THE CONVERGENCE OF COMPUTER PROGRAMMING AND GRAPHIC DESIGN*

David Reed
Chair, Department of Computer Science
Creighton University
davereed@creighton.edu

Joel Davies
Director, Graphic Design Program
Creighton University
joel@creighton.edu

ABSTRACT

Traditionally, computer science curricula have focused on the algorithmic development of software, while design issues such as typography, text layout, and image manipulation have been the domain of graphic design or fine arts programs. With the emergence of the Web as a publishing and programming medium, as well as the availability of high-level development tools, the distinctions between algorithm and presentation are blurring. Today, a programmer needs to understand and apply principles of graphic design in order to develop applications that are usable and attractive to the user. Likewise, a graphic designer developing electronic media must understand and apply programming principles to control the dynamic behavior of the media. This paper addresses the convergence of programming and graphic design, from both the perspectives of a computer scientist and a graphic designer. Simple and practical design principles are presented that can be integrated into a variety of computer science courses.

INTRODUCTION

Computer science and graphic design are disciplines that evolved from very different backgrounds. Computer science traces its roots to mathematics and engineering, and may be roughly defined as the study of computation. Here, the term computation encompasses all aspects of problem solving, including the design and analysis of algorithms, the formalization of algorithms as programs, the development of computing devices for executing those programs, and the networking of those devices to effectively

share resources and computational power [1]. In contrast, graphic design traces its roots to the fine arts and marketing, and may be defined as the process and art of combining text and graphics to effectively communicate ideas [2]. Graphic design, which is sometimes referred to as visual design, focuses on issues such as typography, the use of color, text layout, and image processing in order to present information and convey impressions effectively.

On the surface, it would appear that computer science is concerned with dynamic behavior: the execution of algorithms, while graphic design is concerned with static presentation: the portrayal of information. Recent technological developments, however, have made such generalizations dubious at best. Largely due to the emergence of the World Wide Web as a publishing and programming medium, the distinctions between algorithm and presentation are blurring. A programmer creating an interactive Web page or a front-end to a server-side database must make presentation choices that affect the usability of his or her program. Even the most ingenious program will not be able to compete in the software marketplace if users are unable to understand its workings. Likewise, a graphic designer creating online media must understand and be able to integrate programming in order to add dynamic content and interactivity. Common features on Web sites, such as mouse-sensitive graphics, timer-controlled events, and form submissions, all require programming to control their behavior.

The convergence of programming and graphic design extends beyond the Web. To support the development of applications that are intuitive and usable by a diverse population of users, most modern programming languages provide libraries of graphical user interface (GUI) controls, such as buttons, text fields, and pull-down menus. A competent programmer must master these components and integrate them into the design of useful and marketable software. Likewise, the development tools used by graphic designers require intimate knowledge of algorithms and object-oriented design. A graphic designer must master these concepts in order to create effective documents and media.

The rest of this paper addresses the convergence of programming and graphic design, from both the perspectives of a computer scientist and a graphic designer. Simple and practical design principles are presented that can be integrated into a variety of computer science courses.

A COMPUTER SCIENCE PERSPECTIVE

A survey of computer science texts from five or more years ago reveals a predominantly text-based mode of programming (e.g., [3] [4]). Programs in languages such as Ada, C, and C++ tended to utilize the command line for user interactions, prompting the user with text messages and displaying results as plain text. While Java did provide some support for graphical user interfaces, many texts covered these topics only marginally, instead focusing on console I/O for most examples. The design of program interfaces was only marginally part of computer science curricula, usually relegated to special-topic courses such as Human Computer Interaction (HCI).

In recent years, this distinction between algorithms and interface has blurred, primarily due to the increased importance of Net-centric computing and the standardization of GUI libraries.

