Accessibility Survival Guide for Instructors

This document contains the results of a Subject Area Study on accessibility issues in the curricula of Computer Applications and Office Systems (CAS), Computer Science (CS), and Computer Information Systems (CIS). The study was conducted during Winter Term, 2014, and included the participation of the following people:

- **Supada Amornchat** - Alternate Media Formats Technician, Distance Learning
- **Lisa Brandt** - Alternate Media Formats Technician, Disability Services
- **Angel Chisemet** - Assistive Technician
- **Marc Goodman** - Adjunct Instructor and Subject Matter Expert, CIS
- **Haris Gunadi** - Alternate Media Specialist, Disability Services
- **Gayathri Iyer** - Instructor and Subject Matter Expert, CS
- **Kaela Parks** - Disability Services Director
- **Phyllis Petteys** - Accessibility Specialist, Disability Services
- **Karen Sorensen** - Accessibility Advocate, Distance Learning
- **Susan Watson** - Instructor and Subject Matter Expert, CAS

The overall purpose of the study was to identify accessibility issues that are specific to the teaching of CAS, CS, and CIS. Within that purpose, we identified the following goals:

- To provide instructors with guidance on how to develop new course materials that were accessible to their diverse students.
- To provide instructors with basic information about accessibility requirements and the accommodation process for students with disabilities.
- To provide Disability Services staff with information on creating accessible alternatives to existing course content.
- To provide Distance Learning staff with information that could be used to evaluate whether online course shells are accessible.
- To provide background information for the ongoing discussion around our plans for making our instructional materials accessible, so that the full requirements and costs are well understood.

It is written from the perspective of an "accessibility survival guide for instructors," i.e., the practical information an instructor should know when designing new materials, or when a student with an accommodation signs up for their class. It is organized into the following sections:

1. Introduction - Provides background on accessibility issues and the report. Discusses why instructors should be aware of accessibility issues. Explains what the report covers.
2. Computing for the Blind - Discusses the differences between a blind student using the computer and a sighted student using the computer. Describes assistive technologies that are available to our students.
3. Making Programming Labs Accessible - Presents a rubric for evaluating programming labs for their accessibility. Gives several examples of accessible and inaccessible content, and guidelines for creating accessible content.
4. General Recommendations - Provides general recommendations not covered in the section on the programming lab accessibility rubric. Includes information on the accessibility of publisher content for frequently used publishers in our subject areas.
5. Glossary - Gives definitions for "jargon" terms used throughout the report.
6. References - Provides links to several sources used in compiling this report.

This document may be viewed as a single consolidated report, or as a collection of browsable, hyperlinked web pages.

1. Introduction

This section contains important information about students with disabilities and the accommodation process. It is aimed primarily at instructors who have a student with disabilities, and for instructors who are developing new course materials, or revising existing materials. It contains the following content:

- You are Not Alone - There is a wide range of expertise available at PCC to help you with the accommodation process and with supporting your students with disabilities.
- What Does this Document Cover? - What the remainder of the document will cover, and why we chose to focus on those topics.
- The Accommodation Process - A brief introduction to the process when a student with disabilities signs up for a class.
- Courtesy Rules for Students with Disabilities - Some general guidelines and tips on working with students with disabilities.
- Why Accessibility is Important - An overview of why accessibility is important both to PCC and to our students.

1.1. You are Not Alone

You may have a student with a disability who has signed up for your course, and you're not sure what your responsibilities are. Or, you may be designing instructional materials, and you're not sure how to provide equally effective alternative materials to your students with disabilities. Or, you may just be interested in finding out how blind students can learn to program computers. Whatever your interest is, there's one important message to start out with: you are not alone. The accommodation process and the requirements for providing accessible course materials to our students can be challenging, but you are not expected to do everything and to know everything on your own. There are experts in accessibility and on providing effective educational opportunities for students with a wide range of impairments at PCC. Some experts are available in Disability Services, some experts are available in Distance Learning, and some experts may even be in your own department or SAC. You can find help on evaluating where accessibility obstacles are in your course materials, on creating accessible alternatives, on teaching students with disabilities the basic technology skills

they need to succeed in your course, on assisting your students to overcome accessibility issues, and on understanding the accommodation process. We are all here to help you and your student.

In addition to the material contained in this document, some good starting points include the Disability Services web site, and Accessibility for Online Course Content.

1.2. What Does this Document Cover?

There has already been a large amount of work done at PCC on understanding accessibility issues for our online content and for mathematical content. There are also ongoing plans to conduct additional subject area studies. When we began this study, we started by asking the following question: "What are the important differences between the CAS, CS, and CIS disciplines and other fields of study at PCC?" We wanted to address issues that would be useful for other disciplines at PCC, but we wanted to focus on the "most difficult issues" that were unique to our subject areas, rather than "re-inventing the wheel."

Many types of impairments and accommodations are well understood. For example, it is well understood that deaf students will require closed captioning for video media, and will require transcripts of audio media. This will be true regardless of the subject area. It is also well understood that students with some types of cognitive impairments will require more time on assignments and on exams. Once again, this will be true regardless of the subject area. While addressing these types of impairments and accommodations is crucial for delivering accessible educational opportunities, we did not feel that they should be the focus of this particular study.

