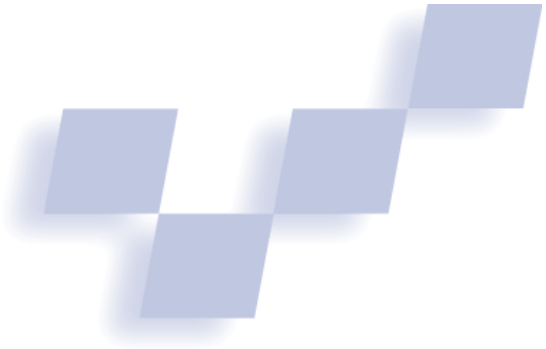
Information Rich Glyphs for Software Management Data



Mei C. Chuah
Carnegie Mellon University

Stephen G. Eick
Bell Laboratories

Three high-dimensional glyphs for viewing software project management data assist in perhaps the most challenging engineering task in modern times—managing large software projects.

In data exploration, glyphs refer to graphical objects or symbols that represent data through visual parameters that are either spatial (positions *x* or *y*), retinal (color and size), or temporal. Common examples of graphical objects include the bars in a bar chart or the points within a scatter plot.

We focus on glyphs for visualizing software project management data. Any large-scale project will have many different classes of resources (lab equipment, staff time, machine cycles, disk resources, interim deliverables, and customer commitments) that must be scheduled and tracked. Inevitably problems will arise and solutions must be found. To support the management process, information systems collect and maintain large status databases. We aim to support and improve the understanding of this information through visualization. Our glyphs are designed to expose patterns among sets of software artifacts and to help identify differences between items.

There are four interesting issues in project management data:

1. *Time*: Project management is time-oriented. Each project has a time in which it must be completed—a deadline. To properly meet deadlines we must track milestones, monitor resource usage patterns, and anticipate delays.
2. *Large data volumes*: Large projects have a lot of data associated with them. For example, a multimillion-line software project may be partitioned into tens to hundreds of subsystems, hundreds to thousands of modules, and thousands to hundreds of thousands of files. Much of this data is unstructured, making mining information from it difficult. Our approach—typical in large projects—partitions the data hierarchically. For example, a software project will have a high-level manager with overall responsibility. These managers may have several supervisors under them, and each supervisor will lead a group of engineers.
3. *Diversity/variety*: Projects commonly have a diverse group of resources as well as resource attributes. Expressing different types of resources (engineers and computers) as well as their attributes (number of code lines and number of errors) requires visual representations flexible enough to convey information about a set of diverse data meaningfully. We designed our glyphs to be versatile so that they can show data for many different software artifacts. Users do not need to learn new visualization structures for each object type. Flexibility results from enabling our representations to show many different data types, including both discrete and continuous domains.
4. *Correspondence to "real world" concepts*: In a project database, data elements usually correspond to "real world" entities or concepts. For example, a *userID-1* data element in a software database represents an actual person, and the element *file-125* corresponds to a source code file. By using glyphs, we maintain the "objectness" of the data elements because all the properties of a data element are grouped together visually.

Glyphs are not a new concept. Chernoff first developed them for multidimensional data.[1] Our work, however, differs from previous efforts because it combines established visualization views (time series, histograms, and rose diagram) to form glyphs. This lets users more easily interpret the glyph by using prior graphic knowledge.

Software project management data

The management systems for large software projects collect a huge amount of project information, frequently organized hierarchically as in Figure 1. The top levels of the hierarchy have fewer aggregate objects while the

0272-1716/98/$10.00 © 1998 IEEE

lower levels have greater numbers of individual objects.

In our system a scatter-plot interface (not shown) allows users to access and filter the objects at different levels of the hierarchy. Users may select and view aggregate objects at any level in any of our information-rich glyph representations. These tools and visual representations let users filter software objects at the upper levels so they can focus on a small number of objects, analyzing them at the lower levels for particular tasks.

Timewheel glyph

Our timewheel glyph is used to visualize time-oriented information. A conventional way to visualize time-oriented information involves a time-series plot—a line chart with time on the *x*-axis and a variable on the *y*-axis. The timewheel glyph displays multiple time series, with each series rotated around a circle, as in Figure 2. To simplify data attribute identification, each attribute is colored according to a rainbow-hue colormap. We used color because it is perceptually discrete.[2] The arrows in Figure 2 show the time increment for each series.