Net-centric computing

The Internet and the Web have changed the way in which many programs today are written. Whereas applications used to be platform-dependent executables that were distributed on disks, many applications today are distributed directly via the Internet. Client-side programming languages such as JavaScript, VBScript, and Java (in the form of applets) allow the programmer to embed code directly into Web pages. Conversely, server-side languages such as PHP and Perl allow the programmer to write applications for tasks such as accessing databases or performing complex calculations. The importance of these types of applications within computer science is highlighted by the ACM/IEEE Computing Curriculum 2001, which identified "Net-centric computing" as one of the core knowledge areas of computer science [5].

The underlying language of Web pages, the HyperText Markup Language (HTML), provides a rich set of GUI control elements that can be utilized by Web-based programs. For example, Figure 1 shows an interactive Web page that integrates text, images, text areas, and a button to provide an interface for translating text. JavaScript code controls the behavior of this page, translating text from one area to the other whenever the button is clicked. Figure 2 shows an interactive Web page divided into frames. When the user enters a login ID and password into the text fields and clicks on the button, a server-side program is executed to access a database and return grades for that particular student. Applications such as these require the programmer to integrate both programming logic and interface design.

Figure 1. An interactive Web page using HTML and JavaScript.

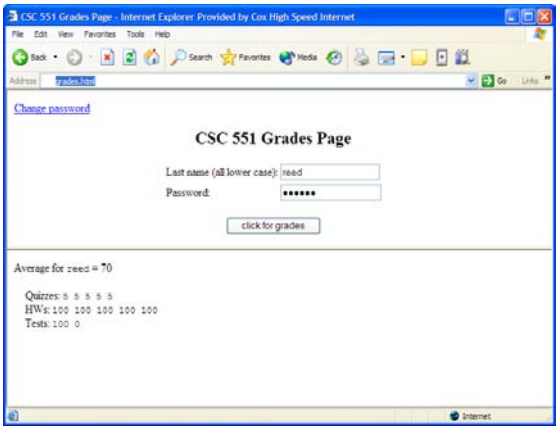

Figure 2. An interactive Web page accessing a server-side database.

Many universities are incorporating Web-based programming courses throughout their computer science curricula. For example, at Creighton University, the Department of Computer Science offers a non-majors course that includes client-side JavaScript programming, an E-commerce course that includes server-side database programming, and a Web Programming course that covers a broad range of client- and server-side

technologies. Proceedings from computer science education conferences refer to a wide variety of such courses (e.g., [6][7][8]).

GUI Programming

As access to personal computers and the Web has expanded, the demand for intuitive and easy to use programs has increased. In addition to languages like JavaScript and VBScript that utilize the Web interface, many modern programming languages provide support for developing graphical user interfaces. For example, Java provides both the Abstract Window Toolkit (AWT) and Swing libraries with GUI controls such as buttons, text fields, text areas, sliders, and checkboxes.

The standardization of GUI libraries in Java may be seen as one feature that has led to its increased use in introductory programming courses. With minimal effort, students can learn to design professional-quality interfaces at the same time they learn the fundamentals of programming. For example, Figure 3 shows an interface for a recent assignment in an introductory Java programming course at Creighton University. Here, text fields, a text area, and buttons are used to control the execution of a Java application.

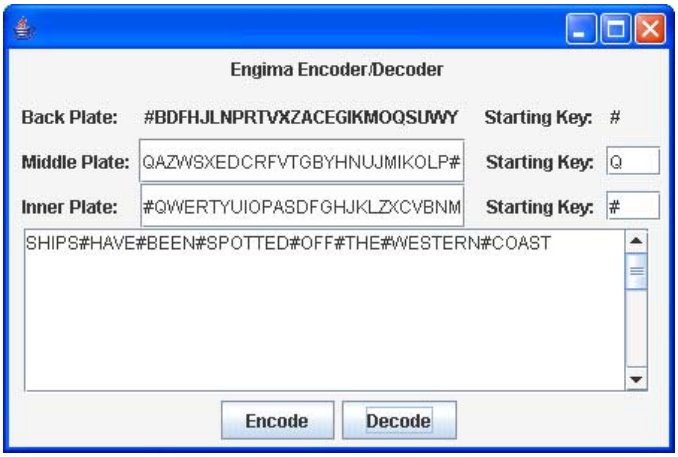

Figure 3. A Java application using Swing.