At their root, CAS, CS and CIS are about interacting with the computer and with applications running on the computer. In particular, a large portion of CS and CIS involves learning to program, which requires interacting with complex applications such as IDEs as well as writing, running, testing, and debugging computer programs (which we will collectively refer to as "programming labs"). For some types of impairments, little additional work is required to make these programming labs accessible. For example, a student with motor control impairments may be unable to use the mouse or keyboard, but can use speech-to-text software such as Dragon Naturally Speaking or Windows Speech Recognition to complete the programming lab. However, this is not the case for blind students and for students with severe visual impairments. There are some fundamental differences between how a blind student interacts with the computer, and how a sighted student interacts with the computer.

We felt that by addressing how programming labs could be made accessible to blind students, we would be tackling one of the most difficult subject area specific issues. We also felt that the insights gained by studying this problem would be useful for understanding more about the requirements for other types of computer related activities, such as teaching a blind student to use Microsoft Word to author a document, or to use Dreamweaver to author a web site. Though we all felt that it was important to address how some of these issues affected students with other impairments, we felt the primary focus of this study should be on blind students and programming labs.

When we began exploring this specific area, we soon realized that there were a large number of factors that affected whether a blind student would be able to complete a programming lab substantially on their own. We carefully considered how this information could be most effectively organized and presented to support multiple uses. Ultimately, we devised the notion of a Programming Lab Accessibility Rubric that contained each factor we had identified as an accessibility standard, along with guidance on how to meet each standard. It is our hope that this rubric will serve multiple purposes:

- To guide instructors during the process of creating or revising course materials, so that they can make choices which have a positive impact on accessibility.
- To provide detailed information to Disability Services so that they can evaluate whether course materials require accessible alternatives when a blind student enrolls in a CAS/CS/CIS class.
- To provide Distance Learning staff with the information they need to assess whether our online course shells meet accessibility requirements.
- To provide background information for the ongoing discussion around our plans for making our instructional materials accessible, so that the full requirements and costs are well understood.

We felt that a detailed Programming Lab Accessibility Rubric could help support all of those purposes.

1.3. The Accommodation Process

Disability Services provides a Quick Reference Guide on faculty responsibilities to students with disabilities. Part of this Quick Reference Guide is a description of the accommodation process. To summarize the process:

1. Students attend an orientation session to understand the accommodation process.
2. The student works with a Disability Counselor to determine an accommodation or set of accommodations that are appropriate for that particular student.
3. The student signs up for courses during the normal registration process.
4. The student's Disability Counselor will send an Approved Academic Accommodations (AAA) Form to the course instructor. This form will include the specifics of the accommodations that the student will need to succeed in the course.
5. Before the course starts, Accessibility Technicians and Alternate Media Specialists will review the course materials and create or acquire alternate formats of those materials which provide equal access to the student. In some cases, the creation of new materials may overlap the delivery of the course.
6. Throughout the course, the student may work with an Accessibility Specialist. The specialist can facilitate the accommodation process and work with the student to build technology skills.
7. If the student needs help to overcome an accessibility issue in the course, they may rely on the help of an Assistive Aid, or the course instructor.

There are two potential pitfalls in this process:

- The time between when a student signs up for a course and when the course begins is usually short. If alternate formats for significant amounts of course materials are required, it may be difficult to create or acquire these materials in time. If substantially all of the course materials are accessible, then this will not be an issue. So, it is extremely important to understand accessibility issues when designing new course materials or revising existing materials.
- Accessibility Technicians and Alternate Media Specialists may not also be subject matter experts in the particular area of study. They may require significant guidance from the course instructor on what materials will present accessibility issues to the student, and how to best provide effective alternatives to these materials.

Two key purposes of this document are to address these issues by providing information to instructors on how to create accessible materials in CAS, CS, and CIS, and to provide guidance to Disability Services and Distance Learning on determining whether course materials in those disciplines will cause accessibility issues, and how to best address those issues. We have addressed these issues by providing information on making programming labs accessible.

1.4. Courtesy Rules for Students with Disabilities

Often, when we meet someone who is different from ourselves, we feel unsure about how to relate to them. We want to make them comfortable and put them at ease, but sometimes we don't know the best way to do this. Disability Services has a handy document with tips for working with students who experience disabilities. You may also find the following "Courtesy Rules of Blindness" from Blind.net to be helpful.

> When you meet me don't be ill at ease. It will help both of us if you remember these simple points of courtesy:
>
> 1. I'm an ordinary person, just blind. You don't need to raise your voice or address me as if I were a child. Don't ask my spouse what I want--"Cream in the coffee?"--ask me.
> 2. I may use a long white cane or a guide dog to walk independently; or I may ask to take your arm. Let me decide, and please don't grab my arm; let me take yours. I'll keep a half-step behind to anticipate curbs and steps.
> 3. I want to know who's in the room with me. Speak when you enter. Introduce me to the others including children, and tell me if there's a cat or dog.
> 4. The door to a room or cabinet or to a car that is left partially open is a hazard to me.
> 5. At dinner I will not have trouble with ordinary table skills.
> 6. Don't avoid words like "see." I use them too. I'm always glad to see you.
> 7. I don't want pity, but don't talk about the "wonderful compensations" of blindness. My sense of smell, taste, touch or hearing did not improve when I became blind, I rely on them more and, therefore, may get more information through those senses than you do--that's all.
> 8. If I'm your houseguest, show me the bathroom, closet, dresser, window--the light switch too. I like to know whether the lights are on or off.
> 9. I'll discuss blindness with you if you're curious, but it's an old story to me. I have as many other interests as you do.
> 10. Don't think of me as just a blind person. I'm just a person who happens to be blind.
> 11. You don't need to remember some "politically correct" term, "visually impaired", "sight challenged" etc. Keep it simple and honest, just say blind.
>
> In all 50 states the law requires drivers to yield the right of way when they see my extended white cane. Only the blind may carry white canes. You see more blind persons today walking alone, not because there are more of us, but because we have learned to make our own way.

