THE Smalltalk REPORT

SIGS PUBLICATIONS

Publishers of JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, C++ REPORT, SMALLTALK REPORT, THE X JOURNAL, REPORT ON OBJECT ANALYSIS & DESIGN, OBJECTS IN EUROPE, and DIRECTORY OF OBJECT TECHNOLOGY

Table of Contents

July–August 1994 **Vol3 No9**

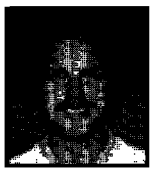
Features

Software metrics for the Smalltalk practitioner 4
William Cole
Evolutionary metrics is an important type of metrics that should be collected for all projects. Bill describes a variety of metrics that can be collected over the lifetime of a project that will provide feedback with respect to the reliability and quality of your applications.

Floating toolbox in Smalltalk/V 9
Wayne Beaton
Digitalk's Smalltalk/V product allows Smalltalkers to take advantage of many native host facilities. One tool that is missing in this library is a floating toolbox window. Wayne provides details to show you how you can implement such a feature.

Columns

Project Practicalities A brief look at size metrics 14
Mark Lorenz
Mark discusses metrics that measure the size of different aspects of a Smalltalk application class library. He covers heuristics for examining metrics for the number of instance variables and the number of methods per class.



Smalltalk Idioms Using patterns: Design 16
Kent Beck
How do you effectively make use of patterns? Like any new "tool," defining new patterns is important. How they are put into practice is what's important, and Kent discusses just how to do that.

The best of comp.lang.smalltalk New net resources 19
Alan Knight
The Internet holds a vast array of source code for many different application domains. This month, Alan provides a sampling of some of those libraries, detailing the domains for which they were developed and how you can get them.

Getting Real Responsibly designing 21
Rebecca Wirfs-Brock your objects' data
An important aspect of object design involves the choice of instance variables for a class. Often overlooked in many methodologies, Rebecca provides some detailed advice on "discovering" the instance variables for your classes.

Departments

Editors' Corner & Note from the Publisher 2
Product Announcements 24
Recruitment 27

The Smalltalk Report (ISSN# 1056-7976) is published 9 times a year, every month except for the Mar/Apr, July/Aug, and Nov/Dec combined issues. Published by SIGS Publications Inc., 71 West 23rd St., 3rd Floor, New York, NY 10010.
Subscription rates 1 year (9 issues): domestic, $79; Foreign and Canada, $114; single copy price, $8. To submit articles, please send electronic files on disk to the Editors at 509-885 Meadowlands Drive, Ottawa, Ontario K2C 3N2, Canada, or via Internet to pugh@scs.carleton.ca Preferred formats for figures are Mac or DOS EPS, TIF or GIF formats. Always send a paper copy of your manuscript, including camera-ready copies of your figures (laser output is fine).
POSTMASTER: Send address changes and subscription orders to: The Smalltalk Report, P.O. Box 2027, Langhorne, PA 19047. For service on current subscriptions call 215.785.5996. PRINTED IN THE UNITED STATES.

Editors' Corner

Welcome to the new SMALLTALK REPORT! We had promised you a new look and a new format in the Fall. As you can see we are ahead of schedule—just like a Smalltalk project normally is! With the expansion in the size of THE REPORT, we will be adding new columnists, features, and departments in coming issues. As we move to our new format, it's a good opportunity to thank the columnists who have been with us since the inception of the THE SMALLTALK REPORT. Special thanks to Rebecca Wirfs-Brock, Juanita Ewing, Kent Beck, and Alan Knight.

We have just returned from the Object Expo conference in New York City, where for the first time there was a well-attended conference track dedicated solely to Smalltalk-related issues. We got a first look at a number of new developments in the Smalltalk arena that we will review in upcoming issues. Smalltalk/X is a new highly portable implementation of Smalltalk from Tomcat/Claus Gittinger in Germany that features a compiler to generate fast native machine code and "makes possible independent class libraries, binary distribution and stand-alone applications."

ParcPlace featured the new 2.0 release of the VisualWorks environment. ParcPlace is positioning the product as a tool for generating client–server solutions and plans to go head to head against O-O 4GL's such as PowerBuilder. In an approach similar to that taken by Digitalk with their PARTS product, ParcPlace is emphasizing the "ability to create basic database applications without any Smalltalk or SQL programming" with the full power of the underlying Smalltalk engine to be used when traditional programming is required. The new release features major enhancements designed to permit the manipulation of relational data as objects.

JOHN PUGH

PAUL WHITE

The IBM booth featured its new VisualAge product while Easel demonstrated their Object Studio family of Smalltalk-based products including their new Synchrony tool. With numerous companies offering training and support services also in attendance, there was a very strong Smalltalk presence.

As Rick Friedman mentions in his note, we are beginning to plan out the program for the Smalltalk Solutions '95 conference next February. Watch out for more details in future issues.

We hope you enjoy your "new" SMALLTALK REPORT.

—The Editors

A note from the Publisher

RICHARD P. FRIEDMAN

During our three years of publication, we have observed a steady growth in the number of Smalltalk users around the world. In fact, we estimate that there are now more than 50,000 avid Smalltalk programmers worldwide and growing by 33% each year. Loyal, committed, and passionate about the language, they fervently seek to find solutions to system development questions using Smalltalk. In the past six months we have seen a spike in the number of Smalltalk-based projects being implemented and a dire need for recruiting knowledgeable and experienced Smalltalk programmers. IBM and Easel have joined Digitalk and ParcPlace in the competitive vendor foray. Clearly, the Smalltalk market is hot.

In keeping with the expanding marketplace and the need for practical information, THE SMALLTALK REPORT will be evolving into a larger publication. Each issue will present more insightful articles, source codes, experience reports, tutorials, and industry news than ever before. Also, THE SMALLTALK REPORT sports a design facelift that gives it a more colorful and easy-to-read magazine format; and it remains the one source you can depend on for savvy advice, useful tips, and thoughtful perspectives on all Smalltalk dialects.

THE REPORT will come to life when it sponsors the Smalltalk Solutions '95 conference in New York City, February, 1995. Virtually all the Smalltalk gurus and leading OO methodologists will be lecturing; the conference technical chair is SMALLTALK REPORT coeditor John Pugh. It will be an opportunity to test-drive all the Smalltalk products at once and get all your questions answered. Smalltalk has grown up, and we are pleased to give it its own magazine and conference.

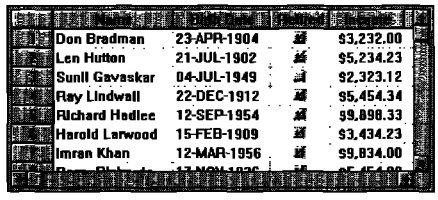
SMALLTALK SOLUTIONS 95

Powerful Spreadsheets, Now in Smalltalk/V®

WidgetKit™/Professional (WKPro) brings proven and powerful spreadsheet DLLs to Smalltalk/V. And the spreadsheet power is as easy to use as WindowBuilder™ Pro/V. WKPro consists of the FarPoint Professional DLLs, Smalltalk wrappers that integrate the controls into the Subpane hierarchy, and Smalltalk classes that allow the controls to be placed and edited interactively in WindowBuilder Pro/V. WKPro enables you to quickly build solid, powerful, reusable, and maintainable UIs for your Smalltalk/V applications.

Graphical Widgets

WKPro includes graphical controls to display pictures (BMP, PCX, & GIF) in spreadsheet cells or separately. Animation too.



	Don Bradman	23-APR-1904	$3,232.00
	Len Hutton	21-JUL-1902	$5,234.23
	Sunil Gavaskar	04-JUL-1949	$2,323.12
	Ray Lindwall	22-DEC-1912	$5,454.34
	Richard Hadlee	12-SEP-1954	$9,898.33
	Harold Larwood	15-FEB-1909	$3,434.23
	Imran Khan	12-MAR-1956	$9,834.00

High-Powered Spreadsheets

You get a spreadsheet similar to Microsoft's Excel™: formulas, drag and drop, and row/column resizing. There are 11 cell types, control of color, formatting, multiple selection, and locking. The spreadsheets have printing, load, and save capability. The functionality is factored into a hierarchy of 7 classes. Choose the one that's right for your application.

Virtual Spreadsheets Too

WKPro includes virtual spreadsheet capability that enables you to load only the visible data.

File System Widgets and More

WKPro includes DirectoryList, DriveList, FileList, and DirectoryFileList controls. You get input validation widgets for the cell types. Use them for spreadsheet cells or by themselves. UIs built with WKPro are portable to all the supported platforms.

No runtime fees for applications developed with WK/Pro. It includes complete documentation, full source, and free support to registered users for the first 90 days.

WIDGETKIT/PROFESSIONAL

NEW! **For Win $395**
NEW! **For Win32 .... $395**
For OS/2 ..... $495 *(4Q94)*

WidgetKit/Professional requires WindowBuilder Pro/V.

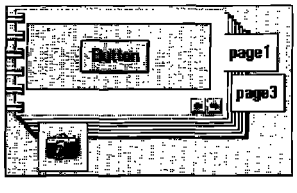
All the DLL functionality of FarPoint Professional is packaged for easy use in WindowBuilder Pro/V. WKPro is compatible with Team/V™ and ENVY®/Developer. Support subscription available.

...And CUA'91 Controls Are Easy Too!

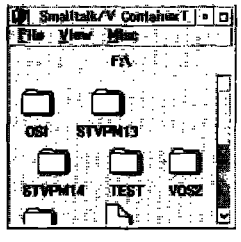
WidgetKit™/CUA'91 is a library of CUA'91 controls for Smalltalk/V. CUA'91 controls provide a distinctive and powerful user interface. WidgetKit/CUA'91 makes them easy to use and portable. Place and edit the controls interactively with WindowBuilder™ Pro/V. WidgetKit/CUA'91's specialized editors give you easy access to all of the control's attributes.

Notebooks, Cached for Performance

CachedNotebooks provide the CUA'91 notebook control. Performance is dramatically improved by dynamic page loading. You get complete control of orientation, tabs, alignment, color, binding, and caching.



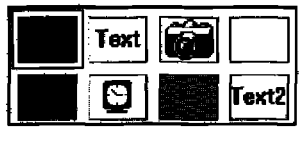
Containers



CuaContainers provide text or icon representations of items they contain. Items can be dragged and dropped between containers. Supports icon, name, text, tree, and detail views. CuaContainers can hold objects of any type.

Value Set and More

CuaValueSet provides a way for users to select from icon and text choices with a mouse click. WidgetKit/CUA'91 also provides full support for the rest of the CUA'91 controls, including slider and spin button.



For WindowBuilder Pro/V

WindowBuilder Pro/V lets you build Smalltalk/V user interfaces fast. Place the controls and edit them interactively. Increase consistency, ease maintenance. Call for a free brochure.

No Runtime Fees

No runtime fees for applications developed with WidgetKit/CUA'91. It includes complete documentation, full source, free support to registered users for the first 90 days, and a 30-day money-back guarantee.

WIDGETKIT/CUA'91

NEW! **For OS/2 ..... $295**
For Win ..... $295 *(3Q94)*
For Win32 ..... $295 *(3Q94)*

WidgetKit/CUA'91 requires WindowBuilder Pro/V. WidgetKit/CUA'91 is compatible with Team/V and ENVY/Developer. Includes DLLs. User interfaces built using WidgetKit/CUA'91 are portable to supported platforms. Support subscription available.

SHARE OBJECT SYSTEMS INC.

Objectshare Systems, Inc.
5 Town & Country Village, Suite 735
San Jose, CA 95128-2026
Fax 408-970-7282
CompuServe 76436,1063

Call to order today (408) 970-7280

9 AM to 5 PM PST, Monday through Friday
30-day money-back guarantee

Software metrics for the Smalltalk practitioner

William Cole

It has been accepted that defect prevention is a more optimal approach to software quality than defect correction. However, many development efforts still rely on defect detection as the primary method for ensuring software quality. This is due not so much to a lack of concern for quality as to the pragmatic reality of software project mortality rates. In many organizations, the success rate of projects started versus completed is sometimes less than 30%. Given this level of mortality in software projects, it is not surprising that project managers are unwilling to invest resources for up-front reliability and quality engineering efforts. In many cases resources for quality efforts are not invested until the project has survived to the test and integration phase. Unfortunately, this is too far into the development cycle for defect prevention techniques to be of any value.

This is why the majority of software quality assurance (QA) efforts drive system reliability to acceptable levels using traditional defect detection techniques (unit and integration testing) techniques. Not surprisingly, defect detection QA approaches for high-reliability systems often demand more resources than those expended in the design and coding phases of development.

The Tom Gilb approach to process measurement says anything you measure in the development process is valuable. However, the practical reality of running software metrics programs leads this author to a more conservative stance. Even with long-standing software metrics (Halstead, McCabe) that have been in use for several years, there is still a high level of interpretation based on the language implementation, the type of application, etc. The logistical aspects of software measurement are also an issue, given the analysis of large data sets along with the need to provide timely feedback.

