Visualization of Software Metrics

Marlena Compton

Software Metrics

SWE 6763

April 22, 2009

Abstract
Visualizations are increasingly used to assess the quality of source code. One of the most well developed visualizations used for depicting source code is the treemap. This paper sets forth the basic principles of data visualization and how they have been applied to the visualization of software using treemaps and CK metrics.

1. Introduction

With software systems growing larger and larger, there is a need to be able to quickly assess large amounts of software for different attributes such as complexity or quality. Treemaps, a type of visual representation for hierarchical data, have been successfully used to represent source code and its complexity, visually. The success behind the application of this particular type of visualization to source code lies in the adherence of the author of Treemaps to Edward Tufte's principles of data visualization. In this paper, I will trace the path from understanding what makes a good visualization to how treemaps successfully apply these techniques for software visualization. In the next section, I will summarize some of Tufte's characteristics for creating excellent visualizations with integrity. In section 3, I will discuss metrics that are frequently applied to software. In section 4, I will give some of the history of treemaps and describe the different types of treemaps. In section 5, I will bring the previous sections together and show the characteristics of good data visualization as they are applied with Treemaps and how Treemaps can be used to represent the structure of object oriented software and the application of metrics to.

2. Tufte's principles of data visualization

Before data visualization can be employed for any purpose, there must be an understanding of what constitutes a good visualization. In his book, The Visual Display of Quantitative Information, Edward Tufte explains the principles that make a good data visualization. These principles are very useful in assessing the quality of data visualizations. High quality visualizations represent extremely large sets of numbers in a very small space. The data they represent is clear to the viewer and not distorted in any way. A good visualization allows the viewer to easily compare the information it is presenting. Data can be studied with the finest micro-level detail but also forms another layer of information at the macro level. Everything in an excellent visualization fits together like pieces in a puzzle in much the same way as a great painting or other work of art. This means that the description and labeling of the data are tightly integrated with statistics being shown (Tufte 13).

Tufte introduced several terms used in describing visualizations. Data ink refers to ink in a graphic that represents the statistics or other information in the graphic. It excludes marks such as gridlines (Tufte 91). The data ink ratio is the amount of data ink divided by total ink used to print graphic (Tufte 93). Data ink can be used for multiple purposes such as in a stem and leave plot. Numbers contained in this type of graphic represent a distribution. Because of the layout of the stem and leave plot, each number contributes to an overall picture of the distribution(Tufte 139, 140). Chartjunk is extra decoration that appears in a graphic but is not related to the

information contained in the graphic. The moire effect is an optical effect which occurs when "a design interacts with the physiological tremor of the eye to produce the distracting appearance of vibration and movement," (Tufte 107).

Integrity in a visualization is very important. There are several characteristics that apply when assessing the integrity of a graphic. Labeling should be used extensively to dispel any ambiguities in the data. Explanations should be included for the data. Events happening within the data should be labeled. All graphics must contain a context for the data they represent. Numbers represented graphically should be proportional to the numeric quantities they represent. Number of dimensions carrying information should not exceed dimensions in the data. Variation should be shown for data only and not the design of the graphic(Tufte 77).

3. Metrics and Software Visualization

While treemaps were originally conceived to represent a hierarchical structure, they can also show metrics about the structure. In the case of software, commonly used software metrics can be expressed using treemaps. Some of the most commonly used metrics for software are CK metrics (Kan Section 12.2.3 page 7). Weighted methods per class (WMC), is the number of methods in a class, unless a complexity metric is applied to each method (Chidamber, Kemerer 482). This complexity is typically cyclomatic complexity (Kan Section 12.2.3 page 7). In the case where method complexity is not taken into account, WMC is the sum of all weights calculated (Chidamber, Kemerer 482). Depth of inheritance (DIT) is the largest length from a node to the root of an inheritance tree (Chidamber, Kemerer 483). Coupling is measured based on Coupling Between Objects or (CBO) and occurs when the methods of one object act on the methods of another object (Chidamber, Kemerer 486). Cohesion is measured with the metric Lack of Cohesion in Methods (LCOM) (Chidamber, Kemerer 488). Since high cohesion is desirable, a higher number for this method indicates higher complexity which can be less desirable (Kan Section 12.2.3 page 7).

