NISS

Visualizing Software Changes

Stephen G. Eick, Paul Schuster, Audris Mockus,
Todd L. Graves, and Alan F. Karr

Technical Report Number 113
December, 2000

National Institute of Statistical Sciences
19 T. W. Alexander Drive
PO Box 14006
Research Triangle Park, NC 27709-4006
www.niss.org

Visualizing Software Changes

Stephen G. Eick,[*] Paul Schuster[†]
Visual Insights
{eick, pschuster}@visualinsights.com

Audris Mockus[‡]
Bell Laboratories
audris@research.bell-labs.com

Todd L. Graves[§]
Alan F. Karr[¶]
National Institute of Statistical Sciences
tgraves@lanl.gov; karr@niss.org

27 December 2000

Abstract

Visualizations of software changes are presented that complement existing visualizations of software structure. The principal metaphors are matrix views, cityscapes, bar and pie charts, data sheets and networks. Linked by selection mechanisms, multiple views are combined to form *perspectives* that both enable discovery of high-level structure in software change data and allow effective access to details of those data. Use of the views and perspectives is illustrated in two important contexts: understanding software change by exploration of software change data and management of software development.

---

[*] Room 1B-519, Visual Insights/Lucent Technologies, 263 Shuman Blvd., Naperville, IL 60566

[†] Room 1B-507, Visual Insights/Lucent Technologies, 263 Shuman Blvd., Naperville, IL 60566

[‡] Room 2F-319, Bell Laboratories, 263 Shuman Blvd., Naperville, IL 60566

[§] Currently at Los Alamos National Laboratory, Los Alamos, NM 87545. Research supported in part by NSF grants DMS–9208758 and SBE–9529926 to the National Institute of Statistical Sciences.

[¶] P.O. Box 14006, Research Triangle Park, NC 27707–4006. Research supported in part by NSF grant SBE–9529926 to the National Institute of Statistical Sciences.

1 Introduction

A fundamental problem in software engineering for large systems is changing the code, to add new functionality, accommodate new hardware, support new operating environments and fulfill increased user expectations. In an ideal world, software architecture would anticipate and facilitate future changes. In reality, the architecture is imperfect, and incorporates compromises forced by time and cost constraints. As a consequence, an immense burden falls on the change process, which becomes complex, costly, hard to manage and difficult even to understand.

Data on software changes are widely available from version management databases. A compelling opportunity, then, is to use these data to enable understanding and management of the change process.

However, in many settings, the scale and complexity (see §2.3) are daunting. Even handling the data is an issue: custom scripts and tools must be created to extract and manipulate the data to put them in proper form for analysis (Mockus, *et al.*, 1999). Visualization is a natural, effective (and perhaps essential) way to deal with scale and complexity. We have developed a number of visualization tools, which not only facilitate rapid exploration of high-level structure in software change data, but also serve as a powerful visual interface to the details of the data. In this paper we describe these tools, and illustrate their application to two key problems. The first problem is *Understanding Software Change*, especially in order to formulate questions not raised previously, to be answered by more formal statistical analyses. The second problem is *Management of Software Development*, for which precise, quantitative results of formal analyses may be less important than the rapid, qualitative understanding of the current status of a development project that visualizations afford.

The remainder of the paper is organized as follows. The setting for our research, the software change process and the data are summarized in §2. Abstractions and principles underlying the visualizations are articulated in §3, where the visual metaphors — *views* — that we employ are described. Applications of the visualizations to the two problems identified above are presented in §4 (Understanding Software Change) and §5 (Management of Software Development). We conclude, in §6, with a discussion and evaluation of the tools.

2 Software Changes and Data

Our definition of a change to software is data-driven: a change is any alteration to the software recorded in the change history database. With some simplification, changes fall naturally into three main classes (An, Gustafson & Melton, 1987; Swanson, 1976). Adaptive changes add new functionality to a system (for example, caller ID in a telephone switch), or adapt the software to new or changed hardware, or to other alterations in its environment. Corrective changes fix faults in the software. Perfective changes (also called “re-engineering”) are intended to improve the developers’ ability to maintain the software (in particular, to make additional changes in the future) but do not by themselves alter functionality or fix faults.

2.1 Setting

The tools presented here were developed in the context of an uncommonly rich data set: the entire change history of a large, fifteen-year old real-time software system for telephone switches. Currently, the system comprises 100,000,000 (numbers are approximate) lines of source code (in C/C++, SDL, a proprietary state description language, and other languages) and 100,000,000 lines of header and make files, organized into some 50 major subsystems and 5,000 modules. For our purposes, a *module* is a directory in the source code file system, so that a code module is a collection of several files. Each release of the system consists of some 20,000,000 lines of code. More than 10,000 software developers have participated in the project over the last fifteen years.

Though the need for visualization is especially acute for projects of this magnitude, our visual tools are broadly and widely applicable within the software development process. Although the original ideas were conceived in the context of large scale software production, but they are equally useful for smaller department-sized projects: in §5 we illustrate their use for daily management of a 25-person development organization building a 250,000 line software system aimed at business intelligence.

2.2 The Change Process

The changes to the source code follow a well-defined, institutionalized process, whose main, hierarchical components are:

Features (for example, call waiting or credit card billing) are the fundamental requirements unit by which the system is extended.

Initial Modification Requests (IMRs) are the high-level design information by means of which the changes that implement a feature are transmitted to the development organization. Typically there are hundreds of IMRs per feature.

Modification Requests (MRs) translate IMRs into the low-level design information representing the work to be done to each module; thus there are multiple MRs per IMR. Described differently, an IMR is a problem, while an associated MR is all or part of the solution to the problem. The supervisor responsible for an IMR distributes the work to developers as MRs.

The developer to whom an MR is assigned “opens” the MR, makes the required modifications to the code, checks whether the changes are satisfactory (within a limited context, i.e., without a full system build), and then submits the MR. Code inspections and integration and system tests follow.

Deltas are editing changes to individual files in order to complete an MR. A file is “checked out” by a developer, edited and then “checked in.”

2.3 Change Data

Data pertaining to the change history of the code itself reside in a version management system (similar to SCCS (Rochkind, 1975)), which tracks changes at the feature, IMR, MR and delta

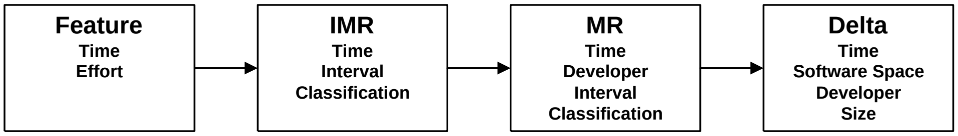


Figure 1: The Software Change Process and Associated Data. The hierarchy of the process flows from left to right. Software space, size, and interval can be defined at higher levels by aggregation. (For example, the size of a feature is the sum of the sizes of the associated deltas.)

levels. Within the version management system, the structure of the change data is as follows (see Figure 1):

Feature. In addition to descriptions, data for features include time and effort, the latter inferred from developers' time records using methodology described in Graves & Mockus (1998, 1999).

IMR. Each IMR has an extensive record (89 fields in all) containing priority, date opened and closed, and the point in the development process when it was initiated (requirements, design, coding, testing, field operation).

MR. Each MR has a parent IMR, dates and affected files, and an abstract — English text describing the change and the reasons for it. There is no explicit format on how and what information is entered for that field; however, its purpose is to allow other developers to understand what change was made and why.

Delta. The data for each delta list the parent MR and the date and time when the change was submitted to the version management system as well as numbers of lines added, deleted, and unmodified by that change.

This level of detail preserves the capability to build earlier versions of the software, which is necessary in order to serve customers with older versions.

Conceptually, the fundamental components of the change data are:

Time: When a change was made.
Software Space: Which files were changed, and which lines were added and deleted.
Developer: The person who made the change.
Size: How many modules, files, and lines were affected.
Effort: How many developer-hours were required to perform the change.
Interval: How long the change took, in calendar time.
Classification of the change as adaptive, corrective or perfective.

Figure 1 indicates the form and level of aggregation at which the version management database contains these components. The classification data have problematic aspects discussed in Eick, *et al.* (1998).

3 Principal Visualization Metaphors

A fundamental problem in visualizing software changes — and in information visualization generally (Card, Mackinlay & Shneiderman, 1999) — is to choose effective visual representations (metaphors) for data that are not inherently physical. The goal is an insightful rather than a faithful depiction of the data. As noted in §1, visualizations are especially effective if they both facilitate rapid exploration of high-level structure in software change data *and* serve as effective interfaces to the details of the data.

For purposes of visualization, the components of software change data listed in §2.3 can be grouped into two categories. The first — *Indices* — are the independent variables in views. Ordinarily, these are time, classification, software space and developer. The second category is *Responses*, which are the dependent variables in views. Size, effort and interval are typical responses. The views we present show one or responses as functions of one or more indices. In some cases, the value of one or more responses is used to filter the view of another. The indices and responses can be used to define a taxonomy of views, which we do not elaborate upon here.

Of the components of change listed in §2.3, only time has an apparent physical representation. While structured (for example, as a network defined by links between modules, as in Figure 10) software space is not intrinsically physical. Software space and developer can, as categorical variables, index matrix and landscape views effectively (§3.1). Size, effort and interval, as numerical variables, can be mapped onto several visual attributes, such as color, size or shape.

We now describe the six primary forms of visualization used in this paper: matrix views (§3.1), cityscape views (§3.2), bar and pie charts (§3.3), data sheets (§3.4) and network views that relate changes to software structure (§3.5). Each visual component, *view* for short, embodies a visual metaphor or representation of the data. In our implementation, the views are interactive and function both as visual displays and as an environment for visual analysis. Obviously the six visual tools that we consider do not exhaust all possibilities; instead they are examples chosen to show the power of visualization in specific settings and the metaphors that we have found most useful in our analyses. Nor do the components exist in isolation from one another: especially in §5, *perspectives* (§3.6) — multiple, linked views of different aspects of the change data, using a different visual metaphor for each — are central.

3.1 Matrix Views

Matrix views (Bertin, 1981) are effective for displaying one or more responses as a function of two numerical or categorical indices. The view itself, as in Figure 2, is a two-dimensional grid with rows corresponding to one index and columns to the other. Cell (i, j) contains a glyph depicting the values of one or more (usually numerical) responses when the row index is i and column index is j. The responses are encoded as visual attributes — color, texture, shape and size — of the glyph.

