Friday 2:00

Towards Visualizing Size Measures of Large Systems

A. S. M. Sajeev, Wendy Wang, Aaron Quigley and Peter Eades

Department of Computer Science and Software Engineering

University of Newcastle, Australia

Towards Visualizing Size Measures of Large Systems

A. S. M. Sajeev*, Wendy Wang*, Aaron Quigley* and Peter Eades*

ABSTRACT: Software Size is one of the fundamental measures, which is often used to estimate cost and effort of software projects. We argue that visualization is a useful mechanism to understand and analyse software sizes of large systems.

This paper discusses the issues involved in visualization, and demonstrates the application of those issues in visualizing software size. We use the Vector Size Measure (VSM) for size measurement, and Virtual Reality Modeling Language (VRML) as the implementation platform for visualization. The demonstration shows that several useful factors of software size can be captured easily in a visual medium.

Introduction

Software Size is one of the fundamental measures, which is often used in cost and effort estimation of software projects. Size measures such as source lines of code and Function Points (Albrecht, 1979) are used in the industry, even though, often they are criticized for the lack of theoretical underpinning and/or inability to measure all aspects of size; for instance, Albrect's function points is shown to be weak in measuring problem complexity (Jeffery, Low, and Barnes 1993).

Often a single size value of a system is not meaningful enough for software developers. A system consisting of hundreds of thousands of components may be built in various phases. Resources that need to be put into different subsystems could vary. Thus it becomes necessary to be able to study the size of different components, or even different "collections" of components. A large set of numbers is not very useful in such cases. Visualization seems to be the answer just as in many other scientific problems that produce huge data sets (Eades and Zhang, 1996).

We have designed a system that uses a simple "cityscape" metaphor to visualize the software size; the cityscape consists of city-blocks representing packages where each block has buildings representing software components. Since the size measure needs to be available early in the life cycle of the system (for example, in project planning), we measure the size from the specification of the system.

The paper is organized as follows. In Section 2 we discuss software size measures. Section 3 describes visualization issues and Section 4 an implementation of size visualization. Section 5 gives the conclusions.

* *Department of Computer Science and Software Engineering, The University of Newcastle, Australia*

Size Measurement

Around 70% of large software projects fail to meet (a) requirements, (b) deadlines, and/or (c) budget (Jones, 1995). Among the important reasons for this failure is the inability to satisfactorily estimate the *effort* required for the project to meet its requirements within the deadline. To a great extent, the effort depends on the characteristics of the software developed; it is common to estimate effort (or project cost) from a measure of *Software Size* (Ebrahimi, 1999). For example, *COCOMO* (COnstructive COst MOdel) (Boehm, 1981) estimates Effort E using the function:

$$E(s) = as^b f,$$

where s is the software size measured in Kilo Delivered Source Instructions, f is an adjustment factor, and the coefficients a and b are determined from the type of software constructed. Fenton and Pfleeger (1997) suggest that software size, S, is a function, f_S, of *length*, *functionality*, and *problem complexity*:

$$S = f_S(l, f, c)$$

where: l is the total number of entities in a system, f the number of functions a system provides, and c the underlying problem complexity.

Source lines of code (SLOC) is often used as a measure of software size. However, it is not available early in the software cycle. An early life cycle size measure is Function Points. The function point technique is biased towards transaction based systems since the measurement is primarily based on inputs and outputs. In other systems such as compilers, embedded systems, and CAD systems, measuring size based on inputs and outputs are not sufficient; much of the complexity is hidden in the algorithms, data structures and architectures. Jeffery *et al.* (1993), Fenton and Pfleeger (1997) and others have pointed out practical and theoretical shortcomings of Function Points. Since Function Points can be measured from a requirements document, the value is available early in the Software Life Cycle, unlike Lines-of-Code. Extensions and modifications have been proposed for Function Points. Among these, *Mark II Function Points* (Symons, 1991) and *Feature Points* (Jones, 1991) are more widely tested.