Apart from CK metrics, cyclomatic complexity is also a metric that is frequently used to assess software. It could easily be applied in a treemap using. Cyclomatic complexity was designed to show the maintainability and testability of souce code. It is calculated by counting different paths through a program based on usage of the keywords, "if while repeat for and or," (McConnell 395).

4. Treemaps

A treemap is a type of data visualization used to represent hierarchically structured data in a format that allows the viewer to see the whole structure on one screen. The first treemap display was created by Ben Schneiderman for displaying files on his computer. He wanted to know which files on his hard drive were taking up the most space, and was unhappy with viewing the file sizes in a list. He was frustrated because when he viewed his files as a list with each file as a line, he could not see all

of his files at once. He decided to create a way to view all of them at the same time. His goals, in creating this map of his system, were to use space as efficiently as possible, to create something interactive so that information could be obtained more quickly and easily, and to make the visualization very easy to understand and esthetically pleasing (Johnson and Schneiderman, 284).

At the time that Schneiderman began his search for a new type of visualization, there were several methods for viewing the files on a system. These included the command line, an outline format, a window format and a tree. Viewing files on the command line, only immediate sub-directories were visible and windows would obscure each other. (Johnson and Schneiderman, 285).

Hierarchical structures contain 2 types of information. They contain information about the structure itself and information about the content within the structure. There are 3 main ways to view hierarchically structured information. These include listings, outlines and tree diagrams. The disadvantage of listings, is that they do not show structural information well. Outlines can show structure, but because they use indentation, only a few lines can be viewed at once. Schneiderman approximated that when viewing system files in a tree format, roughly 50% of the space on the screen is wasted. Because of the disadvantages in listings and outlines, Schneiderman decided to concentrate his efforts on optimizing a tree display for hierarchical information by decreasing the amount of wasted space in the tree display (Johnson and Schneiderman, 285).

For a new perspective, Schneiderman began perceiving directories as sets. This is the step between having a tree and a treemap. Beginning attempts were made at incorporating venn diagrams into a tree, but again, too much space was being wasted in the display. He then began to work with nesting which advanced him to the final idea of a treemap (Johnson and Schneiderman, 286.

Treemaps consist of nested rectangles representing the entities and sub-entities in a hierarchical structure. The size of the rectangle depends on some type of weighted factor, in the case of a treemap representing a file system, this is the file size (Johnson). In this way, larger directories take up more space in the visualization. Color can also be used to delineate between different types of files (Johnson and Schneiderman, 287).