1.5. Why Accessibility is Important

A Forrester Research report (Accessible Technology in Computing - Examining Awareness, Use, and Future Potential [Commissioned by Microsoft, Conducted by Forrester Research, Inc. 2004]) found that 74% of individuals with mild or severe difficulties/impairments use computers, and 57% of computer users are likely or very likely to benefit from the use of accessible technology due to experiencing a mild or severe difficulty/impairment. The impairments covered in this report are varied, and include vision, motor control, hearing, cognitive and speech impairments. An average of 12% of community college students have disabilities, and many of these students will be required to take (or choose to take) one or more computer classes. These students will be able to succeed in your course if your course follows accessibility guidelines.

The American Foundation for the Blind (AFB) estimates that 21.2 million adult Americans (more than 10% of all adult Americans) reported mild or severe visual impairments. They also estimated back in 2001 that there were 1.5 million visually impaired computer users, including those who are blind based on 1999 US Census data. That number has significantly increased in the 15 years since the 1999 Census.

Providing accessible education to our visually impaired students is important for at least three reasons: it is part of PCC's mission, it is a legal requirement, and it provides students with alternative media formats which support students with different learning styles, not only students with impairments.

1.5.1. PCC's Mission Statement

PCC's Mission Statement says that, "Portland Community College advances the region's long-term vitality by delivering **accessible** [emphasis added], quality education to support the academic, professional, and personal development of the diverse students and communities we serve."

One of the most important principles of accessibility is to provide an equally effective learning experience. Accommodations vary from student to student. Students in higher education are increasingly diverse. Diversity includes physical, visual, hearing, learning, attention, and communication differences. Our goal should be to try to effectively address as many of these differences as possible. When creating a lecture, activity, or assignment, we should be sure to create activities that provide similar levels of achievement of learning outcomes for all of our students.

1.5.2. Web Accessibility Legal Overview

PCC is required to comply with federal disability discrimination laws. These requirements are given in Section 504 of the Rehabilitation Act and the Americans with Disabilities Act (ADA):

- Section 504 of the Rehabilitation Act of 1973 says, "*No otherwise qualified individual with a disability in the United States, …shall, solely by reason of her or his disability, be excluded from the participation in, be denied the benefits of, or be subjected to discrimination under any program or activity receiving Federal financial assistance…*"
- The Americans with Disabilities Act of 1990 prohibits discrimination on the basis of disability by public entities.

While accessibility of the World Wide Web is not mentioned in either of these laws, a Dear Colleague letter from the Office of Civil Rights was sent to every college and university president on June 29, 2010, specifically to make the point that **inaccessible technology required in a course is considered discrimination** under the ADA and Section 504 of the Rehabilitation Act of 1973, unless an equally effective alternative is provided.

A year later, the Office of Civil Rights followed up with a Q & A on the original letter. In this Q & A, they specify that even if no students with disabilities are enrolled in an online course, that course still needs to be accessible or prepared to be made accessible immediately.

In order to comply, Distance Learning consulted the college Web Team on the appropriate web accessibility standard to follow. We selected the commonly used Web Content Accessibility Guidelines (WCAG) 2.0 AA as our standard. By following this standard, access to the web can be greatly improved for people with visual, hearing, mobility and learning limitations. Overall usability for all is usually enhanced, not just usability for students with impairments.

But what then do you do, when some 3rd party platforms, applications and content cannot be made accessible? Let's say a flashcard study aid on a publisher's website isn't accessible to a blind student. Then, allowable accommodations or modifications are those that permit the student to receive all the educational benefits provided by the technology in an equally effective and equally integrated manner with substantially equivalent ease of use (Office of Civil Rights resolution agreement with South Carolina College System, 2/18/13). Since the flashcards are a study tool, find another study tool that is accessible, equally effective, equally integrated with substantially equivalent ease of use. If you can't think of anything, consult with your colleagues, Disability Services, and Distance Learning about options. the Office of Civil Rights wants us to plan ahead, so that a last minute course retrofit isn't required (which often is less than equally effective).

See Accessibility in Online Courses for an overview of the legal regulations and settlements related to IT accessibility. There have been lawsuits over inaccessible:

- College/university websites
- Hardware and software
- Learning Management Systems
- 3rd Party content (including Publisher materials)
- Enterprise tools (email, calendaring, etc.)