In this research we use a size measure called Vector Size Measure (VSM) which measures functionality and problem-complexity in a balanced manner (Hastings and Sajeev, 2000). VSM is an early life-cycle measure that uses requirements specified in an algebraic specification language. An algebraic specification defines the system as a collection of abstract data types (ADTs). The specification separates function signatures from the semantics of the ADT. VSM measures the size of an ADT as a vector of functionality and problem-complexity, where *functionality* is measured as the number of operators and operands in the syntactic section (i.e., specification of function signatures) and the *complexity* as the number of operators and operands in the semantic section of the ADT. The size of a software specification is the vector sum of its constituent ADT specifications. VSM and its associated effort prediction model were tested in a pilot study of 8 industrial projects. The study showed high correlation between vector size measure and effort estimation. VSM is shown to be theoretically sound, as well (Hastings and Sajeev 2000).

Visualization Issues

Often a single size value of a system is not meaningful enough for software developers. A system consisting of hundreds of thousands of components may be built in various phases. Resources that need to be put into different subsystems could vary. Thus it becomes necessary to be able to study the size of different components, or even different "collections" of components. An analysis or measurement tool often provides a large textual output that is available to software engineers, project managers and other users. A user can understand a clear textual description of a small amount of information. However, when dealing with large amounts of information, a vast amount of text quickly becomes cumbersome. Visualization seems to be the answer just as in many other scientific problems that produce huge data sets (Eades and Zhang, 1996).

We identified the following main functional and non-functional issues as important in visualization of software size measures. It should:

- be easy to understand, manipulate, explore and navigate on various levels of detail. (see Quigley and Eades, 2000)
- be based on a visual *metaphor* that gives a visual representation to abstract numbers.
- avoid extraneous information; bureaucratic debris in the form of *'chart-junk'* distracts the engineer, and can often be misleading (see Tufte, 1983).
- capture various factors that constitute software size
- facilitate moving from one view (example, visual size measure) to other views (example, numeric size measure, or even, specification or design documents)
- be scalable; i.e., handle the analysis of large code bases (100,000 to 2 million lines of code) typical of industry sized software developments.

Information Visualization takes advantage of the processing speed and graphical capabilities of computers to visually represent large amounts of information on screen, which a user can then interpret (see Ware 2000). The essential idea in information visualization is that the user's perceptual abilities are employed to understand information. Information Visualization techniques are used for the exploration and analysis of large relational data sets. Using visualization techniques, humans can perceive more patterns linking local features than from text alone. There are key advantages in using visualization and a visual *metaphor* that are pertinent for the visualization of an abstract measure such as VSM for large software system:

- **Scale:** Important information about tens of thousands of code measurements is available for immediate comparison and comprehension.
- **Patterns:** Visualization elucidates patterns that were not anticipated in the analysis. This aids in identifying and remedying problems earlier in the development process.

- **Artifacts:** Patterns, as artifacts from the analysis process, enable errors in the algebraic specifications to become immediately apparent and are invaluable in quality control of the entire VSM process.
- **Multi-level:** *Context-in-detail* techniques are used to understand the small-scale features of a dataset while at the same time maintaining an overall understanding of the dataset. A visual *metaphor* blends this understanding on multiple levels of detail in a contextually intuitive manner.
- **Hypothesis:** Visualization facilitates hypothesis formation about the VSM and their relationship to the software system.

Exploratory data analysis (see Quigley and Eades 1999) is a method used by researchers and statisticians from various fields of science such as software analysis, software re-engineering, geography, medicine and finance. The goal is often to investigate and explore the structure of *large data sets.* Visualization of such data analyses aids the exploration and often elucidates patterns that would otherwise be unapparent in the results. As has been noted software size measures are often used to estimate cost and effort of software projects. The results of the analysis of a software system, consisting of a large number of components are often multidimensional. The visualization of the structure of this multidimensional data set produced from a metrics analysis tool is the goal of this work. Figure 1 shows the various elements involved in the visualization.