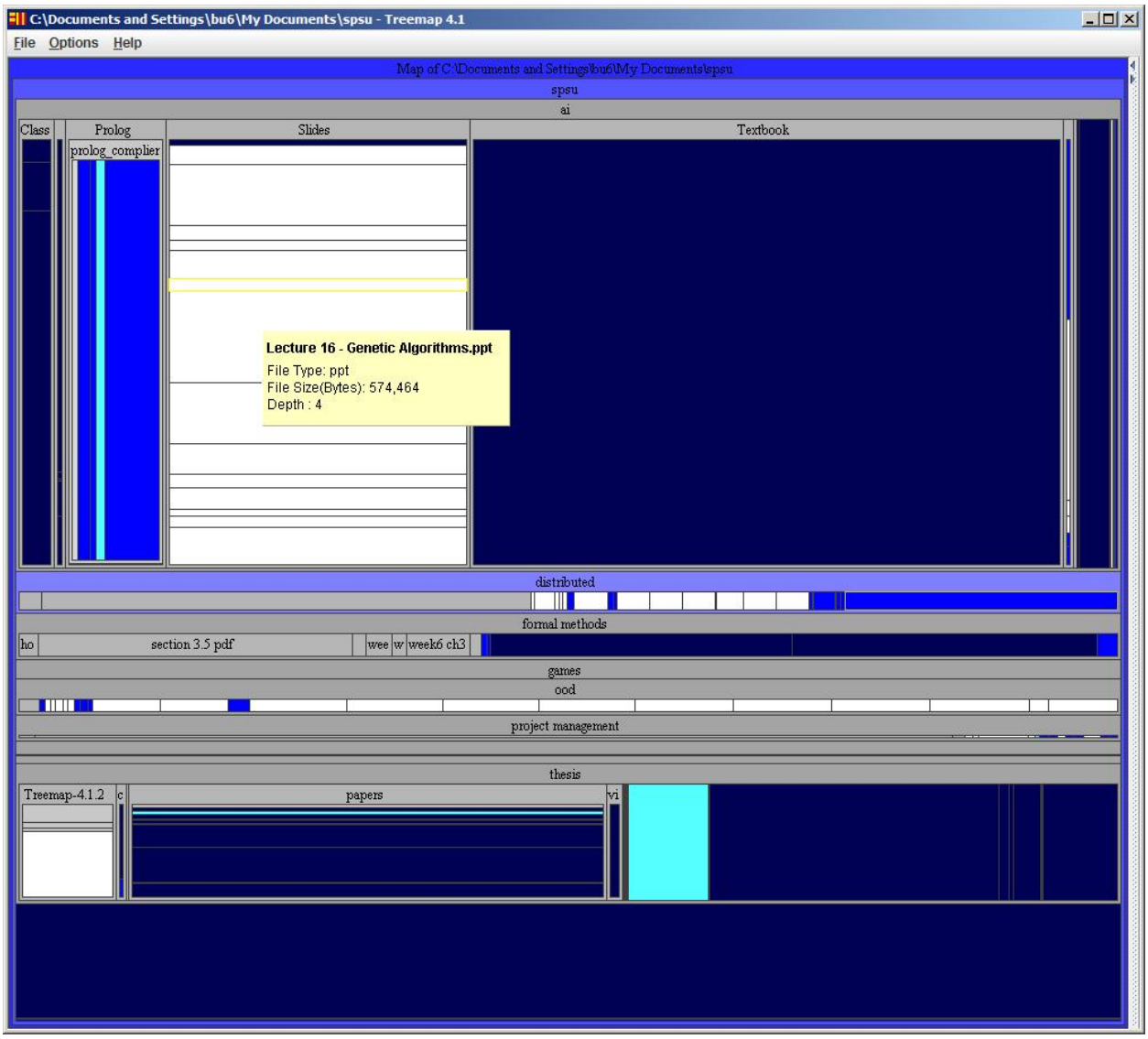

Fig 1. Slice and Dice Layout

The above treemap example shows a directory for organizing files used in classes. The area of the boxes corresponds to their file size. Colors are used to represent different types of files. Adobe acrobat files are dark blue, Powerpoint files are white, executable files are royal blue and zipped files are turquoise. Directories form gray borders around the files and are labeled. In this example, the directory containing files for an artificial intelligence class takes up a little over half of the disk space being used for class files. Nesting occurs within the bounding box of the ai directory to represent subdirectories for slides and prolog. Within the slides directory there are a few files including a set of power point slides about Genetic Algorithms. Interactivity is used to elaborate on the content of the tree. The pop-up box shows the full name of the file, the file type, file size and depth within the tree (Johnson and Schneiderman, 287).

The algorithm used to determine how the rectangles will be laid out on the screen is called the layout algorithm. The three most widely used and studied types of layouts are the original algorithm, called "Slice and Dice," and two variants, the "Clustered/ Squarified" and the "Strip." The clustered layout and the squarified layout were developed at the same time. To avoid confusion, the term "clustered" will be used to refer to these layouts. The slice-and-dice layout uses parallel lines to divide a rectangle representing a larger object into smaller rectangles representing sub-objects. At each level, the orientation of the lines are switched from vertical to horizontal. The clustered layout uses recursion to produce rectangles that have more of a uniform square shape than the slice-and-dice layout. The strip layout produces rectangles that are in order like the slice-and-dice layout. Horizontal rows of rectangles are created in the strip layout(Bederson et. al 834, 835).