The point here is that proposing metrics is the easy part—but validating and interpreting measures into meaningful information that improve the cost efficiency of software development is much more difficult. Therefore, in proposing a series of software measures, we asked ourselves two basic questions:

- What are the concrete goals (in dollar terms) of what you're trying to achieve?
- Are the metrics proposed easily gathered, analyzed and verifiable without too much subjective interpretation?

Our objective here is to present a series of simple software measures to achieve two results: 1) Determine the regions of code which are most volatile, allowing for intelligent project management decisions on where to invest (often limited) QA resources, and 2) Present a straightforward method for capturing and analyzing the measures outlined in this article. We'll call these measures Change Metrics.

The following section presents an introduction to Change Metrics. The third section outlines further interpretations of Change Metrics. The fourth section shows how to implement these metrics within the framework of the Smalltalk language environment. The final section presents conclusions on the material covered in this article.

CHANGE METRICS

There are two basic change measures for OO systems: 1) structural changes to a class and 2) behavior changes (to methods within a class). Within each series, both complexity and productivity measures can be derived. The approach herein takes a larger-grained view than this, focusing instead simply on the version history and number of methods associated with each class version, or:

1. number of versions for a class
2. number of new methods for each class version.

Each of these basic Change Metrics can be viewed from two different perspectives; the distribution of changes within a software system (which are the "hot" classes), and who is doing the changes, or:

1. Distribution of versions across all classes within an application.
2. Changes per developer.

The underlying premise with these measures is that the fewer Versions associated with a class design, the more stable and robust it will be. This is only a general observation. The following will describe each series of Change Metrics in more detail. Generalized trend interpretations are also provided with each measure.

Number of class versions

This relates to the number of versions made to a class. General assumptions derived from this measure:

1. Plotted along a time axis of (t), if the number of class versions increases as the project nears completion, it indicates that the design is unstable.
2. Plotted along a time axis of (t), if this number remains stable or decreases as the project completes, it is a sign that the class design is stable.

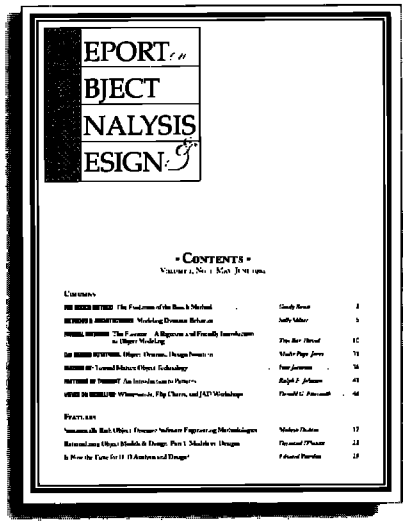
Number of methods for a class

This measure relates to the number of methods associated with each Class Version. General assumptions derived from this measure:

1. An initial growth phase, followed by a stable (or gradually increasing) number of methods per Class is indicative of a robust class design.

New from SIGS!

A SIGS Publication



Report on Object Analysis & Design

Your first step for planning and building object–based software systems.

The *Report on Object Analysis & Design* (ROAD) is a new bi–monthly (6x), advertising–free journal, which focuses on language–independent, architectural concerns about object–oriented analysis, design and modeling. Each issue provides you with in-depth articles addressing the complex questions related to the system architecture prior to when language issues are addressed, including...

- **the fundamental issues related to object modeling**
- **notational schemes for representing A&D models**
- **the processes for performing OOA or OOD**
- **revisions and updates of various design methods**
- **comprehensive comparisons of OOA&D approaches**
- **specifications on which method to use, and when**
- **expert reports on the tools currently available**

And much more.

REPORT on OBJECT ANALYSIS & DESIGN

For software developers and project leaders either currently working on an OT project, or moving toward that goal.

Platform and system independent, ROAD is written for all levels of project complexity.

✂ clip and mail or fax

Yes—Enter my subscription to ROAD at the rate marked below:

- ❑ **Personal**
  - ❑ **1 Year (6 issues) $99**
  - ❑ **2 Year (12 issues) $158 Save 20%!**
- ❑ **Corporate/Library**
  - ❑ **1 Year (6 issues) $199**
  - ❑ **2 Years (12 issues) $358 Save 10%!**

METHOD OF PAYMENT

- ❑ Check enclosed, payable to ROAD (in U.S. dollars, drawn on a U.S. bank)
- ❑ Charge my
  - ❑ Visa ❑ MasterCard ❑ AmEx

ACCT# _____

EXP DATE _____

SIGNATURE _____

SAVE UP TO 20% WITH THIS OFFER!

Name _____

Title _____

Address _____

City _____

State/Province _____

Zip/Postal Code _____

Country _____

P: _____ F: _____

RETURN TO
ROAD, P.O. Box 2027,
Lanchester, PA 17342-2027

To start your subscription to ROAD, mail or fax this coupon today!

2. A great number of changes to methods between successive versions of a class (or wildly fluctuating numbers) points to immature or incomplete design. This can also point to incomplete or poorly understood requirements.

For instance, Figure 1 shows an example graph of the number of methods for a class between successive class versions from a sampled development effort.

Developer counts

Developer counts represent the view of class Ownership from the perspective of the entire system. These counts can be correlated with the above counts to determine:

1. Number of class versions per developer.
2. Number of new methods created per class per developer.

There are many was to correlate developer counts at both the individual Class level and at a synthesis (multiple class) level(s). For instance, Figure 2 shows an example graph of the number classes owned per a group of developers in a measured project;

Developer counts are measures which profile a developers impact and productivity on the system. Traditionally, there has been both resistance and controversy surrounding these types of measures. Personal productivity metrics have complex interdependencies within the context of an organizations sociology. Careful consideration is required in implementing these types of individual productivity measures.

Distribution of changes

An important synthesis of the related metrics in this series, it provides a "change" complexity measure of Classes, based on the number of changes distributed across the entire system within a time domain. See Figure 3 for an example visualization of this metric.

General assumptions derived from this measure:

1. High numbers of changes to source code units are indicative of:
   - low requirements stability/completeness,
   - poor design quality within a source code unit,
   - excessively tight "coupling" of source code units,
   - resulting in a domino effect of change propagation.
2. Clustering of changes within subsystems can indicate present and future reliability and maintainability problems. This allows for the intuitive modeling of future errors using the Change Prone Module equation.[1]

INTERPRETATION

As with most software metrics, interpretation must be limited to

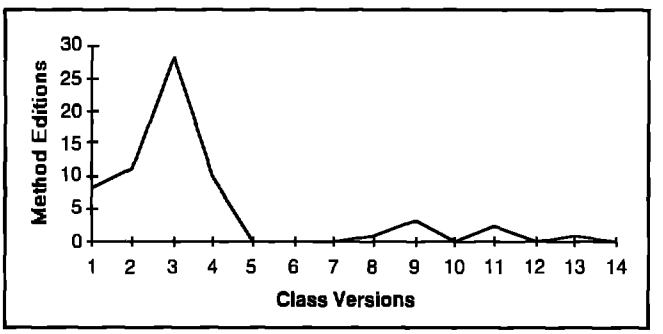
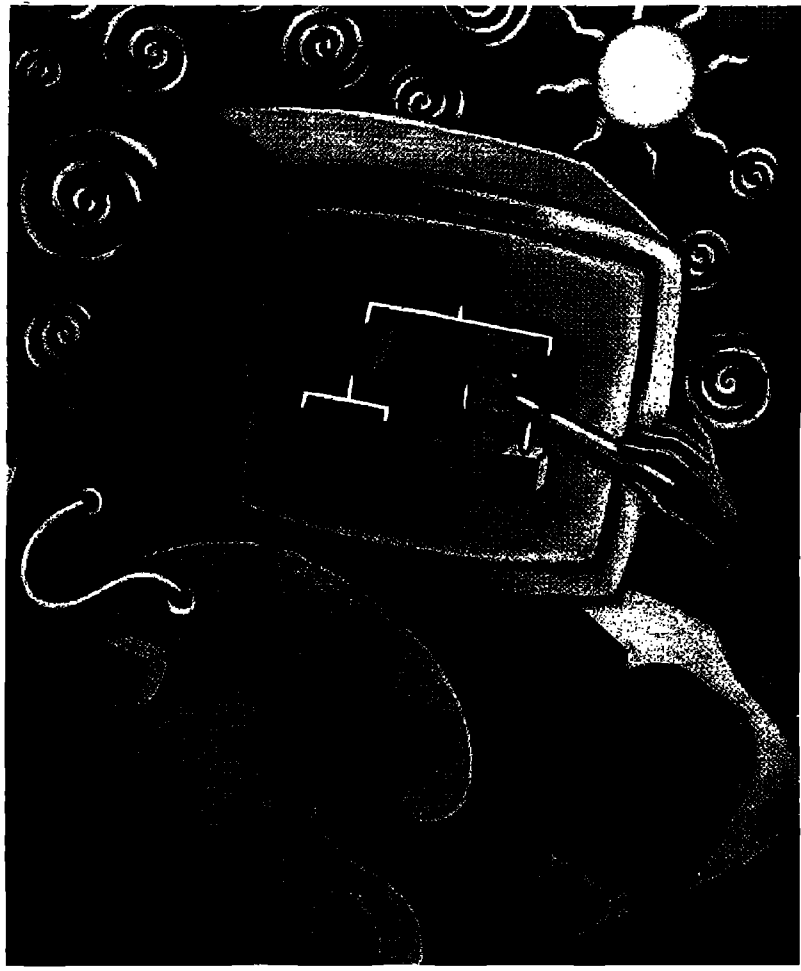


Figure 1. Number of methods/class.



Object EXPO europe

Europe's Largest OOP Conference and Exhibition

"Mastering the Art & Science of Object Technology"

Now in its fifth successful year, **Object Expo Europe** is still the largest OOP conference and exhibition in Europe. More than just sessions and exhibits, **Object Expo Europe** is a festival of object-related events and people.

Be a part of this once-a-year event that will bring the entire European OOP community to London's premiere venue, the Queen Elizabeth II Conference Centre in Westminster. Mark your calendars now, and send for more information today!

What's special about Object Expo Europe '94:

An all new technical programme, for developers to managers!

Choose from over 50 in-depth lectures focusing on new aspects of OOP, including analysis and design, distributed object technology, standards, languages and environments, ODBMS, and project management strategies—the class topics you're most interested in!

More of the speakers you've asked for!

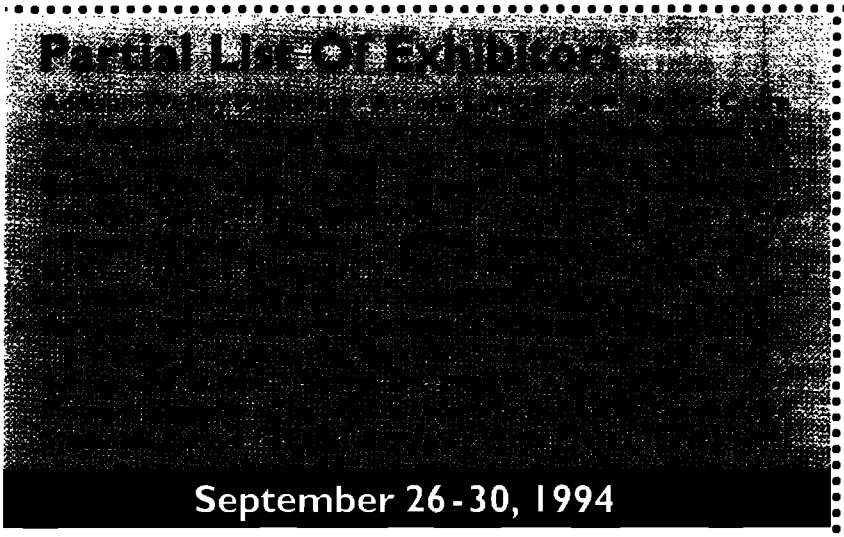
Learn from industry leaders on how to maximise object technology for your company.

All of the newest object technology products on display

Take advantage of this once-a-year event to see and try out an entire spectrum of object technology products.

Exchange ideas and discuss industry advancements

Enjoy Keynote speeches, Birds-Of-A-Feather Roundtable discussions, Product Education Sessions, User Group Meetings, Book Signings, Expert Walk-In Clinics, Welcome Receptions, and much more!



Queen Elizabeth II Conference Ctr., London

Sponsored by

OBJECT magazine · Journal of Object-Oriented programming · C++ Report · computing · KPMG

Presented by

SIGS CONFERENCES

YES!

Please send me more information on Object Expo Europe '94

☐ Free Exhibits Pass ☐ Attending ☐ Exhibiting ☐ Executive Symposium

Name _____

Title _____

Company _____

Address _____

_____

City _____ Province/State _____

Country _____ Zip/Postal Code _____

Day Phone _____

Fax _____

Mail or Fax to: Object Expo Europe '94

Brocus House, Parkgate Road, Newdigate
Surrey, RH5 5AH, UK

voice x 44 (0) 306 631331
fax x 44 (0) 306 631696

OEES7

generalized trend indicators. This is due to the wide range of potential applications and language implementations. Analysis of previous change measure data shows both implicit and explicit relationships to system software quality attributes.[2] Relationships to system quality are dependent on whether there is a positive (high count) or negative (low count) trend associated with changes to a software system.[1] Note also that the terms "increased," "stable," and "decreased" in this case are relative to the average mean number of changes occurring across all classes within a system.