A GRAPHIC DESIGN PERSPECTIVE

Prior to the development of WYSIWYG Web development applications, graphic design students needed to learn HTML in order to facilitate the layout of Web pages. The normal design process involved creating tables in a basic text editor to hold sliced graphics, usually derived from an Adobe Photoshop mockup. In the late half of the nineties, clean-coding WYSIWYG applications, such as Macromedia Dreamweaver and Fireworks, were refined and targeted to the graphic design market (Figure 4). The design process changed to allow designers to move directly from a mocked-up design to a finished and coded layout with only a rudimentary understanding of the underlying code. Design educators could concentrate on teaching user-centric design principles and graphic design elements for Web-based media. If a student wanted to approach a mastery

of HTML and Web programming, design educators could refer their students to Computer Science courses at the same institution.

This period of design education has come to a sharp conclusion due to two developments: the marked increase in demand for dynamic, database-driven Web sites and the integration of object-oriented programming tools in graphic design applications.

Web Design

Expanded and refined cascading style sheet (CSS) browser implementation has brought dramatic changes to the design classroom. Using CSS effectively requires design educators to teach another programming element to students, and it brings a higher level of design and typographic control to design students. The cost of this control is the loss of more WYSIWYG function. The rapid deployment and development of dynamic online content is changing the requirements of professional Web designers, and thus changing the needs of students. Specifically, the availability of open source database solutions, such as MySQL, to facilitate dynamic and custom content delivery to a wide range of users via PHP Hypertext Preprocessor (PHP) and CSS layouts has brought coding back into the design classroom.

Additionally, factions within the Web design community have moved towards a model of design that includes more control over the user experience than the traditional method of creating extendable pages [9]. Design has been trending toward controlled page widths to facilitate reading and communication. Regardless of which view of control a given designer subscribes to, all designers will need to thoroughly understand the capabilities of CSS design implementation in order to create effective concepts. The short-lived days of simply using WYSIWYG software to fully code a site are coming to an end.

Design Tools

Recent releases of Web-based multimedia solutions such as Macromedia Flash now require designers to understand and hand code an object-oriented coding language called Actionscript. Previous versions of Flash wrote code dynamically that could be customized by advanced users, but beginning with Flash MX 2004, Macromedia implemented a code window without contextual prompts to facilitate programming without programming knowledge (Figure 5).

DESIGN GUIDELINES FOR COMPUTER SCIENTISTS

The primary role of a graphic designer is to translate a message from a client to their target audience by creating a visual framework to display the message. Designers use their education in aesthetics and design principles to develop creative solutions to communication problems. The single goal of any design artifact is clear communication between a client and their audience. Artistic expression is, at best, noise that will interfere with this communication. However, strategic aesthetic application and good conceptual execution facilitates this line of communication and reinforces real and perceived relationships between clients and their audience.