In Figure 2, the indices are developers (rows) and software space (columns correspond to modules — subdirectories in the source code tree containing files with related functionality) and two responses are shown: size of changes, which is mapped onto the width of the bar in each cell, and developer, which is redundantly mapped onto color.

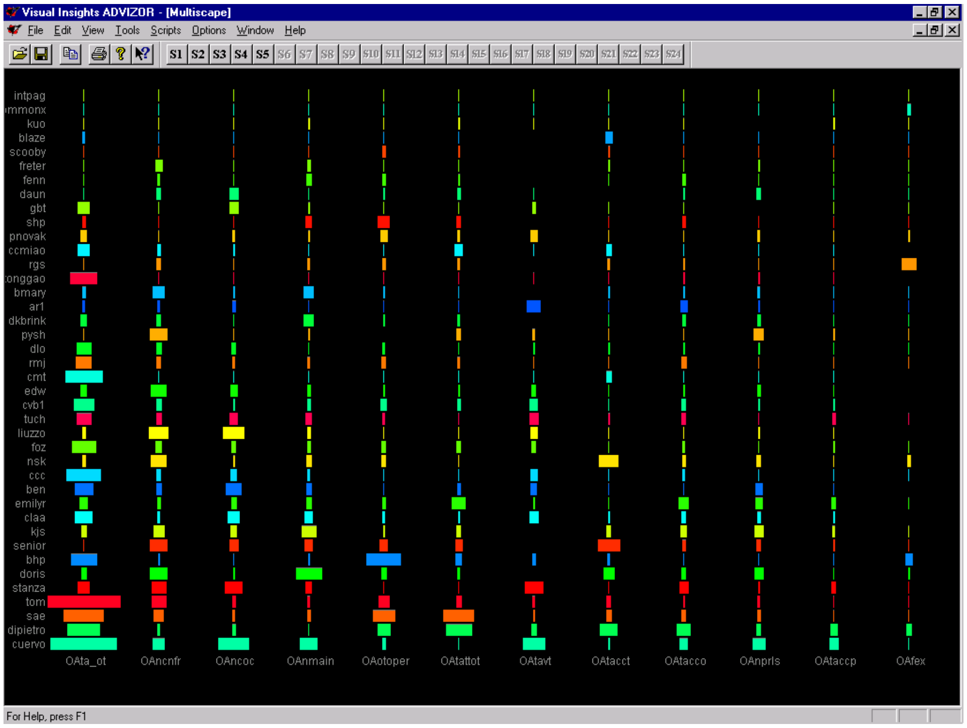


Figure 2: Example of a Matrix View. *Indices:* Rows represent developers, columns are software space, at the level of aggregation of modules. *Responses:* bar width encodes module size, bar color encodes developer.

Figures 8 and 9 contain additional examples of matrix views. In the former, as is sometimes useful for numerical responses, marginal information is displayed along the edges of the matrix view.

3.2 Cityscape Views

Cityscapes (Hill & Hollan, 1991), or three-dimensional bar charts, are three-dimensional extensions of matrix views. There are two indices and one or more responses. Attributes that encode the response variables are, typically, the height and color of the vertical towers comprising the cityscape. In some implementations, the walls of the view are used to display marginal information.

The cityscape view in Figure 3, which has the same indices as the matrix view in Figure 2, uses the heights of vertical towers to encode the number of changes made by each developer to different modules of the code. Color (redundantly but effectively) encodes the number of changes. Figure 9 contains an additional illustration of a cityscape visualization.

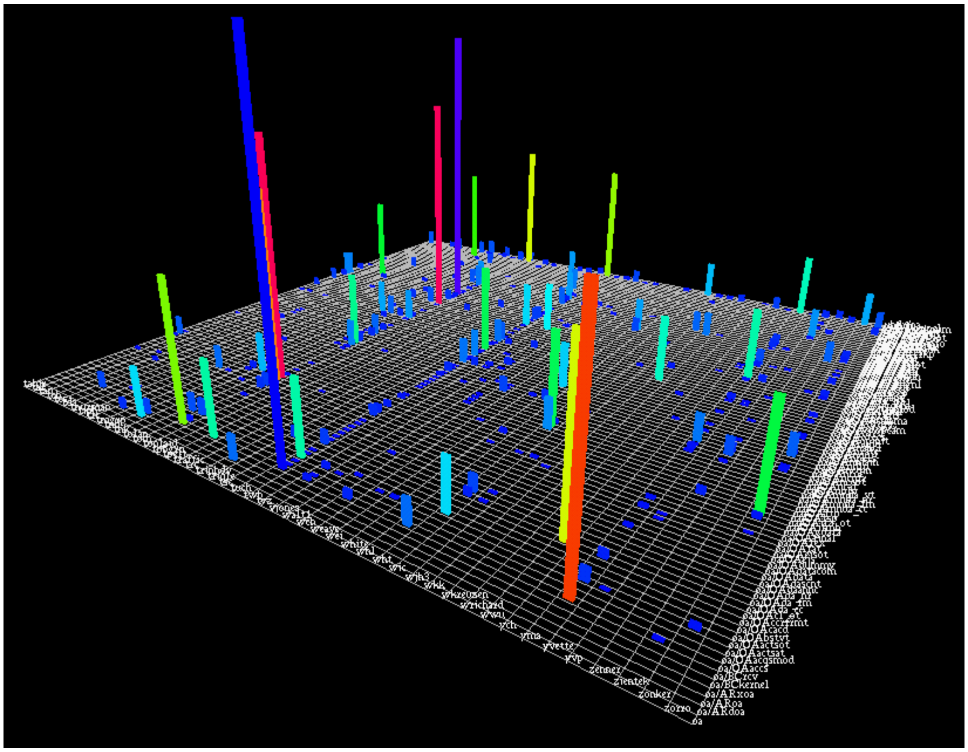


Figure 3: Example of a Cityscape View. *Indices:* Rows represent developers, columns are software space, at the level of aggregation of modules. *Responses:* Bar color and bar height both (redundantly) encode total number of changes.

3.3 Bar and Pie Charts

Simple views such as *bar charts* and *pie charts* complement the richer, more complicated matrices and cityscapes. Although the visual metaphors embodied in bars and pies are well-known, in our implementation we have extended bars and pies in two important ways.

First, bar charts incorporate new mechanisms for *scalability*: by exploiting direct manipulation and interactivity, it is possible to visualize bar charts with thousands of bars. Such data volumes render static bar charts unusable. Figure 7 illustrates. Pie charts, of course, do not scale very effectively.

Second, like all of our tools, bar and pie charts support *selection*, which enables a user, using the mouse, interactively to select an interesting subset of the data. The view incorporates mechanisms for removing (or attenuating) the remainder. Perspectives (§3.6) offer especially powerful ways to implement selection, by transmitting its effects to all views linked to the one in which the selection takes place.

In addition, bar charts are color-coded using *color stacking*. For example, if the bars are showing MRs by programmer and color is tied to MR severity, the colors for each severity will be stacked with each programmer's bar, as in Figure 11. This allows easy comparison of different severities across programmers.

3.4 Data Sheets

A data sheet is a scrollable text visualization (Eick, 1994; Eick, *et al.*, 1997b) providing direct access to individual data elements, such as IMRs, MRs or deltas. Although a sheet is a fundamentally simply a multi-column textual display that can be sorted on the basis of any column, there is one crucial, interactive extension. To deal with scale, as the user zooms out to show more data elements, the text font size shrinks, eventually collapsing into tiny horizontal bars, with the bar length tied to the string width for text variables or encoding the numeric value for numerical variables. (Table Lens (Inxight, 1999) has similar capabilities.)

3.5 Network Views

Although it is complex, software space does have structure. First, there is hierarchical structure (subsystems, modules, files) defined by the architecture of the code. Additional structure is associated with program states, function calls and shared variables. Still other structure may be associated with the change process itself, with modules or files linked, for example, by a history of common changes.

Data Constellations (Wills, 1999) is a visual tool for showing the structure of software space, using a network metaphor. Nodes represent software units (typically, modules), and visual attributes encode measures of association between nodes. In Figure 10 (see §4.6 for discussion), association corresponds to modules' having been changed together in the past (Eick, *et al.*, 1998). Interactive capabilities, including pan, zoom, and filtering, are available.

3.6 Perspectives

As discussed in more detail in §6.3, each kind of view has strengths and weaknesses. Some (particularly matrix, cityscape and network views) support discovery of high-level structure in software change data, but are less effective as interfaces to the details. Data sheets, by contrast, by their very nature provide immediate access to details, but do not convey high-level structure very well. Bar and pie charts are primarily tools to explore and summarize the data, and are also effective selectors.

To exploit the complementary capabilities of different views, we construct *perspectives*, in which multiple views of the same software change data are displayed simultaneously, each using a different visualization strategy. More important, the views in a perspective are *linked*: selection operations (see §3.3) in one view are realized in the others as well. This allows the views that summarize or access details to act as filters for the views that enable understanding of high-level structure.

Figure 5, which contains linked bar and pie charts, is a simple, illustrative example. Sections 4 and 5 present multiple applications of perspectives in the contexts of understanding software change and managing software development.

4 Understanding Software Change

In this section, we show how the visualizations described in §3 function as tools to understand software change by exploring software change data. The setting for such exploration is one of research, in which the visualizations fulfill two principal functions. First, they are interfaces to the data, allowing exploration, browsing, filtering and drill–down in ways that other methods, such as SQL queries to the version management database, simply do not.

Second, the visualizations support the formulation of interesting questions about the software change process, as well as hypotheses about the change process, that would not be posed otherwise. The answers to such questions may be derived in a variety of ways, including additional exploration and formal statistical analyses (e.g., as in Eick, *et al.* (1998) or Graves, *et al.* (1997a)).

As noted in §2.1, our database is the entire, fifteen year change history of a large, real-time software system for telephone switches. Because of the limitations involved in presenting views non-interactively on paper, in this section we restrict attention to changes to one subsystem of the software, which is involved with operator interaction with the switch. This subsystem, created in 1984, has been changed 150,850 times (deltas) by 549 programmers. As shown in the bar chart in Figure 4, tiny amounts of initial development occurred in 1984 and 1985, jumping to approximately 7,000 changes in 1986, becoming nearly 16,000 changes in 1990, and decreasing to just over 6,000 changes by 1998. Data for 1999 are for only part of the year.

