Copyright 1999 IEEE.

Published in the International Workshop on Program Comprehension 1999 (IWPC '99)

May 5-7, 1999 in Pittsburgh, PA, USA.

Personal use of this material is permitted. However permission to reprint / republish this material for advertising or promotional purposed or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE.

Contact
Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / PO Box 1331 / Piscataway, NJ 08855-1331, USA.
Telephone: + Intl. 908-562-3966.

Comprehension with[in] Virtual Environment Visualisations

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

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 by trying to force large amounts of information into a small space, relying solely on two dimensions for the representations. Three-dimensional visualisations are one approach that is being considered to combat these problems, although there are also many different issues that must be considered for these sorts of visualisations. Especially when those visualisations are of intangible software systems. A solution is to use virtual environments as a base for these three-dimensional visualisations, which allows a sense of context to be maintained and also promotes collaboration between the people trying to comprehend the code.

1. Introduction

Software visualisation is an important weapon in the program comprehension armoury. It is a technique that can, when designed and used effectively, aid in understanding existing program code. It can achieve this by displaying information in new and different forms, which may make obvious something missed in reading the code. It can also be used to present many aspects of the data at once.

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. 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 since alternatives also have their own problems. The use of three-dimensions to visualise programs has in recent years attracted attention, and some promising results have been published, for example the work by Young [15].

There are several different ideas as to the strategies programmers use when comprehending code. A good overview of the difference strategies and their interrelationships can be found in Storey et al. [14]. A useful distinction made by these authors is that a mental model is the representation in the programmer's mind and that the cognitive model is the processes and structures that are used to help the programmer form their mental model.

This paper presents software visualisation, virtual environments, metaphors and then a visualisation that has been developed in an attempt to visualise in a useful way ever more complex and rapidly changing code.

2. What is Software Visualisation?

There is not a simple answer to this question. There are many ways of defining software visualisation, each tailored towards the type of visualisation it is intended to support. The range of visualisations covered by these definitions is also large. What is important is that the use of these visualisations as a tool is kept in mind. Friedhoff and Benzon [5], write (p16):

"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."

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, which Frederick Brooks believes (documented by Rheingold [16]) is, and always will be, much more powerful then using computers for artificial intelligence techniques. 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. [7] make a connection with the field of virtual reality (VR), and therefore visualisations that make use of virtual reality as an enabling technology. They 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."

This is important when considering three-dimensional visualisations because, generally, they will use some form of VR technology.

A general software visualisation definition that encompasses these ideas (which are traced fully in [8]) can therefore be written 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."

Figure 1 - Traditional call graph

Existing forms of visualisation, such as the call graph shown in Figure 1, adhere to some facets of this definition in that they use imagery and provide some insights but they do not reduce the perceived complexity of the software under consideration. Some research has tried to move the traditional two-dimensional representations into three-dimensions but this has generally not been a success because these representations do not transfer well [15].

Much of the visualisation done to date of programs has been abstract in nature; geometrical shapes in either 2D or 3D, possibly of differing colours have been used to represent elements of the program code. Real world visualisations tend to visualise the data by using elements from the everyday human world. Whilst these two approaches differ in the representations used, they can be complimentary, and it is very likely that final solutions to any visualisation will encompass some elements of both. It is also important to consider the force the data exerts on the visualisation style. With program code, either style is acceptable because the software is intangible, but for other data sets the data may drive the style to be used.

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 graphical displays and views, and to reduce the perceived complexity through the use of suitable abstractions and metaphors.

3. Using Virtual Environments

(Collaborative) Virtual Environments is a research area that is primarily concerned with the construction of multi-user virtual worlds rather than the work processes or interaction that takes place within those worlds. The research covers all facets of virtual world creation from the underlying graphics engines to varying algorithms for creating the best perceptual experience within the worlds.

Churchill and Snowdon [4] provide a very good description of virtual environments:

> *"As such CVEs provide a potentially infinite, graphically realised digital landscape within which multiple users can interact with each other and with simple or complex data representations."*

Virtual environments can be found in science fiction and fantasy literature where general on-line worlds are described rather than being for one purpose, as is the case with most current virtual environment systems. Examples of these "worlds" are Neal Stephenson's [13] *Metaverse* and William Gibson's [6] *Cyberspace* or *Matrix*.

