Joint EUROGRAPHICS - IEEE TCCG Symposium on Visualization (VisSym '99), May 26-28, Vienna, Austria, 1999

Application of Information Visualization to the Analysis of Software Release History

Harald Gall[1], Mehdi Jazayeri[1], Claudio Riva[2]

[1] Distributed Systems Group, Technical University of Vienna,
Argentinierstrasse 8/184-1, A-1040 Wien, Austria
{gall, jazayeri}@infosys.tuwien.ac.at
[2] Nokia Research Center, Software Technology Laboratory,
P.O. Box 45, FIN-00211 Helsinki, Finland
claudio.riva@research.nokia.com

Abstract. We present our experiences in applying information visualization techniques to the study of the evolution of a large telecommunication software system. We used the third dimension to portray the temporal evolution of the system and color to display software attributes. The visualization was surprisingly successful in uncovering interesting and useful patterns in the system's evolution. To do the visualization, we built a tool that combines off-the-shelf components: a database for storing software release data, VRML for displaying and navigating three-dimensional data, and a web browser for the user-interface. The tool is published on the web. The tool is capable of providing effective views of data that are always kept by software development organizations but are often ignored. Information visualization makes it possible to exploit such historical data about past projects to help in the planning stages of future software projects.

Keywords: software release history, third dimension, color, World Wide Web

1. Introduction

Large software systems undergo significant evolution throughout their lifetimes. Often, the structure of a system after a few releases is very different from the structure it had at its initial introduction. Such drastic changes in the structure are due to new functional requirements on the system after it has been released, improper initial design that does not accommodate enhancements easily, or inadequate performance that can only be ameliorated by major restructuring. The history of a software system's evolution contains information that can be used by managers and system developers in both understanding the reasons for the changes in the current system and in guiding the development of future systems. Unfortunately, such information is only passed on through informal anecdotes without any real understanding. Even though most companies collect lots of data about system releases, such information is rarely used in

future planning. The main reasons for not using the information are the huge volume of data and the difficulty in drawing any coherent conclusions from the data quickly.

This paper reports our experience in trying to make sense of a database of release histories of a telecommunication system consisting of about 10 million lines of code (10 mloc). The company had developed a custom database to maintain data about system releases. The database provided basic tools that an experienced engineer could use to pose queries about the structure of the software. Our goal in the project was to find general ways to discover patterns in the evolution of the software. We identified some key metrics to measure and initially used simple two-dimensional graphs to plot them [8]. Already, these graphs showed significant trends in the system's evolution that were not known to the developers. We then applied three-dimensional graphics and color to visualize further information about the release histories. The techniques were useful in quickly summarizing the large volume of data in the release database [11]. We believe that the techniques are generally applicable to the analysis of software release histories and can be a valuable tool for software engineers and managers in the planning phases of system development and enhancement. Our general conclusion is that three-dimensional information visualization is readily applicable to the analysis of release histories and should become a standard tool in the software manager's toolbox. To our knowledge, no such techniques are currently used or are even being considered in software development organizations.

This work was supported by the European Commission within the ESPRIT Framework IV project ARES (Architectural Reasoning for Embedded Systems). The technique was developed in conjunction with an industrial case study as part of the project ARES.

2. Related Work

Information visualization is currently being explored to examine and comprehend masses of data regarding software systems. Most software visualization work has concentrated on code-level visualization or on performance visualization. Our focus is on structural and architectural level of large software systems. Several papers have addressed this area.

Eick et al. have developed a set of tools for visualizing several classes of data [5] [2] [7]. In particular, SeeSys [1] is a visualization system for displaying the statistics associated with code. Johnson and Schneiderman concentrate on visualizing hierarchical data using Treemaps [18] [12]. The approach uses the screen-filling method for displaying the attribute values of hierarchy components. The color of the region can provide an additional attribute.

Software visualization using three dimensional display is a current research area. Koike developed a 3-D framework to visualize the execution of two parallel/concurrent computer systems [14]. Koike et al. also proposed a 3-D framework for both version control and module management [13] [4]. Ware et al. investigate how to visualize object oriented software in three dimensions [19].