4.1 Basic Statistics of Changes

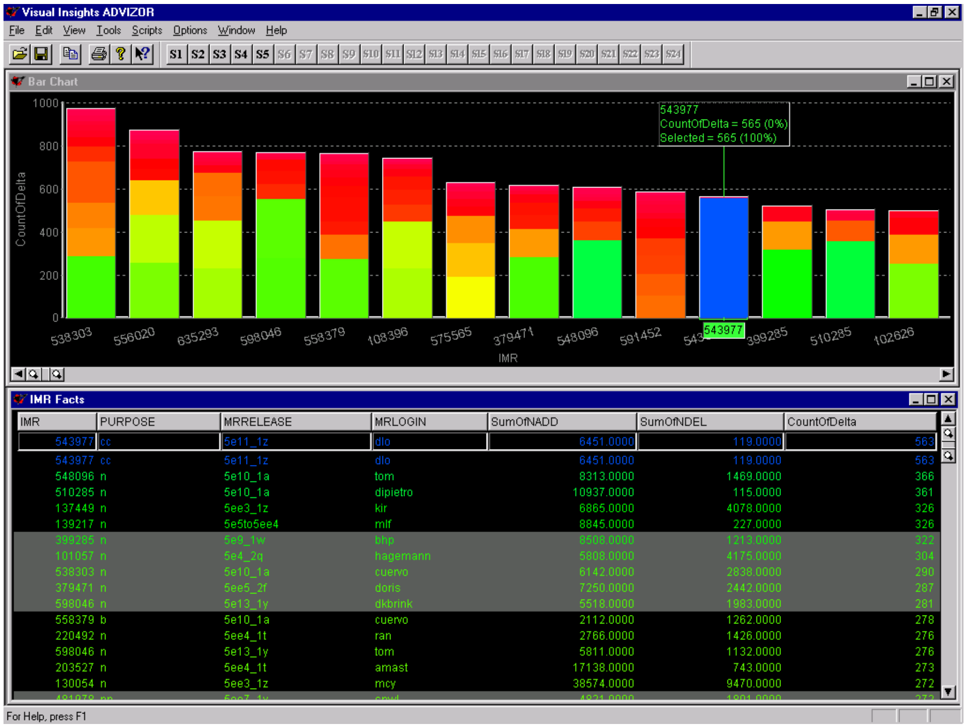
The perspective shown in Figure 4 investigates the structure of changes — specifically, of IMRs. Recall that an IMR describes a problem and has associated to it a set of MRs that solve the problem. For the subsystem under study, Figure 4 shows the number of deltas per IMR, the associated release, number of developers active on the IMR, the number of lines added and the number of lines deleted. The color coding distinguishes these different items for each IMR. In Figure 4, only the 14 IMRs with the largest numbers of deltas are selected, but both the bar chart and the data sheet can accommodate all 16,062 IMRs for this subsystem.

In the bar chart, the user is touching (with the mouse) the blue bar corresponding to IMR 543977, which has an unusually large number of deltas (563) associated with. The data sheet then allows access to detail for that IMR.

4.2 Changes Indexed by Time

Figure 5 is a perspective emphasizing changes as a function of time. It contains a bar chart showing changes by year, a pie chart showing changes by file type and year, and three data sheets listing (1) Deltas, including parent feature and IMR; (2) Module properties, namely, module size and aggregated change sizes (lines added and deleted); and (3) The same module-indexed data as in (2), but sorted by the total number of changes. The colors, which are common to all views, encode the years of changes.

The primary purpose of this perspective is to pose and answer questions regarding the historical pattern of effort and types of files changed. The data sheets give access to details of the data, by



IMR	PURPOSE	MRRELEASE	MRLOGIN	SumOfNADD	SumOfNDEL	CountOfDelta
543977	cc	5e11_1z	dlo	6451.0000	119.0000	563
543977	cc	5e11_1z	dlo	6451.0000	119.0000	563
548096	n	5e10_1a	tom	8313.0000	1469.0000	366
510285	n	5e10_1a	dipietro	10937.0000	115.0000	361
137449	n	5ee3_1z	kir	6865.0000	4078.0000	326
139217	n	5e5to5ee4	mlf	8845.0000	227.0000	326
399285	n	5e9_1w	bhp	8508.0000	1213.0000	322
101057	n	5e4_2q	hagemann	5808.0000	4175.0000	304
538303	n	5e10_1a	cuervo	6142.0000	2838.0000	290
379471	n	5ee5_2f	doris	7250.0000	2442.0000	287
598046	n	5e13_1y	dkbrink	5518.0000	1983.0000	281
558379	b	5e10_1a	cuervo	2112.0000	1262.0000	278
220492	n	5ee4_1t	ran	2766.0000	1426.0000	276
598046	n	5e13_1y	tom	5811.0000	1132.0000	276
203527	n	5ee4_1t	amast	17138.0000	743.0000	273
130054	n	5ee3_1z	mcy	38574.0000	9470.0000	272

Figure 4: Perspective Showing Deltas Indexed by IMR, zoomed in on the fourteen IMRs with the most deltas. Colors show various IMR characteristics. *Bar chart:* number of deltas per IMR and number of lines added and deleted, differentiated by color. *Data sheet:* details for individual IMRs. Discussed in §4.1.

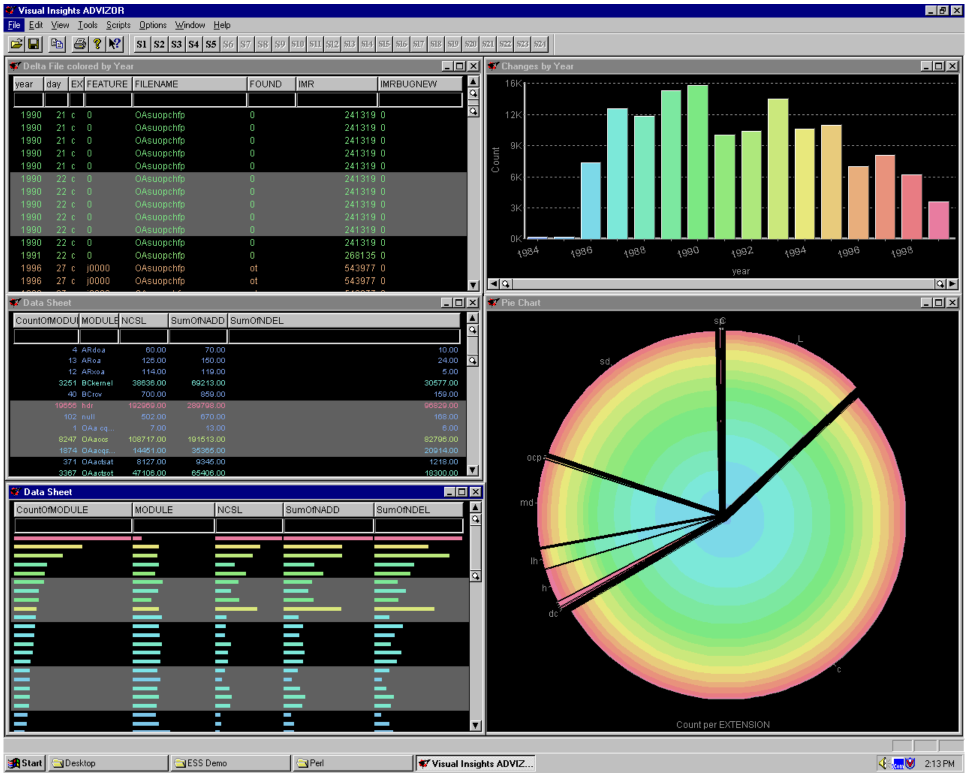
both change and module. One can see in the bar chart in Figure 5, for example, a significant decline in the annual number of changes in recent years.

The pie chart, in which wedges display numbers of changes by file type, shows that just over one-half of the changes have been to C language code, with just over an 1/8 to `sd` (state definition) files, and just under one-eighth to `md` (make files). The colors, representing year as in the bar chart, allow one to see that changes to `sd` files occur relatively more recently than those to other types of files.

The top data sheet serves mainly as a selector, enabling specific deltas to be mapped onto time, size, file type and other characteristics of changes. The bottom two data sheets (which contain the same data, but sorted differently) show that the most frequently changed subsystem is `hdr`. This not surprising, since `hdr` contains (global) header files that define cross-module interfaces.

4.3 Changes Indexed by Developer

The perspective in Figure 6 investigates the question "Who wrote the code?" The two data sheets contain the same information (the one on the left is a compressed version of the one on the right), indexed by developer: number of changes, total number of lines added and total number of lines deleted. Both are sorted by the number of changes, which is mapped onto color. Although Figure



year	day	EX	FEATURE	FILENAME	FOUND	IMR	IMRBUGNEW
1990	21	c	0	OAsuopchfp	0	241319	0
1990	21	c	0	OAsuopchfp	0	241319	0
1990	21	c	0	OAsuopchfp	0	241319	0
1990	21	c	0	OAsuopchfp	0	241319	0
1990	21	c	0	OAsuopchfp	0	241319	0
1990	22	c	0	OAsuopchfp	0	241319	0
1990	22	c	0	OAsuopchfp	0	241319	0
1990	22	c	0	OAsuopchfp	0	241319	0
1990	22	c	0	OAsuopchfp	0	241319	0
1990	22	c	0	OAsuopchfp	0	241319	0
1990	22	c	0	OAsuopchfp	0	241319	0
1991	22	c	0	OAsuopchfp	0	268135	0
1996	27	c	j0000	OAsuopchfp	ot	543977	0
1996	27	c	j0000	OAsuopchfp	ot	543977	0

CountOfMODUL	MODULE	NCSL	SumOfNADD	SumOfNDEL
4	ARdoa	60.00	70.00	10.00
13	ARoa	126.00	150.00	24.00
12	ARxoa	114.00	119.00	5.00
3251	BCkernel	38636.00	69213.00	30577.00
40	BCrcv	700.00	859.00	159.00
19656	hdr	192969.00	289798.00	96829.00
102	null	502.00	670.00	168.00
1	OAa cq...	7.00	13.00	6.00
8247	OAaccs	108717.00	191513.00	82796.00
1874	OAacqs...	14451.00	35365.00	20914.00
371	OAactsat	8127.00	9345.00	1218.00
3367	OAactsot	47106.00	65406.00	18300.00