Since 2009, the National Federation of the Blind has pursued a legal campaign to ensure post-secondary institutions fulfill their obligations to make electronic and information technology accessible to blind students and staff. You may have read about some of these lawsuits and civil rights complaints that had to do with web accessibility. Settlements with the Office of Civil Rights often consume a lot more college resources than if accessibility is factored in from the beginning.

Both PCC as an institution and individual instructors are legally obligated to develop an accessibility action plan and to implement this plan. Distance Learning and Disability Services/Instructional Support are here to help.

1.5.3. Universal Design for Learning

According to Ron Mace, founder and program director for the Center for Universal Design at NCSU, universal design "is the design of products and environments to be usable by all people, to the greatest extent possible, without the need for adaptation or specialized design". Universal Design for Learning (UDL) is the application of this principle to education.

The National Center on Universal Design for Learning provides an overview on how to create curricula that give all individuals equal opportunities to learn. Many of these principles apply to both students with impairments as well as students with different learning strategies, so creating effective alternative content will benefit all of your students, not just students with an accommodation.

The National Center on UDL also provides a set of guidelines and a framework for creating content that meets the needs of all learners. There are also extensive examples and resources for creating materials that follow these guidelines.

2. Computing for the Blind

An examination of the requirements for designing accessible content for blind students must begin with the differences between what a blind student experiences when using the computer, and what a sighted student experiences. These differences can be broken down into two broad categories: differences in the computing environment that the student experiences, and assistive technologies that can be used by the blind student to overcome accessibility barriers.

2.1. The Environment

There are several important differences between the way a blind student is able to use the computer and the way a sighted student can use the computer. The most important differences are related to the way a blind student uses the screen, the mouse, and the keyboard.

2.1.1. The Screen

A blind student will be unable to use the computer screen. Most modern applications use GUIs, so not being able to see the screen often creates accessibility barriers. As a replacement for the screen, the student will get most of their information from a "screen reader" program. A screen reader is a program that describes aloud what appears on your computer screen, speaking the text that's in documents and windows. There is additional information on how screen readers work, and on some commonly used screen readers.

2.1.2. The Mouse

The mouse is primarily a spatial and graphical metaphor for controlling the location of the cursor on the screen. Since the student will be unable to see the screen, they will also be unable to get feedback from the screen on the location of the cursor. Therefore, the student will be unable to use the mouse as well.

2.1.3. The Keyboard

The keyboard will be the primary interface that the student uses to control the computer. They will need to use the computer both to control the operating system (such as Windows, Mac OSX, or Linux), as well as to control any applications they are running (such as a web browser and a development environment). Since modern operating systems and modern applications commonly use GUIs, the student will need to interact with these GUIs using the keyboard. The primary assistive technology for accomplishing this interaction is the use of keyboard shortcuts.

2.2. Assistive Technologies

A number of assistive technologies have been developed that can be used by blind students to overcome accessibility barriers. Many of these technologies are available to PCC students through Disability Services. In some cases, blind students can also receive educational grants to purchase assistive technologies for their personal use. Some of the technologies available include:

- Keyboard shortcuts
- Screen reader programs
- Tactile graphics
- Braille and refreshable Braille displays
- Audio narration
- Other technologies, such as sonification, haptics, and 3D printing

Note that many of these technologies also support Universal Design for Learning and can benefit many of your sighted students, including students with other impairments. For example, screen reader programs can benefit students with certain cognitive impairments and learning disabilities, and keyboard shortcuts can also benefit students with a range of motor impairments.

2.2.1. Keyboard Shortcuts

"Keyboard shortcuts" are keyboard equivalent commands that serve as an alternative for commonly-used actions that are used to control the operating system or application. For example, in Microsoft Word, one can open a document either by selecting "Open" from the "File" menu using the mouse, or by typing Alt followed by F followed by O on the keyboard. Some software publishers also refer to keyboard shortcuts as "accelerators" or "hotkeys," so those terms can be useful when researching whether a particular application is keyboard-accessible. Since a blind student will be using the keyboard as their primary means for controlling the computer, it is essential that keyboard shortcuts are available for all of the applications that the student must use.

2.2.2. Screen Readers

A screen reader program will be the blind student's primary interface for replacing information contained on the screen. The screen reader program doesn't process the image of the screen like an Optical Character Recognition program would. Instead, it gathers information from the operating system (Windows, OSX, or Linux), and uses this information to speak the text. If a program doesn't communicate the proper information to the operating system, the screen reader will not be able to speak its text, even though the interface to that program appears to be textual. For that reason, it's impossible to tell whether a particular program is screen reader compatible merely by looking at it. Instead, each program must be separately tested with the screen reader enabled. Also note that two programs that present substantially the same interface may have different levels of accessibility. For example, on Linux, xterm terminal windows are not compatible with the Orca screen reader while GNOME terminals are compatible, even though the visual difference between the two is minimal.

It is important to note that many screen reader programs offer similar functionality, but are quite different in terms of interface, and browsing modes. This will cause confusion for a student if, for example, you give them instructions on completing a lab using NVDA but the student is using JAWS, or vice versa. It is generally safe to assume that a student will be trained in the use of their own screen reader program before beginning your class, or that Disability Services will work with that student to help them to get the proper training. Therefore, you should be able to avoid including screen-reader specific instructions in most cases.

