Using Benchmarks to Teach and Evaluate User Interface Tools

Brad A. Myers,
Neal Altman, Khalil Amiri, Matthew Centurion, Fay Chang, Chienhao Chen, Herb Derby, John Huebner, Rich Kaylor, Ralph Melton, Robert O'Callahan, Matthew Tarpy, Konur Unyelioglu, Zhenyu Wang, and Randon Warner

Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
bam@cs.cmu.edu
http://www.cs.cmu.edu/~bam

ABSTRACT
As part of the User Interface Software course in the Human-Computer Interaction Institute at Carnegie Mellon University, the students and instructor developed a set of 7 *benchmark tasks*. These benchmarks are designed to be representative of a wide range of user interface styles, and have been implemented in about 20 different toolkits on different platforms in different programming languages.. The students' written reports suggest that by implementing the benchmarks, they are learning the strengths and weaknesses of the various systems. Implementations of the benchmarks by a number of more experienced toolkit users and developers suggest that the benchmarks also do a good job of identifying the effectiveness of the toolkits for different kinds of user interfaces, and point out deficiencies in the toolkits. Thus, benchmarks seem to be very useful both for teaching and for evaluating user interface development environments.

Keywords: Benchmarks, Teaching User Interfaces, Evaluation, HCI Education, Toolkits, User Interface Development Environments.

INTRODUCTION

We have developed a set of benchmarks that cover a variety of user interface styles. Implementing these benchmarks in different toolkits has proven to be a good way for students to learn the features, strengths and weaknesses of different toolkits. The benchmarks are also useful for toolkit creators to evaluate the coverage of their tools, and for developers to evaluate different toolkits.

The "normal" way to teach a User Interface Software course, which has been used in previous semesters, is for the students to do a big group project, so they learn how to use one toolkit in depth. However, the students complained that they did not learn enough about *different* toolkits, and wanted to see how toolkits differed, and the strengths and weaknesses of each. Therefore, the graduate "User Interface Software" course in the Human-Computer Interaction Institute at Carnegie Mellon University is using a new approach for the Spring 1997 semester.[1] The students each individually designed a benchmark as one of their first assignments. These were revised and combined to result in a set of six benchmarks which test a wide range of user interface styles and requirements. An older benchmark from the instructor was added to the set. Each student then independently implemented a benchmark of their choice using four different toolkits. Each implementation was allocated three weeks, and most students were able to learn a toolkit from scratch and create a full implementation of the benchmark in about 20 to 30 hours over the three weeks. The students report that the benchmarks were of the appropriate difficulty, and that they are learning a lot about the various toolkits.

(Note to reviewers: The course will be over in May, so by the final paper deadline, we will have at least 14 more numbers for the tables, as well as more comments. We also expect to get a few more implementations by experts, and a few new toolkits.)

So far, the students have used the following toolkits to implement the benchmarks: Visual Basic, Director, HyperCard, tcl/tk [10], Java AWT, SubArctic [5], Delphi from Borland, MetroWorks PowerPlant for Macintosh,

[1] A description of the course is available at http://www.cs.cmu.edu/~bam/uicourse/1997spring/index.html which includes a link to the assignment for creating the benchmarks.

Microsoft Foundation Classes (MFC) for Windows, Win32 for Windows, and Sk8 for Macintosh.

In order to investigate whether the students' observations about the benchmarks and toolkits were consistent with the results from experienced users, we asked various "experts" to implement the benchmarks in their favorite toolkits. Experienced toolkit users have implemented at least one of the benchmarks in Garnet [8], CLIM [6], Apple's MacApp, GINA++ and GINA/CLM [11], LispView from Sun, tcl/tk, XPCE/SWI-Prolog [12], MrEd [2], and Amulet [9]. The results suggest that the benchmarks are useful for toolkit designers and evaluators, and are helpful when evaluating and creating toolkits. For example, a few developers commented that the benchmarks helped highlight areas where their toolkits were missing important features for certain styles of applic ations.

THE BENCHMARKS

The goals for the benchmarks were that they should be implementable in less than 8 hours by an expert, that the set of benchmarks cover a wide range of user interface styles, that the description should be specific enough so that all implementations of a benchmark would be alike in all essential details, and finally that the benchmarks would be fun to implement. A variety of user interface textbooks were used to help identify the various possible styles of user interfaces, to insure good coverage.

The first assignment in the course was for each student to independently specify a benchmark that met the goals listed above. After the initial benchmarks were graded, we formed groups and combined the benchmarks that addressed the same styles of interface. The result was six benchmarks that cover a variety of interface styles: direct manipulation, forms, animation, text editing, painting (bitmap editing), and multimedia. A seventh benchmark was added, which had been created earlier by the first author to test previous toolkits. Of course, a number of other benchmarks should be added to cover other kinds of user interfaces, such as 3D and multi-user applications, as discussed below.

Note that although the first author of this paper is a toolkit creator, all but one of the benchmarks were created by the class members before they were familiar with the toolkits, and therefore the benchmarks are less likely to be biased in favor of any toolkit.

The next sections describe the seven benchmarks we have so far. A complete description of all the benchmarks is available at `http://www.cs.cmu.edu/~bam/uicourse/1997spring/bench/index.html`

Direct Manipulation Benchmark

The direct manipulation benchmark is a simple interface for playing card games. It is representative of tasks for which the user must position and operate on objects that are close analogies to real-world objects. In these tasks, the spatial relationships of objects are significant, and the objects can be manipulated with the mouse. Often some objects have fixed positions in the workspace, and the act of placing other objects into them or onto them triggers some behavior (for example, dragging an object to the trash can). There should be feedback about what will happen, which is often called "semantic feedback." Another important issue is the Z-order so that overlapping objects are handled appropr iately.