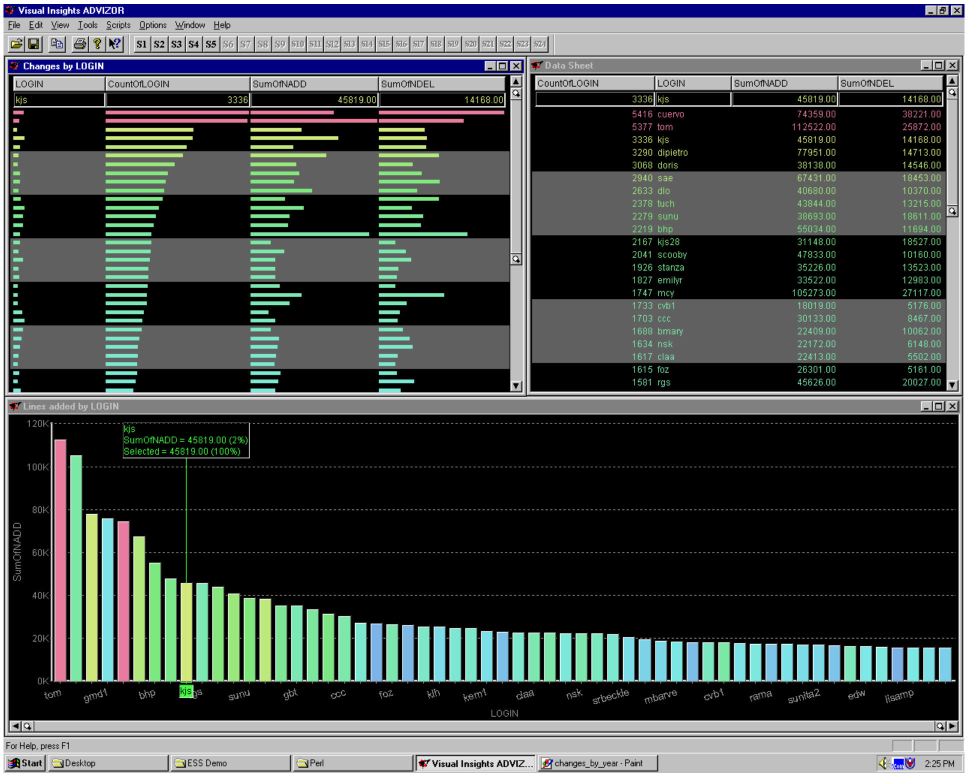
CountOfMODULE	MODULE	NCSL	SumOfNADD	SumOfNDEL

Figure 5: Perspective Showing Changes Indexed by Time. Color encodes year. *Bar chart:* numbers of changes by year. *Pie chart:* numbers of changes by file type. *Data sheets:* individual deltas (top) and changes aggregated by module (middle) and subsystem (bottom). The bottom data sheet shows the conversion of numerical values in a data sheet to bars, as described in §3.4. Discussed in §4.2.

6 does not convey it directly, the display has been filtered interactively to show only the top 10% of the programmers, measured by number of changes.

The data sheets corroborate the 80%–20% rule (twenty percent of the programmers make 80 percent of the changes). In this case, the rule is closer to 90%–10%: the top 50 of 549 programmers made the majority of the changes. The two most active programmers, *prog1* (actual developer names have been masked) and *prog2*, made 5,416 and 5,377 changes (together, 7% of the changes), respectively, adding or deleting 112,580 and 138,494 lines of code. The third most active programmer, *prog3*, made (only) 3,336 changes.

The bar chart shows number of lines added, indexed by developer, with color continuing to represent the number of changes. That number of changes and lines added are positively but not perfectly correlated is clear from this view; a statistical analysis could be performed to quantify the correlation, or for comparison to other subsystems.



LOGIN	CountOfLOGIN	SumOfNADD	SumOfNDEL
kjs	3336	45819.00	14168.00

CountOfLOGIN	LOGIN	SumOfNADD	SumOfNDEL
3336	kjs	45819.00	14168.00
5416	cuervo	74359.00	38221.00
5377	tom	112522.00	25872.00
3336	kjs	45819.00	14168.00
3290	dipietro	77951.00	14713.00
3068	doris	38138.00	14546.00
2940	sae	67431.00	18453.00
2633	dlo	40680.00	10370.00
2378	tuch	43844.00	13215.00
2279	sunu	38693.00	18611.00
2219	bhp	55034.00	11694.00
2167	kjs28	31148.00	18527.00
2041	scooby	47833.00	10160.00
1926	stanza	35226.00	13523.00
1827	emilyr	33522.00	12983.00
1747	mcy	105273.00	27117.00
1733	cvb1	18019.00	5176.00
1703	ccc	30133.00	8467.00
1688	bmary	22409.00	10062.00
1634	nsk	22172.00	6148.00
1617	claa	22413.00	5502.00
1615	foz	26301.00	5161.00
1581	rgs	45626.00	20027.00

Figure 6: Perspective Showing Changes Indexed by Developer. *Data sheets:* Numbers of changes, lines added and lines deleted. *Bar chart:* Number of lines added, indexed by developer. Discussed in §4.3.

4.4 Size of Changes Indexed by Release

Figure 7 is a perspective containing two bar charts that show the size of changes indexed by release. In both charts, each release corresponds to two bars, one showing lines added, the other showing lines deleted. Releases are ordered by time in the lower chart, with color encoding release number (and, indirectly, time). Major releases — measured by size — are clearly distinguishable from minor ones. The upper bar chart uses the same colors as the lower one, but in it releases are ordered according to lines of code added or deleted, providing an effective selector to focus on details of the data.

4.5 Activity Indexed by Developer and Software Space

Here we address a variant of the question in §4.3: “Who changed which parts of the code?” (That is, who worked in which parts of software space?). Figure 8 contains a matrix view showing the number of changes indexed by developer (horizontal axis) and module (vertical axis), in this case the most informative unit of granularity of software space. Developers are sorted by total number of changes, shown as a histogram beneath the matrix view. The two bar charts illustrate different forms of drill–down access to details of the data. That on the left focuses on a few selected

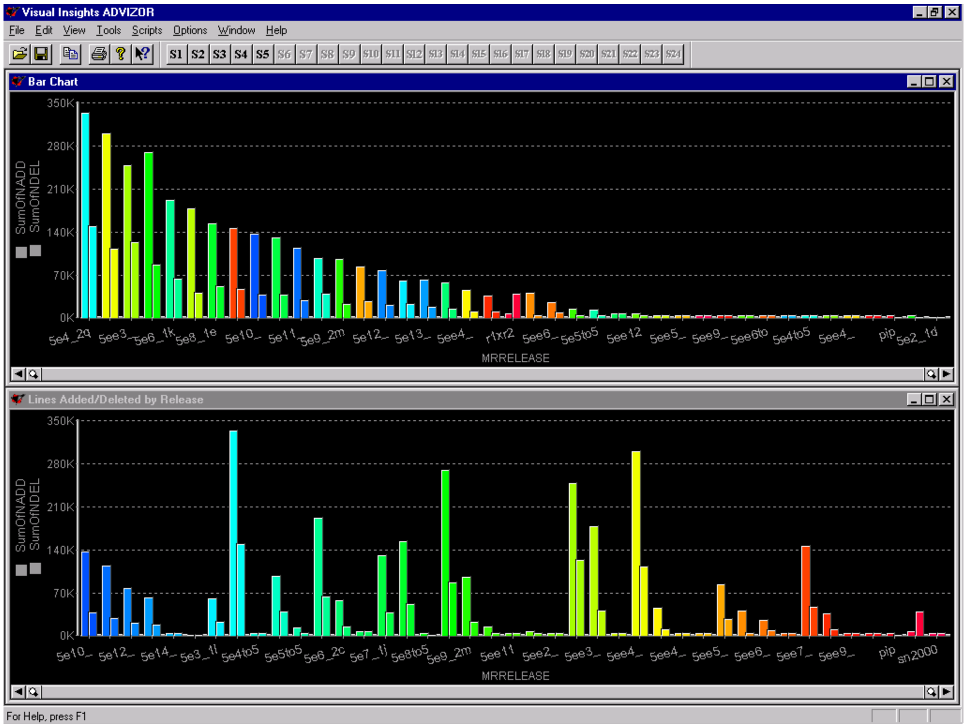


Figure 7: Size of Changes Indexed by Release. *Top:* Changes sorted by size (number of lines of code added or deleted). *Bottom:* Changes sorted by release number. Color encodes release number. Discussed in §4.3.

developers, that on the right shows the activity over software space for the most active developer.

The perspective in Figure 9 has similar goals. All views in this perspective are colored by developer, affording easy matching of corresponding data in different views. The bar charts index changes by module and developer separately, in effect showing marginal statistics from the matrix and cityscape views, and also allowing filtering by either module or developer. This allows one, for example, to identify the programmers who worked on particular modules or to determine where particular programmers worked.

The matrix view, in which rows represent developers and columns represent modules, encodes number of changes as the width of the bars in each cell, while the cityscape view encodes the number of changes as the height of the tube. The large red tube in the cityscape and the large red bar in the matrix view each represent the same 778 changes.

