DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TORONTO

CSC428F/2514F

HUMAN-COMPUTER INTERACTION

Lecture 13

USER INTERFACE DEVELOPMENT TOOLS 2

13.1 Architecture of user interface software.....2
13.2 Higher-level tools.....2
13.3 Card-based prototyping tool — HyperCard.....3
13.4 Time-based prototyping tool — Director 3
13.5 The NeXT Interface Builder 4
13.6 Language-based tools.....4
13.7 The Tcl language.....5
13.8 Some Tcl/tk code from the TimeStore prototype.....5
13.9 The power of the Tcl/tk system 11
13.10 Application frameworks 11
13.11 Garnet: an amazingly comprehensive system 13

Ronald Baecker
Professor of Computer Science,
Electrical and Computer Engineering, and Management
University of Toronto

13.1 Architecture of user interface software

Application
Higher-level Tools
User Interface Toolkit
Windowing System
Graphics Library
Operating System
Hardware

13.2 Higher-level tools

Language-based tools
- State transition network
- Context-free grammars
- Event languages
- Declarative languages
- Constraint languages
- Screen scrapers
- Database interfaces
- Visual programming
- Scripting language, e.g., Tcl

Application frameworks, e.g, MacApp, OLE

Model-based generation, e.g, ITS

Interactive graphical specification
- Prototypers, e.g., Director
- Cards, e.g., HyperCard
- Interface builders, e.g., Visual Basic, NeXT
- Data visualization tools
- Graphical editors, e.g., Peridot

13.3 Card-based prototyping tool — HyperCard

Features of HyperCard
- A system accessible to non-programmers
- An interface builder!!!
- A direct manipulation system
- A hypertext system
- A prototyping tool
- An extensible system
- An object-oriented system

Some limitations to HyperCard
- Interface limitations
  - Best for card-based interfaces
  - Not good for interactive text manipulation
  - Not good for sketching, gesture-based interfaces
- HyperTalk language limitations
  - No arrays
  - Little (weird) program structure
  - Few debugging tools

13.4 Time-based prototyping tool — Director

Features of Director
- System for computer animation, structuring images and their changes over time
- Theatre as the unifying metaphor — actors, stage, etc.
- Scripting language — Lingo
- An extensible, object-oriented system

Limitations to Director
- Very low-level specification of interfaces
- Lingo language not much better than HyperTalk

13.5 The NeXT Interface Builder

Widgets predefined as part of object-oriented class library
Interactive selection, positioning, tailoring of widgets
Connecting widgets to represent dependencies,
    communication via message passing
Behaviours coded in object-oriented language
Role in Rhapsody???

13.6 Language-based tools

Various formalisms for describing interaction as part of a
    “User Interface Management System” (UIMS)

State Transition Networks
    Arcs represent possible actions from a given state
    Best in highly-moded contexts where choices are limited

Context-free Grammars
    Best for textual command languages

Event Languages
    Well suited for handling multi-threaded input

Declarative Languages
    Best for highly patterned interactions, e.g., menus & forms

Constraint Languages
    Best for defining relationships between dialogue objects
    Increasingly, constraints build into object systems or
        toolkit intrinsics

Scripting Languages and Environments, e.g., Tcl/tk

13.7 The Tcl language

Similar to other UNIX shell languages such as the Bourne shell, the C shell, the Korn shell, and Perl

Only a few fundamental constructs and relatively little syntax, making it easy to learn

Enough programmability (variables, control flow, procedures) to allow one to create complex scripts

Tcl is also an interpreter for the scripting language

Tcl commands derive additional power from *substitution* — variable, command, and backslash

Tcl commands can be combined into procedures using Tcl control structures

Tcl also has:

- Arrays and lists
- Advanced control structures
- String manipulation
- File access
- Access to Tk widgets
- Access to other X facilities
- Interapplication communication facilities
- C interfaces