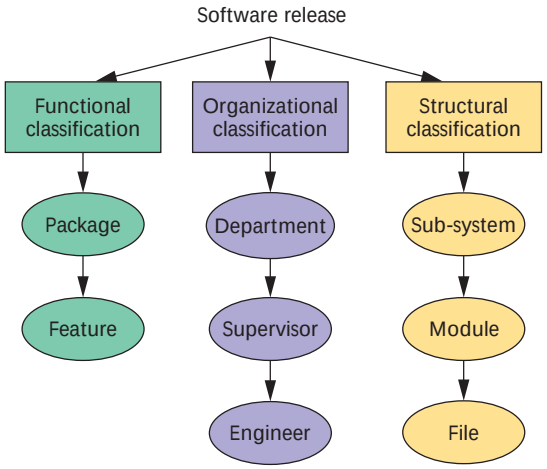
Figure 3 shows 16 software releases using the timewheel glyph interface. From Figure 3, we can easily pick out two major trends: the increasing trend and the decreasing or tapering trend. The increasing trend glyph looks like a prickly fruit (the objects outlined in white in Figure 3). The tapering trend glyph, on the other hand, looks like a coconut husk or a hairy fruit (the objects outlined in red in Figure 3).

In addition to showing the global trends of an object, the timewheel glyph also helps in examining differences in trends within an object. For example, in Figure 2 the overall trend of *userId-1* is tapering; however, the *anew* and *dnew* attributes (two variables measuring newly written code) possess two divergent increasing trends. Because the *aerr* and *derr* attributes (two variables measuring error density) are tapering, we can deduce that at the start of their career *userId-1* fixed errors, but later moved on to developing new code. Error fixing evidently accounted for an important portion of *userId-1*'s activities because it corresponds to the dominant trend.

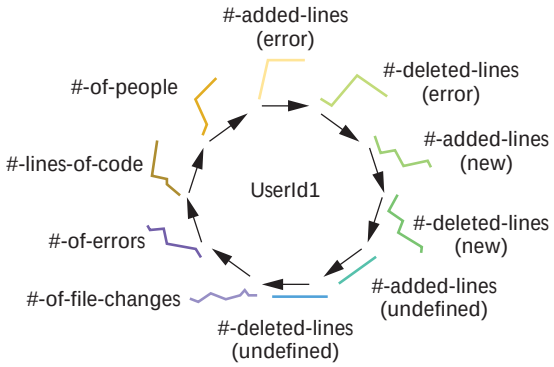
An obvious and traditional way to arrange a set of time series on a 2D plane would be to lay them out linearly. For tasks involving browsing or searching for gestalt patterns, the circular layout may be more effective than a linear layout for five reasons:

1. It reduces the number of eye movements per object.
2. It's less susceptible to local patterns.
3. It weakens the reading order implications.
4. It separates objects with less space, leading to gestalt perception
5. It's rotation invariant

Let's consider the ramifications of these characteristics in order.



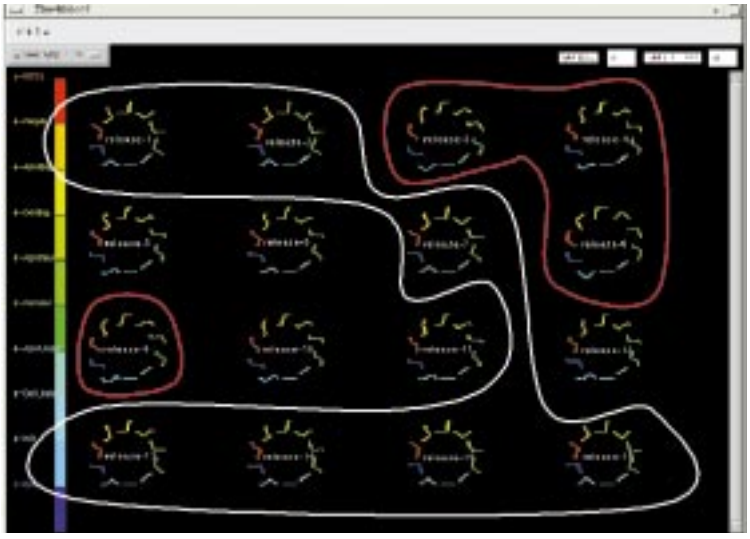
1 Information for a software project presented hierarchically.