4.6 The Span of Changes

The *span* of a change — the number of software units involved (depending on resolution, files or modules) — was introduced in Eick, *et al.* (1998) as a measure of the increasing difficulty to change legacy software over time. There are three primary reasons to expect that changes touching more files will be more difficult to accomplish. First is the necessity to get expertise about unfamiliar files from other developers; this is especially vexing in large-scale software, where each developer has

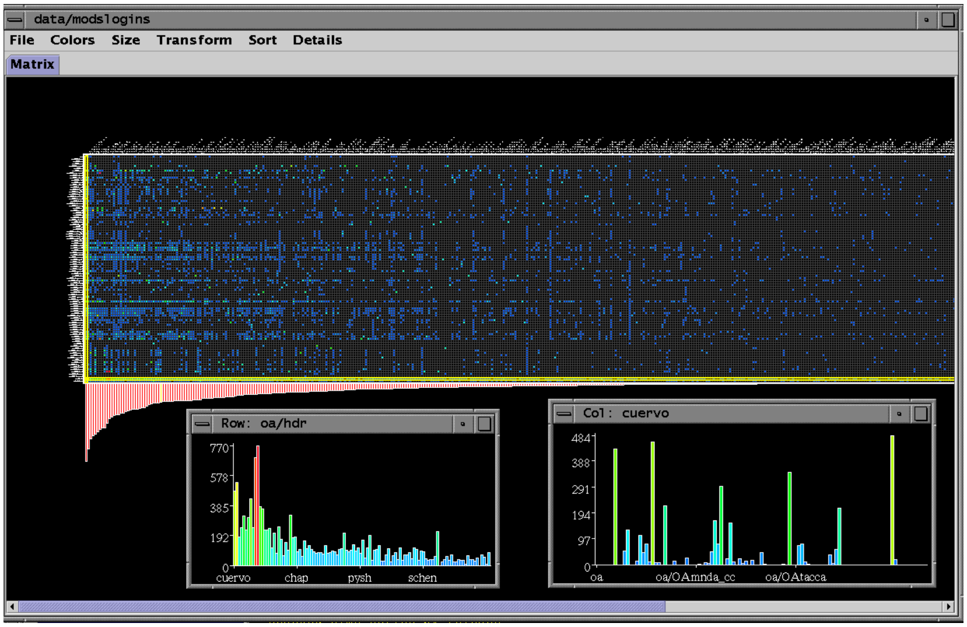


Figure 8: Number of Changes Indexed by Developer and Module. *Matrix view:* changes by developer (columns) and module (rows). *Left bar chart:* Total number of changes for selected developers. *Right bar chart:* Number of changes by module for a single developer. Discussed in §4.5.

only localized knowledge of the code. Second is the breakdown of encapsulation and modularity: changes spanning multiple files are more likely to modify interfaces. Third is the size: touching multiple files increases the size of changes.

In particular, an increase in span over time represents increasing difficulty to change the code, as well as deterioration of its original modular architecture. Figure 10 uses Data Constellations (§3.5) to display software space structured by means of IMR linkages. In this view, nodes correspond to files. The strength of the relationship between two files is a (normalized) measure of the frequency with which both are changed as part of the same IMR:

$$\text{Strength}(i, j) = \frac{N_{i,j}}{\sqrt{N_i N_j}}, \tag{1}$$

where $N_{i,j}$ is the number of IMRs under which files i and j were both changed, and N_i is the number under which i was changed.

Changes over time can be shown by animating the view in Figure 10 or with multiple views corresponding to different time periods (Eick, 1998a).

5 Management of Software Development

As evidenced by the number of late and canceled projects, software development management is not easy. In this section we present a set of perspectives illustrating the use of visualizations as part

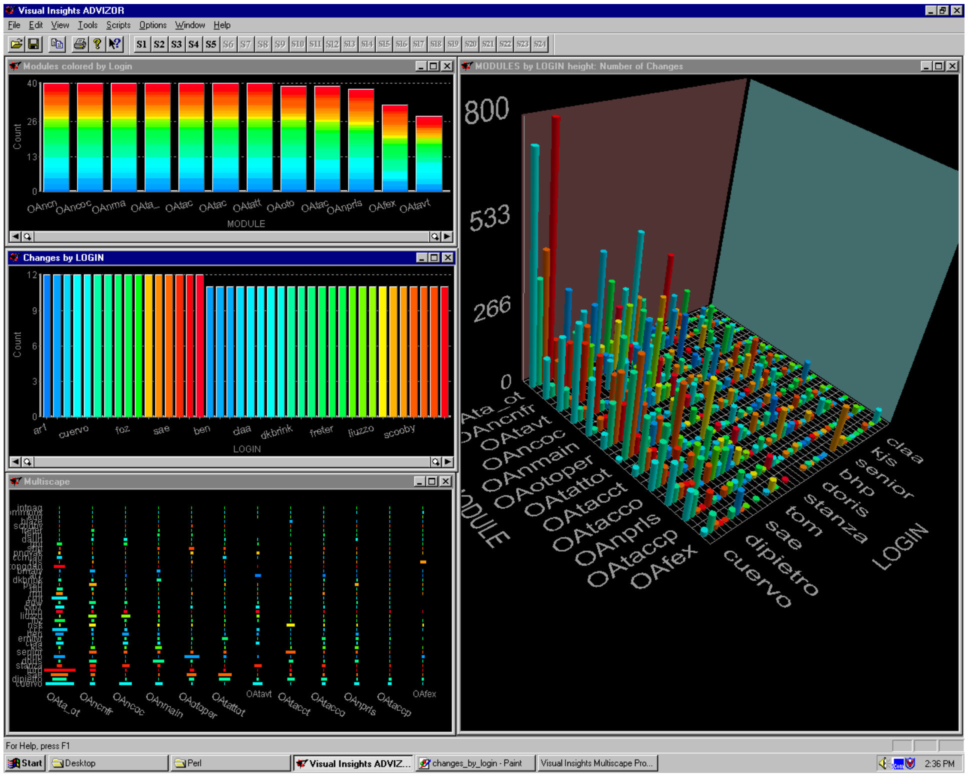


Figure 9: Number of Changes Indexed by Developer and Module. *Bar charts:* Changes by module (top) and by developer (bottom), with color encoding developer. *Matrix view* and *Cityscape view:* number of changes indexed by developer (columns) and module (rows). Discussed in §4.5.

of software development management. As compared with understanding software change (§4), this setting is characterized by need for rapid analyses that guide management decision–making and support specific courses of action. The analyses may be less detailed or less formal than in the research setting of §4, and the conclusions are primarily qualitative.

Three issues will be addressed: MR status, MR severity and developer activity. The views presented here are derived from a single project to a new software product. Taken together, they constitute a case study on the role of visualization in software development management. As noted in §2.1, the setting here is relatively small: 25 developers and a 250,000-line code base.

The perspectives presented here are highly coherent. All contain the same five views: two bar charts showing MRs indexed by initiating developer and assigned developer (for repair), two pie charts showing MRs indexed by severity and status, and a scatterplot of MRs over time. The only difference is that they are applied to different sets of data. The data in some perspectives result from a selection operation in others.

5.1 MR Status

As described in §2.2, work flows into a development organization as IMRs describing problems, and is distributed to developers as a series of MRs solving (pieces of) those problems. At the

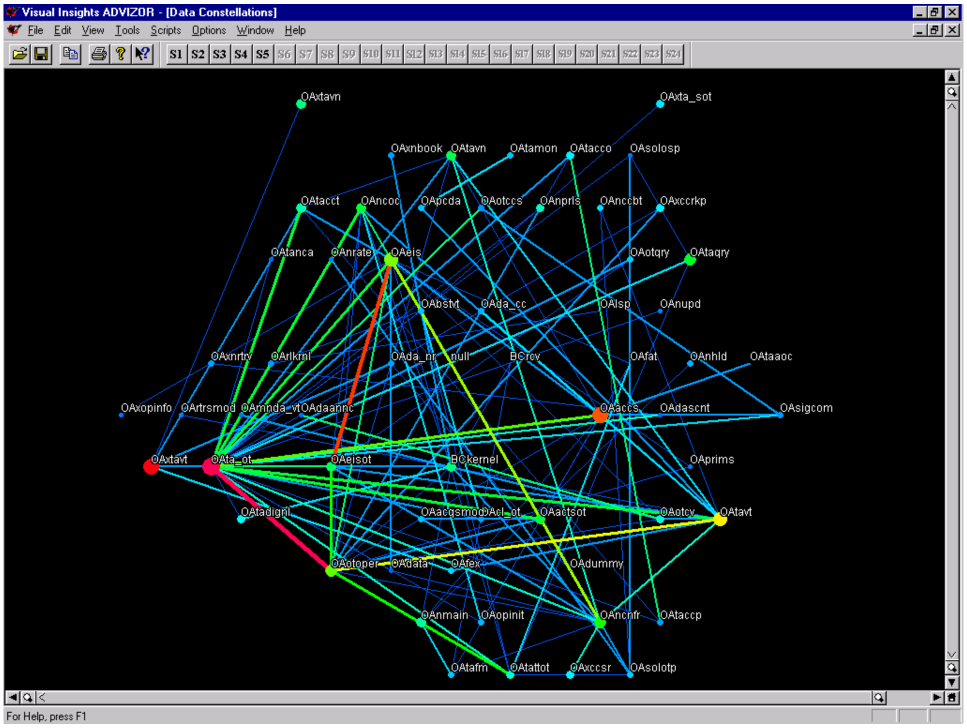


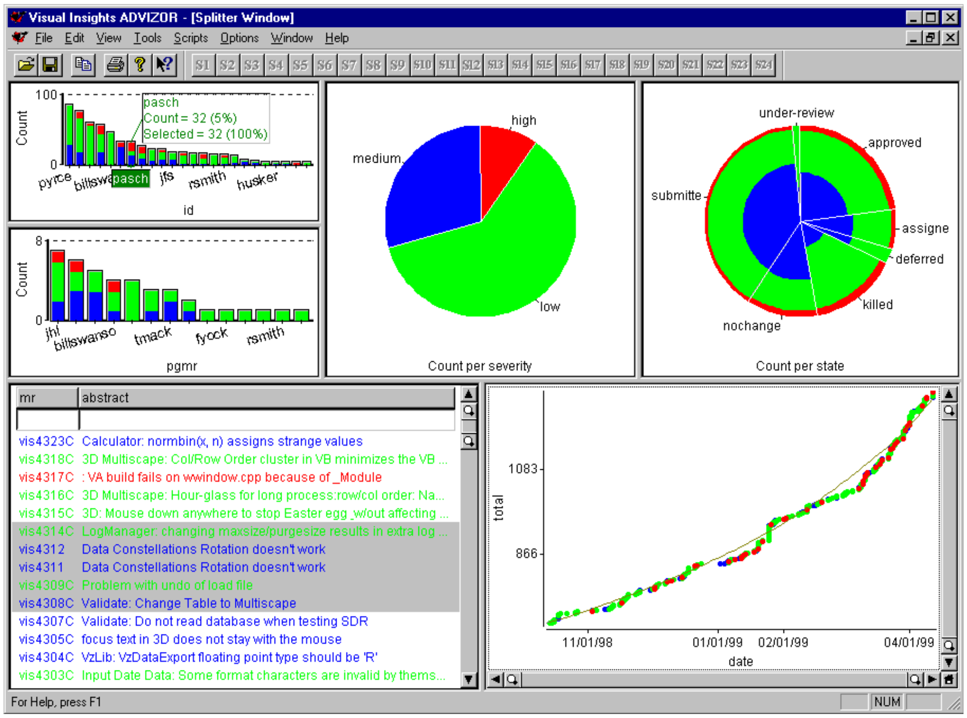
Figure 10: Relationships Between Files Based on IMR Linkages. Nodes are files, and linkage is defined by the relative frequency of files being changed as part of the same IMR. Link color shows the strength of this relationship, defined by equation (1). Discussed in §4.6.

beginning of work for a release, the majority of the MRs will be for new functionality. If work is progressing successfully, MRs should increasingly emphasize bug fixes and tiny enhancements addressing issues found by beta customers. An important challenge for development organizations is to keep track of MR activity, identify bottlenecks and find trouble areas.