13.8 Some Tcl/tk code from the TimeStore prototype

New message module from TimeStore (Figs. 13.1, 13.2, 13.3)

Figure 13.1 Calling the New Message function in TimeStore (Yiu, 1997)

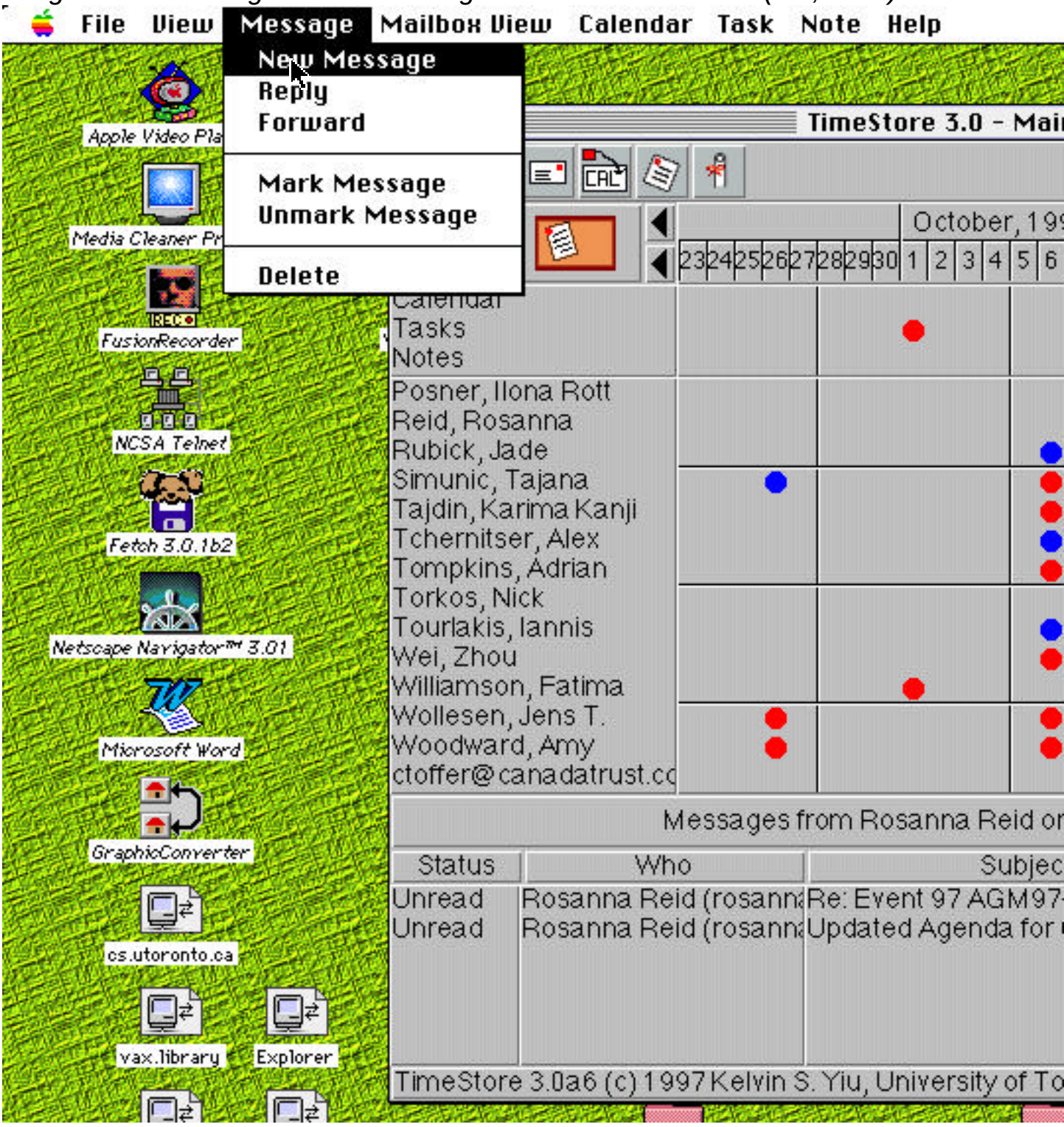

Figure 13.2 A New Message screen from TimeStore (Yiu, 1997)