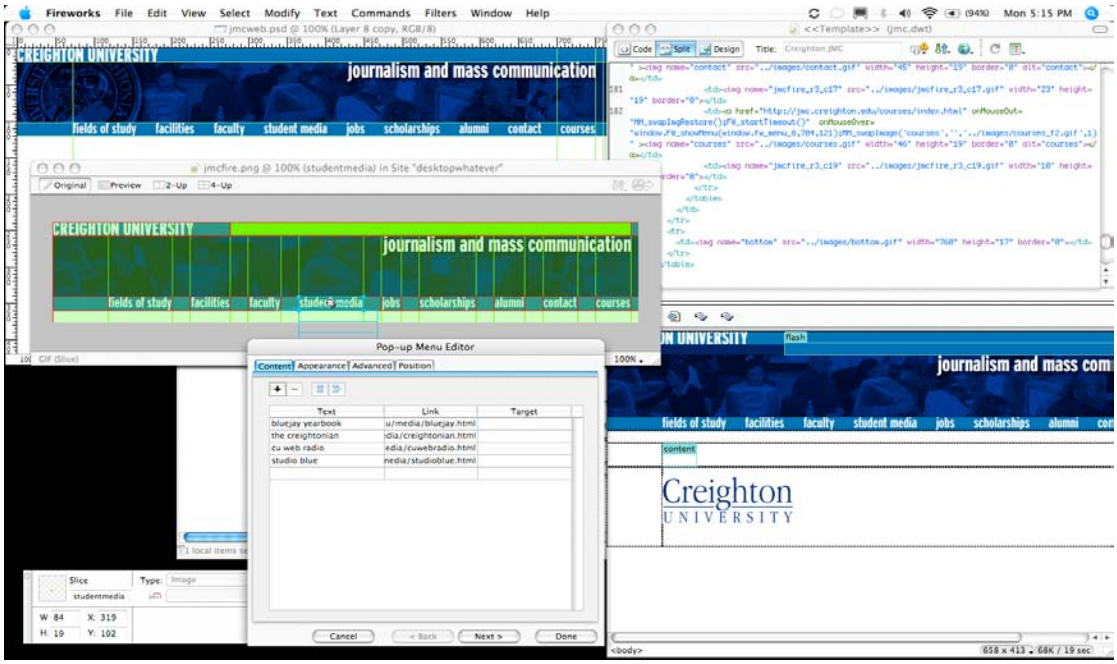

Figure 4. WYSIWYG Design using Adobe Photoshop and Macromedia Studio MX 2004.

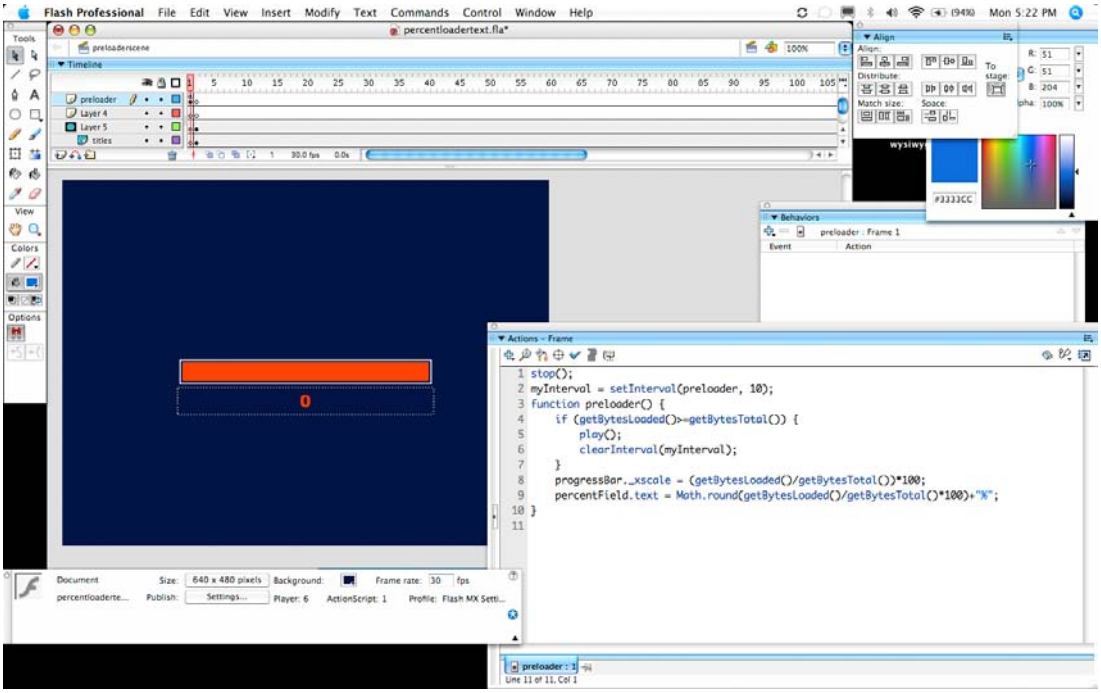

Figure 5. Macromedia Flash Actionscript object-oriented code for a preloading progress bar.