There are several screen readers available on all of the major platforms. Some of the most common include Narrator, JAWS, NVDA, Orca, Speakup and VoiceOver.

2.2.2.1. Microsoft Narrator

Microsoft Narrator comes bundled with Microsoft Windows. It is compatible with Windows itself and some of the more basic applications for Windows, but is not considered a "full featured" screen reader, and will probably not be powerful enough for a blind student to successfully complete a programming lab.

2.2.2.2. JAWS

JAWS (Job Access With Speech) is one of the most powerful and popular screen readers for the Windows platform. However, it is commercial software and is quite expensive (so you might not want to buy a copy just to test your applications with it). However, many of your blind students may already have access to JAWS before they sign up for your class. It is very powerful and supports different modes for browsing through the text in a document. It can allow the student to browse a document section-by-section, if that document has been structured using the proper document headers. It is frequently updated, and has application specific plug-ins that give it a high level of compatibility with many programs. There is a detailed tutorial available.

2.2.2.3. NVDA

NVDA (NonVisual Desktop Access) is a free, open source alternative to JAWS for the Windows platform. It is full featured, but may not be updated as frequently as JAWS. There are several guides and tutorials available. It is a good choice for an instructor who wants to check accessibility for their applications and development environment, but doesn't have access to JAWS.

Most of the test results described in the page on screen reader compatibility are based on tests conducted with NVDA.

2.2.2.4. Orca

Orca is a free, extensible screen reader program for Linux. It was originally developed at Sun Computers, before moving fully into the open source community. It is integrated into the GNOME desktop environment, and comes bundled with several Linux distributions, including Fedora and Ubuntu. It is possible to install Ubuntu Linux using Orca without using the screen or mouse. Full documentation on using Orca is available. There were a number of tutorials available at one point, but most of those links are broken now (one downside of using open source software). As with the rest of the screen reader programs described here, some applications are compatible with Orca and some aren't. If you are trying to use an application in your class that isn't compatible with Orca, it might work with Speakup instead.

2.2.2.5. Speakup

Speakup is another free, Linux-based screen reader program. It is probably not as well supported as Orca, but may work for some applications that are not compatible with Orca. Speakup works particularly well with "console based" applications such as the Lynx web browser and the Pine mail reader. The interface is quite different from Orca's interface, so a blind student may have to switch back and forth between screen reader programs if you require Speakup in your labs. This is "do-able," but is definitely more difficult for the student than using one screen reader program. A guide for using Speakup is available in unstructured text format. There are also tutorials available.

2.2.2.6. VoiceOver

VoiceOver is built-in, advanced screen-reading technology integrated into the Mac OS X operating system. VoiceOver enables users with visual disabilities to control their computer using a rich set of keyboard commands and gestures. To turn on VoiceOver, press Command-F5 on your mac. Here is a link to the VoiceOver Info Guide. VoiceOver provides many ways to read text. You can read text a word, line, sentence, or paragraph at a time. You can hear words and characters spelled phonetically. You can also read text using VoiceOver gestures. AFB AccessWorld Magazine says, "Text Edit is the simple word processor that comes on all the macs. This is the application in which VoiceOver performed best. It is possible to write and edit documents, cut and paste text, and save documents as files in rich text format. This editor does not include a spell checker or other advanced word-processing functions."

2.2.3. Tactile Graphics

Tactile graphics allow visual, graphic information to be conveyed through touch. Tactile graphics are a combination of raised surfaces including textures, lines, Braille labels, points, shapes, and scale (see sample images below). Tactile graphics are frequently made for maps, diagrams, charts, graphs, geometric figures, organic shapes, and building layouts that are difficult to describe in text alone. Students with visual impairments can learn the same information from touching the tactile graphic as a sighted student would from looking at it. Tactile graphics are used mainly by students with visual impairments and sometimes by students with learning disabilities. They can help communicate and convey important visual content to blind students when that visual content cannot be adequately described in a text format.

Important images that cannot be adequately described in a text format should be made into tactile graphics. These often are complex images, like: diagrams, maps, graphs, charts, workflow charts, building layouts, geometric figures, and organic shapes. To determine whether you need a tactile graphic, here are some questions to ask yourself. You **don't** need a tactile graphic if:

- The information is a repeat of facts provided in surrounding text.
- The information could be described adequately in text.
- The real object is available and safe to touch.

Otherwise, a tactile graphic would be appropriate, especially if the information is necessary for the student to complete an assignment or participate in class. (Source: The American Foundation for the Blind from Ike Presley & Lucia Hasty. Techniques for Creating and Instructing with Tactile Graphics. Copyright © 2005. New York: American Foundation for the Blind).