Virtual environments provide a good basis for three-dimensional visualisations because they enable all aspects of the definition of software visualisation (given in the previous section) to be exploited. This does require the visualisation to be designed with the virtual environment in mind and appropriate metaphors used with thought. Using a virtual environment is not necessarily a recipe for a good, usable and useful visualisation.

3.1 The Use of Metaphors

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. [9] 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."*

It is important to note that most (if not all) virtual environments are implemented in some form of virtual reality system. This means that discussions about the benefits of virtual reality are applicable when considering virtual environments.

According to Benford et al. [1] 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 [11] 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.

3.2 Reasons for Using Virtual Environments

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 world, 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, [12], when writing about an Information Visualizer System from Xerox Parc 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 matching 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 [2] 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 Chalmers' 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 ([2]), it would seem appropriate to choose interactive 3D virtual environments as a good way of visualising software and programs.

4. *Software World*

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

A benefit of virtual environments it that it is possible to cause the generation of a new environment which can be accessed upon user demand if the number of data items exceeds a certain threshold. A disadvantage of this is that then decisions have to be made as to how the divisions between environments are made and how the visualisation is able to cope with, for example, evolution in a way that the user can understand. With program code and software systems the evolution of visualisations is important because every system faces maintenance (and hence change) of some description during it's life. It is also important to consider the size of the data sets to be visualised at each level, and to design the visualisations accordingly to try and display these items for the expected inner (realistic) number range.

One solution to the problem of visualising program code in three-dimensions by using virtual environments is the *Software World*. This visualisation has been developed to try and achieve all the aims of software visualisation identified in the definition in Section 2.

Software World makes much use of real world metaphors through its use of, for example, cities. The use of user populated virtual environments is the driving force for this choice of real world metaphor as the basis of the visualisation. In creating a virtual environment

that does not widely differ from the world experienced on a daily basis, the perceptual abilities of the users can be freed to concentrate on the data under investigation.

4.1 Code to Visualisation: The Mapping

The *Software World* is a visualisation that encompasses real world items based on city and urban developments and cartography in an attempt to deal with visualising software systems of differing sizes in a coherent manner. At the detailed levels, the code is shown as buildings and districts of a city environment, whilst the higher level detail is shown in a map style using overview and amalgamation techniques to fit a possibly large amount of information into a finite space. Nesting within graphical objects when the user is distant is another way to deal with large and/or varying amounts of information. The use of an urban environment allows the natural perception of users to be harnessed [10].

The world has different levels, which can be briefly described as

- **World**; flattened, overview picture (atlas style) not necessarily countries as would be known in standard geography but shows different elements of the visualisation at a very high level and the relationships between those elements.
- **Country**; each element shown in the world view is a country. It provides a way of splitting the items in the world down one level without the detail that is provided by the next level down.
- **City**; shown within countries, as the next level of granularity. These cities are composed of sub-areas but try to ease the navigation burden through the use of standard urban navigational aids.
- **Districts**; there can be from 1..n of these in a city, the number depending on the information to be represented in the visualisation. They group together related aspects of the software and provide groupings to be used when moving from a higher level of abstraction to a more detailed level.
- **Streets/Buildings/Gardens/Monuments**; these show the detail of the visualisation and provide the next level of abstraction down from the districts. They also act as legibility features and landmarks of the city.
- **Inside Buildings/Gardens**; this is the finest level of detail, where detailed direct mappings from the code to the visualisation can be made. The use of walls, for example, also provides a way of displaying extra information to the user such as text. Text needs this sort of context in a three-dimensional space to be understandable and visible from many directions.

Software World is targeted at the visualisation of Java code. Some of the items to be visualised are the files, packages, classes, methods and variables and the various attributes of these elements that constitute the full code. This information is essentially a cross reference database of the elements and to provide easier navigation, information retrieval and ultimately understanding, these relations may be shown in the visualisation as a form of hypertext style link. The Java language elements can be mapped to the different visualisation levels as shown in Table 1.

For Java, each file is an object (class), therefore each district is an object. For those objects containing other objects (or files containing several objects) then a district can have sub districts, one for each object. The language definition of Java allows certain assumptions to be made, and the auxiliary classes in a file (since they are not public) are used only by the code in the main class. This is why allowing many classes at the same scope level to be sub-districts is a reasonable mapping decision.

At the same level as the attributes of the class (district), other urban items such as gardens/parks and monuments can be use to represent file and class attributes that are not methods. These items can also serve as navigational aids based on urban planning, as documented by Lynch [10].