The perspective shown in Figure 11 is an entry point in meeting these needs. It leads to the more focused perspectives, designed to address more specific questions presented in §5.2. The basic perspective shows MR counts indexed by *Creator* — the programmer or tester who discovered the problem and initiated the MR; the *Programmer* to whom the MR was assigned; *Severity* of the MR — `high`, `medium` or `low`; and the current *Status* of the MR — `submitted`, `under_review`, `approved`, `assigned`, `deferred`, `killed`, or `no_change`. In addition, the perspective contains a data sheet serving as a user interface to MR abstracts and a scatterplot of numbers of MRs as a function of time (augmented by a smooth fitted trend curve). As the following examples illustrate, this perspective functions both to show patterns within the MRs and as an interface to the details.

Basic Information. The *Creator* and *Assigned* bar charts in Figure 11 show who is initiating and fixing the MRs. For example, the perspective shows that three programmers currently are assigned more than five MRs, which may be cause for concern.

The *Severity* pie chart shows that approximately 1/8 of the MRs are high severity, a bit more than 1/4 are medium, and about 2/3 are low, suggesting that too many problems are accorded MR



mr	abstract
vis4323C	Calculator: normbin(x, n) assigns strange values
vis4318C	3D Multiscape: Col/Row Order cluster in VB minimizes the VB ...
vis4317C	: VA build fails on wwindow.cpp because of _Module
vis4316C	3D Multiscape: Hour-glass for long process:row/col order: Na...
vis4315C	3D: Mouse down anywhere to stop Easter egg _w/out affecting ...
vis4314C	LogManager: changing maxsize/purgesize results in extra log ...
vis4312	Data Constellations Rotation doesn't work
vis4311	Data Constellations Rotation doesn't work
vis4309C	Problem with undo of load file
vis4308C	Validate: Change Table to Multiscape
vis4307C	Validate: Do not read database when testing SDR
vis4305C	focus text in 3D does not stay with the mouse
vis4304C	VzLib: VzDataExport floating point type should be 'R'
vis4303C	Input Date Data: Some format characters are invalid by thems...

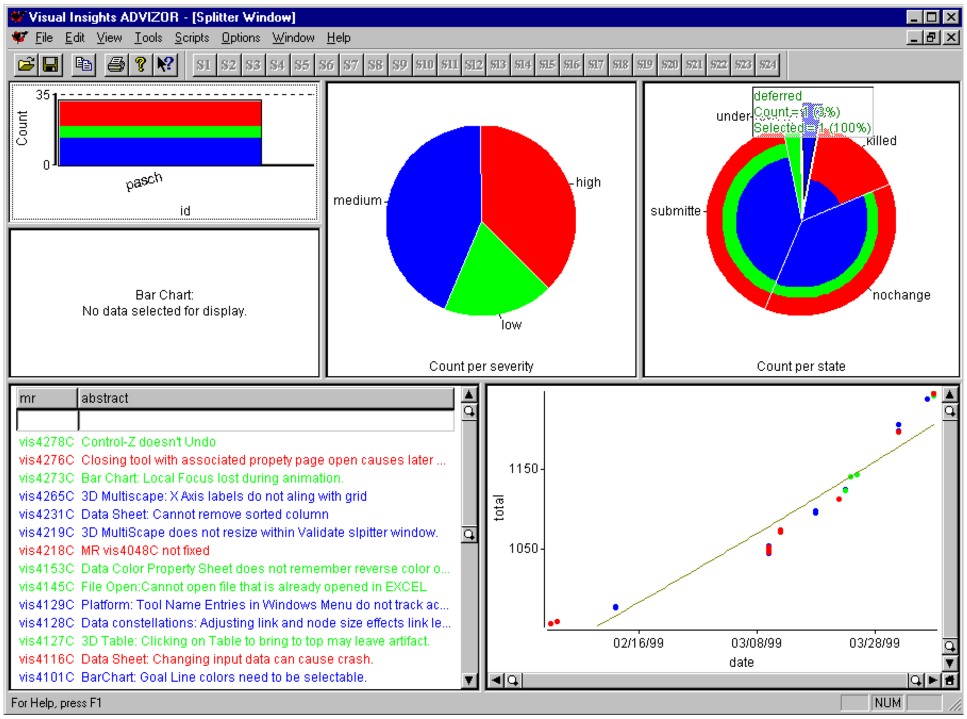
Figure 11: Perspective showing MRs Indexed in Multiple Ways. *Bar charts:* MRs indexed by submitting programmer (top) and assigned programmer (bottom). *Pie charts:* MRs indexed by severity (left) and status (right). *Data sheet:* MR abstract. *Scatterplot:* Cumulative number of MRs, indexed by time. Discussed in §5.1.

status that could have been dealt with otherwise. Possibly confirming this, the *Status* pie chart indicates that no action is taken in response to about 1/4 of the MRs — those whose status is `no_change` or `killed`. Not surprisingly, most `killed` MRs are of low severity. One positive message in the *Status* pie chart is that most MRs have been resolved (those in the `approved`, `deferred`, `killed`, `no_change` and `submitted` states).

The increasing slope (convexity) of the fitted line in the *MR Count* scatterplot indicates that new MRs are being discovered at an increasing rate. While the implications of this increase are unclear, it is unlikely to have been discovered without visualization tools.

Activities of One Developer. In Figure 11, programmer *prog* (interactively labeled with the mouse in the upper bar chart) stands out. This person initiated 32 MRs, 5% of the total, but more important (as shown the large red bar) discovered an unusually high percentage of severe MRs. From the standpoint of understanding the skills of particular programmers, this is worth investigating.

Figure 12 focuses on *prog*. It contains the same views as in Figure 11 — it was created by the selection operation shown there — but applied to only to the data for *prog*. Thus, the upper bar chart and the *Severity* pie chart contain the same information. The *Status* pie chart shows that this person has been assigned no MRs for repair. This is also reflected in the absence of the lower bar chart, which in Figure 11 displayed MRs assigned for repair. Clearly, *prog* devotes significant



mr	abstract
vis4278C	Control-Z doesn't Undo
vis4276C	Closing tool with associated propety page open causes later ...
vis4273C	Bar Chart: Local Focus lost during animation.
vis4265C	3D Multiscape: X Axis labels do not aling with grid
vis4231C	Data Sheet: Cannot remove sorted column
vis4219C	3D MultiScape does not resize within Validate slpitter window.
vis4218C	MR vis4048C not fixed
vis4153C	Data Color Property Sheet does not remember reverse color o...
vis4145C	File Open:Cannot open file that is already opened in EXCEL
vis4129C	Platform: Tool Name Entries in Windows Menu do not track ac...
vis4128C	Data constellations: Adjusting link and node size effects link le...
vis4127C	3D Table: Clicking on Table to bring to top may leave artifact.
vis4116C	Data Sheet: Changing input data can cause crash.
vis4101C	BarChart: Goal Line colors need to be selectable.

Figure 12: Perspective showing MRs Initiated by Programmer *prog*. The views are the same as in Figure 11, but applied only to data for *prog*. Discussed in §5.1.

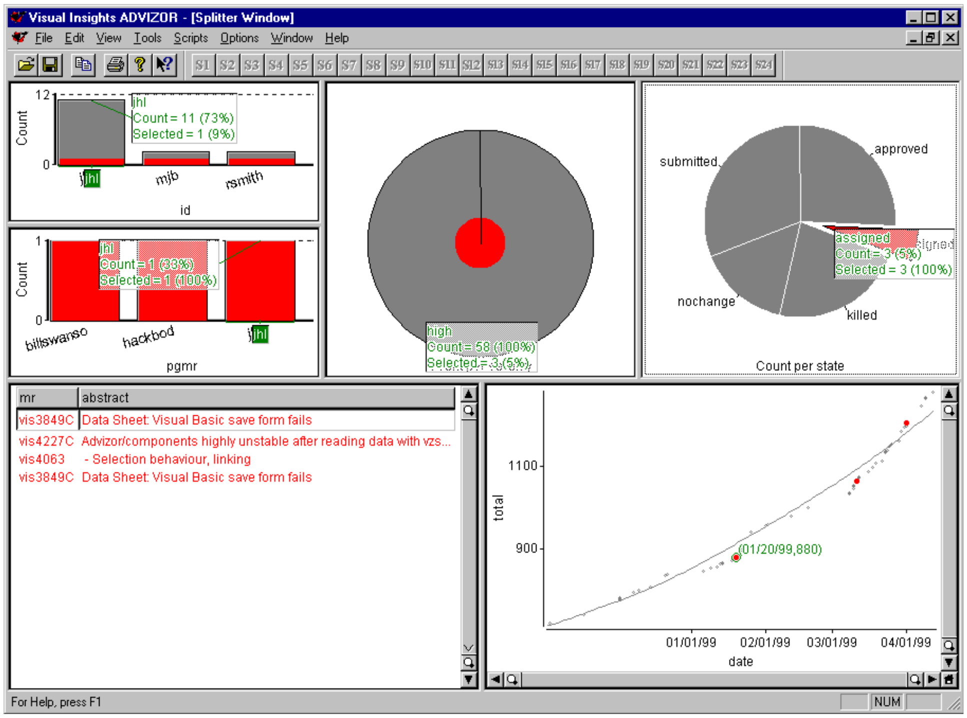
effort to testing. To the extent that *prog*'s activities may deviate from assigned duties or that *prog*'s skills as a tester were not being exploited, this information is very valuable. The same pie chart also shows that all of the MRs discovered by *prog* have been resolved, although one has been `deferred`. Using the data sheet to access the abstract for the deferred MR (not shown) allows a manager to confirm that `deferred` is the correct disposition.