2.2.3.1. Creating Tactile Graphics

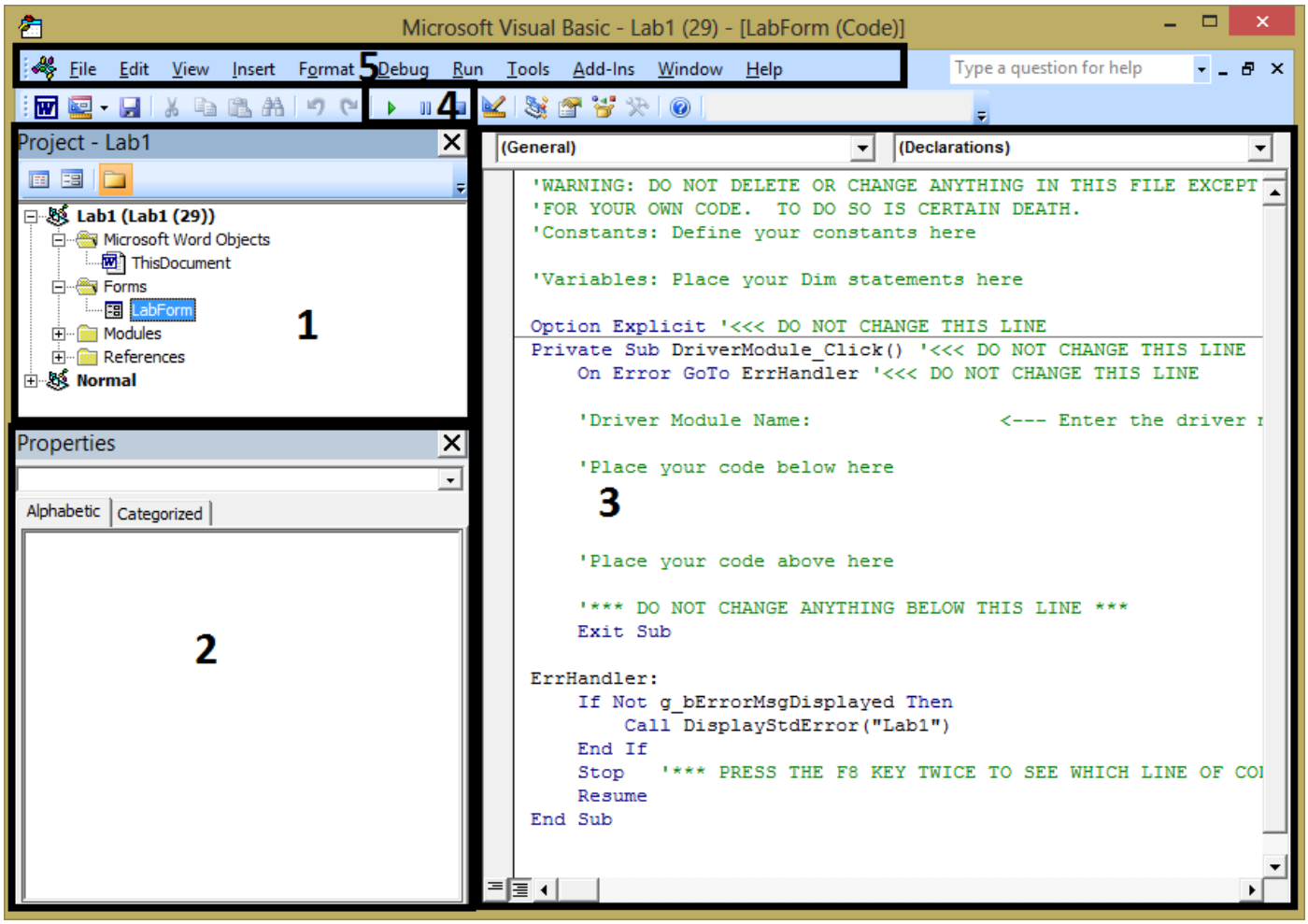
Supada Amornchat (supada.amornchat@pcc.edu) from Distance Learning can work with you to create tactile graphics for your students. However, there are limitations to designing a tactile graphic. A tactile graphic shouldn't have too many details because there needs to be space enough to discern the different demarcations. In order to assist you, she will need the image itself, as well as image descriptions. Each of these requirements is explained in more detail below.