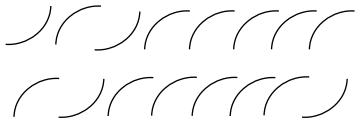
2 Attributes identified by color in a timewheel glyph.

Reducing eye movements per object. Cropper and Evans as well as Danchak found that the visual angle over which the eye is most sensitive is 0.088 radians (5 degrees).[3] Cropper and Evans subsequently stated

> The presentation of information in "chunks" ... which can be taken in one fixation will help to overcome the limitations in the human input system in searching tasks.



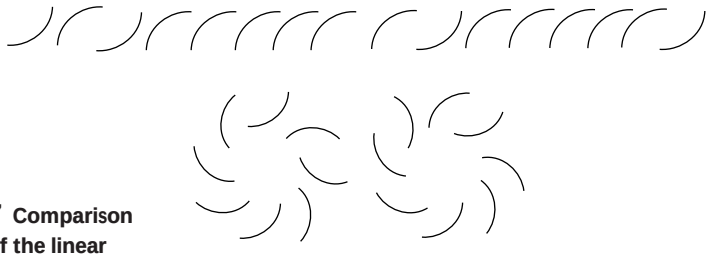
3 Timewheel interface for 16 software releases. Similar patterns are grouped by red and white outlines to identify trends.



4 Linear layout of object-1 and object-2.

5 Circular layout of object-1 and object-2.

6 Cyclic local pattern.



7 Comparison of the linear and circular layouts illustrated in Figures 4 and 5.

Laying out the time series in a linear fashion as in Figure 4 requires more than one eye fixation. Laying them in a circular fashion as in Figure 5 allows all of the time plots to be taken in with one eye fixation. The circular layout reduces the glyph pattern's diameter and better matches the eye's field of vision.

A limit exists on the number of object attributes that can be displayed in the timewheel for it to fit within the area of an eye fixation. As a rough approximation, for a viewer 15 inches from the screen, a visual angle of 5 degrees translates into a circular area on the screen with a radius of 0.65 inches. Our experience is that we can comfortably encode 10 variables in that area. It may be possible to encode as many as 15 variables before the display becomes too dense to interpret.

An alternative layout scheme arranges the time series in rows (Figure 4). This reduces the number of eye fixations; however, the user might begin to cluster the data by rows because we are conditioned to it from reading text. This would adversely affect your ability to sense the overall time patterns in the glyphs. Another possibility positions the time series out in 3D space and encodes properties along the z-axis. The drawback to this 3D layout is occlusion: The first few series occlude the others.

Less susceptible to local patterns. Linear ordering highlights local patterns. For example, in Figure 4 the cyclic local pattern shown in Figure 6 forms a dominant visual impression. The local pattern formed here occurs because our perceptual system groups the two time series based on the gestalt principle of closure.[4] This grouping, however, is spurious because the object attributes have no ordering.

Local grouping effects emphasize the perceptual differences between the two rows in Figure 4. By comparison, the circular placement suffers less because the symbols are not placed directly next to each other. As an example, Figure 5 shows that the rows in Figure 4 are in fact quite similar—one is merely a rotation of the other.

Reading order. A linear layout encourages users to read the plots from left to right. Since the attribute types are unordered, this may cause false impressions. For example, more importance could be placed on the series at the beginning or end. Unlike linear ordering, a circular layout positions each time series at the same distance from the glyph center. In this way, the time series' position has a much weaker ordering implication. Reading order is another reason why the two rows in Figure 4 appear different. The top object has cyclic patterns at its beginning and end, while the bottom object has two opposing patterns.

Less separation for gestalt perception. The circular layout creates a strong gestalt pattern out of individual time series. We recognize the circular pattern because it is a common geometric shape. On the other hand, the linear layout ties the time series together only through spatial proximity. As a result, for us to see the boundaries between objects, we have to leave a lot more white space between the series than in the timewheel case.