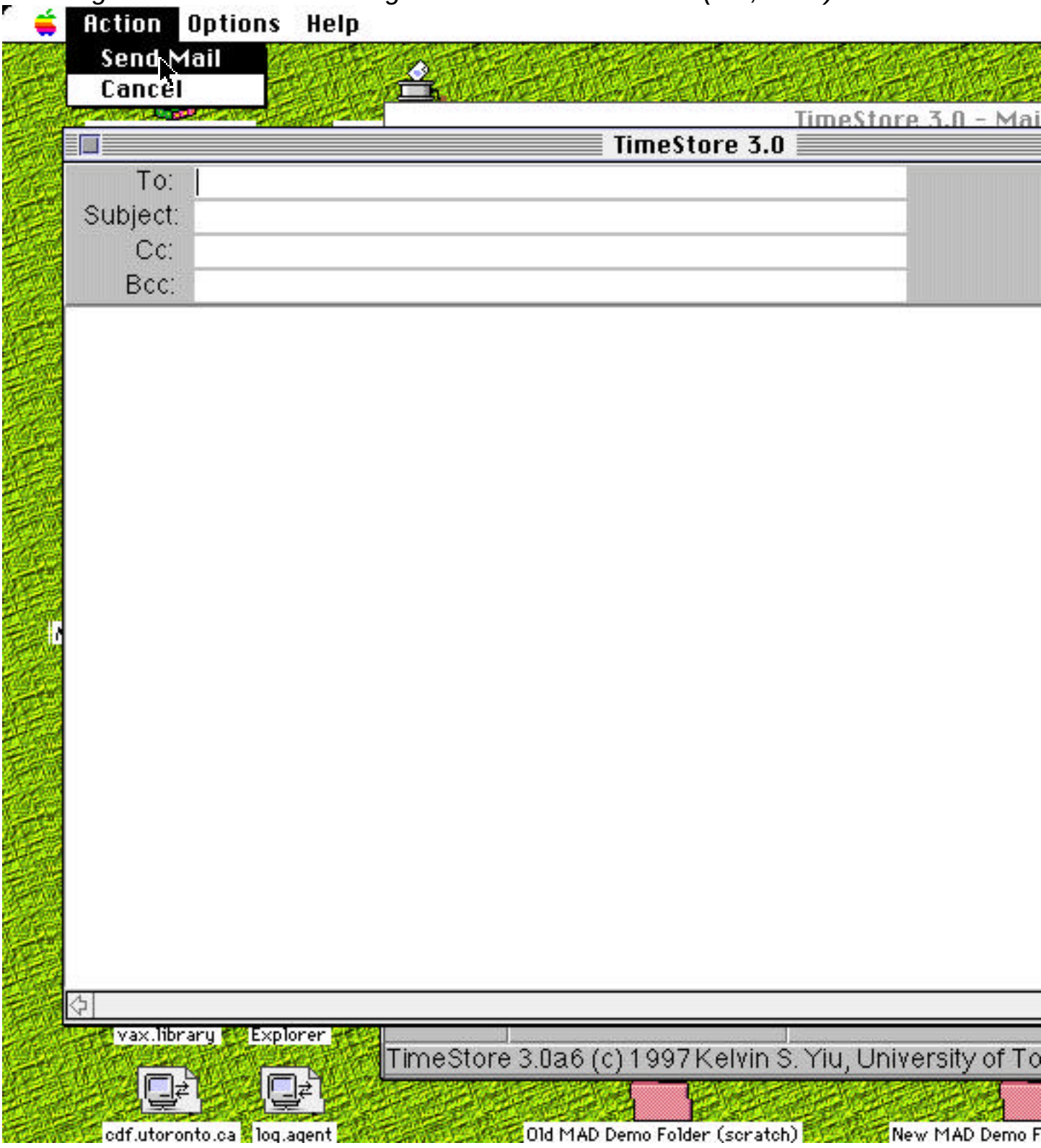

Figure 13.3 The Tcl/tk for the New Message function (Yiu, 1997)

```
#
# FILE: ts_newmail_menu.tcl
#

#eval destroy [winfo child .]
wm title . "TimeStore 3.0"
wm iconname . "TimeStore 3.0"

. configure -bg grey75

# Check for screen resolution. Use different size fonts for different screen
# resolution (ie. a small font for 640x480, a larger font for 1280x1024).
# May use a math formula to scale font size later.

source ts_sysfont.tcl
menu .menu -tearoff 0

# Creates the "Action" Menu, with options "Send Mail" and "Cancel"
# Relates "Send Mail" and "Cancel" to corresponding C functions.

set m .menu.action
menu $m -tearoff 0
.menu add cascade -label "Action" -menu $m
$m add command -label "Send Mail" -command "ts_sendmail"
$m add command -label "Cancel" -command "quit_editor"

# Creates "Options" menu. Note that option "Word wrap" can be checked on or off
# using a procedure "change_wrap" defined in ts_rest_of_editor.tcl.

set m .menu.options
menu $m -tearoff 0
.menu add cascade -label "Options" -menu $m
$m add check -label "Word Wrap" -variable wordwrap -command "change_wrap"

# Creates "Help" Menu. If option "About TimeStore" is chosen,
# it generates a dialog box.

set m .menu.help
menu $m -tearoff 0
.menu add cascade -label "Help" -menu $m -underline 0
$m add command -label "About TimeStore" -underline 0 -command "tk_dialog
     .dialog {About TimeStore} {Version 3.0 alpha 4\n(C) 1997 Kelvin Yiu and
     University of Toronto} {} 0 OK"

. configure -menu .menu
```

```
#
# FILE: ts_mail_headers.tcl
#

# Defines Email Message Headers

frame .header -relief raised -borderwidth 1 -bg grey75
frame .header.l1 -borderwidth 1 -bg grey75

# Define "To:" field and corresponding text area and add them to the header.
set l .header.l1
label $l.to_l -text "        To:" -width 8 -relief raised -font $font1
     -highlightthickness 0 -borderwidth 0 -justify left -bg grey75
entry $l.to_e -relief sunken -borderwidth 0 -highlightthickness 0
     -background white -width 46 -font $font1 -textvariable v1
pack $l.to_l -side left
pack $l.to_e -side left
pack $l -side top -expand yes -fill x

frame .header.l2 -borderwidth 1 -bg grey75
set l .header.l2

# Define "Subject:" field and corresponding text area and add them to the header.
label $l.to_l -text "Subject:" -width 8 -relief raised -font $font1
     -highlightthickness 0 -borderwidth 0 -justify left -bg grey75
entry $l.to_e -relief sunken -borderwidth 0 -highlightthickness 0
     -background white -width 46 -font $font1 -textvariable v2