An important assumption needs to be addressed here concerning the use of ENVY/Developer. Ordinarily, projects tend to start out with long intervals between Class versions. As things progress and the project gets closer to completion, the interval between Class versions tends to decrease significantly. It is important to bear this in mind when interpreting the counts for number of versions per Class.

More than 75 KLOC of Smalltalk code was analyzed during the course of validating our hypothesis. All of the assertions in the third section were validated to some degree. We are intentionally vague on the validation of these measures as our sample of code was limited to this single application. Also, our motive was to use these measures in the most general sense; we sought to avoid complex analysis and by extension, complex interpretation.

IMPLEMENTATION

It is understood that many of the implementation details outlined in this Section are at best rudimentary. They are only intended to provide a general implementation framework for establishing a change metrics collection program within the Smalltalk environment. Further details of implementing a change metrics collection program, along with analysis, report generation, and visualization of metrics data collected will be the subject of a follow-on paper.

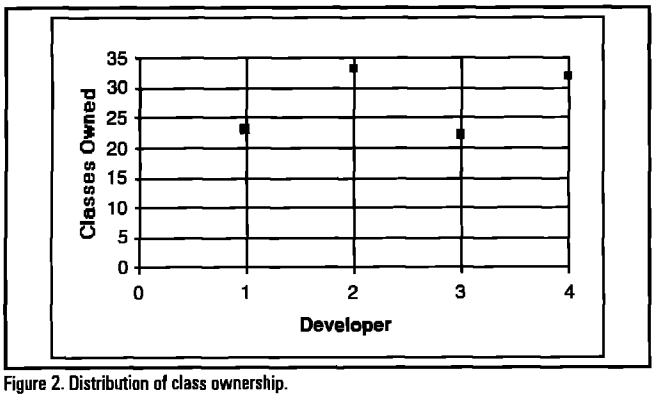


Figure 2. Distribution of class ownership.

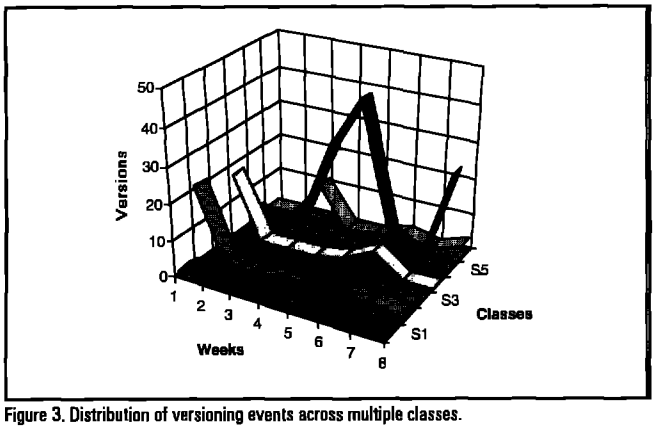


Figure 3. Distribution of versioning events across multiple classes.

The ENVY/Developer team programming tool from Object Technology makes it possible to quickly collect change metrics from an ENVY/Developer repository. In ENVY all method and class editions are marked with a unique timestamp. The repository can be queried for all available editions of a particular class, or all method editions for a particular method, and their timestamps compared.

In this way it is easy to determine several possible metrics about the rate of change in an application including:

1. The total number of editions of a method or class over its lifetime.
2. The number of method changes whose timestamp falls between any two class versioning events.
3. The number of method and class editions whose timestamp falls between any two arbitrary units of time.

In addition, ENVY/Developer provides a record of the class owners of all classes in a repository. It is simple to query all classes in the repository for their owners and assemble a dictionary relating users to the classes they own.

CONCLUSION

As mentioned earlier, several measures of stability, complexity and productivity of a software development effort can be derived from these Change Metrics. Possibilities also exist for using these measures to refine existing birth–death reliability modeling techniques.[3]

No reference is needed to the application domain of the sampled software system. The intent has been is to outline an approach general enough for multiple implementations. An effort has also been made to abstract the key concepts to allow application of these measures to a wide range of language technologies.

It is important to note here that Change Metrics are not a defect prevention technique. In many ways it can be viewed as a large-grained defect detection strategy that identifies groups or subsystems of classes that are more defect prone than others. The efficiency and quality of a software development effort can thus be enhanced by qualifying those areas where testing efforts can be best applied.

Another key benefit of Change Metrics is that they are a back-end quality activity. In most organizations, obtaining resources for quality initiatives is easier in the test and integration stage of development when visibility of a project escalates. Change metrics also allow for more timely feedback on a software system's level of quality, being simpler to gather and to interpret than many traditional SQ measures.

However, if one tries to divine more detailed information from Change Metrics, the results will be less effective as the measurements become more finely grained. Again, this is a large-grained detection approach to software quality.

Those interested in obtaining the Smalltalk code used to capture Change Metrics from an Envy/Developer repository can

continued on page 13

Floating toolbox in Smalltalk/V

Wayne Beaton

Many applications written for Windows use windows that float above the workspace in place of tool boxes fixed to the workspace. This technique has the advantage of being completely free-form, allowing users to organize their work spaces any way they like.

Microsoft has published THE WINDOWS INTERFACE: AN APPLICATION DESIGN GUIDE, which details how user interfaces should appear and function in Windows. Curiously, the text does not mention floating windows, but the Visual Design Guide, software provided with the book, does.

Typically, floating windows have a mini caption bar to distinguish them from regular windows. According to Microsoft, the caption bar should be nine pixels in height and the system box, in the left top corner, should be twelve pixels in width. Figure 1 shows an example of a floating window that could be provided in a window building application.

Windows does provide support for floating windows. However, it provides no support for mini caption bars. The only programming mention that I have been able to find is in an obscure piece of code provided as an example for Microsoft Visual C++ that creates a captionless window and draws everything itself. Fortunately, the same can be accomplished in Smalltalk/V Windows.

The combination of Windows and Smalltalk/V Windows permits the programmer to draw on a TopPane just as easily as any other kind of pane. Windows even tells us when, where and what to draw through its event mechanism.

The class FloatingTopPane has been created, as a subclass of TopPane, to support floating windows. The name FloatingTopPane is perhaps incorrect, in that its instances do not have to float. I have selected this name to keep clear the purpose of the pane—it has been designed to float, and any other use would be a violation of interface guidelines.

The class has been designed in such a way that it can be used in place of TopPane using the same public interface. Child panes can be added using the addSubpane: method; the children use framing blocks the same way as they do with a TopPane.

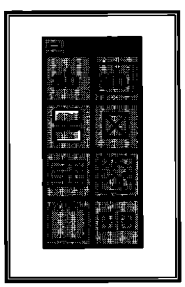


Figure 1. A floating window containing icons.

```
openAbove: anApplicationWindow
    "Open myself as a floating window above
     anApplicationWindow."
    self
        addView:
            (FloatingTopPane new
                owner: self;
                parent: anApplicationWindow;
                when: #opened perform: #opened:;
                    yourself);
        openWindow
```

Figure 2. Example code: How to create a window that "floats" above another.

A floating window is just a child window that remains on top of its parent window. To make a FloatingTopPane float is a simple matter of specifying a parent window at creation time. This is accomplished in Smalltalk/V by setting the parent of the floating window to the parent window. The open method for a ViewManager subclass might look like Figure 2.

When a window is created, a style is passed in to determine the appearance of the window. The style is an integer, which presumably means something to the Windows' API. This integer is answered from the method defaultFrameStyle in ApplicationWindow. FloatingTopPane overrides this method and answers the styles WsPopup and WsBorder. The combination of these styles creates a window with a border and no caption bar or sizing frame.

Due to a bug in V, whether it is specified or not, a window will have a caption bar. The fix for this bug, the method **ApplicationWindow>>wmSize:with:** as provided by Digitalk in a maintenance upgrade has been included with the code.

Through a strange sequence of events, an instance of FloatingTopPane is asked to draw itself when it receives a message to display. The display method simply draws the caption bar in the correct color (depending on whether the window is activated or not) and then draws the system box.

Getting the children to draw themselves uses existing code—no changes were required to facilitate this. However, in order to get children to size themselves correctly, the method **ApplicationWindow>>resize:** had to be overridden to take into account the caption bar that Windows is no longer responsible for drawing. The new version simply takes the rectangle parameter and adjusts it to leave room for the caption bar.

Getting the window to respond to such activites as dragging is another matter. Windows doesn't think that the caption bar is there, so there is no place to grab in order to drag the window to a new location. To facilitate dragging, it is the programmer's responsibility to understand the mouse click, provide a drag outline and to perform the actual move. Further , if the mouse is clicked in the system box, then the system menu should appear.

When the left button is clicked with the mouse in the confines of the window (but not in the confines of a child window), the window is sent the button1Down: message with the point at which the click occured in the window's local coordinates. Similarly, the window will receive the messages button1Move: and button1Up: when the mouse is moved or released with the left button pressed, respectively.

The method **FloatingTopPane>>button1Down:** (shown in Fig. 3) first checks to see if the button was clicked with the mouse inside the system box.

This can easily be determined by checking to see if the rectangle outlining the system box contains the point of the click. If so, the message doSystemMenu is sent to perform any system box activity. Otherwise, if the mouse click occured inside the rectangle outlining the caption bar, then the window is prepared to be dragged.

Since Windows will not perform any of the dragging functions for us automatically, it must be simulated. When a window is dragged, an outline, showing the area the window would occupy should the button be released, is moved about the screen maintaining a relative offset from the current mouse point. To do this, the mouse points, in screen coordinates, must be maintained.

The point of the mouse click is given in coordinates local to the window. These coordinates are converted into screen coordinates and remembered in instance variables. The variable startPoint holds the point where the mouse was first clicked; lastPoint holds the position of the most recent mouse event.

The difference between the last point and the start point is used to determine the current position of the drag rectangle and ultimately, the new location of the window. Figure 4 shows the method that computes the drag rectangle; the frame rectangle of the receiver, which is the rectangle defining the screen area occupied by the window, is translated by the difference of the last point and the start point. The drag rectangle is drawn by XORing this rectangle on the screen.

```
button1Down: aPoint
    "Private - The left button is down at aPoint. If
    aPoint is in the system box, then pop up a menu.
    If aPoint is in the caption bar, then prepare to drag."
    | screenPoint |
    (self systemBoxRectangle containsPoint: aPoint)
        ifTrue: [self doSystemMenu]
        ifFalse: [
            (self captionBarRectangle containsPoint: aPoint)
                    ifTrue: [
                        screenPoint := aPoint
                            mapClientToScreen: self.
                        self
                            captureMouseInput;
                            startPoint: screenPoint;
                            lastPoint: screenPoint;
                            displayReverseDragOutline]
                    ifFalse: [super button1Down: aPoint]]
```

Figure 3. FloatingTopPane>>button1Down.

```
outlineRectangle
    "Private - Answer a rectangle defining my
     dragged frame."
    ^self frameRectangle
        translateBy: (self lastPoint - self startPoint)
```

Figure 4. FloatingTopPane>>outlineRectangle.

```
activeCaptionColor
    "Answer the color to use for an active caption
     (title bar)."
    ^UserLibrary getSysColor: ColorActivecaption
```

Figure 5. The method ColorManager class>>activeCaptionColor.

To ensure that all subsequent mouse events are received, the message **Window>>captureMouseInput** is sent. Normally, mouse events are sent to the window containing the cursor. Calling this message will ensure that all future mouse events are sent to the receiver.

When the message button1Move: is received, the drag outline is hidden, the new position of the mouse is remembered, and the drag outline is then redisplayed in the new position.

The method **FloatingTopPane>>button1Up:** hides the drag outline, remembers the new position of the mouse, and moves the window to the new location by calling the Windows API SetWindowPos. The message clearMouseCapture releases the capture of mouse events exclusively for the receiver.

I have not implemented the use of the system menu, partially because my initial effort didn't work quite right, and partially because I didn't need it that badly. I have provided a hook for it in the method **FloatingTopPane>>doSystemMenu** that can be included by someone with more enthusiasm for such a creature than myself.

The ColorManager

The display methods in the class FloatingTopPane make reference to the class ColorManager, which is a class implemented to handle colors.

Through the control panel, the user has complete control over the colors that windows will use. These colors are not represented in ColorConstants, but they can be determined by calling standard APIs. The ColorManager class implements such calls.

I have included the subset of the methods implemented in ColorManager used by the FloatingTopPane class. Implementing the balance of the methods should be a simple matter.

The method **ColorManager class>>activeCaptionColor**, shown in Figure 5, answers the color to be used by an active caption bar.

It calls the User API getSysColor. Similarly, the method **ColorManager class>inactiveCaptionColor** answers the color of an inactive caption bar. The method **ColorManager class>>black** answers the value for ClrBlack in the pool dictionary ColorConstants.