The top row in Figure 7, for example, contains the same information as the bottom row and is divided by the same amount of white space. It is hard to see the division between the top objects, while it is much easier to see the division between the bottom objects. Instead of white space we could use a bounding box to indicate object boundaries for the linear layouts, although this adds to the density of the display and may distract the user.[3]

Rotation invariant. Some aspects of our shape perception depend on orientation. For example, we have difficulty recognizing faces and facial expressions from upside down images. Our engineering assumption when designing our timewheel glyphs, and our subjective experience using the glyphs, is that this class of symbol perception is rotation invariant. Our ability to perceive patterns in timewheels is independent of their orientation. Timewheels, in our experience, are particularly effective for showing global symmetry.

Some situations call for a linear ordering rather than a circular ordering. These include, for example, tasks where we want to sense the change in a natural ordered progression of time series, such as multiple time series corresponding to several different releases of a software product. Because the circular layout has a very weak ordering implication, it cannot express the ordered progression of the various time series. The circular layout also proves less appropriate for tasks involving pair-wise comparisons. The time series within a timewheel rotate differently, making it more difficult to compare them.

We designed the timewheel for gestalt comparisons between a set of time trends related to one object with that of another.

Another shortcoming of the circular layout is that it has lower information density or "data to ink ratio."[5] In Figures 4 and 5 the information density is approximately 1.78 time series per square centimeter using the linear layout and 1.26 using the circular layout. The circular layout is slightly less efficient because it wastes the space in the middle.

Besides the horizontal linear ordering, we investigated vertical orderings. For our visualizations we prefer horizontal layouts, since both screens and people tend to have horizontal aspect ratios. This may be because our eyes are arranged side-by-side rather than top-to-bottom on our heads.

Another common way to represent time data maps the *time* data attribute to real time, then animates the visualization display. Although effective for identifying outliers, animations prove less effective than traditional time-series plots for determining overall time patterns.

3D wheel glyph

The 3D wheel encodes the same data attributes as the timewheel, but uses the height dimension to encode time. Each variable is encoded as an equal slice of a base circle, and the radius of the slice encodes the variable's size as in a rose diagram.[6] Each variable is also colored in its own discrete, shaded color.

Figure 8 shows the 16 releases from Figure 3 using 3D wheel glyphs. An object with a sharp apex (as in the right-most glyphs in Figure 8) has an increasing trend through time. An object that balloons out (as in the lower-left glyphs in Figure 8) has a tapering trend.

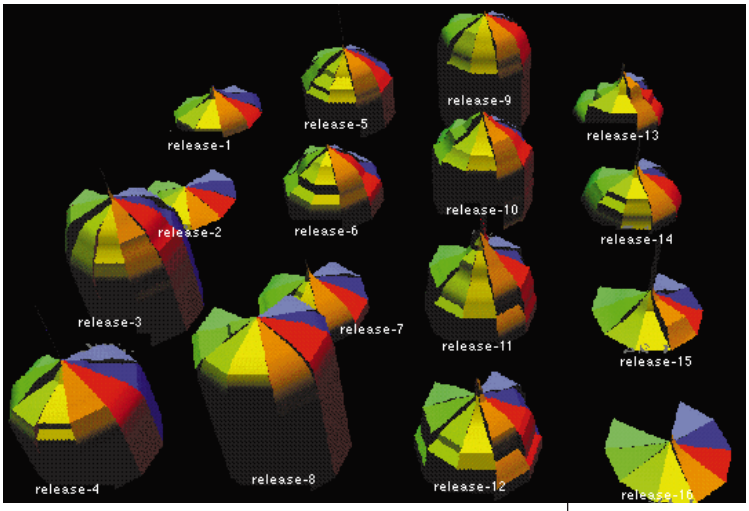
The 3D wheel shares the advantages of the timewheel over linear ordering methods. However, unlike the timewheel, the 3D wheel allows users to perceive the dominant time trend through its shape rather than through the global pattern formed by the series. As a result, it is easier to identify overall time trends using the 3D wheel. Accurately determining the trend of any particular time series within the 3D wheel glyph (or making pair-wise trend comparisons) proves more difficult because human depth perception is not as powerful as our ability to compare 2D spatial positions. It is also harder to identify divergences in trends within a 3D glyph because of occlusion and perspective. Even with occlusion in the 3D wheel, it is still much less than if we were to lay out the time series over the z-axis.