Visual Representation

A good visual representation of the multi-dimensional data can effectively convey information and hence understanding to the user but a poor representation can confuse or worse, mislead (see Tufte, 1997). The challenge in multidimensional visualization is to develop methods that produce pictures of high quality. The increasing power and storage capacity of computers means that the results or much larger and more complex analysis can be processed and rendered on screen. The fact that larger analyses can be drawn is not enough for visualization. It is easy to draw a great deal of graphical information but the limits on both the available screen space and more importantly human cognition are quickly reached (see Quigley and Eades 2000). If the picture cannot easily be viewed on screen, for example due to poor resolution, or the sheer volume of graphical information, then the visualization can no longer be considered useful to the user. Numerous viewing schemes have been proposed to overcome this problem (Feng 1997 and Huang 1999). Our solution is to use a scaleable *metaphor that* gives a visual representation to abstract numbers on varying levels of detail as required.

A scaleable *metaphor* should allow the visualization to be viewed on multiple levels of detail or *abstraction* as necessary. Our method for the creation of the abstractions recursively groups the results of the analysis of sub-systems and components. This method is particularly appealing for software analysis as one particular level in the grouping (an abridgement) can be drawn on screen. The choice of level allows the results to be concisely (i.e. more abstractly) represented with fewer graphics on screen. The abridgement of the analysis results in a loss of precision. This has to be measured against the fact that a greater understanding of the overall system is now possible. A visualization scheme based on a multi-level viewing still has the ability to mix levels of abstraction, allowing greater detail in certain areas of interest, while maintaining an abstract view of the rest of the results.

Scaleable metaphors

A *metaphor* is an object that is considered to possess similar characteristics to another object one is trying to describe. In the case of multidimensional data we are trying to describe the interrelationships among a large set of related results, which are often expressed as ordinal values. A *scaleable visual metaphor* provides a set of symbolic elements in the form of a visual lexicon. This lexicon must form a cohesive group of elements for representing the multidimensional results. The metaphor must be scaleable as it has to represent the results of an analysis on various levels of abstraction, as required.

The visual elements (symbols) of the metaphor can be classified in terms of *sensory* and *arbitrary* symbols. Sensory symbols derive their expressive power from their ability to use the perceptual processing power of the brain without learning (see Ware 2000). Arbitrary symbols have to be learned, i.e. they have no perceptual basis. For example, a picture of a bird is a sensory symbol whereas the word bird in English or *tori* in Japanese must be first learned before it can be understood and hence used as a symbol. The choice of metaphor should be simple so as to avoid introducing new arbitrary representations as these are often hard to learn, easy to forget and embedded in culture and applications.

The goal of visualization in this case is to produce a perceptually efficient visual format to aid in effort prediction in a software development. Each ADT can be considered as an entity with various attributes that need to be visualized. These attributes may typically be classified as *nominal* or *ordinal.* A nominal attribute is simply a labeling function, such as a sub-system being labeled mission-critical, high-priority, typical or low-priority. In visualization these attributes are often visualized using visual attributes that have no inherent ordering, such as color, texture or micro-texture. An ordinal attribute does have place in a sequence of things such as a numeric value. An example of an ordinal attribute is the number of operators and operands in the semantic section of the ADT. In visualization these attributes are often expressed in terms of height, width, depth or luminescence of visual elements.

Along with entities and their attributes there are *inter*-relationships that define structures and patterns that relate entities to each other. There are also *intra*-relationships, which must be considered within the ADT. Any example of an important inter-relationship for measuring software, is the **"part-of"** relationship. It can be used to classify ADTs belonging to the same module, sub-system or component for example. In visualization, relationships are often expressed in terms of lines connecting related entities together or entities encompassing other entities in the case of an inclusion relationship. The choice how to represent inter and intra relationships must be derived from the choice of visual metaphor so as to maintain a consistent set of visual elements.

Multidimensional data

Visualization is used so that a user can extract meaningful and useful information from a vast amount of otherwise unrelated data. Hypothesis formation, as was stated previously, is often the motivating factor in the use of a visualization tool. When the amount of measures that must be displayed is low such as two or three then classical plotting techniques such as those shown in Figure 2 and Figure 3 are often used. When the amount of measures becomes larger then often the naïve approach is to encode each one of these new measures with a value in terms of red, green or blue. This approach treats each color as a *data dimension.* This allows for three extra measures and if we extend this approach to texture (smooth to rough) and micro-texture we can display 8 separate measures in a simple three-dimensional data plot with spheres (for example) representing entities. Problems with this from a HCI perspective immediately become apparent; people perceive color and texture differently and the

interplay between color scales makes it difficult to identify 'how-much' is the contribution of a particular color. However with eight such measures identifying patterns, such as a collection α of rough red spheres with a fast micro-texture swirl may be possible. But interpreting what a large collection such as α spatial aligned with a collection of smooth blue spheres with a slow micro-texture swirl (β) is difficult., Identifying distinct groups such as α,β can often be easily achieved whereas building higher order inferences i.e. an understanding of how α,β relate, often requires human intervention.

