Systems.
Computer, pp11-24, December 1993.

[Stas92] J. T. Stasko.
Three-Dimensional Computation Visualization.
Georgia Institute of Technology, Technical Report GIT-GVU-92-20, 1992.

[Stor96] M.-A. D. Storey, H. A Müller, K. Wong.
Manipulating and Documenting Software Structures.
Chapter in Software Visualization, edited by P. Eades and K. Zhang, World Scientific Publishing Co., November 1996, pp244-263, ISBN 981-02-2826-0.

[Tapp94] R. Tapp, R. Kazman.
Determining the Usefulness of Colour and Fonts in a Programming Task.
3rd Workshop on Program Comprehension, November 14-15, 1994, Washington, D. C., IEEE Press, pp 154-161.

[Walk95] G. Walker.
Challenges in Information Visualisation.
British Telecommunications Engineering Journal, Vol. 14, pp17-25, April 1995.

[Ware93] C. Ware, D. Hui, G. Franck.
Visualizing Object Oriented Software in Three Dimensions.
CASCON '93 (IBM Centre for Advanced Studies), Conference Proceedings, pp 612-620. Toronto, Ontario, Canada, October 1993.

[Youn96] P. Young.
Three Dimensional Information Visualisation.
University of Durham, Computer Science Technical Report 12/96.

[Youn97] P. Young, M. Munro.
A New View of Call Graphs for Visualising Code Structures.
University of Durham, Computer Science Technical Report 03/97, April 1997.

[Youn98] P. Young, M. Munro.
Visualising Software in Virtual Reality.
Proceedings of the IEEE 6th International Workshop on Program Comprehension, June 24-26, 1998, pp19-26. Ischia, Italy, IEEE Computer Society Press.

[YoungOnline] P. Young's work can be found online at http://www.dur.ac.uk/~dcs3py/.