The scatterplot in Figure 12 provides information about *prog*'s work patterns. Evidently *prog* did preliminary testing in early February, worked on other activities during the second half of February and first week of March, and then tested aggressively during the second and third weeks of March.

5.2 High-Severity MRs

High-severity MRs represent critical problems in need of immediate attention. Key metrics for a development organization are the number of open high severity MRs, how long they have been open, who is working on them and why they have not been resolved.

Figure 13, which is derived by filtering the perspective in Figure 11, focuses on open, high-severity MRs. It allows managers to see at once that there are only three open, high-severity MRs, assigned one each to programmers *prog1*, *prog2* and *prog3*. The oldest of these was initiated on January 20, and the most recent on April 1. (Views are based on data through April 28.) The oldest MR was both initiated by and assigned to *prog3*. This suggests that perhaps the long time to clear



mr	abstract
vis3849C	Data Sheet: Visual Basic save form fails
vis4227C	Advizor/components highly unstable after reading data with vzs...
vis4063	- Selection behaviour, linking
vis3849C	Data Sheet: Visual Basic save form fails

Figure 13: Perspective showing High-Severity, Open MRs. This filtered version of Figure 11 indicates that three are currently being worked. Discussed in §5.2.

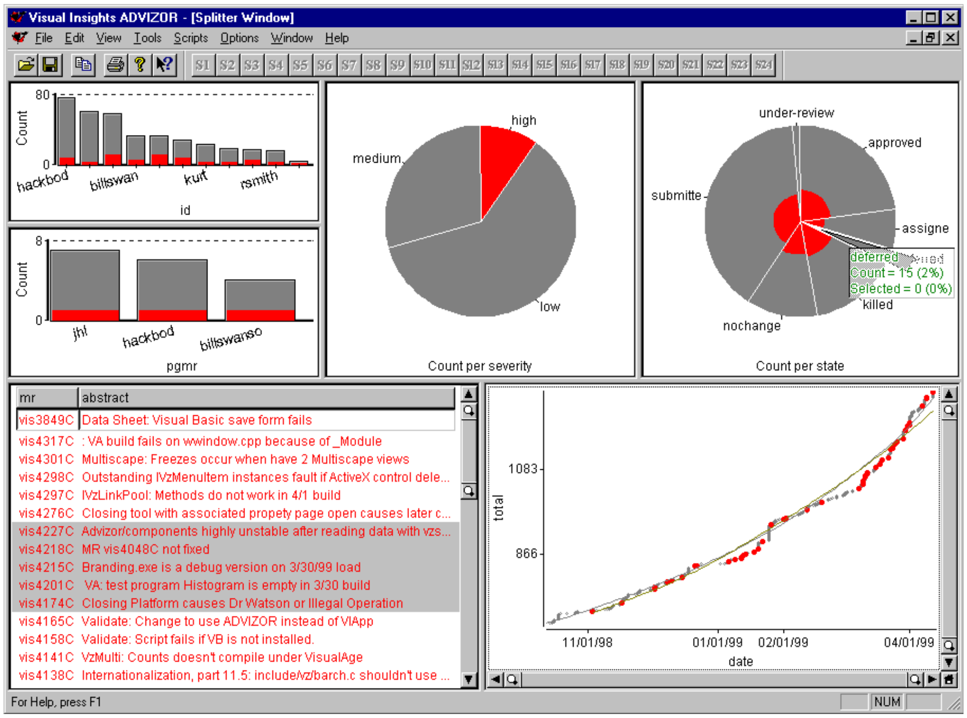
this defect results from the absence of a separate tester advocate.

Figure 14 contains the same perspective as Figure 13, but shows both open and closed severe MRs. The bar chart shows that no high-severity MRs have been deferred to the next release.

5.3 MR Quality

High numbers of `killed` and `no_change` MRs are an indication of poor MR quality. Unnecessary MRs waste the resources used to investigate them. Potential remedies include additional training for members of the testing organization.

Figure 15, derived from Figure 11 by filtering to include only system testers, investigates `killed` and `no_change` MRs. It reveals that all MRs initiated by the testers corresponding to the two leftmost bars — both new staff members — were resolved as either `killed` or `no_change`. Similarly, nearly two-thirds of MRs initiated by *prog* were `killed` or `no_change`. Ordinarily, such high rates would be cause for concern and management attention. In this case, however, further investigation indicated that these three programmers were performing self tests and identifying problems, and then assigning MRs to themselves.



deferred
Count = 15 (2%)
Selected = 0 (0%)

mr	abstract
vis3849C	Data Sheet: Visual Basic save form fails
vis4317C	: VA build fails on wwindow.cpp because of _Module
vis4301C	Multiscape: Freezes occur when have 2 Multiscape views
vis4298C	Outstanding IVzMenuItem instances fault if ActiveX control dele...
vis4297C	IVzLinkPool: Methods do not work in 4/1 build
vis4276C	Closing tool with associated propety page open causes later c...
vis4227C	Advizor/components highly unstable after reading data with vzs...
vis4218C	MR vis4048C not fixed
vis4215C	Branding.exe is a debug version on 3/30/99 load
vis4201C	VA: test program Histogram is empty in 3/30 build
vis4174C	Closing Platform causes Dr Watson or Illegal Operation
vis4165C	Validate: Change to use ADVIZOR instead of VIApp
vis4158C	Validate: Script fails if VB is not installed.
vis4141C	VzMulti: Counts doesn't compile under VisualAge
vis4138C	Internationalization, part 11.5: include/vz/barch.c shouldn't use ...

Figure 14: Perspective showing both Open and Resolved High-Severity MRs. The views are the same as in Figure 13, but data for both open and resolved severe MRs are displayed. Discussed in §5.2.

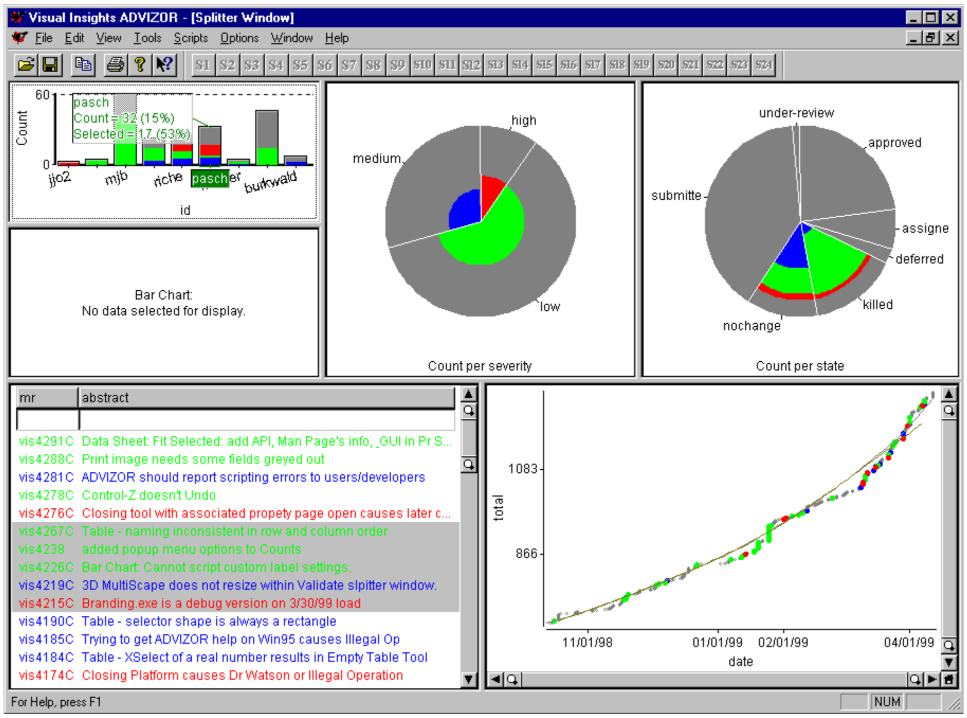
6 Discussion

6.1 Relation to Previous Research

This research builds on visualizations of software structure and previous research on software changes in three significant ways. First, it focuses on visualization perhaps the most fundamental activity in software engineering — software change. Second, the research has created a suite of perspectives and visual tools for historical analysis of software change. Finally, we have extended visualization into operational aspects of software development.

Historically, some of the most exciting examples of software visualization have involved algorithm animation. One of the earliest and most exciting examples was Baecker's videotape picturing sorting algorithms (Baecker, 1981). This work focused on helping programmers understand algorithms. Other significant early research efforts involving algorithm animations include Brown's examples (Brown, 1988) and Stasko's algorithm animation systems (Stasko, 1990). Our visualization focus, of course, is not on algorithm animation but rather on visualizing changes.

Another focus in software visualization has been on visualizing code itself. Baecker, et al. provide techniques for efficiently typesetting programs (Baecker, *et al.*, 1990). A related approach to visualizing software text and line-oriented statistics such as age, developer and release involves SeeSoft™ and its many applications (Ball & Eick, 1996; Eick, 1998b,a; Eick, *et al.*, 1997a; Eick,



pasch
Count = 32 (15%)
Selected = 17 (53%)

Bar Chart:
No data selected for display.

mr	abstract
vis4291C	Data Sheet: Fit Selected: add API, Man Page's info, _GUI in Pr S...
vis4288C	Print image needs some fields greyed out
vis4281C	ADVIZOR should report scripting errors to users/developers
vis4278C	Control-Z doesn't Undo
vis4276C	Closing tool with associated propety page open causes later c...
vis4267C	Table - naming inconsistent in row and column order
vis4238	added popup menu options to Counts
vis4226C	Bar Chart: Cannot script custom label settings.
vis4219C	3D MultiScape does not resize within Validate slpitter window.
vis4215C	Branding.exe is a debug version on 3/30/99 load
vis4190C	Table - selector shape is always a rectangle
vis4185C	Trying to get ADVIZOR help on Win95 causes Illegal Op
vis4184C	Table - XSelect of a real number results in Empty Table Tool
vis4174C	Closing Platform causes Dr Watson or Illegal Operation

For Help, press F1

Figure 15: Perspective Showing MRs Resolved as `killed` and `no_change`. Discussed in §5.3.