Infobug glyph

The infobug glyph looks like an insect, hence the name. The infobug's wings, head, tail, and body represent four important classes of software data.

The wing

Each wing represents a time series with time running from top to bottom. The x-axis on the left wing encodes the number of code lines, and the x-axis of the right wing encodes the number of errors. Usually, increases in code bring about comparable increases in number of errors. This results in symmetrical insect wings.

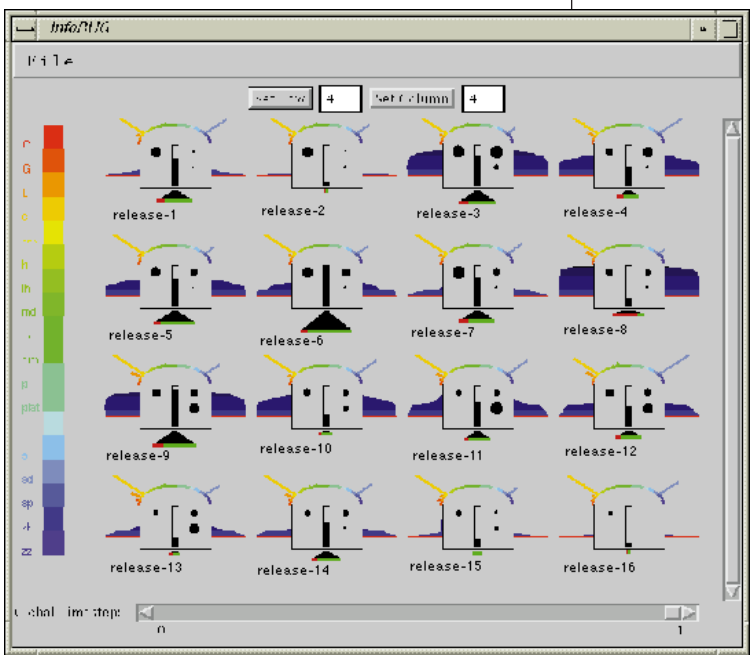


8 The 16 software releases shown in Figure 3 as illustrated with the 3D wheel interface.

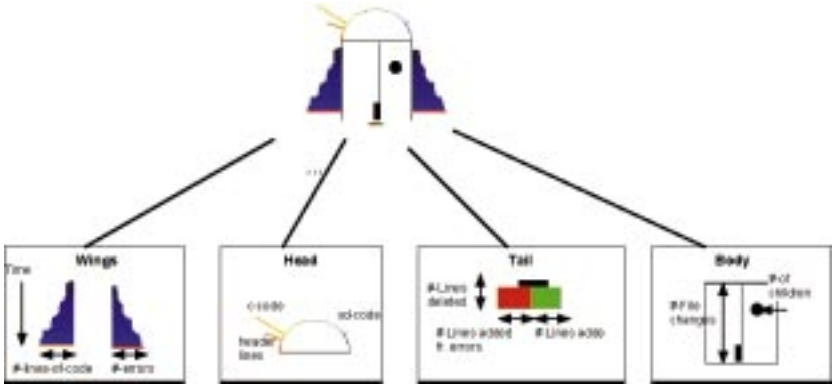
Increases in code not accompanied by similar increases in errors may imply poor testing of the component. On the other hand, increases in the number of errors not accompanied by similar increases in code could mean that the existing code is inherently difficult, has architectural problems, or is poorly written and in need of re-engineering. Nonsymmetrical wings help identify these cases.

Wing position (whether starting at the top or bottom) indicates the time at which the project started, while the wing shape shows whether the number of code lines and the number of errors found are increasing, decreasing, or static with time.

Through animation the infobug glyph can show information at different times within the project. Clicking on the wings selects a time-slice, causing the head, body, and tail to update. The selected time slice shows up as a red band on the infobug wing. The time component for all infobug glyphs can be changed simultaneously by using the slider at the bottom of the interface (Figure 9).