Table 1 - Actual mappings from Java code to graphics

Visualisation Level	Code Element
World	The software system as a whole.
Country	Directory structure, which maps to the packages in Java.
City	A file from the software system.
District	Class (contained within the specific file and hence city in the visualisation).
Building	Methods.

from Sun and is taken from the String class. The method

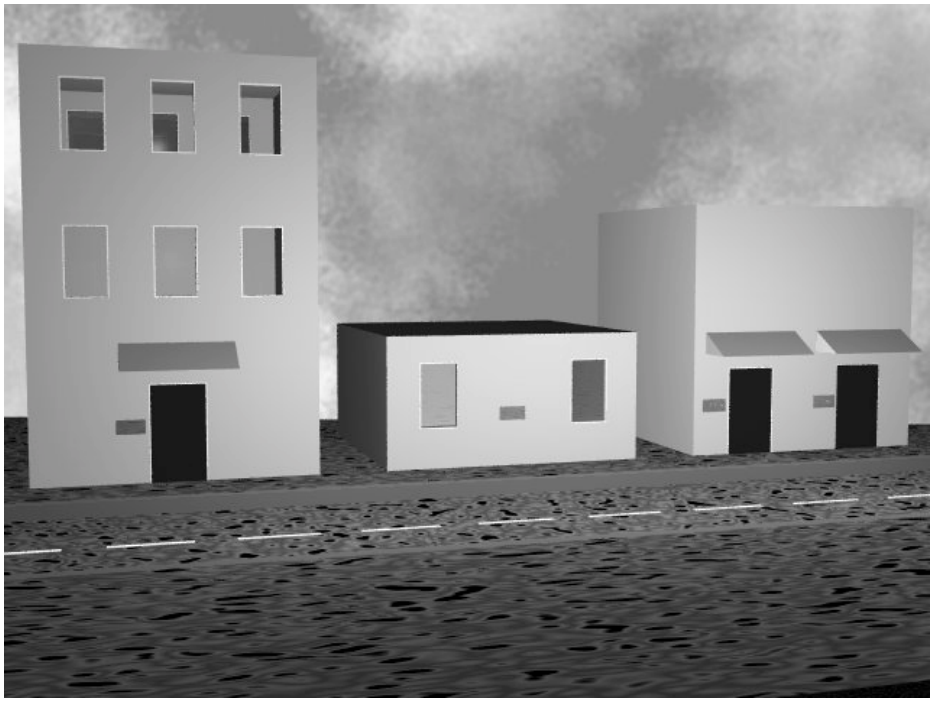

Figure 2 - A street from a *Software World*

Within districts, the methods (buildings) are laid out in a block (grid). The reason for this is that it means that assumptions about the relationships (if any) between methods do not have to be made. It also allows for easier, sensible, evolution of the visualisations as it can easily be shown to the user where changes have occurred in the underlying code. If a district (or even a city with districts) gets too dense then the visualisation will represent it as such. It is then a visual indication that restructuring of the code could be beneficial. The user of the visualisation can then "bulldoze" the crowded areas!

As an example of this the street scene, Figure 2, represents part of a district (String) and shows buildings (compareTo, toCharArray, and substring). The size of the buildings represent the number of lines of code (normalised), the number of doors represents the number of parameters, and the number of windows shows the number of variables declared in the method. The colour of the building shows whether the method is private or public. The name of the method is on a plaque by each door, and for those buildings with no doors (if there are no parameters) one plaque is placed on the building for information.

The code that this visualisation is based on is from the 1.1.x distribution of the Java programming language signatures of the illustrated methods can be seen in this code snippet:

```
public int compareTo(String anotherString)
{ . . . }

public char[] toCharArray()
{ . . . }

public String substring(int beginIndex, int
endIndex)
{ . . . }
```

4.2 Exploration and Query

There are two main, general, activities involved in program comprehension. The first of these is overview browsing and exploration, where the code is scanned for items that may be of interest. The second is a more directed form of exploration, in that the start point and item of interest is known and by moving to this point in the code detailed browsing and exploration can then take place. In summary these activities can be known as exploration and query respectively. They also map to the top-down and bottom-up strategies of program comprehension.