3-D graphics for displaying hierarchical structures have been development by Card, Mackinlay and Robertson [17]. Their work provides ConeTree which is a technique for displaying hierarchical objects in 3-D space. Improvements of ConeTree are described by Carrière and Kazman [3] and by Koike [15].

3. Visualizing the Software Release History

The case study's architecture is organized as a *layered system*. The structure is a tree hierarchy with four levels. The top level is the *system level*. It is based on the *subsystem level* (second level), *module level* (third level) and *program level* (fourth level). Each level consists of one or more elements. The elements in each level are named corresponding to the names of the levels: *subsystems*, *modules*, *programs*.

Table 1 shows an example of the software release history. Each row represents the version numbers of a *program* element. The first and second column (labeled Sub and Mod) contain respectively the *subsystem* and *module* to which the *program* belongs. The third column (labeled Prog) contains the name of the *program*. The columns labeled from 1 to 20 represent the twenty system releases. The version number of each element is the RSN of the release where it had the latest change. For example, if a *program* changes its implementation at release 1, 2 and 5 then its version numbers are the sequence < 1 2 2 2 5 5>.

The database is populated with 20 releases of the software product. Each release contains 8 *subsystems*, 47 to 50 *modules* and 1500 to 2300 *programs*.

Table 1. An example of the database in which the software release history is stored.

Sub	**Mod**	**Prog**	**1**	**2**	**3**	**4**	**5**	**6**	**…**	**20**
Sub1	Mod A	Prog A	1	2	2	2	5	5	…	18
Sub1	Mod A	Prog B	0	2	3	4	5	0	…	20

Following the success of the 2-dimensional graphs in the previous study [9], we decided to explore the application of three-dimensional visual representations for examining the software release history. The purpose of the representation is to present the data contained in the evolution history in a more comprehensible way.

The software release history consists of three entities: time, system structure and version numbers. These entities are visualized in one three-dimensional diagram. The coordinates are called *x*, *y*, *z*. The coordinate *z* stores the time information. This coordinate is expressed in release sequence number (RSN). The system structure is displayed by 2-D or 3-D graphs. The graph is spatially positioned along the coordinate *z* at the value of its own RSN. Version numbers are shown using colors. Colors are associated with version numbers through a color scale. The structure elements are painted according to their own version number. Fig. 9 provides an example of color scale. In this way, each color represents an attribute value.

Four 3-D graphical objects are used to visualize the abstract elements of the system structure: cubes, spheres, lines and bars. The composition of these graphical objects

represents a graph and serves to visualize the system structure of one release. Cubes represent modules of the system structure (i.e. *subsystems*, *modules* and *programs*). The color of the cube denotes its version number. A sphere represents the topmost module of the system structure (i.e. *system*). The color denotes the release sequence number (RSN) of the system structure. A line represents the "contains" relationship between two modules. A percentage bar is a graphical object that offers a compact representation of a group of elements. It is composed of a set of colored blocks. Each block has two properties: relative size and color. The relative size is proportional to the percentage of modules that have the same version number. The color depends on the version number through the color scale. For comparative analysis, size is the most effective perceptual data-encoding variable [6].

Section 6.1 presents examples of three-dimensional visualization. Section 6.2 shows the use of 3-D diagrams for examining the historical evolution. Section 6.3 presents 2-D representations obtained projecting the three-dimensional graph.

3.1 3-D Visualization of structure

The case study has a tree hierarchical structure. A tree hierarchy can be visualized by 2-D and 3-D tree graphs. 3-D graphs are implemented using the Cone Tree technology [17]. 2-D graphs use the same Cone Tree technology in two dimensions. Fig. 7 shows the whole structure of one system release of the case study. The whole structure of the system is visualized in one view. The 3-D layout allows the viewer to navigate the system structure. Fig. 8 is a zoom on one *subsystem*.