CONCLUSION

I have provided my implementation of floating windows in the class FloatingTopPane, a class created in order to employ floating windows with minicaption bars.

The implementation is simple but incomplete; I have chosen not to provide the system menu, nor have I provided for any keyboard movement.

When creating objects like the floating top pane, it is important to consider look and feel guidelines like those published by Microsoft for Windows. Using these guidelines will ensure that applications will appear and function similar to other applications and provide a more comfortable environment for the user. ♀

Wayne Beaton is a senior member of the development team at The Object People. He can be reached at The Object People in Ottawa, Canada, at 613.225.8812, or by email at wayne@ObjectPeople.on.ca.

Listing 1. FloatingTopPane class.

```
TopPane subclass: #FloatingTopPane
instanceVariableNames:
    'startPoint lastPoint '
classVariableNames: ''
poolDictionaries:
    'WinConstants '
category: 'Windows-Mini Caption Bar'!

!FloatingTopPane class methodsFor: 'examples' !

example1
    "Open an empty floating window over the Transcript."
    self new
        parent: (Transcript views at: 1);
        open! !

!FloatingTopPane methodsFor: 'activating' !

activate
    "Private - I have been activated. Redisplay my caption bar."
    self invalidateRect: self captionBarRectangle.
    ^super activate!

deactivate
    "Private - I have been deactivated. Redisplay my caption bar."
    self invalidateRect: self captionBarRectangle.
    ^super deactivate! !

!FloatingTopPane methodsFor: 'initializing' !

initGraphics
    "Private - Initialize the graphics tool for the receiver.  Borrowed from
     the super class, except that I need a Pen, not a TextTool."
    handle = NullHandle ifTrue: [^nil].
    graphicsTool := Pen forDC: nil medium: self.
    "Can't use foreColor: or backColor: because it sets the graphicsTool's
     foreColor & backColor of the window's childrens"
    self propertyAt: #foreColor put: self foreColor.
    self propertyAt: #backColor put: self backColor.! !

!FloatingTopPane methodsFor: 'events-buttons' !

button1DoubleClick: aPoint
    "Private - Left button is double clicked. If the double click occured in
     the system box, then close myself."
    (self systemBoxRectangle containsPoint: aPoint)
        ifTrue: [self close]
        ifFalse: [super button1DoubleClick: aPoint]!

button1Down: aPoint
    "Private - The left button is down at aPoint. If aPoint is in the system
     box, then pop up a menu. If aPoint is in the caption bar, then
     prepare to drag."
    | screenPoint |
    (self systemBoxRectangle containsPoint: aPoint)
        ifTrue: [self doSystemMenu]
        ifFalse: [
            (self captionBarRectangle containsPoint: aPoint)
                ifTrue: [
                    screenPoint := aPoint
                        mapClientToScreen: self.
                    self
                        captureMouseInput;
                        startPoint: screenPoint;
                        lastPoint: screenPoint;
                        displayReverseDragOutline]
                ifFalse: [super button1Down: aPoint]]!

button1Move: aPoint
    "Private - As the user moves the mouse around with the left button
     down, show an outline of where the window would go if the button
     is released."
    | screenPoint |
    handle = WindowHandle queryCapture
        ifTrue: [
            screenPoint := aPoint mapClientToScreen: self.
            self                                    continued on next page
```

continued from page 11

Floating toolbox in Smalltalk/V

```
                displayReverseDragOutline;
                lastPoint: screenPoint;
                displayReverseDragOutline]
        ifFalse: [super button1Move: aPoint]!

button1Up: aPoint
    "When the user lets up the button, move the window to the new
     location."
    | screenPoint |
    handle = WindowHandle queryCapture
        ifTrue: [
            screenPoint := aPoint mapClientToScreen: self.
            self
                displayReverseDragOutline;
                lastPoint: screenPoint;
                clearMouseCapture.
            self handle
                setWindowPos: nil
                rectangle: self outlineRectangle
                fs: SwpNozorder | SwpNosize]
        ifFalse: [super button1Up: aPoint]! !

!FloatingTopPane methodsFor: 'private-accessing' !

lastPoint
    ^lastPoint!

lastPoint: aPoint
    lastPoint := aPoint!

startPoint
    ^startPoint!

startPoint: aPoint
    startPoint := aPoint! !

!FloatingTopPane methodsFor: 'private-style' !

defaultFrameStyle
    "Private - Answer my style."
    ^WsPopup | WsBorder! !

!FloatingTopPane methodsFor: 'private-menu bar' !

buildMenuBar
    "Private - I have no menu bar."! !

!FloatingTopPane methodsFor: 'private-captionBar' !

captionBarHeight
    "Private - Answer the height of the caption bar."
    ^9!

captionBarRectangle
    "Private - Answer the rectangle outlining my caption bar."
    ^self captionBarRectangleIn: self rectangle!

captionBarRectangleIn: aRectangle
    "Private - Answer the rectangle outlining my caption bar. Assume
     aRectangle is my frame."
    ^(aRectangle leftTop leftAndUp: 1)
        extent: aRectangle width + 2
            @ self captionBarHeight! !

!FloatingTopPane methodsFor: 'private-client rectangle' !

clientRectangleIn: aRectangle
    "Private - Answer the rectangle, inside my frame, where my children
     go. Assume aRectangle is my frame."
    ^(aRectangle leftTop down: self captionBarHeight - 1)
        rightBottom: aRectangle rightBottom! !

!FloatingTopPane methodsFor: 'private-displaying' !

display
    "Display myself."
    self
        displayCaptionBar;
        displaySystemBox!

displayCaptionBar
    "Private - Display the caption bar."
    | box fillColor |
    box := self rectangle.
    self pen
        foreColor: ColorManager black;
        backColor:
            (self isActive
                ifTrue: [ColorManager activeCaptionColor]
                ifFalse: [ColorManager inactiveCaptionColor]);
        rectangle: (self captionBarRectangleIn: box)!

displaySystemBox
    "Private - Display the system box in the top left corner."
    | systemBoxRectangle insetRectangle |
    systemBoxRectangle := self systemBoxRectangle.
    insetRectangle := (0@0
        extent: systemBoxRectangle width - 6 @ 3)
            centerIn: systemBoxRectangle.
    self pen
        foreColor: ColorManager black;
        backColor: ColorManager lightGray;
        rectangle: systemBoxRectangle;
        backColor: ColorManager white;
        rectangle: insetRectangle! !

!FloatingTopPane methodsFor: 'private-displaying drag outline' !

displayReverseDragOutline
    "Private - Display the drag outline using the display's pen."
    self
        displayReverseDragOutline: self outlineRectangle
        using: Display pen!

displayReverseDragOutline: aRectangle using: aPen
    "Private - Display aRectangle as the drag outline using aPen."
    aPen
        saveDC;
        setRop2: R2Notxorpen;
        foreColor: ColorManager black;
        place: aRectangle leftTop;
        box: aRectangle rightBottom;
        restoreDC!

outlineRectangle
    "Private - Answer a rectangle defining my dragged frame."
    ^self frameRectangle
        translateBy: (self lastPoint - self startPoint)! !

!FloatingTopPane methodsFor: 'private-system box' !

doSystemMenu
    "Private - The system menu has been clicked. I guess that a menu
     should be opened."!

systemBoxExtent
    "Private - Answer the extent of the rectangle outlining the system
     box."
    ^self systemBoxWidth @ self captionBarHeight!

systemBoxRectangle
    "Private - Answer the rectangle outlining the system box."
    ^self systemBoxRectangleIn: self rectangle!

systemBoxRectangleIn: aRectangle
    "Private - Answer the rectangle outlining the system box, assuming
     aRectangle is the window's frame."
    ^(aRectangle leftTop leftAndUp: 1)
        extent: self systemBoxExtent!

systemBoxWidth
    "Private - Answer the width of the system box. This constant comes
     right from Microsoft's GUI guidelines."
    ^12! !
```

```
!FloatingTopPane methodsFor: 'windows events' !

resize: aRectangle
    "Private - When I resize, make sure that my child windows leave room
     for the caption bar."
    ^super resize: (self clientRectangleIn: aRectangle)!

wmSize: wordInteger with: longInteger
    "Private - Process the window resizing message.  From a Digitalk
     maintenance upgrade (2.01)."
    | extent |
    "do nothing if being minimized"
    wordInteger = Sizeiconic ifTrue: [ ^nil ].
    extent := ( ( WinPoint new:4 )
        uLongAtOffset: 0 put: longInteger ) asPoint.
    ( extent x = 0 or: [ extent y = 0 ] )
        ifFalse: [ self resize: ( 0 @ 0 extent: extent ) ].
    ^nil! !
```

Listing 2. ColorManager class.

```
Object subclass: #ColorManager
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries:
    'WinConstants ColorConstants '
category: 'Environment'!

!ColorManager class methodsFor: 'system colors' !

activeCaptionColor
    "Answer the color to use for an active caption (title bar)."
    ^UserLibrary getSysColor: ColorActivecaption!

inactiveCaptionColor
    "Answer the color to use for an inactive caption (title bar)."
    ^UserLibrary getSysColor: ColorInactivecaption! !

!ColorManager class methodsFor: 'colors' !`

black
    "Answer the color constant for black."
    ^ClrBlack!
```

Software metrics

continued from page 8

email the author at billcole@mercury.interpath.net, or 73363.276@compuserve.com.

Refining the implementation of Change Metrics to address error reintroduction as a result of defect correction will be the topic of a future paper. ♀

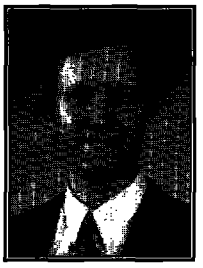
Acknowledgement

The author wishes to thank Kyle Brown for his assistance in preparing this article.

References

1. Levendel, Y, Reliability analysis of large systems: defect data modeling, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 16(2), 1990.
2. Myers, G. SOFTWARE RELIABILITY, 1976, p. 340.
3. Ehrlich, W. K., Lee, S. K., Molisani, R. H. Applying reliability measurement: A case study, IEEE SOFTWARE ENGINEERING, March 1990.

William Cole is President of JumpStart Systems, a Smalltalk consulting firm in Cary, NC. He can be reached at 919.460.1583 or by email at billcole@mercury.interpath.net.



A brief look at size metrics

MARK LORENZ

Last month, we took "A brief look at inheritance metrics." This month, we will continue looking at O-O metrics, again drawing heavily* from my book.[1]

An area of historical interest in software metrics is the desire to measure the size of a piece of code. As usual, the differences encountered in O-O systems necessitate the use of different metrics. In this article, I'm going to discuss some design metrics dealing with class and method size.

SIZE METRICS

Class size

Class size can be measured in a number of ways, including:

- number of instance methods
- number of class methods
- number of instance variables
- number of class variables

In this article, we'll look at the *number of instance methods* in more detail.

Figure 1 shows the results from a number of O-O projects. We see that most of the projects averaged in the ten to fifteen methods per class range. I have noted a couple of points worth discussing on the results—UI-intensive projects and longer-term projects. We believe, through further detailed study, that:

- UI classes tend to have more methods to service the controls available to the user;
- new requirements are typically placed on mature classes over time, causing them to grow in size;
- classes nested deeper in the inheritance hierarchy tend to have fewer methods, since they merely extend their superclasses' capabilities.

The *public* methods indicate most accurately the amount of work for which a class is responsible.

The number of methods in a class relates to the amount of collaboration being used. Larger classes may be trying to do too much of the work themselves instead of putting the responsibilities where they belong. They are more complex and harder to maintain.

Smaller classes tend to be more reusable, since they provide one set of cohesive services instead of a mixed set of capabilities.

On an individual class basis, I use an upper threshold of 20 for the number of instance methods in a model class and 40 for UI classes to identify anomalies. I use a lower threshold for *averages* across a number of classes: 12 for model classes and 25 for UI classes.

ACTION PLANS

Possible actions when anomalies are found in class size

Hold design reviews to examine the class to see if some of the methods don't make sense to be included in this class' responsibilities. There may be undiscovered class(es) or misplaced responsibilities. Look carefully at method names and ask yourself questions such as

- Is this something I would expect this class to do?
- Is there a less obvious class, such as an event, that has not been defined?

When examining a class, focus on public methods of a class.

Take a look at the *instance variable usage* metric to see if there is a way to divide the class along optimum method lines.

METHOD SIZE

Method size can also be measured a number of ways:

- number of message sends,
- number of statements, and
- number of lines of code.

In this article, we'll look at the *number of message sends* as a more style-independent metric for measuring method size.

Figure 2 shows some project results for message sends. Our rule of thumb is nine for an upper anomaly threshold. Large numbers may indicate function-oriented code and/or poor allocation of responsibilities.