pack $l.to_l -side left
pack $l.to_e -side left
pack $l -side top -expand yes -fill x

frame .header.l3 -borderwidth 1 -bg grey75
set l .header.l3

# Define "Cc:" field and corresponding text area and add them to the header.
label $l.to_l -text "        Cc:" -width 8 -relief raised -font $font1
     -highlightthickness 0 -borderwidth 0 -justify left -bg grey75
entry $l.to_e -relief sunken -borderwidth 0 -highlightthickness 0
     -background white -width 46 -font $font1 -textvariable v3
pack $l.to_l -side left
pack $l.to_e -side left
pack $l -side top -expand yes -fill x

frame .header.l4 -borderwidth 1 -bg grey75
set l .header.l4

# Define "Bcc:" field and corresponding text area and add them to the header.
label $l.to_l -text "       Bcc:" -width 8 -relief raised -font $font1
     -highlightthickness 0 -borderwidth 0 -justify left -bg grey75
entry $l.to_e -relief sunken -borderwidth 0 -highlightthickness 0
     -background white -width 46 -font $font1 -textvariable v4
pack $l.to_l -side left
pack $l.to_e -side left
pack $l -side top -expand yes -fill x

pack .header -side top -expand no -fill x
focus .header.l1.to_e
```

```
#
# FILE: ts_rest_of_editor.tcl
#

# This code is used when creating new messages, tasks, notes and calendar events.

# Define the text area for editing and the corresponding scrollbar
frame .editor -borderwidth 0
frame .editorbottom -borderwidth 0
text .editor.edit -width 80 -height 25 -relief ridge -borderwidth 1
     -highlightthickness 0 -background white -font $font3 -xscrollcommand
     ".editorbottom.xscroll set" -yscrollcommand ".editor.yscroll set"
scrollbar .editorbottom.xscroll -command ".editor.edit xview" -width 15
     -orient horiz -borderwidth 1 -bg grey75
scrollbar .editor.yscroll -command ".editor.edit yview" -width 15
     -borderwidth 1 -bg grey75
label .editorbottom.panel -bitmap @nothing.bmp -width 15 -height 15 -bg grey75

# Add the editor and the scrollbar
pack .editor.edit -side left -expand yes -fill both
pack .editor.yscroll -side right -expand no -fill y
pack .editor -side top -expand yes -fill both
pack .editorbottom.xscroll -side left -expand yes -fill x
pack .editorbottom.panel -side right -expand no
pack .editorbottom -side bottom -expand no -fill x

# Define the procedure that toggles between allowing
# and disallowing word wrapping.
proc change_wrap args {
    global w wordwrap

    if {$wordwrap == 1} {
      .editor.edit config -wrap word
    } else {
      .editor.edit config -wrap none
    }
}

# set default word wrap
set wordwrap 1

bind . <Destroy> "quit_editor"
```