3.2 3-D Visualization of historical evolution

The main purpose of our approach is to use 3-D diagrams to examine the historical evolution of the system structure. Three-dimensional diagrams help to visualize both historical and structural information. The third coordinate is expressed in RSN. For each value of RSN the associated system structure is visualized by one 2-D tree graph. Fig. 1 shows an example. The 3-D diagram represents 10 releases of one of the *subsystems*. For each release, the structure of one *subsystem* is visualized. The structure contains all its *programs* and *modules*. In Fig. 1 the *program* elements are visualized by percentage bars. In Fig. 2 the same *subsystem* is visualized without percentage bars. In this way, *program* elements are distinctly visible.

3.3 2-D Visualization

2-D visualizations can be obtained by projecting the 3-D diagram onto 2-D space. Fig. 2 shows how to make a simple projection by compacting and rotating the diagram. The picture obtained by the projection reveals a concise and informative representation of system evolution. Fig. 3, Fig. 4, Fig. 5 and Fig. 6 contain examples obtained in this way. Each picture visualizes the evolution of one *module* element through the percentage bars (RSN is vertically directed from up to down like in Fig. 6). Each row is associated with a system release. For each system release (from 1 to 20), the percentage bars are displayed. The percentage bars show with the same color the percentage of modules that have the same version number. This representation highlights the

critical times—when major changes seem to have happened—in the evolution of the whole *module*.

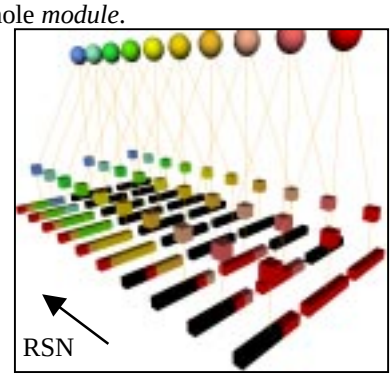

Fig. 1. Visualizing the history

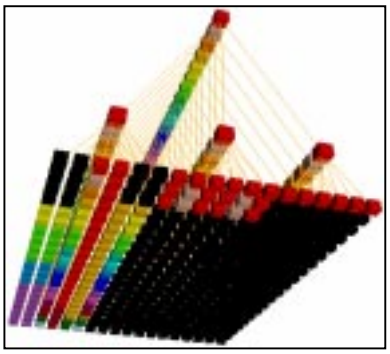

Fig. 2. Rotating the 3-D graph

4. Visualizing to understand

We applied the graphical representation to the examination of the case study. This section presents the results and the advantages we achieved by visualizing the data contained in the database. A more detailed description of the results, of interest to software engineers, can be found in [11][16].

3-D Layout

Three coordinates allow to visualize both system structure and historical evolution of the system. The viewer can perceive both structural and historical information looking at one view. Most important, the viewer can view the changing system structure over time. This important view never comes out in software projects. The viewer can also navigate into the representation to find the best perspective for looking at the data. The advantages are summarized below:

- Visual perception of the structure: in understanding abstract information human minds create visual representations to simplify the process. Visualization can provide these representations for the viewer and relieves his/her mind. In this way imagination and creativity are free to address new ideas. Moreover, three-dimensional layout, rendering and shading is pleasant for humans because they can simulate the reality to which our mind is naturally accustomed.
- Visual retrieval of data: the visualization system we have developed allows the user to click on an element to retrieve its properties. For example, clicking on a *program* element the user can extract its name and version number. This is faster and more attractive that setting up a textual query and submitting it to the database. 3-D layout combined with this functionality allows navigating the system structure and quickly retrieving the information stored in the database.
- 3-D Navigation: the user can virtually navigate the 3-D graphical space for examining the system structure. Users can choose the best point of view to concentrate

on specific data sets. New representations can be easily obtained by just rotating, zooming, projecting or moving the graph.