Aspects of this style occur commonly in direct-manipulation interfaces. For example, in the Macintosh Finder, moving files into folders triggers an actual file-system operation. In the Windows Solitaire card game, only certain moves are legal, and the rules of the game require a distinction between cards that are on top of other cards and cards that are underneath.

In summary, this benchmark tests:

- Dragging objects with the mouse.
- Invoking different actions depending on where the objects are dropped ("drag and drop").
- Providing feedback about the operation that will happen if the objects are dropped (semantic feedback).
- Maintaining and changing the Z-order of objects.
- Loading and viewing GIF color pixmap images.

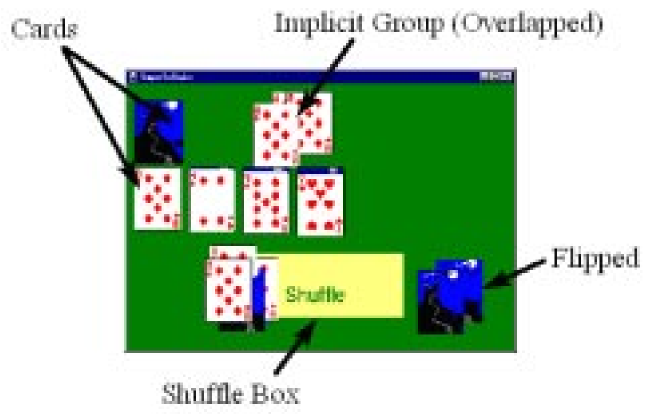


Figure 1: Example screen from the benchmark for direct manipulation implemented in Visual Basic.

This application will present a window containing 52 objects representing playing cards. The user can move them around the window, "flip" them to show either the front or the back, and move a group of cards into a "shuffle box" object that rearranges them (showing semantic feedback when the objects are dragged over the shuffle area before the mouse button is released). Cards can also be brought to the front of the Z-order. They are implicitly grouped for movement and flipping based on the way they overlap: if the user clicks on an object and drags it, all the objects that are on top should move as well. Figure 1 shows one possible implementation. So that the programmer doesn't have to waste time with drawing, 53 GIF format files are provided with all the card faces and the back of the cards. [2]

Forms Benchmark

This benchmark is a set of forms, mimicking the user interface for an installer program. Many database front ends, data entry, and monitoring systems use such forms to structure the interaction with the user.

This benchmark tests several aspects of a user interface toolkit:

- The ability of a toolkit to create a structured set of forms with various widgets.
- How hard it is to specify complex dependencies between the elements of the forms.
- The ability to handle constraints on the values that are allowed for various fields.
- The ability to have dependencies among the widgets, such as the default value or enabling of one widget depending on the value of other widgets.
- The ease with which form structures can be reused in multiple parts of the interface.
- The ease with which toolkits can provide a user with control over time-consuming processes without locking up the interface.
- Support for accelerators such as a default button in the dialog boxes, and hitting the TAB key to move from field to field.
- Support for specific kinds of widgets and containers, such as "option buttons" that pop up a set of choices, and putting check boxes into a scrolling group inside a dialog box.

This benchmark pretends to install five files, called Editor, Spell Checker, English Dictionary, French Dictionary, and Help. It obeys the constraints that the Editor must be installed, and the Spell Checker may only be installed if one of the Dictionaries is installed.

There are two primary paths through the installer, the Easy Install path and the Custom Install path. In the Easy Install path, the user is given no choice about what files can be installed; the files that are installed are the Editor, the Spell Checker, and the English Dictionary. In the Custom Install path, the user gets a choice of what files to install, and after installing them, the user may return to install more files. A scroll bar shows the progress of the (pretend) install, during which time the user is allowed to abort by hitting the cancel button.

[2] We felt GIF was the most portable format, but it would be easy to supply alternative formats, if necessary. The files are available with the benchmark descriptions.

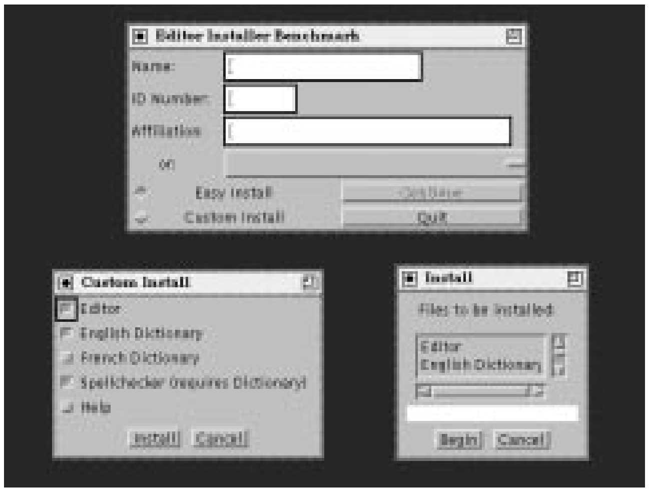


Figure 2: Example of three screens from the benchmark for forms implemented using Java AWT.