Steffen & Sumner, 1992). The central metaphor used by SeeSoft is literal: lines of code are presented by lines of pixels that preserve file structure, length and indentation. Other attributes are mapped onto line color. Implicitly, these visualizations can display data associated with changes (age is an example), but they are not designed to visualize changes themselves. (See Stasko, *et al.* (1998) for additional aspects of software visualization.)

Although attention to software change and maintenance is long-standing, statistical analysis and visualization of change data (and of software metrics in general) on a large scale have occurred only recently. In an early paper, Ebert proposes several visual metaphors for understanding software metrics (Ebert, 1992). In another stream of related work, the statistical analyses and models focused on *code decay* — decayed code is harder to change than it should or can be — have elucidated particular aspects of software change data (Eick, *et al.*, 1998; Graves, *et al.*, 1997b,a; Graves, 1999; Graves & Mockus, 1998; Staudenmayer, *et al.*, 1998), as well as pointed clearly to the need to visualize such data. In other examples (Eick, *et al.*, 1998; Graves, *et al.*, 1997a), the visualizations are focused not on the software data themselves, but instead of the results of statistical analyses of the data, using Data Constellations (Wills, 1999) and SiZer (Chaudhuri & Marron, 1999).

6.2 Implementation

The views and perspectives presented in this paper are implemented via Visual Insights' ADVIZOR™ (Visual Insights, 1999b), which runs under Microsoft Windows. ADVIZOR™ is neither targeted nor restricted to software change data. The capabilities of ADVIZOR™ exploited here are to create multiple individual views (matrix and cityscape views, bar and pie charts, data sheets and network views), and to link these (via mouse-based selection operations) into perspectives. Perl and Visual Basic scripts were used to extract and manipulate data on subsystems from the version management database (Mockus, *et al.*, 1999). The perspectives described in §5 are in daily operational use. The concepts and questions underlying the views are, of course, quite generic, and transfer readily to other environments.

6.3 Evaluation of Various Views

Each of the six classes of views described in §3.1–3.5 has strengths and weaknesses, which we summarize here. One clear motivation for combining views into perspectives is to exploit the differing strengths.

Matrix Views. A strength of matrix displays is that many cells are visible; also, there is no overplotting. A single image can present the information contained in thousands or tens of thousands of cells. On a high-performance, 1280 × 1024 resolution monitor, cells of 10 × 10 pixels are easily seen. In extreme cases, cells as small as a few pixels or perhaps even a single pixel may work. Pan–and–zoom capabilities needed to deal with matrix views that are too big to fit on the screen are intuitive and implemented easily. In simple cases, the view can be scrolled in the same way as a spreadsheet, but sophisticated controls are also available, which more effectively maintain *focus+context* (Card, Mackinlay & Shneiderman, 1999).

In matrix (and cityscape) views, the structure of software space is irrelevant to creating the view: because there is no natural order to modules, the columns (say) of the view can be arranged in any order. The price of this ease, however, is that these views cannot effectively relate structure to other variables.

Cityscape Views. By comparison with matrix views, cityscapes are more compelling, but at the expense of decreased scalability. Using 3D towers to encode information requires more pixels per cell than a matrix view. Occlusion is another problem: tall towers toward the front of the cityscape obscure smaller ones toward the rear. These problems can be overcome to some extent by rotating, thresholding, or using other interactive techniques, but these techniques are not as intuitive as for matrix views.

Bar and Pie Charts. These simple views seem most effective as selectors linked to other views in a perspective. As implemented in ADVIZOR™, bar charts scale effectively to indices with large numbers of values. Pie charts do not scale well under any circumstances.

Data Sheets. In isolation, data sheets are not compelling visualizations. Their strength is immediate access to details of the data, which is especially effective when they are linked to other views.

Network Views. The strength of network views is that they can reveal high-level structure that is impossible to discern by other means (because the structure is too complex or too complicated to describe). Three principal, related weaknesses are lack of scalability, inability to display multiple link characteristics and overplotting (Ware & Frank, 1996). Scalability is a consequence of the fact that accessible, comprehensible graphs require many empty pixels. Link attributes other than width and color have not yet been used effectively. (Length conveys node associations.) Network layout algorithms (Fruchterman & Reingold, 1991; Gansner, *et al.*, 1993; Munzner, 1997) attempt to locate strongly associated nodes near one another, which helps minimize link overplotting, but the problem remains severe. Techniques employing metaphors for underlying structure in the network (Lamping & Rao, 1994; Munzner, 1997; Robertson, *et al.*, 1993; Schaffer, *et al.*, 1996) are avenues for future exploration.

7 Conclusions

Numerous factors generate strong impetus to visualize software change data. Change is fundamental and inevitable in large software systems. The payoffs — intellectual and economic — from understanding change well and managing it effectively are significant. Data about software change are collected routinely, but are too massive and complex to be dealt with by formal analyses alone.

In the research reported in this paper, we have created a number of visualization concepts for software change data, and implemented them both individually and linked to form perspectives. The utility and value of these perspectives for exploring change data and for managing software development have been demonstrated using real data and real problems.

References

An, K. H., Gustafson, D. A., and Melton, A. C. (1987). A model for software maintenance. In *Proc. Conf. Software Maintenance* (Austin, TX) 57–62.

Ball, T. A., and Eick, S. G. (1996). Software visualization in the large. *IEEE Computer* **29(4)** 33–43.

Baecker, R. (1981). *Sorting Out Sorting*. Videotape.

Baecker, R. & Marcus, A. (1990). *Human Factors and Typography for More Readable Programs*. Addison–Wesley, Reading, MA.

Bertin, J. (1981). *Graphics and Graphic Information Processing*. Walter de Gruyter & Co., Berlin.

Brown, M. H. (1988). *Algorithm Animation*. MIT Press, Cambridge, MA

Card. S. K., Mackinlay, J., and Shneiderman, B. (1999). *Readings in Information Visualization*. Morgan Kauffman, San Francisco.

Chaudhuri, P., and Marron, J. S. (1999). SiZer for exploration of structures in curves. *J. Amer. Statist. Assoc.* (to appear)

Ebert, C. (1992). Visualization techniques for analyzing and evaluating software measurement. *IEEE Trans. Software Engrg.* **2(2)** 80–86.

Eick, S. G. (1994). Graphically displaying text. *J. Computational and Graphical Statist.* **3(2)** 127–142.

Eick, S. G. (1998). Maintenance of large systems. In Stasko, *et al.* (1998).

Eick, S. G. (1998). A visualization tool for Y2K. *IEEE Computer* **31(10)** 63-69.

Eick, S. G., Graves, T. L., Karr, A. F., and Mockus, A. (1997a). A Web laboratory for software data analysis. *World Wide Web* **1** 55–60.

Eick, S. G., Mockus, A., Graves, T. L., and Karr, A. F. (1997b). Web-based text visualization. In Bandilla, W., and Faulbaum, F., eds. *Softstat '97 Advances in Statistical Software* **6** 3–10.

Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., and Mockus, A. (1998). Does code decay? Assessing the evidence from change management data. *IEEE Trans. Soft. Engrg.* (to appear)

Eick, S. G., Steffen, J., and Sumner, E., Jr. (1992). SeeSoft — A tool for visualizing line-oriented software statistics. *IEEE Trans. Software Engrg.* **18(11)** 957–968.

Fruchterman, T., and Reingold, E. (1991). Graph drawing by force-directed placement. *Software – Practice and Experience* **21(11)** 1129–1164.

Gansner, E. R., Koutsojos, E., North, S. C., and Vo, K.-P. (1993). A technique for drawing directed graphs. *IEEE Trans. Software Engrg.* **19** 214–230.

Graves, T. L., Karr, A. F., and Mockus, A. (1997). Modeling software changes. In Minder, C. E., and Friedl, H., eds., *Proceedings of the 12th International Workshop on Statistical Modelling*.

Graves, T. L., Karr, A. F., Marron, J. S., and Siy, H. (1997). Predicting software faults using change history. *IEEE Trans. Soft. Engrg.* (to appear)

Graves, T. L. (1999). Finding clusters in network link strength data. Submitted to *Technometrics*.

Graves, T. L., and Mockus, A. (1998). Inferring change effort from configuration management databases. *Metrics 98*.

Graves, T. L., and Mockus, A. (1999). Identifying productivity drivers by modeling work units using partial data .Submitted to *Technometrics*.

Hill, W. C., and Hollan, J. D. (1991). Deixis and the future of visualization excellence. *IEEE Visualization Conf. Proc.* 314–320.

Inxight, Inc. (1999). Table Lens. Information available on line at `www.inxight.com/Products/Developer/AD_TL.html`.

Lamping, J., and Rao, R. (1994). Laying out and visualizing large trees using a hyperbolic space. *Proc. ACM Symp. on User Interface Software and Technology* 13–14.

Mockus, A., Eick, S. G., Graves, T. L., and Karr, A. F. (1999). On measurement and analysis of software changes. Technical Report, National Institute of Statistical Sciences.

Munzner, T. (1997). H3: Laying out large directed graphs in 3d hyperbolic space. *IEEE Inform. Visualization Conf. Proc.* 2–10.

Robertson, G. G., Card, S. K., and Mackinlay, J. D. (1993). Information visualization using 3D interactive animation. *Comm. ACM* **36(4)** 56–71.

Rochkind, M. J. (1975). The source code control system. *IEEE Trans. Software Engrg.* **1(4)** 364–370.

Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., and Roseman, M. (1996). Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Trans. Computer–Human Interaction* **3(2)** 162–188.

Stasko, J., Domingue, J., Brown, M. H., and Price, B. A. (1998). *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA.

Stasko, J. (1990). Tango: A framework and system for algorithm animation. *IEEE Computer* **23(9)** 27-39.

Staudenmayer, N. A., Graves, T. L., Marron, J. S., Mockus, A., Siy, H., Votta, L. G., and Perry, D. (1998). Adapting to a new environment: how a legacy software organization copes with volatility and change. *Proc. Fifth International Product Development Management Conf.*, Como, Italy, 1998.

Swanson, E. B. (1976). The dimensions of maintenance. *Proc. 2nd Conf. Software Engrg.* (San Francisco) 492–497.

Visual Insights (1999b). Visual Insights ADVIZOR. Information available on-line at `www.visualinsights.com/advizor`.

Ware, C., and Frank, G. (1996). Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Trans. Graphics* **15(2)** 121–140.

Wills, G. J. (1999). NicheWorks – interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*. **8(2)** 190-212.