Observations on the historical evolution

The 2-D representations make evident some useful trends in the historical evolution of the modules. As and example, we examine the 2-D representation in Fig. 6. This view can be obtained as a projection of the 3-d representation of Fig. 2. It visualizes the evolution of *program* elements belonging to a *module*. At release 1 (first row) all programs have the same version number 1 (red color). At release 2 (second row) 96% of *programs* have version 2 (pink color) and the rest (4%) have version number 1 (red color). This means that a majority of *programs* (96%) changed their implementation and therefore the *module* has been extensively modified. The motivation for such a big modification may be found by direct inspection of the code or *module*'s documentation. Whatever the reason, such a view is a cause of concern and should trigger a closer examination.

Still in Fig. 6, at release 8 the *module* has its last major modification. In fact at release 8 (eighth row) the large dark green zone shows that many *programs* have changed their implementation. Then from release 8 until release 20 many of these *programs* maintain their version number: for each release from release 8 to 20 the green color zones are the biggest ones. Between release 8 and 20 a small fraction of *programs* change their version number: in Fig. 6 this is reported by the regions on the right colored with blue, purple and dark green. The *programs* change at releases 9, 10, 11, 12, 14, 15, 17, 19. It is clear that the *programs* have stabilized.

Changing rates

Changes of version number are visualized as changes in colors. This allows a qualitative and intuitive indication of the changing rates. High changing rates are identified by regions where colors change very quickly. Low changing rates are identified by regions with plain color. In Fig. 3, the first two *modules* have very high changing rates, almost all the *programs* change their version number every release. This anomalous behavior of the module was already detected with the statistical analysis [9]. In Fig. 4, the third *module* has very low changing rates, only at release 19 and 20 some *programs* change their version.

Growing rates

In visualizing the *program* elements, we decided to use a black color to represent those elements that have been removed by the module or that have not been implemented yet. The system has a common structure in all the releases, and the black colored elements are elements not present in a particular release. This visualizes the growing rate of the *module* in terms of amount of *program* elements. In fact, the size of the black region is associated with the amount of not-present *programs*. The modification in size of this region is an indication of the growing rate. If the black region diminishes with time, the *module* is adding *programs*. If the black region is not present, it means that the *module* has reached its maximum size. If the black region increases with time, the *module* is removing *programs*. In Fig. 3, the third *module* has

high growing rates. In Fig. 5 the eighth *module* on the second row has been removed from the system.

Patterns
One advantage of visualization is that it enables and exploits the humans' pattern matching skills. This ability relies on the human visual system that is able to detect regions characterized by repetitive use of the same shape, the same color, the same filling or the same texture. Identification of patterns is needed to discover dependencies and relationships between system elements. An example can be identified in Fig. 5 where all the *modules* of one subsystem are visualized using the 2-D.

Considering Fig. 5, we can identify three *modules* that have the same color filling. They are the last *module* on the right of the first row, the fifth and the sixth *modules* on the second row. The common attribute is that their representations have a large region, filled with the same pink color. This region begins at release 2 and extends until release 20. This means that for each *module* many *programs* have the same version number 2 (pink color) and these *modules* don't change in any of the releases. This observation could lead to identify relationships or commonalties among the *programs* or could lead to identify more generally that the *programs* of different *modules* have the same behavior.

Comparisons
The visual representation allows the user to make comparisons between different data sets:

- Comparisons among modules: 2-D graphs and 3-D graphs allow to visualize the structure of the system. With this representation the viewer can navigate through the data to compare the modules of the system. Different views of the same configuration of modules can make such comparisons easier. Comparing colors the viewer automatically makes comparisons of values.
- Comparisons of different measures: a limitation of the case study is that it contains data only about version numbers. Therefore, it is not possible to investigate different measures. The idea is that by comparing the same modules visualized with different attributes, it should be possible to identify commonalties or differences.
- Comparisons of different releases: the historical (temporal) evolution of a *module* can be visualized in a compact form that shows the changes of its *programs*. Such view allows the comparison of the same *programs* in different releases and the historical evolution of different *modules*.