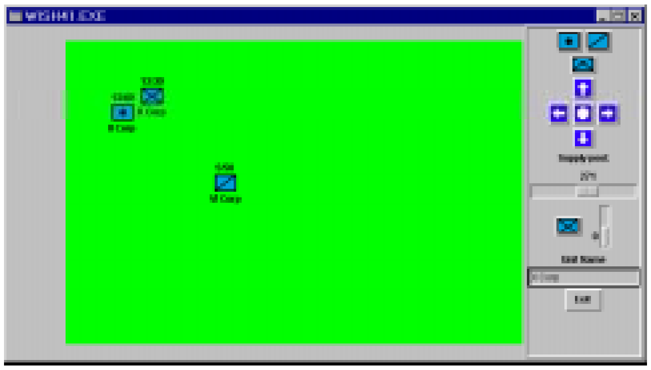


Figure 3: Example screen of the benchmark for animation implemented using tcl/tk on a PC.

Some general features are that hitting the tab key should move from field to field in a reasonable order, so it is easy to enter the values. The default action in each form (usually OK or Install) should be denoted (usually by a thicker border) and hitting the RETURN (or Enter) key should activate it. Figure 2 shows some sample screens from one possible implementation.

Animation Benchmark

This benchmark uses a war game setting, which is representative of many games. It is designed to test:

- How well the toolkit handles animations and other time-based behaviors.
- How easy it is to create new objects dynamically at run time and to select objects showing custom feedback.
- Having multiple objects constrained to move t ogether.

To focus the benchmark on interaction with animation, we have simplified the game and simulation aspects. Also, the icons are quite simple so no time is needed to draw fancy pictures.

In the simulation for the benchmark, troops have the properties of position, velocity, velocity maximum, food, food maximum, name, type, and a list of orders. The battlefield has width, height, a main food storage, and a list of troops. On each turn, the simulation updates all the dynamic properties of the objects in the simulation according to wall clock time. The user can create new troops, and specify their strength, velocity and type. Users can select a troop while it is in motion and update its speed and direction. Troops die when they run out of food. Figure 3 shows an example screen.

Text editing Benchmark

This is a simple text editor with which users may open one document to edit. It supports multiple fonts and styles, and mouse-based text editing. The interface styles include menus, dialog boxes, and shortcuts (function keys and tool-bars). This benchmark is representative of the interfaces found in such programs as Notepad and Wordpad (Windows 95 and NT 4.0). In addition to this, we feel that this benchmark is indicative of most routine text editing tasks.

The main feature that makes this benchmark be more than just the editor that is supplied by the toolkit is the requirement that selected text can be dragged to the toolbar to change the formatting. The intention is to see if customized editing manipulations can be easily added to the built-in facilities.

In general, the editor should have the look and feel of the underlying system and toolkit in which it was created. It should have a standard menu with File, Edit and Format menus, which contain the usual Open, Save, Exit, Cut, Copy, and Paste operations, and Bold, Italic and Underline. There should also be a toolbar containing icons for at least Bold, Italic and Underline. Figure 4 shows an example.

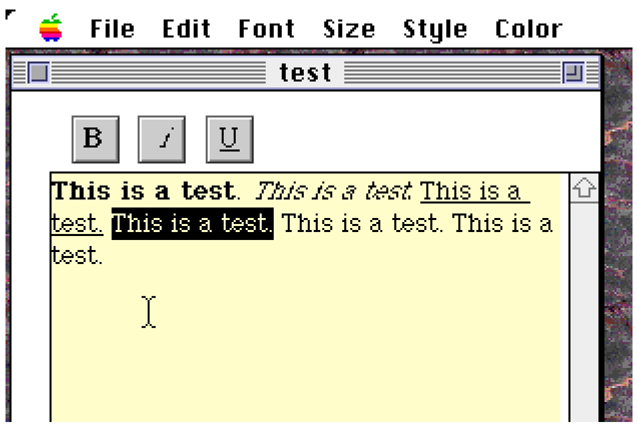

File
Edit
Font
Size
Style
Color
test
B
I
U
This is a test. *This is a test.* This is a test. This is a test. This is a test. This is a test.

Figure 4: Example implementation of the benchmark for text editing using CodeWarrior’s PowerPlant on a Mac.

In summary, this benchmark tests:

- Support in the toolkit for multi-font, multi-line mouse-based text editing.
- Support for cut, copy and paste of text to the system’s clipboard or “cut buffer.”
- The customizability of the text editing facilities to support application-specific requirements.

Painting Benchmark

Although most toolkits have built-in support for *drawing*, some do not support *painting*. The distinction is that painting requires the ability to set, edit and clear individual pixels in a bitmap (sometimes called a “pixmap” if in color). The picture must be saved in an offscreen buffer in case the drawing on the screen is covered and uncovered by a different window. This benchmark is representative of interfaces used for creating and editing images stored as bitmaps. It will allow the user to draw lines, paint lines with a square brush, and erase portions of the bitmap in a single drawing window. The user will be able to select from at least four colors and three line thicknesses from pallets. The user may also select rectangular regions of this window to cut, copy, or paste. It is similar in style to Microsoft Paint, KidPix, MacPaint, PaintShopPro, Adobe Photoshop and Fractal Painter. Figure 5 shows an example.

Some things in most painting tools that are not included are drawing curves, spray painting, entering text, filling regions, polygons, etc.

In summary, this benchmark tests:

- Support for editing individual pixels of the screen.
- Support for copying and drawing to off-screen buffers.
- Support for drawing with various colors and sizes of objects (different “brushes”).
- Support for cut, copy and paste of images to the system’s clipboard or “cut buffer.”