9 The same 16 software releases shown in Figure 3 and Figure 8 illustrated by the infobug interface.



10 Infobug glyph broken down into the head, wing, body, and tail sections.

The head

A particular software component may consist of several different types of code, for example C, C++, Java, or VRML. The code consistency of software components indicates the components' capabilities and purposes. For example, knowing that VRML is used suggests 3D representations. Examining such data might also show changes in development practices and in the requirements for a particular software component.

The head shows the relative code sizes, by type, for a given time slice. The code type is color-coded. The color scale for the encoding appears at the left of the interface in Figure 9. The color scale on Figure 9 is discrete, with 18 different hues. Although the scale looks continuous, the different hues are clearly visible and easily distinguishable on a computer monitor. The software component in Figure 10, for example, consists of C code, SD (state definition) code, and header files. By interactively changing the time component, we can obtain information on how the different code types evolve. Such changes give us hints about a software component's changing needs.

The tail

In the triangle-shaped bug tail, the base encodes the number of code lines added and the height encodes the number of code lines deleted. The tail base further divides into two parts: code added due to error fixing (red) and code added for new functionality (green). Figure 9 shows that most of the releases consist of code added for new functionality, except for bug fixing release-8.

Tail shape determines the ratio between the number of code lines added to lines deleted. A short, squat triangle like the one for release-8 shows a high added-to-deleted ratio. The triangle shapes for most of the other releases are less squat, indicating a lower added-to-deleted ratio.

The body

Often important, component size reflects the extent to which a component affects the project. It is encoded in two ways: through the number of altered files and through the number of child objects a software component contains. The bar in the middle of the infobug body shows the absolute number of file changes. The sizes of the black circles on the body encode the number of child components contained within the current objects. Child-group size helps us gauge whether a software object is wide (related to many other components) or narrow (related to only a few other components). The type of child objects encoded depends on the software hierarchy of the system being analyzed. Our system, for example, is based on the software hierarchy shown in Figure 1.

In our experience, the infobug wings and head may dominate our perception of the glyph and make it seem less integrated—it may not be viewed as a whole. This contrasts with Chernoff faces, where the different facial features are naturally viewed together because of our lifelong training in recognizing faces. This contributes directly to the main strength of Chernoff faces—our ability to use them for rapid gestalt comparisons. Nevertheless, it is also more difficult to separate the different facial features and determine which feature(s) contribute to a particular facial expression (visual pattern). Thus, it may be difficult to interpret the face glyphs when mapped to data. Infobugs make gestalt comparisons slightly harder, but simplify analyzing the glyph and separating the specific parts of the glyph (sections of the data) contributing to a visual pattern. In the future, we hope to experiment with the saliency of different infobug components and determine their effectiveness. We would also like to explore the importance and effect of integration, and ultimately hope to strike a good balance between integration and separability.

Glyph design philosophy

Three guiding principles motivated our glyph designs: small multiples, established visualizations, and information rich glyphs.

Small multiples

Small multiple designs contain a small number of visually rich representations arranged on a grid, where each representation must be derived from the same design structure. Figures 3 and 9 show examples of small multiple designs. Each visualization contains 16 glyphs arranged on a 4×4 grid, with each glyph built from the same circular or insect-like design structure. Small multiple designs are effective for three reasons:

1. *Visual constancy*: Because the same design structure repeats, we tend to focus on the change in data from one representation to the next rather than on change in graphical representation.
2. *Economy of perception*: From a small multiple design, "An economy of perception results; once viewers decode and comprehend the design for one slice of data, they have familiar access to data in all the other slices."[7]
3. *Uninterrupted visual reasoning*: "Their multiplied smallness enforces local comparisons within one eye-span, relying on an active eye to select and make contrasts rather than on bygone memories of images scattered over pages and pages."[7]

Established visualizations

We combine well-known, established visualization views (time series, histogram, and rose diagram) to form our glyphs. This solves an important problem with information rich glyphs discovered by Chernoff in 1973.[1] Chernoff mapped data attributes onto features of faces in the hope of exploiting a person's innate ability to group faces in interesting ways. The problem, however, is that salient facial features do not necessarily conform to important information relationships.[8] In addition, facial groupings may be subjective and not consistent for all users.