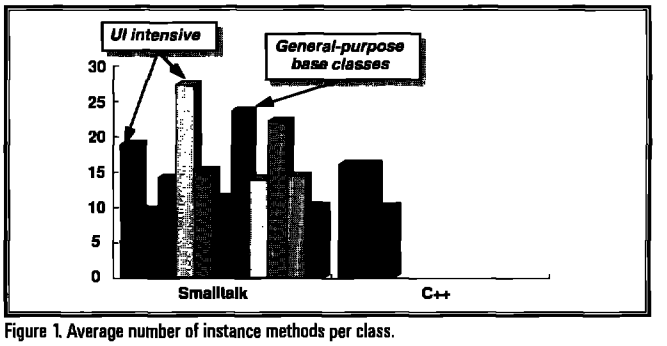


Figure 1. Average number of instance methods per class.

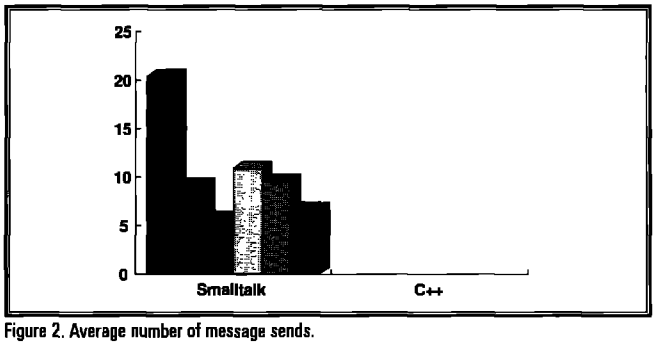
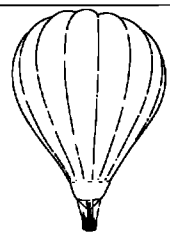


Figure 2. Average number of message sends.

* This article draws heavily from the work I did for the book, with the publisher's blessing. Rather than note every reference, I will refer you to the book for an extensive discussion of OO metrics.

Mark Lorenz is founder and president of Hatteras Software, Inc., a company that specializes in helping other companies use object technology effectively. He welcomes questions and comments at 919-851-0993 or at 71214,3120@compuserve.com.



Following are a number of affecting factors for this metric.

Key classes. Classes at the center of your application will often be larger and more complex.

UI classes. User interface classes will have some large methods, such as the initial layout of the window contents. This may skew your averages higher.

Active versus passive classes. Classes that take an active role in driving the behavior of the system will generally be larger and more complex than passive, data-providing objects.

Accessing methods. Methods that allow access to state data in classes are typically very short. This will skew your averages lower.

Language. C++ method sizes tend to run higher than Smalltalk. I use a higher threshold for C++ projects to identify anomalies. Coding in a hybrid language, such as C++, also allows the developer to write code in methods that are outside the "O-O part" of the language. This code will not, of course, be counted in the number of message sends but certainly relates to the size of the method. We wouldn't want to ignore 100 lines of non–O-O code and count a couple of message sends (member function invocations). Of course, we don't generally want our developers writing non–O-O code anyway!

Action plans

So what do you do if your message sends are beyond the rule-of-thumb threshold? Look at the methods individually or in small groups and decide if an action would improve your design. If so, provide mentoring for developers who are writing larger methods. They are probably falling back on old habits and writing their methods in a more serial, function-oriented fashion instead of requesting services from other objects. Larger numbers of smaller methods are, in general, a better O-O design.

SUMMARY

We have taken a brief look at O-O metrics dealing with class and method size. In particular, we examined a class' number of instance methods and a method's number of message sends. We have seen that there are factors that affect the meaning of these measurements and have taken a look at possible action plans for anomalies. ♀

GLOSSARY

Accessing method. A method that is used to *get* or *set* an instance variable. Accessing methods allow you to perform laissez-faire initialization. They are usually very short, almost standard, methods that are left out of some measurements.

Anomaly. A deviation from the common result.

Heuristic. A guideline based on trial-and-error usage. A rule of thumb.

Key class. A class that is central to the business domain being automated. A key class is one that would cause great difficulties in developing and maintaining a system if it did not exist.

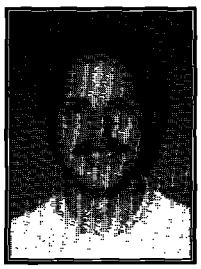
Measurement. The determination of the value of a metric for a particular object.

Metric. A standard of measurement. Used to judge the attributes of something being measured, such as quality or complexity, in an unbiased manner.

Threshold value. A measurement value that has been determined through project experiences to be significant in terms of desirable or undesirable designs, with some margin of error. Generally, these will be tunable over time as you gain experiences specific to your business and teams.

Reference

1. Lorenz, M. Object-Oriented Software Metrics: A Practical Guide, Prentice Hall, Englewood Cliffs, NJ, 1994.



KENT BECK

Using patterns: Design

I kind of ran out of steam towards the end of that last series on creating new objects. I think the message—that many of the most important objects are not the ones you find by underlining nouns in a problem statement—is still valid. The objects that emerge (if you're watching for them) late in the game, during what is typically thought of as maintenance, can profoundly affect how you as a programmer view the system. By the time I got to the fourth part, though, I was tired of the topic. Those last couple of patterns still deserve some reexamination in the future.

This month I'm making a new start, and once again the topic is too large for a single column. The problem is how patterns can be used. I have presented probably a dozen patterns over the past year, but I haven't said anything about how they are used. Using patterns is a topic of current interest to me, because I've started teaching a three-day course on how to write and use patterns, and my students are asking me how to apply all these great patterns they are writing.

I divide the use of patterns into three categories: explaining designs, designing, and documenting for reuse. This series will address all three uses of patterns, and introduce several new patterns in the process.

EXPLAINING

Even if you don't use patterns explicitly in design, they are a great way to explain the architecture of a system. Look for a paper Ralph Johnson and I wrote in the coming ECOOP '94 proceedings for an example which uses patterns to describe the Hot Draw drawing editor framework (the paper is also available via ftp from st.cs.uiuc.edu).

DESIGNING

Nothing says you have to wait to use patterns until the design is finished. Considering patterns by name during design can result in clearer, more consistent, and more productive designs. If you're working in a team, patterns can become the basis of much more efficient communications between team members. I find that even when I design alone, patterns keep me from taking shortcuts that cost me time down the road.

Kent Beck has been discovering Smalltalk idioms for eight years at Tektronix, Apple Computer, and MasPar Computer. He is the founder of First Class Software, which develops and distributes reengineering products for Smalltalk. He can be reached at First Class Software, P.O. Box 226, Boulder Creek, CA 95006-0226, or at 408.338.4649 (phone), 408.338.3666 (fax), 70761,1216 (Compuserve).

REUSE

The patterns used for explaining and designing are very general purpose, software engineering patterns. They are intended for an audience of professional programmers who are used to making judgments based on experience and taste. There is another programming audience, however, one that is potentially much larger than just hackers. This audience are those who program by necessity, not by choice. They are the biologists, chemists, and business people of the world who can't find a program that does what they want, but they don't want to learn any more about programming than they must to get their job done. Patterns are a great way of communicating how to reuse a framework (as opposed to merely telling them how it works).

I'll deal with all three uses of patterns in the months ahead. I'll start with design—how you can use patterns to help you design better, faster, and with greater confidence.

DESIGNING WITH PATTERNS

The stereotyped Smalltalk programmers design as they program. There's never a "paper" design, instead the design of the program is reflected in the code and changed as a result of manipulations in the browser. Ignore for a moment that this picture doesn't reflect any of the diversity that currently exists in the Smalltalk world, which includes many programmers following explicit, nonprogramming-environment-based methods. What can patterns offer the lone wolf, the "design-as-you-go" programmer?

One great thing about patterns is they provide you with easy-to-digest descriptions of design techniques. Even if you have been using Smalltalk for years, there are likely tricks that other designers know that you don't. Reading and writing patterns gives you a way to communicate your design knowledge with others, and have them fill you in on techniques you don't know.

One of the things I like about patterns is that they force me to be consistent in my designs. After I formulate a pattern I can often go back to my existing programs and see how they would have been better had I consistently used the pattern. An example is using "each" for the parameter to a block used for iteration. I used to get creative with the parameter names, trying to call them something meaningful. By just using "each" I can program faster, because I don't stop to choose a name; I can read the code better, because I know what to expect; and I am more likely to put complex expressions in the block into their own method, because "each" doesn't work well if you are programming a block with 20 lines in it.

Finally, no one ever really just codes for themselves, for the moment. Someone will read your code, even if it is just you six months later. The consistency and explicitness of patterns, if their use is documented, become important sign posts on the road to understanding what you were getting at when you wrote the code in the first place.

Now let's move into an example of using patterns for design. I'll introduce a specification that we need to design to, then alternately tell you about new patterns and show how they advance the design. In real design, you'll be working mostly with patterns you already have in front of you. The purpose of the

discussion isn't to show you a real design session, but rather to convince you that patterns could be valuable during design, and coincidentally introduce some new patterns.

THE PROBLEM: TV WITH REMOTE CONTROL

Here's the example I'll use for the rest of the articles on design with patterns. The problem is to design the software to run a television, including a remote control. For now we'll support two simple stories (some people call them "use cases"): when someone presses a channel button the channel (one of ten) changes, and when the television notices a commercial starting it flashes a light on the mute button of the remote control.

We are given several pieces of software to begin with:

- a keyboard library that returns the last key pressed (0-9 for the 10 channels, -1 for no new key press, and 10 for mute)
- an infrared communication library that sends and receives bytes
- a tuner library that changes channels, and
- an object that watches a video stream and broadcasts messages whenever it notices the start of a commercial

I'm assuming that both the television and the remote control have processors capable of executing objects, and that they are linked with a communication channel based on infrared.

> *Reading and writing patterns gives you a way to communicate your design knowledge with others...*

BIG OBJECTS—OBJECTS FROM THE USER'S WORLD

We'll implement the "change a channel in response to a keypress" story first. The first problem to be solved is finding the large scale objects in the system. The last issue contained a pattern, Objects from the User's World, which addressed this problem. The problem statement of that pattern was, "What are the best objects to start a design with?" Sounds like the right pattern to me.

The solution of Objects from the User's World is to begin the system with objects that the user thinks about. Two objects spring to my mind that fit this criteria. RemoteControl has the responsibility for translating user input into commands (that is, given some user input, the RemoteControl will decide what messages to send to the rest of the system). Television has responsibility for changing channels. I'm imagining protocol in Smalltalk like channel: anInteger.

```
RemoteControl * translate user input into commands
Television * change channels
```

If I want to execute the story, first the RemoteControl has to take the input of a channel button being pressed, then translate it into a message to the Television to change the channel.

Keyboard—Objectified Library and Event

This scenario is plausible as far as it goes, but it isn't nearly detailed enough to begin programming. The next problem we solve is getting the keystrokes into the RemoteControl to begin the computation. The first part of this problem is reading the keystrokes in the first place. Recall that we have a library that lets us read the keystrokes. We need to surface those functions in object terms. Here's the pattern we will use:

OBJECTIFIED LIBRARY

How can you integrate a function library with a system built from objects?

All modern object languages provide facilities for calling functions written using older languages. The design problem is how to integrate the functions conceptually with the rest of the system. The simplest solution is to call the functions wherever you need to in your code. This has the advantage of simplicity, because you don't have to introduce any new objects.

Just calling the functions from wherever introduces problems of its own. First, external libraries change at a rate that isn't synchronized with changes to your objects. New releases of a library may come at a time when you don't want to change much of your system. By scattering the calls to the library all over your system, you make yourself vulnerable to having to touch many parts of your system.

Second, because the world is moving more and more towards objects, any function library is likely to be replaced by objects. Scattering calls to the library means that you will have to revisit much of your design should that happen.

Finally, scattered calls to the function library don't communi-

Smalltalk Idioms

cate well. In particular, you cannot easily answer the common question, "how is the library being used?" Answering this question is important both for making the kind of design updates required above, and to use the library with other objects. Therefore: Create an object to represent the library. Give it external protocol using Smalltalk naming conventions and accepting and returning standard objects.

Using Objectified Library, we create a Keyboard object whose responsibility in the design is to read keystrokes, and which accomplishes it by calling the keyboard library.

Keyboard * read keystrokes
Running the story again, we have the RemoteControl asking the Keyboard for the next keystroke, then translating that into a message to the Television to change the channel.

Notice that the solution to Objectified Library tells us to "accept and return standard objects." What object should the Keyboard return? Here is a pattern that you are probably familiar with that answers this question:

Event
How do you represent interesting events from the outside world?

The simplest solution to representing outside events is to cause a message to be sent to the object which handles the event, passing as parameters what happened. Unfortunately, the parameters are seldom a simple, atomic object. Also, the system is often interested in other, related information from other sensors such as the clock (i.e., a timestamp). Passing all of the information as separate parameters leads to long, unwieldy message names and introduces the risk that some parameters will get out of sync with others.

Simplicity of naming is not the only important issue working against simply sending a message. Many times the parameters to be included with an event are time valued and must be collected immediately to be correct. The collection of the parameters doesn't involve complex processing, so it can easily be done in a lightweight but higher priority process than the process in which the reaction to the event is determined. Splitting the collection of the parameters from their processing introduces the need to create a protocol between the tasks. Simple protocols provide flexibility, and a single object as the protocol is simplest of all.