5. Visualization on the web

To support the 3-D representations reported in this work, we have developed a visualization tool called 3DSoftVis. The tool consists of three components: a database, a 3-D visualization engine and the graphical user interface. The database contains the data regarding the evolution of a software system. The 3-D visualization engine transforms

the data extracted from the database into 3-D models. The user interface presents the graphs to the user through multiple windows and allows the viewer to customize the views interactively.

The tool has been developed for the World Wide Web platform using web components such as Java, VRML (Virtual Reality Modeling Language), JDBC and HTML. The platform-independence technologies which compose the system allow to publish it on the web [20], making it available to any user with access to the network. The major features of 3DSoftVis are summarized below. A more detailed description can be found in [10].

- World Wide Web Application: the web architecture makes the tool accessible from the network. The application and the database reside on a remote server and they can be easily upgraded by developers. On the client side 3DSoftVis requires a minimal configuration and relieves users from installation problems.
- Graphical engine based on VRML: VRML is an easy solution for developing 3-D graphical tools. We are not 3-D graphical experts and the choice of VRML simplified considerably the implementation of the graphical engine. VRML is also a standard web component that can be easily integrated in a web application.
- Collaborative environment: data and that application for their analysis are bound together on the server machine, so that the most updated database and the most recent release of the tool are immediately available. Groups of researchers can use 3DSoftVis by sharing the same working environment without the constraint of having to be located nearby or working on the same machine-platform.

6. Lessons Learned

Our experience with software visualization has been successful in providing insights into the evolution of the software system in the case study. Since the use of 3-dimensional and color visualization is rather rare in software engineering studies, here we summarize some of the more general lessons we have learned in the hope that they will be useful in future studies.

1. Information visualization is a powerful technique for examining huge data sets and for understanding abstract information. Visual representations aid humans comprehension because they change the process of understanding from being a cognitive task to being a perception task. Visual patterns are easy to detect and support discovery of hidden dependencies in the data. These advantages, which have been exploited in other application areas, are also available in the study of abstract entities such as software structure and qualities.
2. Three-dimensional visualizations allow the viewer to visually perceive the abstract information about the structure of a software system. Navigation makes it possible to change the viewpoint and quickly extract the data from the database. It could serve as a functional interface to a configuration management system.
3. Color and region filling are two effective techniques for presenting data and enable to quickly detect patterns. In our work, 2-D visualizations have been used to visually detect unstable modules and to discover unknown dependencies among system

components. The fact that color was helpful in detecting patterns is rather surprising because of the lack of any natural relationship between software and color.

4. The visualization system has been developed with components that are available in the public domain. Without any particular expertise in graphics, we could build a 3-D graphical application in a reasonable time. The web platform allows publishing the tool on the web and simplifying the process of maintaining the database.
5. Evolution history of software systems contains valuable information for both software developers and managers. This information concerns the quality of the system. Visualization is the first technique we know of to make sense of the data that quality assurance groups collect (and usually stay dormant).

7. Summary and conclusions

In this paper, we have presented our experiences in applying information visualization techniques to the study of the evolution of a large telecommunication software system. It rather surprising that the more advanced computer science technologies are not usually applied within computer science itself. Although information visualization has been successfully applied in many application areas, its application to the study of software has been rare. It is certainly not part of any organized software process model. Our experience shows that the planning phases of software development can certainly benefit from the use of information visualization on software release data. Traditionally, data are kept and analyzed by quality assurance groups with hardly any feedback to, or influence on the work of development groups. Information visualization could provide an effective medium of communication and collaboration between the quality and development groups.

Also surprising was the fact that rather sophisticated graphics techniques are readily available and can be used by non-experts to build useful visualization tools. Such tools can then be published on the web for use by large communities of researchers and engineers.

We were surprised by how well color depicted the patterns in the system's evolution. Based on this experience, we believe that color has many potential applications in the display of different software attributes.