In order to create effective design artifacts, the designer must parse the message and have a clear understanding of the demographics of the target audience. At the very least, she should gather data on the level of operating system and browser technology used by the audience. This information is invaluable to create content accessible to all users [10]. Once she has established the basis for this communication, developing strategic visual communication is a far simpler task. Good design is sometimes so transparent that it truly frames and contextualizes the content, with no interference whatsoever. However, bad design is easily noticed as inappropriate use of color, typography, graphics and layout.

While books and articles can be found that provide design advice to the computer scientist from an HCI perspective (e.g., [11]), the graphic design perspective has been generally lacking. The following design guidelines, when applied correctly, can greatly enhance user experience for onscreen media, whether it be interactive Web sites or software applications.

Typography: Typography is the basic unit of onscreen communication. When selecting typography, be sure to honestly evaluate if the font is easy to read onscreen. Simple typefaces, such as Verdana and Arial, have been designed to be highly readable on monitors. In general, it is easier to read sans-serif typefaces (again, Verdana and Arial) onscreen than serifed typefaces (such as Times New Roman) at smaller sizes. Also consider what is communicated by certain fonts - Comic Sans is often used to designate a less formal attitude, but generally undermines the message to create a sense of childishness or naiveté.

Size: Large type is not necessarily easier to read. Consider the size of print in a book compared to the overall size of the page. Proportion of font size to the overall window size is critical to creating readable text. Generally, it is considered ideal to have a measure of 50-80 characters per line, including spaces and punctuation [12]. The measure can vary to match the tempo of the writing, a shorter measure works well for simple sentence structure. Ideally, type measure is set to allow one concept to be delivered per line of text. One can balance measure with CSS line height controls. Longer measures should be balanced with higher line height values, while short measures often do not need additional line height value.

Contrast: It is critical that typography and major graphic elements on the page maintain a large difference in contrast from the background color. If a background pattern or image is used underneath text, it is even more important to maintain a large difference in contrast. Generally, a 60% difference in contrast across two color channels will suffice to maintain contrast between a background color and the text to be read. Since "web safe" colors are expressed as hexadecimal values in 20% increments, this is actually a fairly easy guideline to follow. The easiest method to test for adequate contrast is free and easy: lean back from the monitor and squint your eyes. If you can see the text "jumping" from the background, you have enough contrast.

Color: There are two major concerns to address when using colors: aesthetics and usability. Aesthetically, it is best to have balance in the color palette. When using a bright or intense color, balance it with a dark and muted color. Don't use intense colors next to each other, as this results in a perceived dimensional illusion that can be very disorienting to users. Also limit your palette. With good planning, it is rarely necessary to use more than 4 colors in a given layout. From a usability standpoint, remember that approximately

5% of most audiences suffer from color blindness. Avoid red/green and blue/orange color combinations, unless you use dramatic contrast differences between the two colors [13]. Online tools such as Vischeck [14] are available to help view the effects of colorblindness on Web page designs and graphics.

Hierarchy: Develop a simple hierarchy of information to be placed in your layout. Visually separate different layers of information (navigation, text fields, paragraphs, etc) using color, size, background color fields and negative space. Negative space is often under-utilized due to the use of oversized fonts. By simply reducing font sizes and moving elements of the page apart, a visual hierarchy is easily built and maintained. Strong visual hierarchy is the key to good layout, as it creates the framework through which all elements of design and content are viewed [15].