13.9 The power of the Tcl/tk system

System programming lang.	System integration language (Scripting language)
C, C++, Java	Perl, Tcl/Tk, Javascript
Building components	Plugging together components
Strongly typed	Typeless
Usually compiled	Usually interpreted

Use systems programming language when:
- Need to implement complex algorithms or data structures
- Large data sets are manipulated and speed is critical
- Functionality is well-defined and changing slowly

Use scripting language when:
- Main task is connecting together pre-existing components
- Application includes a GUI (but see other approaches in these lectures)
- Functionality evolving rapidly over time
- Need for extensibility

Scripting on the rise due to GUIs, the Internet, and component frameworks, e.g., ActiveX, OpenDoc, JavaBeans, and due to facilitation of rapid prototyping (Fig. 13.4)

13.10 Application frameworks

Class libraries built on top of a toolkit designed to support particular kinds of applications

Example: MacApp system for implementing applications that adhere to the Mac look and feel (on top of Mac Toolkit)

Example: Garnet (Lisp) and Amulet (C++)

Figure 13.4 Productivity enhancements with Tcl/Tk
(Osterhout, 1997, http://www.sunlabs.com/~ouster/scripting.html, p. 7)

Application	Comparison	Ratio	Comments
Software test and installation	C test application: 272K lines, 120 engineer months. C FIS application: 90K lines, 60 engineer months. Tcl/Perl version: 7.7K lines, 8 engineer months	22-47	C version implemented Tcl/Perl version replac both C applications.
Database library and application	C++ library: 2-3 months Tcl library: 1 week C++ application: 2 months Tcl application: 1 day	8-12 60	C++ version implemen first; Tcl version of libr more functionality.
Display oil well production curves	C version: 3 months Tcl version: 2 weeks	6	Tcl version implement
Query dispatcher	C version: 1200 lines, 4-8 weeks Tcl version: 500 lines, 1 week	2.5-8	C version implemented uncommented. Tcl ver had comments, more f tionality.
Spreadsheet tool	C version: 1460 lines Tcl version: 380 lines	4	Tcl version implement
Simulator and GUI	Java version: 3400 lines, 3-4 weeks. Tcl version: 1600 lines, < 1 week.	2-3	Tcl version had 10-20 functionality, was impl mented first.