Bundling the parameters together into an object introduces its own costs. First, there is the additional complexity in the system of having an additional class. Second, there will be a runtime cost associated with creating the object. However, these costs are easily dealt with later, while the difficulties associated with not having the parameters together will adversely affect the whole design. Therefore: Create an Event object.

Give it protocol to return information describing what happened outside the system to create the event and report what else interesting was happening at the time.

Our Event object will carry along what key was pressed to create the Event. The Event will be a good place to hide the interpretation of the keystroke (i.e., the difference between 0–9 and 10). We'll be able to add testing protocol like isChannel, which will insulate the rest of the system from the details of the keyboard.

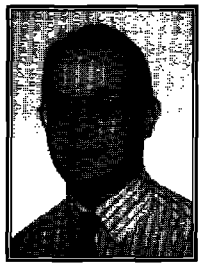
We have to change the responsibility of Keyboard a little to reflect that the Keyboard is responsible for creating the events:

Keyboard * create events from keystrokes
Running the story, we now have the RemoteControl asking the Keyboard for an Event, then using the Event to send a "change channel" message to the Television.

CONCLUSION
This article has used one pattern from the previous issue, Objects from the User's World, and two new ones, Objectified Library and Event, to create the first four objects in our system: RemoteControl, Television, Keyboard, and Event.

That's all I have space for in this issue. Next time we'll decide how the objects are going to be distributed between the two processors, including a nifty pattern I learned while consulting on a telecom project called Half Objects. We'll also get into more of the details of how Events get turned into commands.

If you have good examples of applications of any of the patterns I present, or if you have alternative patterns that you think are better, please let me know. Patterns are just getting started and there are no right answers, only a bunch of committed people searching. ♀



ALAN KNIGHT

New net resources

The Internet, in its new guise as an "information superhighway" is becoming very trendy. There are an ever-increasing number of resources accessible via the net. This column covers some Smalltalk-related resources that have recently become available.

X3J20 MAILING LIST

X3J20 is the designation of the American National Standards Institute (ANSI) committee that's working on a standard for Smalltalk (q.v. THE SMALLTALK REPORT, 2(7):1)).If you're interested in the committee's progress, you can now get updates by e-mail. **Bruce Schuchardt (bruce@slc.com)** writes:

> **The X3J20 Smalltalk standards committee has formed a mailing list for its work. In our 10/93 meeting it was decided that since the committee is public its mailing list should be open for public subscription.**
>
> **If you wish to subscribe to the X3J20 mailing list, send a message to listmaint@slc.com with the body text**
>
> **subscribe x3j20 your-email-address**
>
> **You must send the subscription from the mailing address you list in the body of the subscription request or it will be ignored, i.e., you can't request a subscription for someone else, or for yourself at another address.**
>
> **Problems in using the mailing list should be addressed to x3j20-owner@slc.com. We have had some problems with the mailing list but we hope these have all been ironed out at this point.**

Messages to the list should be sent to x3j20@slc.com.

There is also an electronic archive of X3J20 documents, at the ftp site info.er.usgs.gov. Documents are available as ASCII text, PostScript, or Hypertext Mark-up Language (HTML). They are stored by document number in the directories that are named /pub/smalltalk/approved and /pub/smalltalk/submitted. The file pub/smalltalk/ Index has a list of document descriptions and numbers.

Alan Knight is a consultant with The Object People. He can be reached at 613.225.8812, or by e-mail as knight@acm.org.

SMALLTALK REFACTORY

There have been a number of interesting development tools for ParcPlace Smalltalk from Ralph Johnson's students at the University of Illinois. These include the object debugging extensions (q.v. THE SMALLTALK REPORT, 2(9): 4) and the NewTool extended browser (q.v. THE SMALLTALK REPORT, 3(2):17). The Smalltalk Refactory is another in this series, which assists in making a number of common types of modifications to Smalltalk code. From the user manual:

> **Object-oriented languages are touted as promoting software reuse. However, object-oriented software is usually not reusable when it is first written. Reusable software is the result of several design iterations involving different applications that reuse a common body of code. As the code is incorporated in the different applications, reusability problems become apparent and the body of code must be altered. Each iteration becomes easier and easier as the common code becomes more and more reusable.**
>
> **Since these iterations involve not only new code but also existing code, care must be taken when altering the common body of code. The changes that are made must be behavior-preserving so as not to break the existing applications. These behavior-preserving manipulations that change the design of the reusable code are known as refactorings.**
>
> **...The Smalltalk Refactory is a tool (basically a modified browser) that brings together several refactorings that have been created at the University of Illinois.**

The supported refactorings are:

- Rename instance and class variables, automatically changing all references
- Generate accessor methods for a variable and change all direct references to use the accessors
- Move variables up into a superclass or down to all subclasses, moving methods if necessary.
- Convert superclass to component. This moves a class up the hierarchy and converts its current superclass into an instance variable to which messages get forwarded.
- Change direct instance variable references to a method into calls to accessor methods (generating the accessor methods if necessary)
- Rename a method, changing references.

Renaming a method is particularly interesting, as this is not easy in a dynamically typed environment like Smalltalk. The user manual describes their technique:

> **(Suppose) your application has a class that defines the add: method. After review, it is determined the name addFigure: more accurately reflects its function. Simply changing the name to addFigure: and then browsing all senders of add: will return an enormous list of methods since add: is often called in the context of Collection classes. Very few of these are actually references to your application.**
>
> **To overcome this, the refactory uses a new technique known as lazy refactoring. After the renaming operation has been performed, a Dynamic Analysis is created that monitors all calls to the original method. When these are detected the source code of the caller is altered to refer to the new method. The upshot of all of this is that to correctly perform**

The best of comp.lang.smalltalk

> **the refactoring, your code should be run on a thorough test suite. ...In future versions of the refactory, many powerful, new refactorings will be implemented in the manner along with additional support for this style of refactoring.**

I think this could be a great tool. While it doesn't do anything that couldn't be done by hand in under half an hour, it does it quickly, automatically, and (at least for the static cases) without missing anything. I think this could do a lot to encourage refactoring.

As an academic project, this has a few more rough edges than one would expect in a commercial offering. I came across a trivial bug in one of the menus, and I thought that slower transformations should display a wait cursor. It would also require a bit of work to make it compatible with ENVY/Developer or other version control tools.

While the code is dependent on the details of ParcPlace's browser and metaframework, it should be possible to use this as a basis for something similar in another dialect. It can also serve as a framework for adding your own favorite refactorings.

Don Roberts (droberts@cs.uiuc.edu) wrote the refactoring browser, and included a number of refactorings written by **Dan Walkowski (walkowsk@cs.uiuc.edu)**.

It is available from the normal Smalltalk ftp sites (st.cs.uiuc.edu and mushroom.cs.man.ac.uk). At the UIUC site it is stored as pub/st80_r4/Refactory1.0.st. Those interested in more information about refactorings are referred to William Opdyke's Ph.D. thesis on the subject, available from the same site (in PostScript form) as pub/papers/opdyke-thesis.ps. The pub/papers directory also contains a number of other papers which may be of interest.

COMPUTATIONAL GEOMETRY WORKBENCH

The refactory is of interest to almost anyone working in Smalltalk, since it deals with domain-independent transformations of code. The Computational Geometry Workbench, on the other hand, is a much larger, very domain-specific toolkit. The domain is (of course) computational geometry, mostly of the two-dimensional straight-line variety. The documentation describes it as providing:

> **an environment for creating, editing and manipulating geometric objects, demonstrating and animating geometric algorithms and for implementing new algorithms.**

This toolkit comes from a research group under Jörg-Rudiger Sack at Carleton University. In fact, I was part of this research group a few years ago, and this project is a substantial part of my Master's thesis. If you like it, I'm happy to take credit for large parts of it. If you don't, well, it was long ago when I was young and (even more) foolish. In either case, you should be aware of my obvious bias. As with many projects dependent on student labor, the quality of the code varies widely. Some of it is very good, some of it is surprisingly good for student work, and some of it is not good at all.

The system is written for Smalltalk/V Mac 1.2.1 (that's one version behind the current 2.0). At the rate Digitalk releases V/Mac versions it shouldn't fall any further behind in the next few years. It's a fairly large system, including:

- Basic data structures (stacks, queues, search trees, etc.).
- Not-so-basic data structures(finger trees, splay trees, etc.).
- Robust (knock wood) code for dealing with geometry.
- Some very complex geometric algorithms (triangulation, visibility, shortest-path, and point location are most prominent).
- A geometric object browser and other tools for manipulating geometric objects.
- A framework for algorithm animation.
- Yet another home-grown change control system.
- Many hacks to Smalltalk/V Mac, including color support.
- A clipboard interface, very handy for importing geometric figures into drawing programs and documents.
- Documentation (an uncommon feature with university projects).

It is available by ftp from the normal archive sites or from alfred.ccs.carleton.ca.

OBJECTSHARE UPDATES BY FTP

Objectshare Systems makes the popular WindowBuilder and SubPanes add-ons for Smalltalk/V. Updates of their products are now available by ftp from the UIUC archive site, in the pub/OBJSHARE directory. These are incremental updates, meaning that they must be installed into an image already containing the appropriate version of the product (for example, to upgrade WindowBuilder Pro 1.0 to 1.04).

Alan Knight is a consultant with The Object People. He can be reached at 613.225.8812, or by e-mail as knight@acm.org.

REBECCA
WIRFS-BROCK

Responsibly designing your objects' data

When should you define the data encapsulated within responsibly designed objects? What things should you consider as you pin down the internal representation of object data? Responsibility driven design places great emphasis on object roles and responsibilities. Once initial object behaviors are defined, the design can and should be refined. Refinement often makes the difference between a muddled design and a workable one. It is during refinement that we make a concerted effort to streamline collaborations, factor common responsibilities into class inheritance hierarchies, and clarify the contractual agreements between objects. All this modeling soaks up a lot of design energy!

Even after this effort, we're really not ready to roll into implementation. Not *just* yet. There is a fair amount of object design work left. Whether these details are worked out as the last stages of design or the initial phases of construction is irrelevant. It is these design details that are vital to most business applications.

Initially, in my quest for answers to the design for object data, I searched the writings of authors' with a strong data focus for expert guidance. While their methods emphasize identifying the form of the data encapsulated within objects and showing structural relationships between classes of associated objects, they only give light treatment (if at all) to many tactical considerations. While I found it interesting, their works did not shed much light on what I wanted to find out.

Nuts and bolts implementation details typically are left as an "exercise to the implementor." Even datacentric methods finish before dealing with the complexities of designing persistence into an application. Without a design for dealing with this, our applications are incomplete. However, a generic, one-size-fits-all solution is unrealistic. Over the past three years I have seen several practical solutions to data design for large applications. No one followed a recipe. Each development team wrote their own unique chapter. Each solution focused on different elements of this large problem. These experiences have shaped my opinions.

Rebecca Wirfs-Brock is the Director of Object Technology Services at Digitalk and co-author of Designing Object-Oriented Software. Prior to joining Digitalk, she managed the development of Tektronix Color Smalltalk. Comments, further insights or wild speculations are greatly appreciated by the author. She can be reached via email at rebecca@digitalk.com or by mail address at Digitalk, 7585 S.W. Mohawk Drive, Tualatin, OR 97062.

I don't intend to answer all these "data meets responsibility driven design" questions in this column. I will present a strategy for tackling the design of object data, both persistent and non-persistent. I will outline a practical and straightforward process for developing these details. We'll look at managing structural relationships between collaborators, understanding object persistence, fetching, retrieving and updating parts of objects that do persist, and a general strategy for providing information to and incorporating advice from those tasked with developing external databases for our applications.

Let's start by looking at what data requirements we can glean from our initial object model.

WHAT WE KNOW ABOUT OUR OBJECTS

Use cases describe tasks that a system must perform. Conversations capture the dialog between an actor (either a human or another application) and our object model. Conversations depict sequences of interactions and work steps that must be accomplished to perform an actor-directed task.

As we unfold our model of collaborating objects to supports each conversation, we are developing a sense of objects' dynamic behaviors. We comprehend how actors ask to get at certain information. We know how and why various objects are used in particular situations.

Our emerging object model is populated with objects having responsibilities for knowing

- certain facts,
- how to perform specific tasks, and
- about the existence of others,

designed according to their roles and stereotypes.

For example, in our video kiosk application (which I have borrowed from our design course and will use to illustrate objects' responsibilities and behaviors), a Local Inventory object knows the Videos on hand. Each Video "knows" a number of characteristics (including title, director, category, rating, and so on).

When we record responsibilities this way, we are reflecting in our model how we expect to navigate between objects. To display a list of available movie titles, we'll ask the Local Inventory for that list. We can then display this list. When the user picks a movie from that list, she can either choose to read a detailed description or perhaps to view a video clip before renting. She may decide to reserve that movie for later instead of renting it today.