2.2.3.1.1. The Image

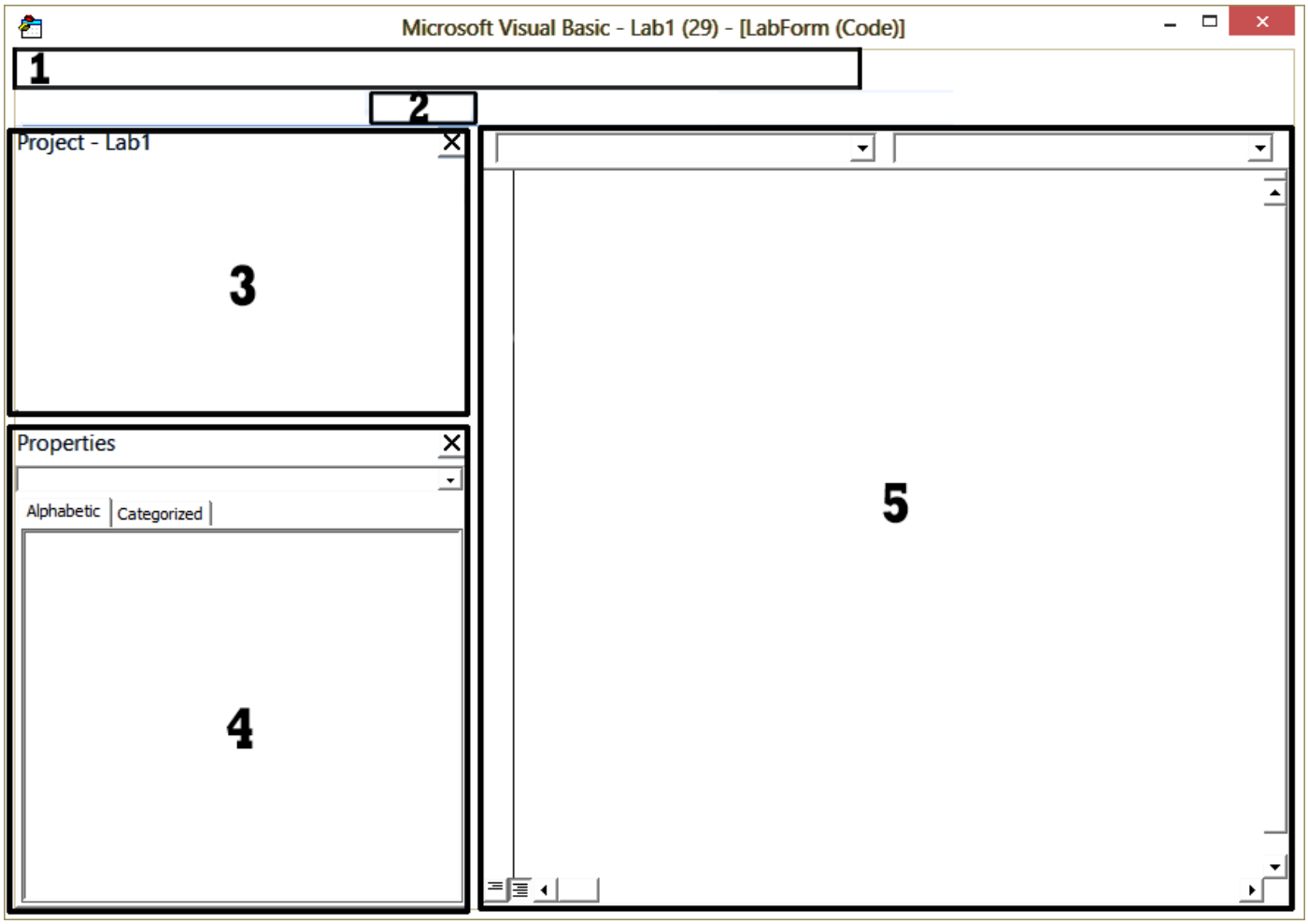
Some example image times include diagrams, maps, pie charts, flowcharts, 3D graphics.

1. Simplify the image by eliminating unnecessary graphic elements of the original image (or tell me what can be eliminated). Because you are a subject matter expert, I'd rather let you eliminate the non-essential pieces.
2. Identify and prioritize how important each graphic element is, so that we know which ones can be eliminated if we have to do so.
3. Export and send an image file as a .jpg, .png, .gif, or .svg format.

Below is an example of a complex image with "touch points" added:



Here is an example of the same image with all the detail eliminated. The order of each section is revised to make it easy for blind students to naviage.



2.2.3.1.2. Image descriptions

The following information will be used to create the tactile graphic:

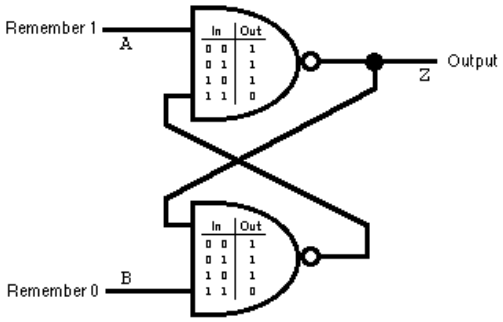
- Title of the image, for example: "Microsoft Visual Basic - Lab 1."
- Description of the image, for example: "This screenshot shows Microsoft Visual Basic user interface, which contains five different panels. Start from pane 1 to 5, Project pane, Properties pane, Code pane, toolbar pane, and menu bar pane in counterclockwise."
- Descriptions of individual elements, corresponding to the touch points above. For example:
    1. Menu bar - Here is a menu bar with various commands. These commands all have keyboard shortcuts. Under the File menu, there is a Save command, which you can run with either Alt then f then s, or by typing control plus s. Under the View menu, there is a Code command, which will show the Code Window for whatever item is selected in the Project Window. Under the Debug menu are commands for executing your program one line at a time, and for putting breakpoints in your program. Under the Run menu are commands for starting and stopping your program.
    2. Run, break, and stop tools - Here is a toolbar with various commands. These commands all have keyboard shortcuts. We will be using the following commands: Save - control plus S, Run - F5, Break or Pause - control plus break, Reset or Stop - Alt then R then R again.
    3. Project Window - This pane contains information about your Word document. Each open Word document has one top-level folder. Then, inside each document are various subfolders. In this class, we will be using the Forms folder for all the labs, and in Lab 5, we will be using a Class Modules folder. Inside the Forms folder, there is an item called LabForm. If you select this item and press F7, the LabForm code will appear in the Code Window.
    4. Properties Window - This pane contains information about the currently selected object. We will be using this pane in Lab 5, when we use it to change the name property of a new class that we create.
    5. Code Window - This pane contains the actual VBA code you will write in the labs. Each lab starts you off with some VBA code that provides a framework for your program. You can ignore most of this starter code except for an area where you will be adding variables, an area where you type in your driver module name, and an area where you will be typing your code.