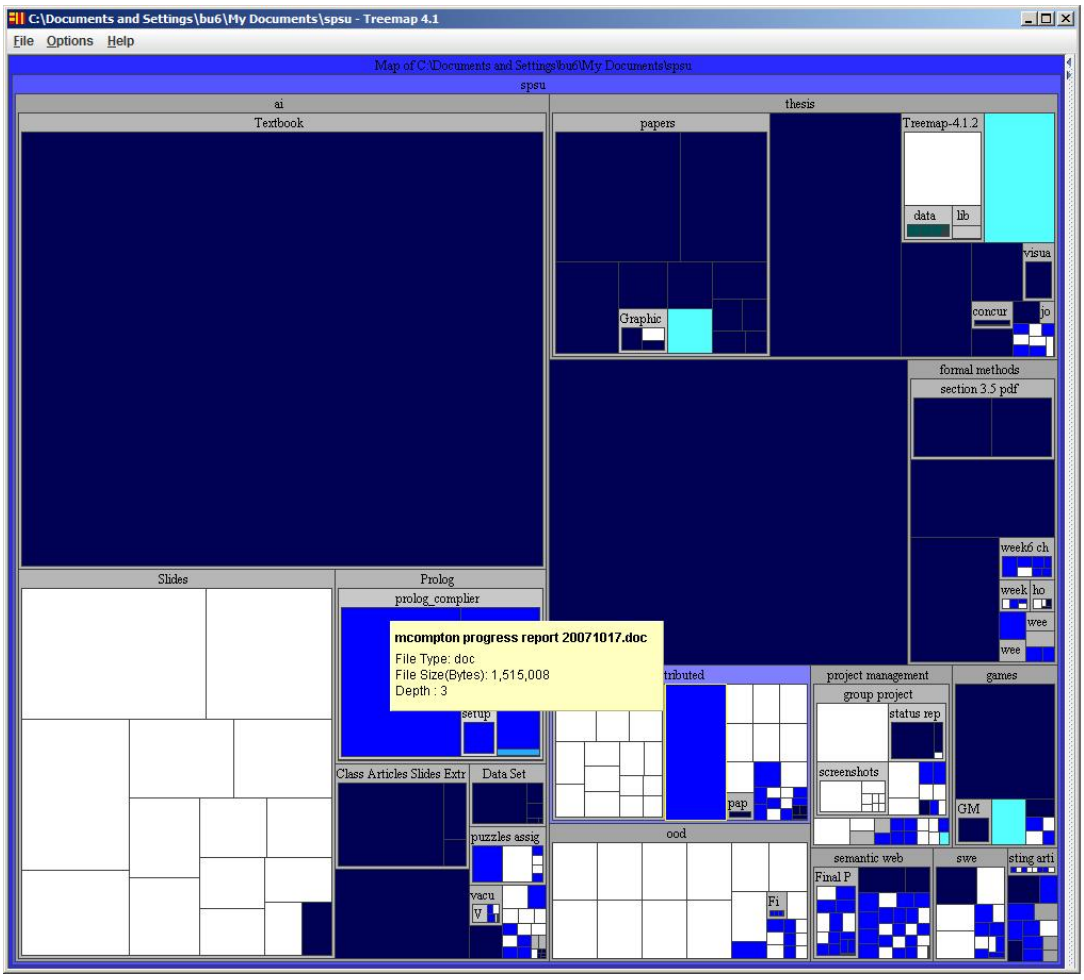

Fig. 2 Clustered Layout

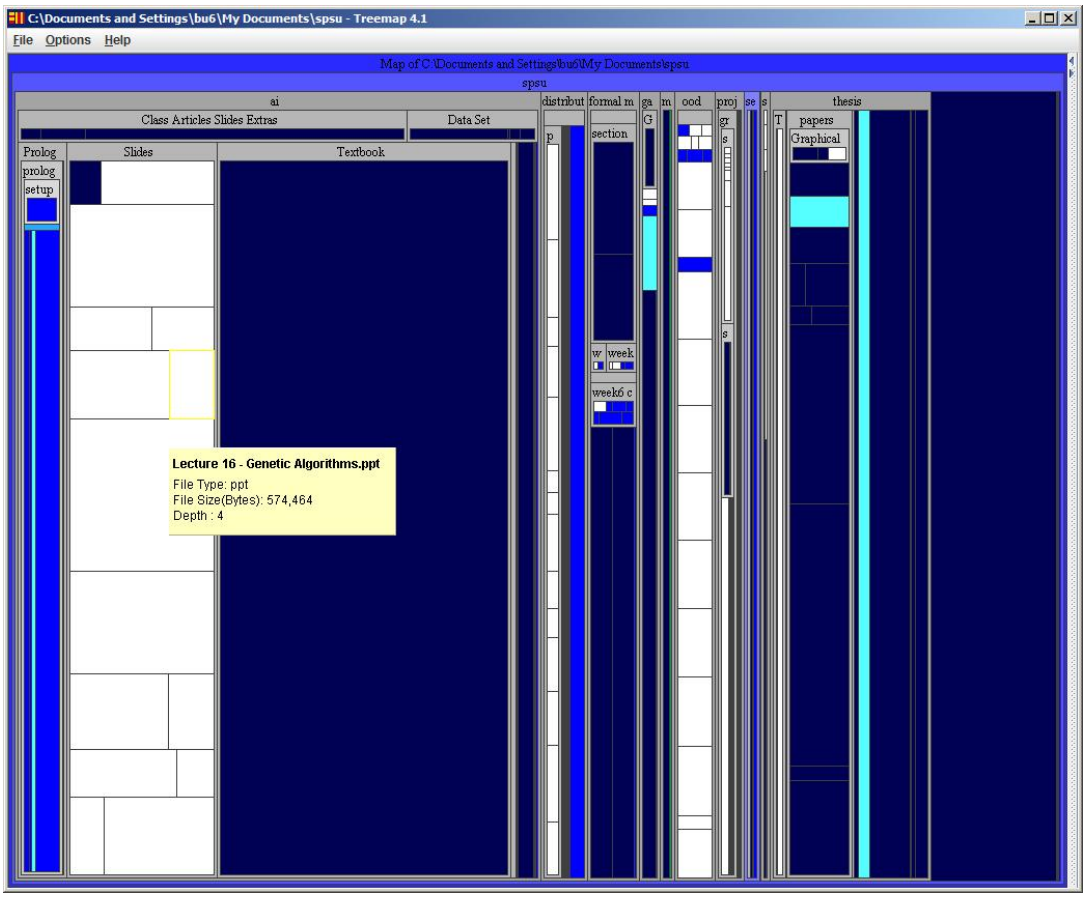

Fig. 3 Strip Layout

Each layout has advantages and disadvantages. The slice-and-dice layout while preserving the order of the data being represented and so produces treemaps in a consistent format, but it creates long, skinny rectangles that are difficult to see, select, label and compare. While clustered layout treemaps are easier to view, if data is added to the dataset, order is not preserved and the entire layout changes. Maintaining the order within a dataset can be a key aspect when a viewer is looking for patterns in the data. Both the slice-and-dice layout and the clustered layout require the eye of the viewer to switch between horizontal and vertical in order to process the treemap. The strip layout was developed to provide a more uniform order among the rectangles (Bederson et. al 834, 835).

The newest type of treemap layout is based on the voronoi tesselation. In previous treemap layouts two dimensional datasets have been presented with one dimension presented as the area of the rectangles and color used to show the 2nd dimension (Bederson et. al 835). Since voronoi treemaps are polygon instead of rectangle based, the multiple sides form a curve which can help an object distinguish itself (Balzer et. al 167).