Consistency: Consistently using all of the above guidelines within a project will significantly raise both the usability and aesthetic qualities of the project. If there are several "screens" or pages in a project, maintain visual consistency throughout the experience. Users become disoriented if the navigation moves to another area of the screen, or if all the colors suddenly change within a Web site. There is great value in providing a predictable, consistent experience for users. If there are a series of related projects, visually brand them as a series by repeating the same fonts, colors and overall layout. Using an external cascading style sheet greatly simplifies this process.

Program effectively: Effective use of external cascading style sheets can greatly simplify the design and maintenance of Web sites of all sizes. Redefining a large set of tags to automate text formatting or extensive use of CSS IDs to define a layout can have the same effect of creating a site-wide design template. Simply changing the style sheet will update the major design elements of any page linked to the CSS file. A small amount of time spent experimenting with the elements of CSS typographic controls will reveal it is possible to create typography that approaches the level of control most designers find in print design applications. Judicious use of line height and margin controls can drastically improve readability on complex pages. An ideal CSS implementation will allow the designer/programmer to strip all the font tags from a Web site, and still have perfect typography.

CONCLUSION

As the computer science and graphic design disciplines converge, it is inevitable that cross-pollination between the two result in a blurring of the lines of professional practices. Computer scientists increasingly need exposure to design trends and principles, so that they might take advantage of lessons learned by graphic designers. Likewise, designers will require the computer science experience to accurately and efficiently code projects, in addition to the ability to comprehend new technologies as they emerge. Collaboration between computer science and graphic design educators is imperative to ensure that each discipline learns from the other and is prepared for future developments.

REFERENCES

[1] Reed, David. A Balanced Introduction to Computer Science, Prentice Hall, 2005.

[2] Azatiko Internet Group, Montreal Web Design Company. Glossary of Internet Terms, http://www.azatiko.com/glossary/graphic-design.php, accessed May 15, 2005.

[3] Roberts, Eric. The Art and Science of C, Addison Wesley, 1995.

[4] Astrachan, Owen. A Computer Science Tapestry, 2nd edition, McGraw-Hill, 2000.

[5] Joint IEEE Computer Society/ACM Task Force for CC2001. Computing Curricula 2001, Journal on Educational Resources in Computing (JERIC), 3(1), 2001.

[6] Reed, David. Rethinking CS0 with JavaScript, SIGCSE Bulletin, 33(1), 2001.

[7] Bloss, Adrienne. Teaching Fundamentals for Web Programming and E-commerce in a Liberal Arts Computer Science Curriculum, Journal of Computing Sciences in Colleges, 16(2), 2001.

[8] Calongne, Cynthia. Designing for Web Site Usability, Journal of Computing Sciences in Colleges, 16(3), 2001.

[9] Lynch Patrick and Horton, Sarah. Web Style Guide, Basic Design Principles for Creating Web Sites, Second Edition, TYPOGRAPHY: Line Length, http://www.webstyleguide.com/type/lines.html, accessed May 16, 2005.

[10] Lynch Patrick and Horton, Sarah. Web Style Guide, Basic Design Principles for Creating Web Sites, Second Edition, INTERFACE DESIGN: Accessibility, http://www.webstyleguide.com/interface/access.html, accessed May 16, 2005.

[11] Gordon, Aaron. User Interface Awareness, Journal of Computing in Small Colleges, 11(2), 1995.

[12] Bringhurst, Robert. The Elements of Typographic Style, Hartley and Marks Publishers, 2004.

[13] Pavka, Anitra. Digital Web Magazine, Accessible By Design, http://digital-web.com/articles/accessible_by_design/, accessed May 16, 2005.

[14] Dougherty, Robert and Wade, Alex. Vischeck: VischeckURL, http://www.vischeck.com/vischeck/vischeckURL.php, accessed May 16, 2005.

[15] Lynch Patrick and Horton, Sarah. Web Style Guide, Basic Design Principles for Creating Web Sites, Second Edition, PAGE DESIGN: Visual Hierarchy, http://www.webstyleguide.com/page/hierarchy.html, accessed May 16, 2005.