Software visualization


Complete definition: Software Visualization is the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of software.

Why visualization?

It was not known how cholera was transmitted. Dr. John Snow had the hypothesis that it gets transmitted via water. To check this, he plotted on the map of the city:
- the deaths of a new epidemic (dots)
- the water pumps (Xs).

The result was that high number of deaths were detected near infected water pump on Broad Street.

This is a picture of ENIAC I (1946).
(http://en.wikipedia.org/wiki/ENIAC)

In 1946 we used to see the programs. In the picture we can see the complexity of the program in how intricate the cables are. And we see who is working on what. This is no longer the case with modern software systems.
A picture is worth a thousand words.

We are visual beings.


How many groups do you see?

Some see 3 groups and some see 4 groups. Those that see 3, see the circle in the center as belonging to the group formed by the two circles at the bottom.
Enclosing clarifies the situation.

Again, no problem in identifying 3 groups when circles are connected with edges.

The same happens when the circles share the same visual shape.
These are some examples of so-called Gestalt principles. According to these, we perceive the world as a whole rather than as a sum of parts.

Two more examples of Gestalt principles.

Our brain is a computer with 3 types of memory:

- Iconic
- Short-term
- Long-term


http://en.wikipedia.org/wiki/Gestalt_psychology
If eyes are computers, visualizations are programs.

These attributes of form are the primitive instructions we can use for building these programs.

Exemplifying Pre-attentive Processing
Colin Ware, Information Visualisation, Elsevier, Sansome Street, San Francisco, 2004. p150
70% of all external input comes through the eye.

To see is often used a synonym for to understand. Do you see my point?

A picture is worth a thousand words.

This picture shows approximately 350 words for a tiny system.
Let’s see what else can we do with 1000 words.

If we display all of them equally, we cannot identify much.

Increasing the font size leads to a tag cloud visualization. The small text is hardly readable, but it still competes for attention.
If we decrease the visual importance of the small text, we still know it's there but the important words stand out more. Still, alphabetical order is not necessarily the most relevant.

Ordering the words eases our task.

An example of what can be done with 1000 words.
UML is a nice visual language for expressing ideas, but it is hardly useful as a visualization.

In this example, we show a small fragment of the model hierarchy in Moose. Still even at this level of zoom, we cannot see the details.
If we zoom out, all we see is the shape of hierarchies, and the shape of classes.


System complexity is a polymetric view that does a better job at showing the shape of hierarchies and of individual classes.


Polymetric views are graphs enriched with metric information.
System complexity of ArgoUML.

Node = class
Edge = inheritance
Node width = number of attributes (NOA)
Node height = number of methods (NOM)
Node color = number of lines of code (LOC)

This picture shows classes (small squares) grouped in packages (large rectangles). The color of the classes is given by the predominant concept. The labels show the package name, but they are cropped when longer that the package width. The classes in the package are arranged to get a ration between width and height as close as possible to the golden ratio (1.61803399).

31 packages, 394 classes and 9 concepts of JEdit.

What to visualize?
Software structure
Software relationships

Class Blueprint shows class internals


Treemaps show the hierarchical structure by filling completely the given space.

The picture shows the files colored by type of ArgoUML 0.26 and it was generated with Disk Inventory (http://www.derlien.com/).
Blue are Documents, Red are Jars, Cyan are Pictures, Cyan are Java


The class is split into 5 layers:
- The initialize layer contains constructor methods,
- The Interface layer contains methods called from outside the class,
- The Internal implementation layer contains methods called from within the class,
- The Accessor layer contains setters and getters,
- The Attribute layer contains the attributes :).

Blue edges represent method invocations. Cyan edges represent attribute access.
The blueprint has a rich vocabulary.

On both dimensions of methods and attributes are mapped metrics. On the color of the methods are mapped different attributes of the method.

The picture on the left shows the blueprint of one superclass and 3 of its subclasses. The subclasses have the same shape and color (given by overriding methods). Hence the name Siamese twins.

The picture on the right shows a class that has two distinct interests, because the methods on top do not have a direct relationship with the methods on the bottom (we see that because there is no blue edge in between).


One straightforward way of representing relationships is to display entities in a circle and draw edges between them. The picture shows classes organized in a module structure and the arrows are dependencies (red=called, green=caller). We know that Units 16 and 18 are called many times, but we do not know exactly where from. The picture is too noisy.
Hierarchical edge bundles clarify dependencies


Hierarchical edge bundles make use of the hierarchical structure of entities to make relationships between larger parts clearer.


This visualization uses hierarchical edge bundles to show the relationships between two hierarchies of data.

A correlation matrix displays the same entities both on the rows and on the columns. Each cell in the matrix is colored based on the strength of the correlation. This notation is useful for identifying similarities (e.g., code duplication).

In this picture, the matrix displays the similarity of vocabulary used in the classes of JEdit. Furthermore, the classes are grouped to reveal clusters of classes that use similar vocabulary.

It took thousands of years to build the first abstract representation of the real world using x and y axis. Software on the other hand, has no physical shape and one challenge is to lay it out so that the distance between entities has a meaning.

This visualization proposes a cartography metaphor to represent software. The entities are distributed based on the vocabulary used.


CodeCity represents the system structure as a city. The packages generate quarters, while the classes are buildings.

http://www.inf.unisi.ch/phd/wettel/codecity.html


In this work, design flaw suspects are highlighted with different colors.

http://www.inf.unisi.ch/phd/wettel/codecity.html

What to visualize?
Software structure
Software relationships
Metaphors
Interaction

Visualization does not guarantee understanding

Not any picture tells a thousand words.
Minimize non-data ink

The 6 actions are:
1. Remove background.
2. Remove legend.
3. Add better graph description.
4. Make the series line black for better contrast.
5. Make the grid lines light gray to be less intrusive.
6. Make the dates to start from the origin to avoid confusions.

The only element debatable to be chart junk is the black border which could be perhaps made gray.


Interesting about this visualization is that each part of the bug bears information, and the result is a pleasant glyph:
- the wings show two time series of lines of code (left) and errors detected (right)
- the antennae show different types of code. For example, the orange line shows the amount of C code.
- The eye shows the amount of inheritance relationships.
- With red and green are shown lines added to correct errors (red) or for new functionality (green).
A small experiment

1, 13, 27, 4, 96
What were the numbers?

Easy!

What's the last advertisement you saw?
Not so easy!

Each visualization provides a language that needs to be learnt.

Visualization is art, too.

Just at the beginning of 20th century artists sought means of expression that would match the industrial age, now, as we step into the information age we seek new artistic means of expression.
Visualization is art, too.

The picture on the top right: http://acg.media.mit.edu/people/fry/revisionist/
The other two pictures were created with Mondrian.

Two nice collections of visualizations are:
http://infosthetics.com/
http://www.visualcomplexity.com/vc/

This picture was created by Michele Lanza

The zeppelin!

The picture shows C grammar dependencies and it was created by Magiel Bruntink, Jurgen Vinju and CWI

The picture shows a novel way of drawing treemaps.

http://lip.sourceforge.net/ctreemap.html