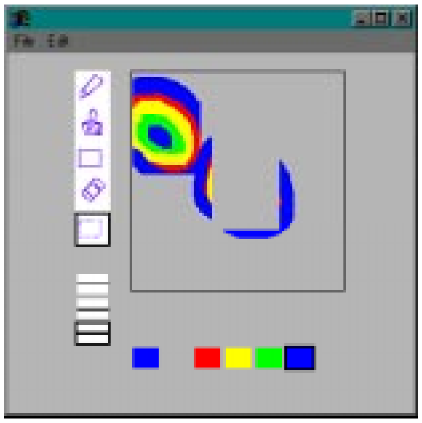


Figure 5: Example of the benchmark for painting impl e-mented with Java AWT on a PC.

Multimedia Benchmark
The multimedia benchmark is designed to test whether the toolkit can handle video and audio. The interface that this benchmark more closely resembles is that of a publicly accessible kiosk such as found in museums, corporate offices, and malls. The style is representative of almost any multimedia-based content program such as encyclopedias and dictionaries ("Encarta," "Bookshelf"), and some games. Specifically, the benchmark is a kiosk for a commercial establishment that provides navigation and store information to the public.

The main part of the display is a map of an imaginary mall. When a store is clicked on, its video can be played in the center. The benchmark also has a database component, which allows each user to log in with a unique name, and then allows the user to type a note associated with each store. These notes are then displayed the next time the same user logs in and clicks on the store.

This benchmark tests:

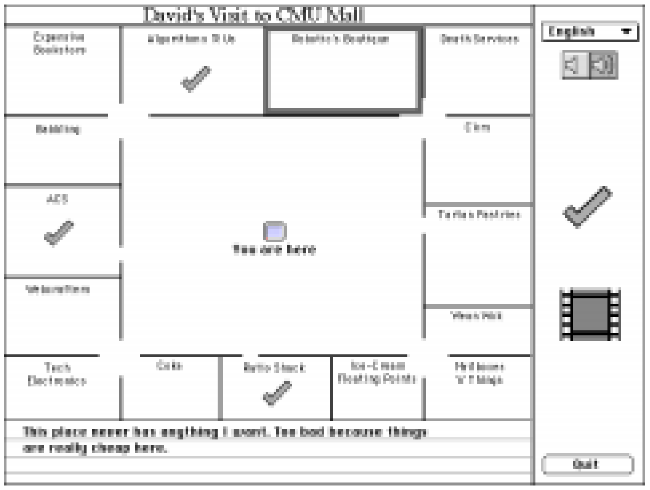
- Support for loading and playing video.
- Support for playing audio synchronized with the video and with user interactions.
- Support for creating a simple database of user annotations which are saved to disk between different executions of the benchmark.

Figure 6 shows an example of the benchmark. To eliminate the time spent on creating the media, sample video and audio are provided in various formats.

Graph Editing Benchmark
As an attempt to evaluate the Garnet user interface development environment, the first author created a benchmark in 1990 and had it implemented using 6 toolkits in 1991 and 1992 [8]. The benchmark description was reformated for the world-wide-web, and included in the list to see if toolkits have changed in the last 7 years.

This benchmark is a simple graph editor, with nodes connected by arrow-lines. The user should be able to select nodes and lines, and delete them and change their line thicknesses. The nodes can be resized and moved, and the lines must stay attached appropriately. Figure 7 shows an example implementation. What this benchmark tests beyond what is in the Cards direct manipulation benchmark is:

- The standard graphical editing mechanisms (selection handles, moving and growing objects).
- Creating new objects dynamically.
- Constraints for keeping the lines attached to objects and the text at the top of the objects when the objects are moved or changed size.
- Support for rounded rectangles, non-editable text fields, and multiple line styles for lines and rounded rectangles.



This place never has anything I want. Too bad because things are really cheap here.

Figure 6: Example of the benchmark for multimedia i m-plemented on a Macintosh with MetroWerks' PowerPlant.

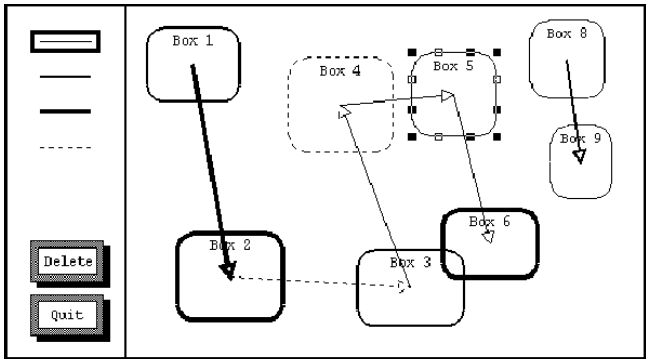


Figure 7: Example of the graph editing benchmark i m-plemented using Garnet on Unix.

THE TOOLS
As part of the class, the students have so far used 11 different tools, and they will be using 2 or 3 more before the end of the term. Figure 8 summarizes all the tools and benchmarks used by the class. For their first assignment, they had to choose a prototyping tool, like Visual Basic, HyperCard or Director (most used Visual Basic). For the next two assignments, they could pick any two tools they wanted. For the last assignment, everyone will be using Amulet, which was chosen because it is a modern, research toolkit that incorporates many features, including constraints, support for Undo, and high-level input and animation models, not found in typical commercial toolkits.[3]

In addition, we have posted requests on various newgroups and mailing lists to try these benchmarks. As a result, "experts" have used 10 tools to implement at least one of

[3] Amulet is being developed as a research project by the course instructor, who is the first author of this paper.

the benchmarks.[4] Figure 9 summarizes their results. The comments generated as a result of these implementations do a good job of highlighting the strengths and weaknesses of the various tools.