References

1. Baker M. J., S. G. Eick, Visualizing Software Systems, AT&T Bell Laboratories, 1994.
2. Ball T. A., Eick S. G., Software visualization in the large, IEEE Computer, April 1996, pp. 33-43.
3. Carrière J. and Kazman R., Interacting with Huge Hierarchies: Beyond Cone Trees, Proceedings of Information Visualization '95, Atlanta, Georgia, Oct, 1995, pp. 74-81.
4. Chu H. and Koike H., How does 3D Visualization Work in Software Engineering ?: Empirical Study of a 3D Version/Module Visualization System, International Conference Software Engineering 98 (ICSE 98), 1998.

5. Eick S. G., Fyock D. E., Visualizing corporate data, AT&T Technical Journal, January 1996, pp. 74-76.
6. Eick S. G., Engineering perceptually effective visualizations for abstract data, in Gregory M. Nielson, H. Mueller, and H. Hagen, editors, Scientific Visualization Overviews: Methodologies and Techniques, IEEE Computer Science Press, February 1997, pp. 191-210.
7. Eick S. G., Steffen J. L., Sumner E. E. Jr, Seesoft - A Tool For Visualizing Line Oriented Software Statistics, IEEE Transactions on Software Engineering, Vol. 18, No. 11, Nov 1992, pp. 957-968
8. Gall H., Hajek K., Jazayeri M., Detection of Logical Coupling Based on Product Release History, International Conference on Software maintenance (ICSM '98), Washington, DC, 1998.
9. Gall H., Jazayeri M., Klösch R., and Trausmuth G., Software evolution observations based on product release history, International Conference on Software maintenance (ICSM '97) (Bari, Italy), pages 160-6, IEEE Computer Society Press, September 1997.
10. Jazayeri M., Riva C., Experiences with developing Native World Wide Web Applications, http://www.infosys.tuwien.ac.at/~riva/Docs/bibl/jaz98a/full/, to submit.
11. Jazayeri M., Riva C., Gall H., Visualizing the Software Release Histories: the use of Color and Third Dimension, ACM Transactions on Software Engineering and Methodology, 1998, submitted, also in Technical University of Vienna, Distributed Systems Group, technical report TUV-1841-98-14.
12. Johnson B., Visualizing Hierarchical and Categorical Data, Ph.D. Thesis, Department of Computer Science, University of Maryland, 1993.
13. Koike H., Chu H.: VRCS: Integrating Version Control and Module Management using Interactive Three-Dimensional Graphics, Proceedings of 1997 IEEE Symposium on Visual Languages (VL'97), 1997, pp.170-175.
14. Koike H., The role of another spatial dimension in software visualization, ACM Transactions on Information Systems, 11(3), July 1993, pp. 266-286.
15. Koike H., Yoshihara H., Fractal Approaches for Visualizing Huge Hierarchies, Proceedings of the 1993 IEEE Symposium on Visual Languages (VL'93), 1993, pp.55-60.
16. Riva C., Visualizing Software Release Histories: The Use of Color and Third Dimension, Master's Thesis, Politecnico di Milano, Milan, Italy, June 1998, also in http://www.infosys.tuwien.ac.at/~riva/Docs/bibl/riva98/full/
17. Robertson G. G., Mackinlay J. M. and Card S. K., Cone Trees: Animated 3D Visualizations of Hierarchical Information, Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '91), ACM Press, 1991, pp. 189-194.
18. Shneiderman B., Tree Visualization with Treemaps: A 2D Space Filling Approach, ACM Transactions on Graphics, Vol. 11, No. 1, 1992, pp. 92-99.
19. Ware C., Hui D., Franck G., Visualizing Object Oriented Software in Three Dimensions, Conference Proceedings of CASCON' 93, Toronto, Canada, October, 1993, pp. 612-620.
20. 3DSoftVis Demo, Technical University of Vienna, Distributed Systems Group: http://www.infosys.tuwien.ac.at/~riva/vis