Any visual metaphor for visualization of the VSM measures must consider such multi-dimensional representation in its design. These considerations should be thought of as an initial validation of the metaphor enabling us to rule out clearly unsuitable metaphors early on in the process.

Implementation

Several scaleable visual metaphors were considered for visualizing the measures generated.

These included:

- A city landscape – with building, streets, blocks and neighbor hoods.
- A mountainous terrain – with mountains, valleys, ridges and plateaus.
- A solar system – with planets, suns, satellites and asteroid fields.
- A roller coaster ride – with flat tracks, big hills and different colored cars.

A city landscape was chosen as it maximized the sensory symbol used while minimizing the number of arbitrary new symbols to be learnt. Further, the choice of a simple yet expressive metaphor should allow for modeling of the metrics in an easy to follow and pleasing manner. The city scene is intuitively easy to understand because it is composed of square objects laid out in a regular symmetric manner. These objects are well defined and produce scenery that is clear, uncluttered and suitable for zooming to obtain the relevant details. The scene generated should be familiar to users of other tools that generate square shapes; this scene simply adds another dimension of information, namely height that relates to functionality. Another reason for choosing a cityscape was that the items in the model would be culturally relevant to the target user group.

In the *cityscape* buildings represent components, city blocks represent subsystems and a city represents a software system (See Figure 4). The height of a building is the functionality measure of the corresponding component, while the color gives a measure of its problem-complexity. One can enter a building by clicking on it to see the inside of a building where specification details are available; this can be easily extended to have rooms in the building providing various artifacts related to that component (e.g. UML diagrams) that are useful for the project managers and software engineers.

System Design

The system has two main components; an Algebraic Specification Language (ASL) tool that generates VSM values from the specification [developed by Hastings (2000)] and another tool that translates the data generated into a three-dimensional visualization (See Figure 5).

Cityscape metaphor

A cityscape visualization is composed of buildings, roads, grass and blocks. Each of these items directly represents the software system it is modeling. Each building represents a component, the height of the building corresponds to the functionality of the component. The color of the building corresponds to its complexity; a color scale similar to that used in maps shows the grading of colors. The roads separate blocks into grass sections with each grass section representing a package.

Users are able to navigate around the city and examine each *component* or *package* in detail.

Generating cityscape

To generate visualization a specification of the system in terms of a series of *adts* is required.

Once a set of *adts* has been obtained the visualization is generated by executing the generator program (Figure 5). It processes each *adt* file sequentially, first obtaining the set of VSM values from the specification, and then using this set to generate HTML and VRML views (Figure 6).

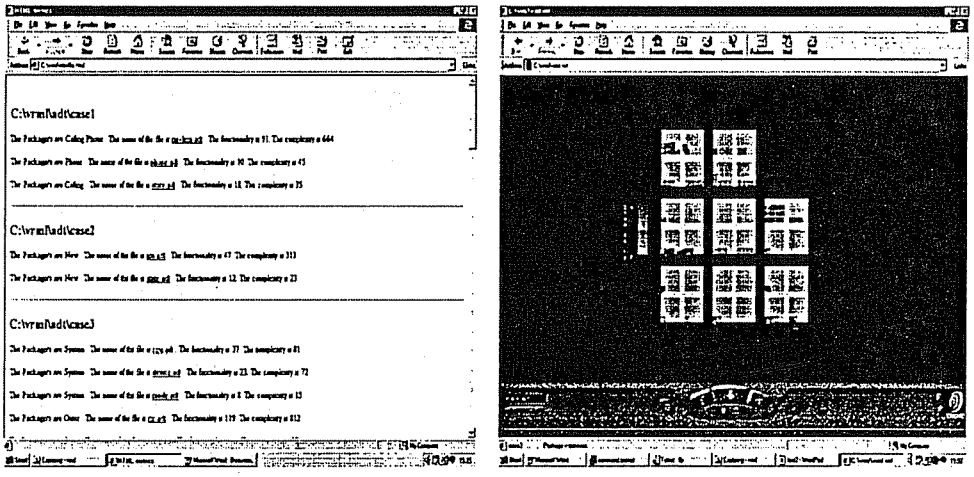

Figure 6 Generated HTML and VRML file.