5. Using treemaps for Software visualization

In their paper, "Visualization-based Analysis of Quality for Large-scale Software Systems, Langelier et. al use a 3-d treemap to perform analysis of the classes that comprise the source code for the popular Eclipse IDE with CK metrics(217). Their purpose was to analyze the Eclipse source code at the macro level. The used the slice-and-dice treemap layout with the size of each node corresponding to the total number of classes with superclasses at the root level divided into sub-classes. Classes are represented by 3-d rectangles. By using three dimensions instead of two dimensions, the property of twist was added to the typical treemap properties of color and size (Langelier et. al 215, 216).

The treemap layout was applied to some of the CK metrics including CBO (coupling), LCOM5 (lack of cohesion), DIT (depth of inheritance tree) and WMC (weighted methods per class). The metrics were mapped as follows: CBO to color, LCOM5 to twist, and WMC to size (See Figure 1, page 216). In representing WMC, classes with higher WMC had taller rectangles and classes with lower WMC had shorter rectangles. Classes with a low CBO, indicating loose coupling, were blue and classes with a high CBO indicating high coupling were red (Langelier et. al 215, 216).

Some ancillary features were created to assist in viewing the treemap. Because viewing three dimensions on a two-dimensional computer screen can create masking of some elements, a camera system was devised to allow the viewer to navigate through the visualization. Filters were created to help in the analysis of some of the metrics. These filters allowed the viewer to emphasize some elements and de-emphasize others (Langelier et. al 219).

The treemap visualizing CK metrics was used to detect design violations. Occurrences of the design violation of low coupling and high cohesion are visible where there is a large aggregation of boxes that are red and twisted. Additionally, this type of visualization makes it possible to easily detect the "blob" anti-pattern which occurs when many lines of code are contained in a few classes with very complex methods. A blob manifests as a tall twisted box linked to smaller boxes. A filter was used to show classes with high complexity. When one of these classes with high complexity is selected, another filter was applied showing related classes. Related classes that visualized as small rectangles indicated possible blobs (Langelier et. al 220).

6. Conclusion and Future Directions

In this paper I have examined how the quality of source code is visualized using CK metrics and applying Edward Tufte's principles of data visualization. I summarized Tufte's principles for data visualizing large data sets of multi-variate data. CK metrics and cyclomatic complexity were summarized as metrics that could easily be applied

to software for visualization in a treemap. In my analysis of treemaps I showed several types of treemap layouts including slice-and-dice, clustered and strip layouts. My summary of Langelier et. al's work showed how the CK metrics and slice-and-dice layout can be used for visualizing large sets of source code.

Future opportunities for visualizing source code with treemaps would be to experiment with implementing the clustered and strip layouts. While the quality of software has been explored using source code, nothing has been written about visualizing software tests and software defects as treemaps. I am currently looking at how effective this would be for large scale systems.

7. References

Bederson, Benjamin B.; Ben Shneiderman; Martin Wattenberg. 2002. Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies. *ACM Transactions on Graphics*. ACM Press. Vol. 21. No 4. Pages: 833-854.

Chidamber, Shyam R. and Chris F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*. IEEE Press. Vol. 20. No. 6. Pages: 476 - 493.

Johnson, Brian and Ben Shneiderman. 1991. Treemaps: A Space Filling Approach to the Visualization of Hierarchical Information Structures. *In Proceedings., IEEE Conference on Visualization Volume*, IEEE Press. Issue , 22-25 Oct 1991 Pages: 284 - 291.

Langelier, Guillaume; Houari Sahraoui; Pierre Poulin. 2005. Visualization-based Analysis of Quality for Large-scale Software Systems. *In Proceedings of the 20th IEEE/ ACM international Conference on Automated software engineering*. ACM Press. Pages: 214-223.

Tufte, Edward R. The Visual Display of Quantitative Information, 2nd. ed. 2001. Chesire, Conneticut: Graphics Press.