It is interesting to note that some vendors declined to try the benchmarks because they said it would take too long since their tools were designed for more complex applications. This seems to be inappropriate — just because a tool can handle complex applications does not mean that it should be hard to learn, or hard to use for small applications!

The following are some observations about the tools that were used to implement the benchmarks:

Visual Basic — Many of the students used Visual Basic Version 4.0 for Windows 95 and NT to prototype the benchmarks. Most students found it to be very easy to learn and appropriate for most of the benchmarks. It is best for forms-based applications with fixed-format screens of widgets, but there is also support for animation and dynamic graphics, and the resulting performance was fine except for the painting benchmark. There is a rich set of predefined objects and the interactive editor was easy to use to lay out the interface. Problems include the difficult and inconsistent syntax of the Visual Basic programming language, lack of object-orientation, lack of data structures and linked-lists, lack of flexibility in the drawing primitives, and no support for selection handles and resizing objects. There are many Visual Basic books, and the students used a variety of them. None seemed particularly excellent or comprehensive, and many important topics were not covered by each book.

Director — The students found MacroMedia Director version 5.0 to be too hard to learn for use as a prototyping tool. Most of the students gave up after just a few hours of trying because it looked so hopeless, and the user interface, documentation, and design of the "Lingo" language were reported to be "terrible" and "pathetic", just "a hodge-podge of ideas." Limitations of the tool, such as only 48 "sprites" (moving objects) and no apparent method for performing required actions (such as changing the picture of a sprite) seemed insurmountable. The only things that resulted were a few movies. An expert in using Director told the class that it could, in fact, do all the prototyping required, but that it might require a long time to learn how.

Hypercard — One student tried Apple's Hypercard version 2.0 for the Macintosh. The interactive editor for placing widgets and graphics was easy to use, but Hypercard seems quite limited by today's standards, and does not have a complete set of widgets. The syntax of the built-in HyperTalk language was hard to use and the error messages were poor.

Java AWT — Many of the students tried out Java and its toolkit called AWT. Java is a new object-oriented programming language and AWT is its cross-platform widget and drawing library. The students used various versions of AWT (v1.02 on Unix, and v1.02 and v1.1 on a PC), and found it still quite fragile and unstable. Due to the limited set of provided widgets (e.g., no ability to have pictures in buttons, an inadequate text-editing widget), and difficult-to-learn layout managers, AWT currently seems most appropriate for small, simple applications. The performance of the applications was somewhat slow, and compilations took a very long time on some platforms. One student used Semantec Visual Café v1.0 and found it to be an effective development environment with a nice interactive layout editor, but it was buggy and not yet adequately documented. The other students just used a Java compiler. In learning AWT, the main problem was the poor documentation. The advantages are the nice language (Java) and that the resulting programs will run on any platform as well as on the World-Wide-Web.

SubArctic — SubArctic [5] is a new research toolkit for Java that hides much of the complexities of AWT. Advantages include lots of built-in capabilities, including animation and constraints. SubArctic is a very powerful and flexible toolkit. Problems include bugs in various implementations of Java and AWT, unacceptably slow performance of the resulting application, and difficulties in figuring out which of the many possible ways of implementing something would actually work.

Delphi — Borland's Delphi runs on Windows NT or 95 and is an interactive environment, like Visual Basic, but the language is Object Pascal and it is fully compiled. The students used Delphi Developers Version 2.0. The students found Delphi easy to learn, even for those that did not know Pascal. Delphi seems most appropriate for Forms-style applications. The graph benchmark was difficult because there is no line component, it is hard to manage dynamically created objects, and there is no support for selection handles or resizing objects. However, the implementation seemed to be more robust than the Visual Basic implementation of the same benchmark. Code also seems to run faster than with Vi sual Basic.

PowerPlant — MetroWerks's PowerPlant version 1.3 is a framework that works with its CodeWarrior environment for C++ (version 11) on the Macintosh. PowerPlant includes an interactive interface builder for laying out widgets. The students reported that PowerPlant is "great provided you know Object-Oriented programming very well" and it was much easier than using the Macintosh Toolbox directly. However, PowerPlant is quite hard to learn since

[4] We would very much like to get more implementations, so please try out one or more benchmarks and send the results to `bam@cs.cmu.edu`.

the documentation is inadequate, and requires that the programmer use the basic "Inside Macintosh" QuickDraw documentation for graphics.

Microsoft Foundation Classes (MFC) — Microsoft's Foundation Classes (MFC) is a C++ framework for developing Windows applications. The students used MFC 4.0 with Visual C++ 4.0 (the compiler environment) on a PC. There is a visual dialog box interface builder, which makes forms style applications quite easy, and good support for text editing, but there is little built-in support that helps with any of the other styles or benchmarks. The AppWizard and ClassWizard in MFC were somewhat useful for generating an outline of some of the code, but using the AppWizard properly seems to require understanding the complete MFC class hierarchy. The students complained that "the VC on-line documentation system is just pathetic," which hindered learning. Although the students were able to create some of the benchmarks, they found MFC to be *very* difficult to learn, and Microsoft reports that it takes 4 to 6 months to learn to use MFC effectively. More fundamentally, MFC suffers from too many inconsistent ways to perform similar actions, probably because it has to be backwards compat ible.