By using established visualization views, we ensure that the visual patterns formed within the glyph will better correspond to real patterns within the data set—and thus recognized by glyph users. In addition, interpretation of the glyph is simplified by using prior graphic knowledge that involves significantly less interpretive subjectivity than with unconventional glyphs.

Information rich glyphs

Our glyphs have visually rich representations so that we can view many dimensions of a data object simultaneously. Each glyph consists of a clustered set of simple graphical artifacts. Users achieve visual grouping of the elements in each glyph by the proximity of the visual elements as well as by arranging the elements to form a familiar object or a common geometric shape (the insect glyph or the circle).

Another powerful method for viewing multidimensional objects is through linked scatter plots.[9] Information rich glyphs have an advantage over linked plots: information rich glyphs preserve the "objectness" of the data elements. In other words, all properties of a particular software component group together spatially. This preservation is important in analyzing software systems because the data elements within that domain conform to real programs, people, or concepts.

Another possibility, using dense glyph fields, densely concentrates stick-figure glyphs into a 2D or 3D space.[10] In such displays, useful data is extracted from the "texture contours" or variation formed within the dense glyph field. A weakness of this approach is that unlike scatter plots, it ignores the use of spatial cues. These cues can effectively reveal relationships among data attributes. It also falls prey to the *"moiré* effect in which the designs interact with the physiological tremor of the eye to produce the distracting appearance of vibration and movement."[5]

Conclusion

We have developed three information-rich glyphs and applied them to a large, multidimensional data set in the software project management domain. The glyphs, coupled with filtering and aggregation, highlight interesting patterns and anomalies in the data set.

A problem with our approach is that the aggregate summaries used in the glyphs may hide important data patterns from the user. To ensure that the user does not lose any crucial information, it is important visually to encode statistics that may indicate the accuracy and importance of a particular aggregate, such as number of data objects within the aggregate, or the standard deviation for all the summary data attributes. We already have some of this information integrated into our glyphs, but hope to include more in the future. ■

References

1. H. Chernoff, "The Use of Faces to Represent Points in *k*-Dimensional Space Graphically," *J. American Statistical Association*, Vol. 68, No. 342, 1973, pp. 361-368.
2. B.E. Rogowitz and L.A. Treinish, "An Architecture for Rule-Based Visualization," *Proc. IEEE Visualization 93,* IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 236-243.
3. T.S. Tullis, "The Formatting of Alphanumeric Displays: A Review and Analysis," *J. of Human Factors*, Vol. 25, No. 6, 1983, pp. 657-682.
4. K.T. Spoehr and S.W. Lehmkuhle, *Visual Information Processing*, W.H. Freeman and Company, San Francisco, 1982.
5. E.R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut, 1983.
6. H. Wainer, *Visual Relations*, Springer-Verlag, New York, 1997.
7. E.R. Tufte, *Envisioning Information*, Graphics Press, Cheshire, Connecticut, 1990.
8. D. Marchette, "An Investigation of Chernoff Faces for High Dimensional Data Exploration," *http://irisd.nswc.navy.mil/CSI803/Dave/chern.html.*
9. R.A. Becker and W.S. Cleveland, "Brushing Scatter Plots," *Technometrics*, Vol. 29, No. 2, 1987, pp. 127-142.
10. R.M. Pickett and G.G. Grinstein, "Iconographic Displays for Visualizing Multidimensional Data," *Proc. IEEE Conf. on Systems, Man, and Cybernetics*, IEEE Press, Piscataway, N.J., Vol. 1, 1988, pp. 514-519.

Mei C. Chuah *is a computer science graduate student at Carnegie Mellon University. She is currently finishing up her dissertation on characterizing interactive visualization methods and automatically generating these methods based on task and data requirements. Her current interests lie in the areas of interactive visualizations, 3D visual representations, and automatic visualization design.*

Stephen G. Eick *is the chief technologies officer of Bell Lab's Visual Insights venture and leads the Interactive Data Visualization Research group. His interests include visualizing databases associated with large software projects and networks, and building high-interaction user interfaces. He has a BA from Kalamazoo College, MA from the University of Wisconsin at Madison, and PhD from the University of Minnesota.*

Readers may contact Chuah at School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, e-mail mei+@cs.cmu.edu.