Choice of VRML

As part of the requirements analysis for this project the following design criteria were identified.

- Ability to model three-dimensional images
- Portability
- Easy installation

- Ease of use
- Lack of need to produce complicated shapes or highly interactive graphics.

VRML was chosen above other graphical languages such as OpenGL, DirectX and Java3D. VRML was identified as having significant advantages over more heavy weight graphical environments.

- VRML can concisely describe complex visual environments in small to medium sized files.
- VRML can be visualized in platform independent browsers via a plug-in.
- VRML is sufficiently lightweight for expressing a wide variety of scaleable visual metaphors.

From interoperability point of view a user only needs to download a plug-in for their existing browser to be able to visualize, explore and monitor the VSM measures. Finally the issue of portability is satisfied since VRML extends the distributed hyperlink nature of HTML. A C++ program was developed to generate the VRML code. A Cosmo (D) player was used to test the generated visualization.

System Operation

The operation of the system can be broken down into two areas; generating the visualization and navigating the visualization.

An example of a visualization produced by this system is shown in Figure 6; this is the view when first loaded. The entire city is displayed on the single screen; users can further explore areas of interest.

In Figure 4 the city has been rotated to offers a better view of the building height (which represents the functionality of the system).

Furthermore, one can focus on different parts of the city using viewpoints. Figure 7 focuses on *Case 6*.

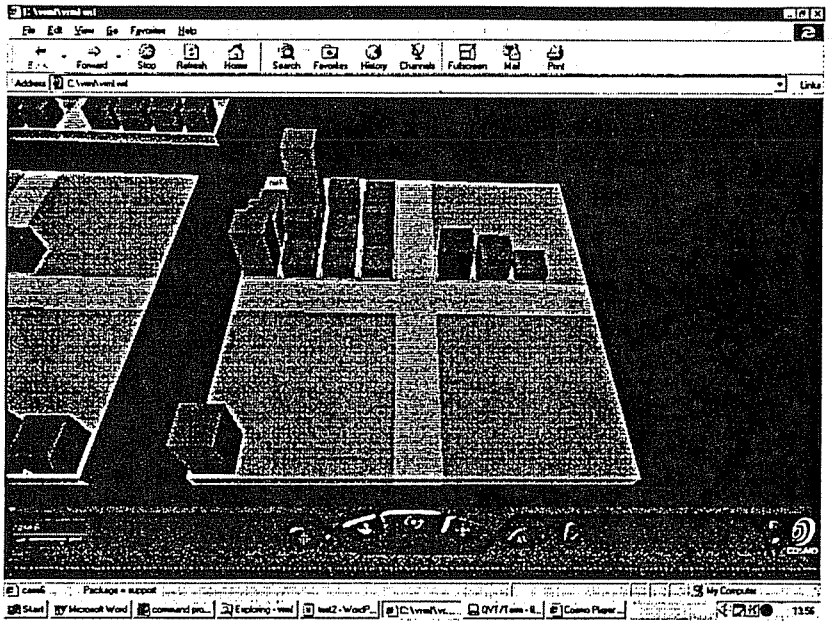

Figure7 Using the viewpoint

Also when the mouse moves over a building the name of the component it refers to with the complexity and functionality values appear in the bottom left hand corner. When the mouse moves over the grass area the name of the package it is related to appear at the bottom. Click a building allows entering the building to view its specification. Similarly, when the road area is clicked, it displays the diagram that shows the relationship between the subsystems.

Types of Users

The system benefits two types of users.

Managers

The system is intended to aid managers when allocating resources. He/she explorers the cityscape to find areas that require different kinds of resources. For example, a block with a number of red (i.e., high complexity) buildings would indicate the place where highly skilled engineers are required, as opposed to a block of short blue (i.e., low functionality and complexity) buildings.