Win32 — Win32 is the lower-level interface to the Microsoft Windows toolkit used by MFC. One student implemented the Cards benchmark directly in Win32 in C, without using MFC. Programming at this level seemed more difficult than MFC, and shares many of the disadvantages of MFC mentioned above.

tcl/tk — Tcl/tk [10] uses its own scripting language (tcl) which is interpreted, and applications created with it will work on PC, Unix or Mac. The students used various versions (7.3/3.6 on Unix, 7.5/4.1 on Windows and Unix, and 7.6.0b2/4.2 for Linux). They reported that tcl/tk was appropriate for small tasks like the benchmarks, and was fairly easy to learn. There are sufficient built-in widgets to cover what was needed. Problems include difficulty in debugging, the requirement of using lots of global variables, total lack of support for modularity, figuring out the right quoting and balancing braces due to the macro-like language, and poor error messages and debugging facilities. A tcl/tk expert also implemented one benchmark (the Cards) using version 8.0a2 and reported that tcl/tk is good for small to medium-size applications where performance is not a critical issue (the performance of the Card benchmark shuffling was reported to be "a bit slow"). The best and worst feature is that the code is unstructured, so it is easy to glue together components and to create a front end to an existing tool, but it is a poor language for large systems.

Sk8 — This tool, pronounced "skate," is an object oriented toolkit that lets users create programs for the Macintosh. It was developed as a research project by Apple (www.research.apple.com) and never officially released. One student used version 1.2 to implement the multimedia benchmark. Although implemented in Common-Lisp, Sk8 provides the programmer with a "English-like" language like AppleScript or HyperTalk which supports a prototype-instance model. There is an interactive GUI builder and good support for different kinds of media including sounds, pictures, and Quicktime movies. Sk8 enabled full implementation of the Multimedia benchmark in fewer lines of code than any of the other tools. On the down side, many parts of the documentation are incomplete or unavailable, and some of its GUI tools are still buggy.

XPCE/SWI-Prolog — XPCE [12] is a toolkit for Prolog, and version 4.9.2/2.8.0 was used by the designer of the toolkit running on the Linux version of Unix on a PC. The developer reported that the toolkit is most appropriate for graphical editors (like the Cards and Graph benchmarks). The developer said that trying the Paint benchmark with the tool was very useful since it "pointed to a weakness in the image change-forwarding code that was easily fixed, making a wider range of applications feas ible."

MrEd — MrEd [2] is a cross-platform toolkit, and version 49/14 was used by the designer of the toolkit on Linux on a PC to implement the Cards and Text Editing benchmarks. The designer reports that MrEd is best for medium to large GUI applications (although the numbers shown in Figures 8 and 9 show MrEd is competitive, so it seems OK for small applications as well). The styles supported make it best for direct manipulation and text editing applications. MrEd uses Scheme as its programming language which requires a lot of memory at run time.

Amulet — Amulet [9] V3.0 for C++ runs on Unix, Windows NT or 95, or the Mac, and the designer of the toolkit (the first author of this paper) used it to implement one benchmark. Amulet incorporates novel object, input, output, constraint, animation, and undo models, which make it appropriate for direct manipulation and animation tasks. (Adding full multi-level Undo to the graph benchmark only increased the size to 241 lines of code, which is still smaller than other implementations that do not provide any undo.) Amulet is also designed to be easy to learn for students, and it includes an interactive builder. Amulet V3.0 does not support text editing, multi-media or painting, so it is not appropriate for those benchmarks.

Garnet — Garnet [8] is a toolkit for Lisp and runs on Unix or Macintosh. Garnet was created by the first author. It is the predecessor to Amulet and is most appropriate for creating direct manipulation, forms and text editing applications. It has built-in constraints and a high-level input model. The graph benchmark was implemented in Garnet in 1990.

CLIM — CLIM [6] is a Lisp toolkit developed by Symbolics which uses the CLOS Common Lisp Object System. In 1992, the graph benchmark was implemented by a CLIM designer using version 1.1 which has high-level support for direct manipulation editors.

MacApp — MacApp is a framework for creating Macintosh applications. In 1990, the graph benchmark was written using MacApp v2.0 which used Object Pascal. MacApp provides high-level support for commands and Undo, but not for graphics or interaction.

GINA++ —GINA++ [11] is a research toolkit in C++ from the German National Research Center for Computer Science. In 1992, the graph benchmark was implemented using version 1.2. GINA++ provides a high-level output model, including support for selection handles around a rectangular object, and a sophisticated Undo model.

CLM/GINA — CLM/GINA [11] is a Lisp implementation of the GINA toolkit from the same group as GINA++. It has similar properties to GINA++.

LispView — LispView is a CLOS interface to the XView toolkit from Sun. In 1990, the graph benchmark was implemented using version 1.0.1. LispView mainly provides an interface to the widgets and underlying graphics primitives, and does not have high-level support for object redrawing, rubber-band lines or interaction.

THE EVALUATIONS

After implementing the benchmark, the implementers filled out a fairly long questionnaire, which asks about the implementation, the toolkit and the benchmark.

The answers from the students show that they got a very good understanding of the various toolkits from the quick implementation of the benchmarks. They were able to identify the strengths and weaknesses of the tools, and were able to evaluate what kinds of interfaces would be most appropriate to implement using that tool.