The *Software World* visualisation is able to support both of these forms of user enquiry about the program code. Exploration at any level is done through the use of

the visualisation and it's metaphor, for example, by locating a place using the overview map, travelling to that place and then exploring on foot the buildings and environment that represents the program code at that point. For example, in Figure 2, exploration has led to a street that is part of a district (String) that shows buildings (methods) that are part of that district (class).

Queries are supported through the use of a query system providing a leap into the visualisation system and through the bookmarking facilities that each user has of the visualisation. These both enable the user to "jump" into the visualisation at a given point without having to explore the higher levels. For example, a query such as "What are the methods of the class String?" would generate the visualisation shown in Figure 2.

5. Results, Future Directions and Conclusions

This paper has shown that it is possible to move away from two and three-dimensional abstract visualisations. One way forward has been presented that makes use of a three-dimensional real world metaphor. The use of this real world metaphor supports the natural perceptive abilities of programmers and maintainers.

There are problems when defining real world metaphors for Java (or any programming language). Once a metaphor has been decided upon then there is the problem of the mapping from language element to the metaphor. Table 1 showed simple mappings, whilst more detailed mappings can be seen illustrated in Figure 2. These detailed mappings for the *Software World* visualisation consider the attributes of the simply mapped elements. The use of the methods from the String class (shown in Figure 2) as the buildings show the simple mapping. The height of the building, number of doors and windows, and the plaques provide detail based on lines of code in the method, number of parameters, number of local variables and the name of the method.

The visualisation has been shown to facilitate exploration and query. By supporting these two activities the *Software World* visualisation is able to sustain top-down and bottom-up comprehension strategies. Comprehension work by Chan and Munro [3] shows the benefit of supporting more than one comprehension strategy.

A future aim is to develop the visualisation to support user communication and collaboration. One intended development is to make use of notice boards within buildings (methods in terms of Java) where functionality and programmer understanding notes can be attached for other users to read. Electronic sticky notes can be used to update and improve the knowledge about the method on these displays, with eventual conversion to sheets on the notice board (a more permanent form of documentation) if the information is of value. The use of workspaces for users of the visualisation is also planned. This workspace in the *Software World* is intended to be an office in the visualisation system which allows users to define and use hyperlinks, bookmarks and keep notes. These features have all been identified as good facets of program comprehension systems [14].

Another future aim is to develop the analysis and mappings to deal with the evolution of the underlying code. The evolution considered to date has been that of insertion and deletion of classes and methods, but not currently low level changes to code. In the *Software World* insertion can be shown by scaffolding around new buildings or districts, with demolition signs and fences around deleted buildings and districts.

Prototype tools have been developed to implement the ideas in the Software World. A parser for Java has been constructed to provide the information for a set of mapping tools that will then generate input into the virtual reality software. Virtual worlds have been created manually, but automation of this process is in progress.

The visualisation presented in this paper, the *Software World*, has shown a way to make use of three dimensions and virtual environments in trying to create program comprehension support techniques to overcome the limitations of traditional techniques. Background information and examples have been presented, along with the intended future directions of the work. It is important that research effort is directed towards new and useful visualisations so that the support mechanisms for program comprehension are improved and refined, especially with the complexity and change program code is subjected to.

Acknowledgements

This work is partly financed by an EPSRC studentship.

References

[1] 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. ISBN 0-89791-765-0. pp77-85.

[2] 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.

[3] P. S. Chan, M. Munro.
PUI: A Tool to Support Program Understanding
Proceedings of the IEEE 5th International Workshop on Program Comprehension, Dearborn, Michigan, May 28-30, 1997, pp192-198.

[4] E. F. Churchill, D. Snowdon.
Collaborative Virtual Environments: An Introductory Review of Issues & Systems.
Virtual Reality, Vol. 3, No. 1, 1998, pp3-15.

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

[6] W. Gibson.
Neuromancer
Paperback edition publiched in 1995 by Voyager, an imprint of HarperCollins Publishers. ISBN 0 00 648041 1

[7] 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.

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

[9] 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.

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

[11] 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.

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

[13] N. Stephenson.
Snow Crash
Paperback edition published by ROC (Penguin Group) 1993. ISBN 0-14-023292-3.

[14] M.-A. D. Storey, F. D. Fracchia, H. A. Müller.
Cognitive Design Elements to Support the Construction of a Mental Model During Software Visualization.
Proceedings of the IEEE 5th International Workshop on Program Comprehension, Dearburn, Michigan, May 28-30, 1997, pp17-28.

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

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