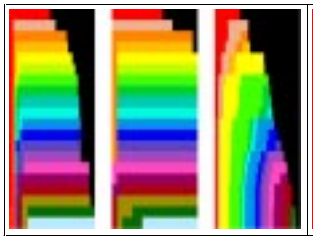

Fig. 3. Subsystem A

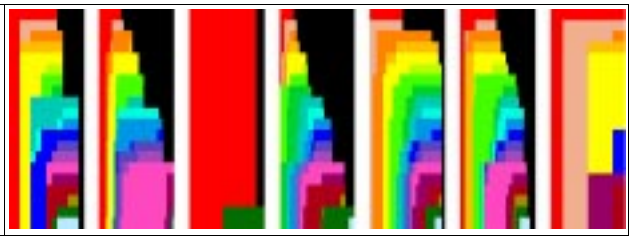

Fig. 4. Subsystem B

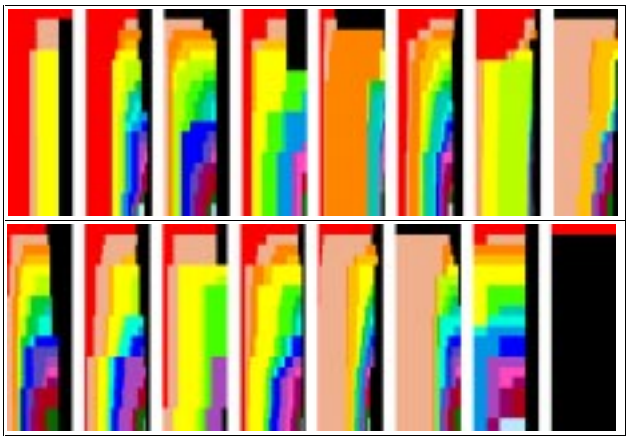

Fig. 5. Subsystem C

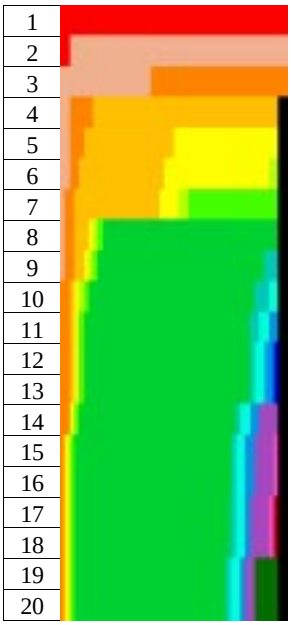

Fig. 6. A module

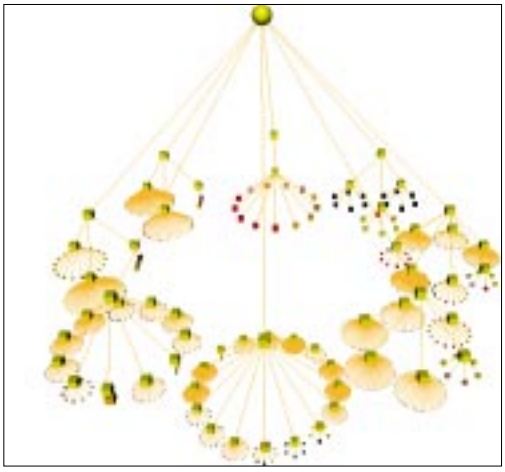

Fig. 7. The structure of the system

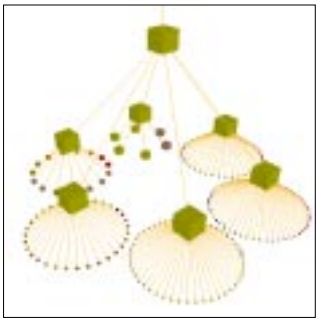

Fig. 8. Focusing on one subsystem

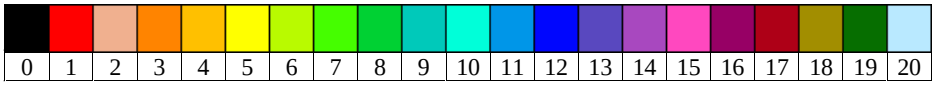

Fig. 9. The color scale