Figure 8 shows a summary of the numerical results for the number of lines of code, and the time the students spent on the implementation. This shows that the students were mostly able to create complete implementations of the benchmarks in about 16 hours of work (ranging from 7 to about 30 hours), and it took about 500 lines of code to completely implement the benchmarks, ranging from 90 lines to about 1200. Figure 9 summarizes the numbers reported by the experts on implementations of the benchmarks with various toolkits.

Given the quite small sample, it is probably not appropriate to use these numbers as a way to compare toolkits, since the individual differences among the programmers may dominate the differences between toolkits. Also, it is important to note that all of these are self-reported times, so there is some chance of underreporting, especially for the longer times which people might feel embarrassed to report. Still, some trends are interesting:

- The Paint benchmark seems to be harder to implement in most toolkits. It was especially difficult to implement the cut and paste feature. The text editor benchmark seems easy, but only if the toolkit has an adequate and customizable text widget.
- Most students found Visual Basic to be easy to use and learn, and to be sufficient to implement benchmarks like these. The other "prototyping" tools (Director and HyperCard) were much less successful.
- The interactive and interpreted environments, like Visual Basic, tcl/tk and Delphi seem to result in less work and fewer lines of code than the compiled environments like Java, PowerPlant or MFC.
- The range of the number of lines of code seems fairly consistent between the students and the experts (Figures 8 and 9), which suggests that the students' implementations are reasonable.
- Comparing the 1990 implementations of the graph benchmark with the newer toolkits shows that not much has changed — Amulet, like its predecessor Garnet, is good for this style of program, and it still takes a fair amount of time and lines of code with other toolkits. Visual Basic appears to be about as good as the earlier Lisp toolkits.

LIMITATIONS AND FUTURE WORK

An important limitation is that the benchmarks do not cover all types of interfaces. In particular, interfaces for distributed and multiple-user applications are not covered, and neither are 3D interfaces. A vendor claimed that none of the benchmarks were appropriate to his toolkit designed for process-control software. It would be great to have additional benchmarks for these other types of applications, which would then be added to the set for use in evaluating other toolkits.

Another important limitation is that benchmarks mainly measure the learnability and suitability of the tools for small applications. This leaves out issues of structure, design, analysis, and performance that become more important for large applications. However, we feel that *all* tools should still be easy to learn, so at least the benchmarks will be an appropriate test of some aspects. Also, the analyses by the students suggest that even with programs the size of the benchmarks, they were able to see where the problems of scale would arise with toolkits like Visual Basic and tcl/tk.

In using the benchmarks for teaching the course, some students now complain that they were not learning any tool in depth, as they would if there was a single large project using one tool. Clearly, there is a trade-off, and possibly a course could be designed with two implementations of the benchmarks (3 weeks each) and then a bigger project (for 6 weeks).

key: **bold**=full implementation, normal=about 90% implemented; *italic*=very partial implementation

Lines of Code	Visual Basic	Director	Hyper Card	tcl/tk	Java AWT	Sub-Arctic	Delphi	Power Plant	MFC	Win32	Sk8
Animation	**450 / 565 / 360**			**398 / 629**	1003 / **585**				**350**		
Card Game (DM)	**300**	*10*			300	**428**			**600**	**530**	
Connected Graph	313						568				
Forms	**329 / 354**		200	**483 / 399**	**765**		**316**		**141***		
MultiMedia	86							800			**97**
Paint	454	*12*			**1220**			**1128**	808 / **940***		
Text Edit	90	*74*		**90**		*100*		**100**			

Implementation Time (in hours)	Visual Basic	Director	Hyper Card	tcl/tk	Java AWT	Sub-Arctic	Delphi	Power Plant	MFC	Win32	Sk8
Animation	**4.0 / 8.0 / 3.0**			**21.75 / 12.0**	10.25 / **18.9**				**7.0**		
Card Game (DM)	**7.0**	*5.0*			11.0	**13.5**			**6.0**	**15**	
Connected Graph	5.25						11.0				
Forms	**5.5 / 14.0**		7.0	**13.0 / 14.0**	**17.0**		**18.73**		**30.0**		
MultiMedia	11.0							32.0			**10.2**
Paint	21.0	*10.5*			**20.0**			**27.71**	27.6 / **31.0**		
Text Edit	2.0	*7.0*		**30.0**		*7.0*		**6.0**			

Figure 8: Results for the number of lines of code, and time (in hours), for the implementations by the students of the benchmarks. Some benchmarks were implemented by more than one student, and their numbers are separated by slashes (/). Lines of code marked with an asterix (*) do *not* include code automatically generated by Wizards and interactive tools.

Lines of Code	Garnet	CLIM	MacApp	GINA++	LispView	CLM, GINA	tcl/tk	XPCE/ SWI-Prolog	MrEd	Amulet
Animation										
Card Game (DM)							**206**		**287**	
Connected Graph	**183**	**331**	**1026**	**550**	**500**	**273**				**212**
Forms										
MultiMedia										
Paint								**480**		
Text Edit									**177**	

Implementation Time (in hours)	Garnet	CLIM	MacApp	GINA++	LispView	CLM, GINA	tcl/tk	XPCE/ SWI-Prolog	MrEd	Amulet
Animation										
Card Game (DM)							**4**		**2**	
Connected Graph	**2.5**	**4.5**	**9**	**16**	**16**	**20**				**1.42**
Forms										
MultiMedia										
Paint								**4.3**		
Text Edit									**1.3**	

Figure 9: Results for the number of lines of code and time (in hours) for the implementations by experts using various toolkits. The first six implementations were performed in 1991 and were previously published [8], and the others are new.