Table 1. Each row of the table describes one or two applications that were implemented twice, once with a system programming language such as C or Java and once with a scripting language such as Tcl. The **Ratio** column gives the ratio of lines of code or development time for the two implementations (>1 means the system programming language required more effort). In most cases the two versions were implemented by different people. The information in the table was provided by various Tcl developers in response to an article posted on the comp.lang.tcl newsgroup. Scripting languages showed less improvement in situations where they were used for the first implementation, most likely because the second implementation benefited from the experiences of the first.

13.11 Garnet: an amazingly comprehensive system

The mother of all interface software tools (Fig 13.4)

Garnet video (Siggraph Video Review 97, #13)

Fig. 13.5 Structure of the Garnet system (Myers et al., 1990, from BGBG, p. 359)

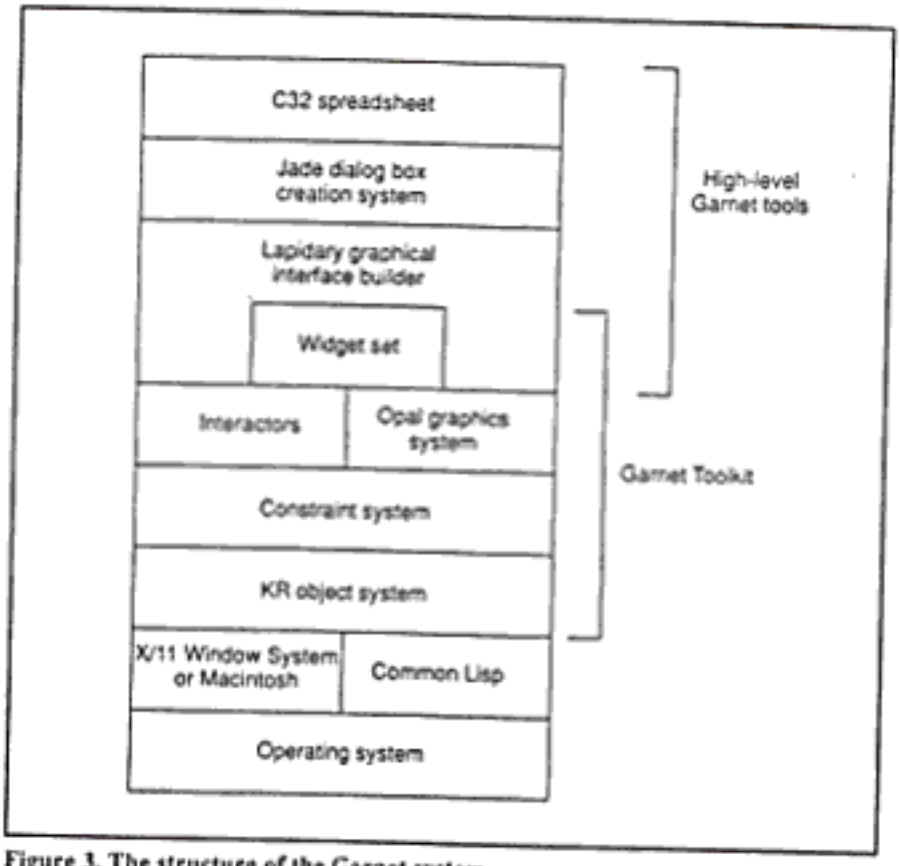

Figure 3. The structure of the Garnet system.

Lapidary
Draw graphics
Demonstrate inter-active behaviour
(programming by example)