Data modelers look at an emerging object model differently than I have been trained. Both of our perspectives are useful. When a data modeler looks at relationships between data (or between classes of objects), she sees the possible ways to get at one piece of information from another. A data modeler needs to understand the relationships between objects. Traditionally, data modelers have built structural models showing cardinality, data ownership, and creation and update rules in order to understand the significance of those relations. The data modeler looks at my object model with an eye towards building a consistent logical data model. Physical modelers are concerned with designing data base schemas that are correct *and* high performance.

There are important differences in our concerns. During design initially focus on object behaviors. It is my firm belief that

these need to be understood before one steps into the design of each object's encapsulated data. Data modelers seek information about access patterns and structural relationships! Are we at cross purposes? Not really. What we can do is tell them about our application's impact upon their data designs.

We have been focusing on how our objects collaborate to support actor-directed tasks. So, with a modest effort we can add these details to our model:

- We can define a first cut at our objects' instance variables.
- We can identify what parts of our objects are persistent.
- We can tell how to uniquely identify any persistent object.
- We can describe the ways we traverse our objects to accomplish specific tasks.
- Conversation by conversation, we can determine which objects will be accessed and how they will be modified.
- We can record which conversations cause specific objects' information to be created, modified, updated, stored and retrieved.
- We can determine how we'd ideally like to fetch persistent information in order to "materialize" our application objects from external storage.
- We should be able to tell how long objects need to exist in our application.

Besides simply digesting this information, it is equally important that database designers critique our design. We are providing them with preliminary information that will remain preliminary until we believe it to be sound. They can help validate our work. They can spot inconsistencies and gaps in our thinking. They can recommend alternative ways to build our objects that are more flexible, place less demands on databases, or take into consideration other applications' needs.

Database design is so crucial to business applications that database specialists are integral members of many object analysis and design teams. Data designers add a vital perspective.

STEREOTYPES AND PERSISTENCE

We can also look to object stereotypes to identify persistency demands. While there are no hard and fast rules, here are some typical situations.

Controllers perform a cycle of action. *Coordinators* pair client requests with objects that can perform a specific service. *Service providers* typically perform a single operation on demand. *Interface objects* support communication between objects within our program and external systems or users.

These objects typically don't define persistent information. However, they may be designed to be configurable. In this case they are constructed to contain configurable "facts" that dictate how they perform their varied tasks. Often we store this configurable information externally and reinitialize these objects and their classes prior to use. This allows us to change certain aspects of our application without having to redefine these classes.

On the other hand, coordinators and controllers do initiate requests to retrieve other persistent objects. For example, in our video kiosk design, the Session Manager (a coordinator) asks a CustomerDB object (a database interfacer) for a particular customer object. Controllers may pass along references to these persistent objects. The Session Manager passes along a customer to a Kiosk Transaction object tasked with coordinating either renting, reserving or previewing a movie.

Objects may generate and store a history of their work. Service providers and coordinators might generate and store this information themselves or, more likely, collaborate with others having the specialized knowledge do perform this task. For example, in our kiosk design, a record is printed for each transaction. Transactions are logged and the local and store inventory is adjusted. To do these tasks, we can design a Transaction Record object to contain transaction specific information.

Information holders maintain values that other objects can ask about. These kinds of objects may or may not be persistent. Transaction Records may themselves be made to persist, or they may be interpreted by Printers and Transaction Loggers to generate persistent information.

Consider another example from our video kiosk. We model a Customer object that knows facts about customers, including their allowable transactions. Certain customers, based on their credit record and recent level of business are granted special privileges. While we may externally store a lot of information relating to a customer, we don't automatically equate all of it with our application's view of a customer.

We may formulate a complex query asking whether the number of rentals the customer has made in the last month is greater than a certain number and whether bills have been paid on time. Based on the answer to that query we can construct a customer object that only holds the information needed for our video kiosk application, including a list of permissible customer transactions. We might generate that list of permissible transactions rather that read the information directly in some customer record in a database.

Structurers maintain relationships between objects. Structurers hold references to other objects that refer to others in a complex network. A key decision to make is how much information to retrieve and store at any point. The ideal is to get just what you need, unless it is easy to get more than you want. Access patterns can be determined by analyzing our model's behavior a use case at a time. Complex retrieval and storage strategies, which demand support via a data access framework, are beyond the scope of this column.

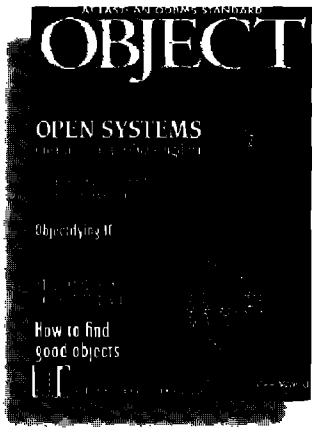
PRACTICAL CONSIDERATIONS

We need to make tradeoffs in our object and database designs. When we implement a solution, we make tactical tradeoffs that limit our design's potential capabilities. They also enable us to build practical solutions. In our Smalltalk programs we connect one object to another (e.g., make one object "know" about another) because that is how we see them collaborating. Connecting one object to another is unidirectional, unless we explicitly make it otherwise.

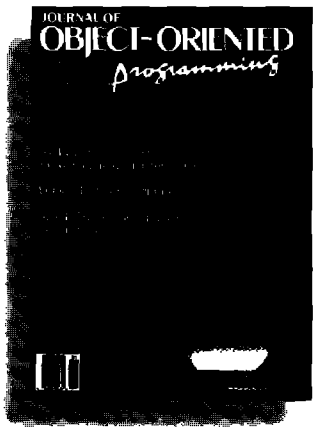
For example, since the Local Inventory knows of Videos on hand, then the Local Inventory may contain direct references to Video objects. We might store them in a dictionary with the title as a key. If we need Video access to be more

continued on page 28

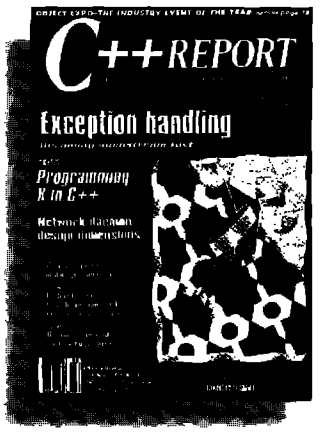
Your best tools for object-oriented software development...



The manager's guide to implementing object technology. The "point of entry" for software management infusing objects into their work environment. Filled with how-to advice, usable strategies, and real world experiences.

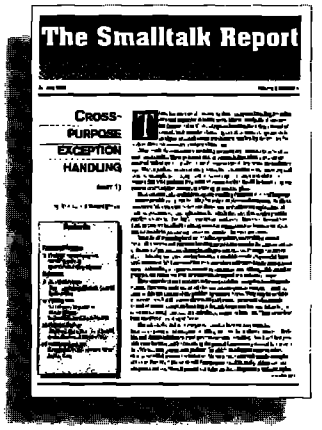


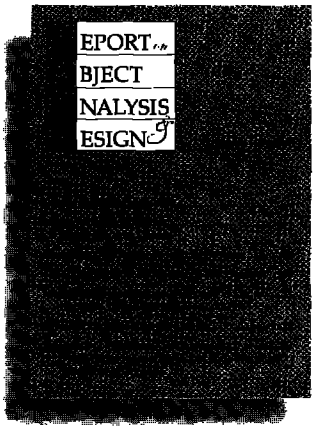
Written for programmers and developers using OOP techniques. International in scope. Code intensive, practical, technical. Breakthrough peer-reviewed papers and invited columns. Now in its 7th year.



Informs C++ developers on how to get the most out of the language. Ideas and techniques for increasing your productivity with C++. Code-intensive, functional tips and tricks for C++ users on all levels and platforms.

Filled with "how-to" advice for Smalltalk users at all levels and in all dialects. The best way for Smalltalk programmers to maximize the language's potential.





Addresses language-independent, architectural concerns about O-O analysis, design and modeling. Platform and system independent, ROAD is written for software developers and project leaders.

SIGS PUBLICATIONS

Yes, I want my subscription to the following publications to begin immediately. If not completely satisfied, I may cancel at any time and receive a full refund of the unused portion.

- □ *Object Magazine* (1 year, 9 issues) $39
- □ *JOOP* (1 year, 9 issues) $59
- □ *C++ Report* (1 year, 9 issues) $69
- □ *The Smalltalk Report* (1 year, 9 issues) $79
- □ *ROAD* (1 year, 6 issues) $99

TOTAL_____

Method of Payment:

□ Bill me, Attn: _____

□ Check Enclosed (Payable to *SIGS Publications*)

□ Charge My: □ Visa □ MasterCard □ American Express

Card#_____ Exp ____

Signature _____

Name _____

Title _____

Company _____

Address _____

City _____

Province/State_____ Postal Code/Zip _____

Country _____

Phone_____ Fax _____

Return this coupon by mail or fax, or call to start your subscriptions.

Mail: SIGS Publications, Inc., P.O. Box 2027, Langhorne, PA 19047

Fax: 215-785-6073

Phone: 215-785-5996

Important: Non-U.S. orders must be prepaid. Please add $35 per subscription year for air service. Checks must be in U.S. dollars drawn on a U.S. bank.

Product Announcements

Product Announcements are not reviews. They are abstracted from press releases provided by vendors, and no endorsement is implied.

Vendors interested in being included in this feature should send press releases to THE SMALLTALK REPORT, Product Announcements Dept., 91 Second Ave., Ottawa, Ontario K1S 2H4, Canada, 613.225.8812 (v), 613.235.8256 (f).

UNISQL delivers first ODMG-based C++ and Smalltalk links

UniSQL, Inc., announced the availabilty of two new object-oriented database interfaces that support the Object Data Management Group's (ODMG) emerging specifications for industry-standard C++ and Smalltalk object-oriented programming language bindings. The announcement was made at the Object Expo National Conference and Exposition. The UniSQL Smalltalk Interface products let C++ and Smalltalk developers build mission-critical application that utilize UniSQL's object/relational database and middleware capabilities in a way that is fully transparent to C++ and Smalltalk language environments.

UniSQL provides integration to C++ and Smalltalk developers via the firm's UniSQL/M Multidatabase System. This system is able to support object capabilities, such as methods, inheritence, user-defined data types, and composition, with systems that have no such capability.

Both the UniSQL C++ Interface and UniSQL Smalltalk Interface are available for Version 2.1. They will also be available in July for the UniSQL/X Database Management System and the UniSQL/M Multidatabase System on UNIX servers running DEC OSF/1, HP-UX, IBM AIX, and DEC/OSF/1, with DOS/Windows support.

UniSQL, Inc., 9390 Research II, Suite 200, Austin, TX 78759-6544, 512.343.7297 (v), 512.343.7383 (f)

ParcPlace object-oriented tool provides point & click database application creation

ParcPlace Systems, Inc., has introduced VisualWorks 2.0, a major new release of the company's object-oriented client and server tool. VisualWorks 2.0 emphasizes ease-of-use and features a Database Application Creator that allows corporate developers to create basic database applications without any Smalltalk or SQL programming.

VisualWorks' Database Application Creator comprises three components: an ObjectLens to view and manipulate relational data as objects; a Visual Data Modeler to create a visual map between relational data and objects; and DataForms—intelligent reusable objects that allow database manipulation without SQL programming.

ParcPlace Systems, Inc., 408.720.7514 (v), snichols@parcplace.com (email)

Easel Corporation signs licensing agreement for SOM and DSOM technology

Easel Corporation announced that it signed a licensing agreement with IBM for its System Object Model (SOM) and workstation Distributed SOM (DSOM) technologies. Under the terms of the agreement, Easel will incorporate the SOM and DSOM technologies into its Object Studio family of object-oriented application development tools.

The integration of SOM and DSOM into Easel Corp.'s Object Studio family will enable developers to build object-oriented applications containing interoperable components that are reusable and platform independent. The company will deliver SOM- and DSOM-compliant products in the next major release of its Object Studio product family.

SOM and DSOM are standards for creating object-oriented class libraries and for reusing objects. It allows an object to be created in one development language and accessed from a different language. SOM is the first cross-language, cross-platform implementation of the Object Management Group's Common Object Request Broker Architecture (OMG's CORBA) specification.

Easel Corp.'s Object Studio family comprises the ENFIN Smalltalk client/server application development environment and Synchrony, the first of a new generation of business object management tools that integrates the design, assembly and reuse of business objects. Business objects are high-level objects that mirror an organization's business process and enable developers to rapidly build enterprise-scalable applications that help run a business.

Easel Corp., 25 Corporate Drive, Burlington, MA 01803, 617.221.2100 (v), 617.221.6899 (f)

European Smalltalk Summer School

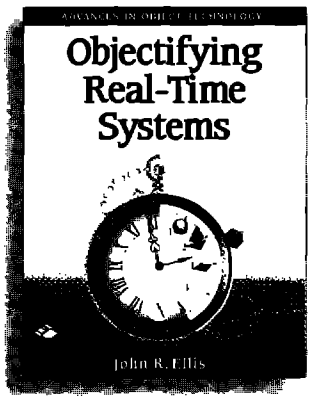
European Smalltalk Users Group (ESUG), organizes its second Smalltalk Summer School in Cork, Ireland, from September 5–9, 1994. This will be a unique opportunity for attendees to meet well-known European Smalltalkers from both the academic and industrial fields, and to gain hands-on experience of Smalltalk's most advanced techniques. Non-European Smalltalkers are of course welcome.