Engineers

The engineers, who design and develop the system, can enter the buildings and view the specifications, UML diagrams etc. (The current implementation only shows the specifications.)

Discussion

One advantage of using the cityscape metaphor is that it has discrete elements, namely, buildings that could be easily mapped on to software components, and buildings can be grouped to represent a subsystem. However, the disadvantage is that it is not easy to show the relation between different subsystems. Consequently, there is not a single representation of the overall size of the system. Displaying related systems in close proximity is often used in visualization. Another possibility is to

represent different relations between components using different symbols such as overhead bridges, pipes underneath etc.

The VRML implementation facilitates scalability by providing an inbuilt mechanism to zoom to any location. The visualization captures the size of the different components; it is easy to see which part of the city are dense, in terms of functionality (height of buildings), complexity (color) or number of components (number of buildings); this can help managers allocate resources, for instance, according to the need. It is easy to provide further information needed for engineers and managers "inside" each building.

Our future work includes exploring better metaphors that will allow us to increase the *information density* of visualization, provide views of varying granularity, and animation for tracking changes to size from factors such as "requirements creep". One method to allow coarse views of the system is to have automatic view selections depending on how far the user is from the object. The current prototype has a further limitation that every block is of the same size. Even though, it provides a pleasant appearance, in practice, some blocks could be extremely crowded with buildings and the others largely empty. One possibility is to generate the separation (i.e. roads) dynamically.

The Vector Size Measure we used is based on algebraic specifications. The visualization techniques, however, can be easily used with other size measures. In those cases, we need to identify the parameters of size measurement and map them to the properties of the cityscape.

Acknowledgement

The algebraic specification of case studies used in visualization, and the measuring tool were supplied by Tim Hastings.

References

Albrecht, A. J. (1979): Measuring application development productivity, in *Proceedings of IBM Applications Development Symposium*, 83-92.

Ames, A. L., Nadeau, D. R. and Moreland, J. L., *VRML 2.0 Sourcebook*, John Wiley, 1996.

Eades, P. and Zhang, K. (editors) (1996), *Software Visualization*, World Scientific.

Fenton, N. E. and Pfleeger, S. L. (1997): *Software Metrics: A Rigorous and Practical Approach, 2nd Edition*. London: PWS Publishing Company.

Feng, Q.W. (1997): *Algorithms for drawing clustered graphs*, PhD thesis, The University of Newcastle, Australia 1997.

Hastings, T. E. and Sajeev, A. S. M. (2000), A Vector Based Approach to Software Size Measurement and Effort Estimation, *IEEE Transactions on Software Engineering* (Accepted).

Hastings, T. E. (2000): *Measuring Software Size and Predicting Development Effort of Contemporary Software Systems*, PhD Thesis (Submitted), Monash University.

Huang, M (1999): *On-line animated visualization of huge graphs*, PhD thesis, The University of Newcastle, Australia 1999.

Jeffery, D. R., Low, G. C. and Barnes, M. (1993): A Comparison of Function Point Counting Techniques, *IEEE Transactions on Software Engineering*, 19(5), 529-532.

Jones, C. (1991): *Applied Software Measurement*. New York, NY: McGraw-Hill.

Quigley, A. (1997): *Visualizing a reverse engineered system structure with dynamic 3-D clustered graph drawings*, in proceedings SoftVis 1997, 25-29.

Quigley, A. Eades, P. (1999): *ProVEDA: A scheme for Progressive Visualization and Exploratory Data Analysis of clusters*, in proceedings of SoftVis, 67-74.

Quigley A. Eades, P. (2000): *FADE: Large scale layout and Visual abstraction*, Eighth international symposium on Graph Drawing (to appear).

Symons, C. R. (1991). *Software Sizing and Estimating: Mk II FPA*. Chichester, England: John Wiley.

Tufte, E. (1983): *The Visual Display of Quanitive Information*, Graphics Press.

Tufte, E. (1990): *Envisioning Information*, Graphics Press.

Tufte, E. (1997): *Visual Explanations*, Graphics Press.

Ware, C. (2000): *Information Visualization: Perception for design*, Morgan Kaufman