RELATED WORK

There is very little prior discussion about ways to teach a user interface software course, outside of the SIGCHI Curricula for HCI [3]. Benchmarks are widely used for evaluating the performance of hardware and compilers (such as the SPECMark [13]), but we do not know of any previous attempts at using benchmarks for evaluating the effectiveness of tools or libraries. Programs that print "Hello World" using the terminal output functions (like `printf`) have been implemented using 188 different languages http://www.latech.edu/~acm/HelloWorld.shtml), and programs that print the lyrics to "99 Bottles of Beer" have been implemented in 180 languages (http://www.ionet.net/~timtroyr/funhouse/beer.html) but these do not test any UI toolkits. There have been many studies of programmers (e.g., [1]), which consistently show an order of magnitude difference in capabilities, which is why the numbers in the tables might not be reliable. A comprehensive survey for evaluating toolkits [4] was developed, but it seems much more fun and educational to implement a benchmark than to fill out a long survey. Surveys will also highlight different aspects. Surveys might identify the features which are present or missing, but are less likely to show which toolkits are easier to learn or are more effective for which types of applications. There are many previous surveys of user interface tools (e.g., [7]), which include discussions and comparisons, but none of these are based on implementations of benchmarks.

CONCLUSION

Using benchmarks seems to be a useful technique for students, toolkit developers, and toolkit users. Creating benchmarks and then implementing them four times proved to be an excellent way to give the students a feeling for the comparative strengths and weaknesses in a variety of toolkits. Despite skepticism of the tool vendors, the students were moderately successful at learning and using even "professional" toolkits like MFC and PowerPlant in a three-week period. The benchmarks also have proven to be useful to toolkit developers since they highlight missing features and difficult parts. Finally, the benchmarks seem to be useful for evaluating toolkits. If we can get enough implementations, then comparisons of the code size and times might be more meaningful. Even if not, a person considering various toolkits should be able to find a benchmark similar to the planned interface, and then try implementing the benchmark in the various toolkits. The results reported here suggest that these implementations should not take longer than a few days per toolkit, and the information learned during the implementations will be quite helpful and meaningful.

ACKNOWLEDGMENTS

We would like to thank the many people outside the class who implemented the benchmarks in various toolkits: Matthew Flatt, Ray Johnson, Jan Wielemaker, Hans Muller, Scott McKay, Christian Beilken, Markus Sohlenkamp, and John Pane.

This research was sponsored by NCCOSC under Contract No. N66001-94-C-6037, Arpa Order No. B326, and partially by NSF under grant number IRI-9319969. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

REFERENCES

1. Curtis, B., "Fifteen Years of Psychology in Software Engineering: Individual Differences and Cognitive Science," in *Proceedings of the 7th International Conference on Software Engineering*, 1984, IEEE Computer Society Press. pp. 97-106.

2. Flatt, M., "MrEd," 1997. Department of Computer Science - MS 132, Rice University, 6100 Main Street, Houston, TX 77005-1892, (713)527-8101. mflatt@cs.rice.edu, http://www.cs.rice.edu/~mflatt/mred.html.

3. Hewett, T.T., *et al.*, eds. *ACM SIGCHI Curricula for Human-Computer Interaction*. 1992, ACM Press: New York, NY. ACM Order Number: 608920.

4. Hix, D. "A Procedure for Evaluating Human-Computer Interface Development Tools," in *Proceedings UIST'89: ACM SIGGRAPH Symposium on User Interface Software and Technology.* 1989. Williamsburg, VA: pp. 53-61.

5. Hudson, S.E. and Smith, I. "Ultra-Lightweight Constraints," in *Proceedings UIST'96: ACM SIGGRAPH Symposium on User Interface Software and Technology.* 1996. Seattle, WA: pp. 147-155. http://www.cc.gatech.edu/gvu/ui/sub_arctic/.

6. McKay, S., "CLIM: The Common Lisp Interface Manager." *CACM*, 1991. **34**(9): pp. 58-59.

7. Myers, B.A., "User Interface Software Tools." *ACM Transactions on Computer Human Interaction*, 1995. **2**(1): pp. 64-103.

8. Myers, B.A., Giuse, D., and Vander Zanden, B., "Declarative Programming in a Prototype-Instance System: Object-Oriented Programming Without Writing Methods." *Sigplan Notices*, 1992. **27**(10): pp. 184-200. ACM Conference on Object-Oriented Programming; Systems Languages and Applications; OOPSLA'92.

9. Myers, B.A., *et al.* "Easily Adding Animations to Interfaces Using Constraints," in *Proceedings UIST'96: ACM SIGGRAPH Symposium on User Interface Software and Technology.* 1996. Seattle, WA: pp. 119-128. http://www.cs.cmu.edu/~amulet.

10. Ousterhout, J.K. "An X11 Toolkit Based on the Tcl Language," in *Winter.* 1991. USENIX: pp. 105-115.

11. Spenke, M., "Gina++ and GINA for Lisp," 1992. P.O. Box 1316, D-W-5205, St. Augustin 1, Germany, +49 2241 14-2642, spenke@gmd.de.

12. SWI, "XPCE," 1997. University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands, +31 20 5256121, xpce-request@swi.psy.uva.nl, http://www.swi.psy.uva.nl/projects/xpce/home.html.

13. System Performance Evaluation Cooperative, "SPEC Benchmark Suite Release 1.0," 1989.