The program will include one-track tutorials, as well as multiple-track workshops, demonstrations or experience reports. Also, rooms will be available for spontaneous meetings. Smalltalk vendors will be able to make demonstrations free of charge. The National Software Directorate will support a special promotion of Smalltalk to the Irish Software Industry in parallel to ESUG. This school is linked to a COMETT-projected supported by the EEC. We look forward meeting you. Do not miss this major European Smalltalk event!

For more information, please contact ESUG by email at esug@ibp.fr or by writing to Annick Fron at AFC Europe, Les Maurettes "Le Grimaud," Avenue du Docteur Lefebvre, 06270 Villeneuve-Loubet, France; +33.92.028653.

From The People That Bring You This Magazine...

New Release!



Objectifying Real-Time Systems

by John R. Ellis

The basic concepts of object-oriented programming are discussed, establishing a common understanding of objects.

•

Provides instruction on how to create each of six RTOOSA Requirements Model products.

•

Anyone interested in developing object-based real-time systems should read this book.

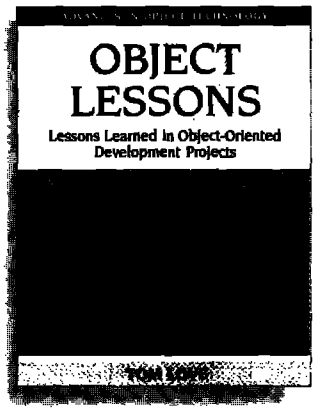
•

Accompaning diskette contains the source programs of examples throughout the book that enable the reader to experiment and verify exectutions without having to key in code.

•

(525 pages with diskette)

Best Seller!



Object Lessons

by Tom Love

Trade secrets of Tom Love, Vice President of IBM Consulting Group are revealed demonstrating the "how to's" of putting theory into practice.

•

An "insider's view" of some major companies' successes and failures relating to object-oriented software projects. Save countless hours and dollars by learning from their costly mistakes.

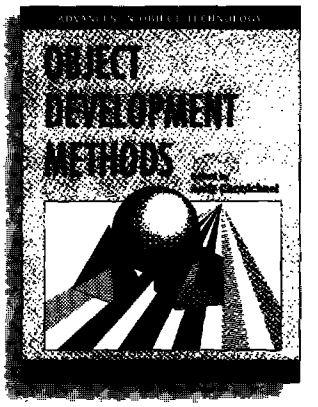
•

This book is written for those making decisions on design and management of large-scale commercial object-oriented software.

•

(276 pages)

New Release!



Object Development Methods

edited by Andy Carmichael

Addresses how object-orientation can be applied to systems analysis and design.

•

A comprehensive survey is included comparing the leading methodologies of Booch, Texel & Rumbaugh among others.

•

The common concepts and underlying structure of each methodology is an important theme explored within this book.

•

This book proves an invaluable reference guide for those still exploring various methodologies and their benefits/drawbacks, and for those who have already made a methodology selection.

•

(380 pages)

COMPLETE MONEY- BACK GUARANTEE

Available at selected bookstores.
Distributed by Prentice Hall.

SIGS BOOKS

SEND TO: SIGS Books, Inc., Attn: Circ. Dept.
71 West 23rd Street, 3rd Floor, New York, NY 10010
FOR FASTER SERVICE, FAX TO: 212/274-06

YES! Please send me the following book(s). If I am not totally satisfied, I may return the book(s) within 14 days and receive a complete refund.

- ❑ *Objectifying Real-Time Systems* by John R. Ellis (ISBN: 0-9627477-8-5) $44
- ❑ *Object Lessons* by Tom Love (ISBN: 0-9627477-3-4).....$44
- ❑ *Object Development Method* edited by Andy Carmichael (ISBN: 0-9627477-9-3).....$39

Name _____
Title _____
Address _____
Company _____
City/State/Zip _____
Country/Postal Code _____
Phone _____ Fax _____

METHOD OF PAYMENT
❑ Check enclosed (payable to SIGS Books)
❑ Charge my credit card:
❑ Visa ❑ MasterCard ❑ AmEx Exp Date_____
Credit Card # _____
Signature _____

Shipping/Handling: For US orders, please add $5 for shipping/handling, Canada add $10; Foreign add $15;
Important: NY State residents add applicable sales tax.

SBTPA

JOIN THE FORERUNNER IN CLIENT/SERVER TECHNOLOGY

SmallTalk Developers

SYSTEMHOUSE

SHL utilizes leading-edge client/server technologies to solve the diverse needs of organizations worldwide. Recently recognized as *the current leader in client/server consulting and development,* The Gartner Group, a prominent technology research organization based in Stamford, CT projects we have *the best prospects for client/server consulting and systems integration success by 1995,* of all companies in the industry.

Leading the industry in transforming business processes through information technology, we employ 4,800 associates internationally and enjoy annual revenues in excess of $750 million.

We seek expert SmallTalk Developers who possess exceptional technical skills and business advisory experience. These positions are located in our rapidly expanding MINNEAPOLIS, MINNESOTA practice. Extensive travel to clients' sites may be required.

- **Senior Technical Architects/Technical Architects**
- **Senior Systems Engineers**
- **Programmer/Analysts**

You must have demonstrated experience in the following:

- SmallTalk V with Parts or SmallTalk 80 with Visual Works with either Team V or Envy Tools
- Experience with Object Oriented Design and Object Oriented Analysis using Responsibility Driven Design methodology utilizing CRC, or Object Introduction Diagram
- Experience with associated CASE Tools a plus

Discover why "The Systems Integrator Newsletter" named us the *best Client/Server Systems Integrator of 1992 and the Rising Star of Systems Integration.* Send resume, indicating position of interest, in confidence to: **Manager of Human Resources-SMALLTALK, SHL, 1901 North Naper Blvd., Naperville, IL 60563-8895. FAX: 708-505-9158.** Equal Opportunity Employer M/F/D/V

Recruitment

For information on advertising in the Recruitment Section, contact Michael W. Peck at 212.242.SIGS

micado SoftwareConsult GmbH is one of the leading system houses in Germany for object oriented languages. It has an expert team with wide experience in development and customer support. Due to the astounding growth of the object oriented market in Germany, we are currently seeking the following freelance OO professionals:

Smalltalk Designers and Developers

If you welcome new challenges and if you want to explore your career opportunities please send or fax your resume to

micado SoftwareConsult GmbH
Reutherstr. 1a-c D-53773 Hennef
Tel. (49)2242-871-450 FAX -455
Compu-Serve 100024,2444

Tech Specialists
Precision Staffing. Superior Service.

We currently have numerous opportunities requiring 1+ years experience with:

- SMALLTALK
- Object Oriented Design
- OS/2
- C++
- C

For immediate consideration, please FAX or mail resume to:
Tech Specialists
5711 Six Forks Rd.
Raleigh, NC 27609
(919) 870-5100
(919) 870-7274 Fax

For consultants and businesses alike looking for success, we are THE company of choice. Tech Specialists is a professional services firm that provides IS, Software Development and Engineering professionals on a contract basis. For opportunities with a real future, call us to explore the possibilities.

Call for Writers

THE Smalltalk REPORT

is seeking expert reports, tutorials, and technical papers. Articles should be instructive, product-neutral, and technical.

Submit papers, discuss story ideas,or request Writers' Guidelines from:
John Pugh and Paul White, Editors
THE SMALLTALK REPORT
855 Meadowlands Dr. #509, Ottawa, ON K2C 3N2
613.225.8812 (v), 613.235.8256 (f)
john@objectpeople.on.ca

Editorial topics include:

Applications
- Commercial, engineering & scientific applications
- Applications frameworks
- Project management
- Vertical (application) and horizontal (system) class libraries
- Portability issues
- Object library management

Project management
- Rapid prototyping
- Version management
- Application management
- Team organization
- Organizing for reuse
- Introducing Smalltalk into an organization

Tools
- User interface builders
- Object editors
- Application development tools
- Project management tools
- CASE tools

Language issues
- Inheritance
- User interface paradigms
- Concurrency
- Persistent objects and databases
- Distributed Smalltalk issues
- Performance issues
- Typing
- Metalevel programming

(Competitive stipend paid)

Product Announcements

Digitalk releases EHLLAPI component for PARTS Workbench

Digitalk, Inc. is shipping the PARTS Communications Wrapper for EHLLAPI (Emulator High-Level Language Application Programming Interface), a component for PARTS Workbench for OS/2, which allows integration with legacy mainframe systems by adding visual, object-oriented, client-server technology.

The Parts Assembly and Reuse Tool Set (PARTS), is the first component-based client/server integration framework ever offered. It makes the assembly and reuse of so˜ware components from different technologies a distinct and easier process than creating the components, which allows for much faster delivery of applications with reduced maintenance cost. These components can be written in Smalltalk/V, C, COBOL, or other languages. The PARTS Workbench is the framework for integrating these components.

The PARTS Communications Wrapper for EHLLAPI is designed for corporate users who need GUI front ends to their legacy 3270/5250 mainframe systems. It allows corporate users to easily convert 3270/5250 text screens to graphical screens, giving their users a clean, intuitive interface without having to change the host application.

The EHLLAPI Wrapper supports IBM Communications Manager and other EHLLAPI products compatible with Communications Manager on OS/2. Now mainframe data can easily be integrated with graphical user interfaces, Smalltalk/V, C, COBOL, database servers that use SQL, and Lotus Notes.

Digitalk, 5 Hutton Center Dr., 11th Floor, Santa Ana, CA 92707, 714.513.3000 (v), 714.513.3100 (f).

BOK Technologies introduces GraphicsObjects/V for Smalltalk/V for Windows, Win32, OS/V, and Macintosh

BOK Technologies Inc. announced GraphicsObjects/V, a high-level and platform-independent graphics class library for Smalltalk/V. The library speeds up the cross-platform development on interactive graphical applications written in Smalltalk/V for Windows, Win32, OS/2, and Macintosh. GraphicsObjects/V contains a rich set of classes that encapsulate the platform graphics API and existing low-level graphic methods implemented in Smalltalk/V, thus providing a portable layer that isolates Smalltalk/V programmers from platform dependencies. GraphicsObjects/V programming API (access methods) is uniform, consistent and identical for all its implementations, which insures the cross platforms portability. Code created with GraphicsObjects/V is portable across Smalltalk/V, Windows, Win32,OS/2, Macintosh and future platforms.

The library provides in-depth support for objects including geometric shapes (Bezier curves, polygons, ellipses, wedges, arcs, etc.), a structured graphics hierarchy (collection, set, dictionary of graphics, etc.), a complete 2D geometrical transformations (rotation, scaling,. etc.), prebuilt shapes (arrows, nodes, etc.), drawing attributes (line styles, brushes, etc.), and text blocks that fully support multiple fonts, styles, and colors. The library also provides methods allowing to display, perform hit testing, position, align, and arrange graphics objects. Developers can save and retrieve graphics objects using the storing/retrieving mechanism defined in Smalltalk/V. The library is written entirely in Smalltalk/V and all source code is included. It is available for Smalltalk/V, Windows, Win32, OS/2, and Macintosh.

BOK Technologies, Inc., 5476 Trans-Island Ave., Montreal, PQ, H3W 3A8 Canada, 514.485.6690 (v), 514.485.2095 (f), 72730.655@compuserve.com (e-mail)

Getting Real

continued from page 22

flexible, we can build other mechanisms to access and retrieve information.

Objects can be "wired together" via instance variable references, or more loosely coupled via event mapping. In contrast, if there is a way to get at one piece of information from another in a relational database, you can. If we need to be infinitely flexible we may keep video information in a database and access it through a query.

It may not be sufficient to say that one object "knows" about another. *How* does it know? And, even more intriguing, *when* does it need to know of another? Once structural aspects are decided, strategies for fetching and retrieving persistent information can be worked on.

If we don't have enough room to internally store all Video information inside our application, we can retrieve this information from an external source on demand. In any case, we still describe our Local Inventory object as being responsible for knowing and managing the kiosk's videos. If we place space, performance, or flexibility constraints on our design, we just have to be more clever about how our Local Inventory determines the information it is responsible for knowing.

CONCLUSION

I've learned several lessons from observing and participating in large development efforts. Big efforts require framework support and consistent design guidelines that are followed. Ad hoc solutions to data access and retrieval don't scale. Defining consistent ways to define the instance variables for objects and handle persistent information smoothes out the development effort.

Sometimes, though, it is the simple, easy to do things that make a big impact. It pays to involve data designers early on. Identify persistence requirements and the "shape" of data requests as soon as practical, but not before you understand your model. Build complexity as you need to. Expect designs to evolve and mature. It's pleasantly surprising to me how much we do know about our objects' data requirements if we've really thought through our initial design. ♀