2.2.3.2. Sample Images

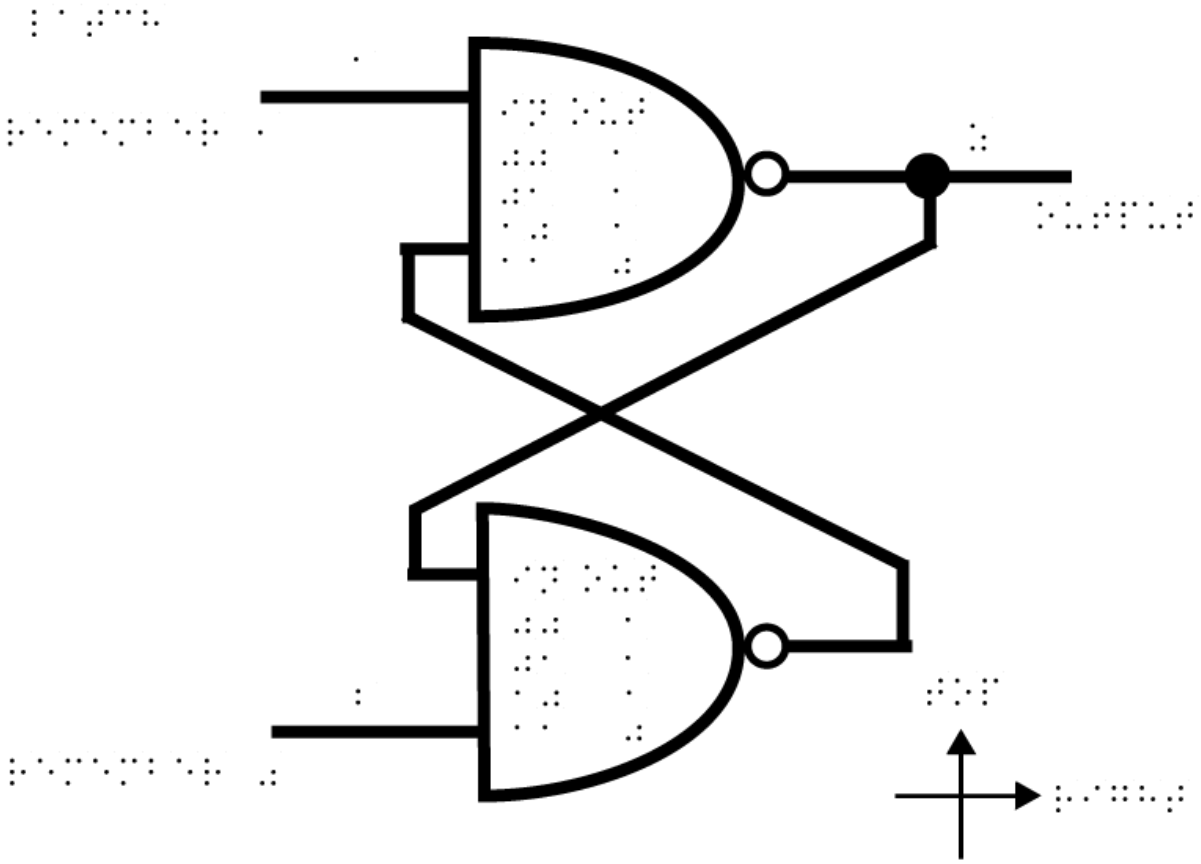
Here are some sample tactile graphics that Supada created that demonstrate various features that are useful for blind students.

2.2.3.2.1. A Latch Diagram

Below is an image of a latch logic circuit diagram:

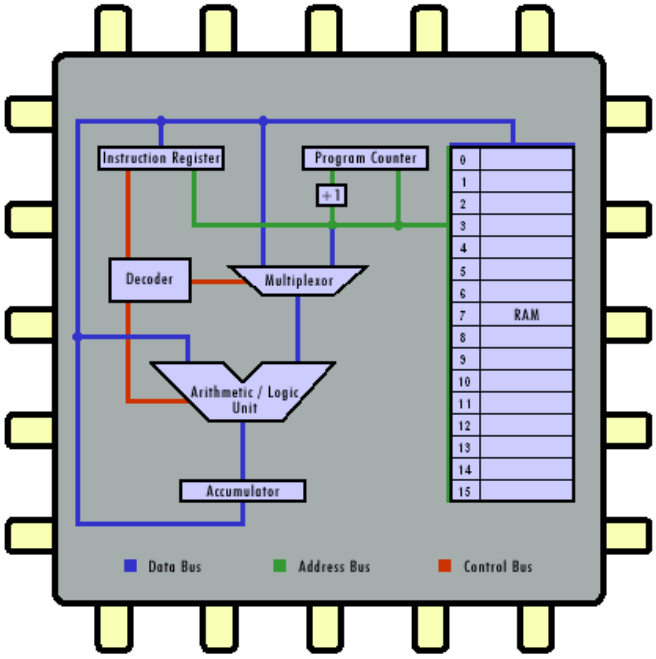


Here is an example of the same image converted to a tactile format, with Braille text:

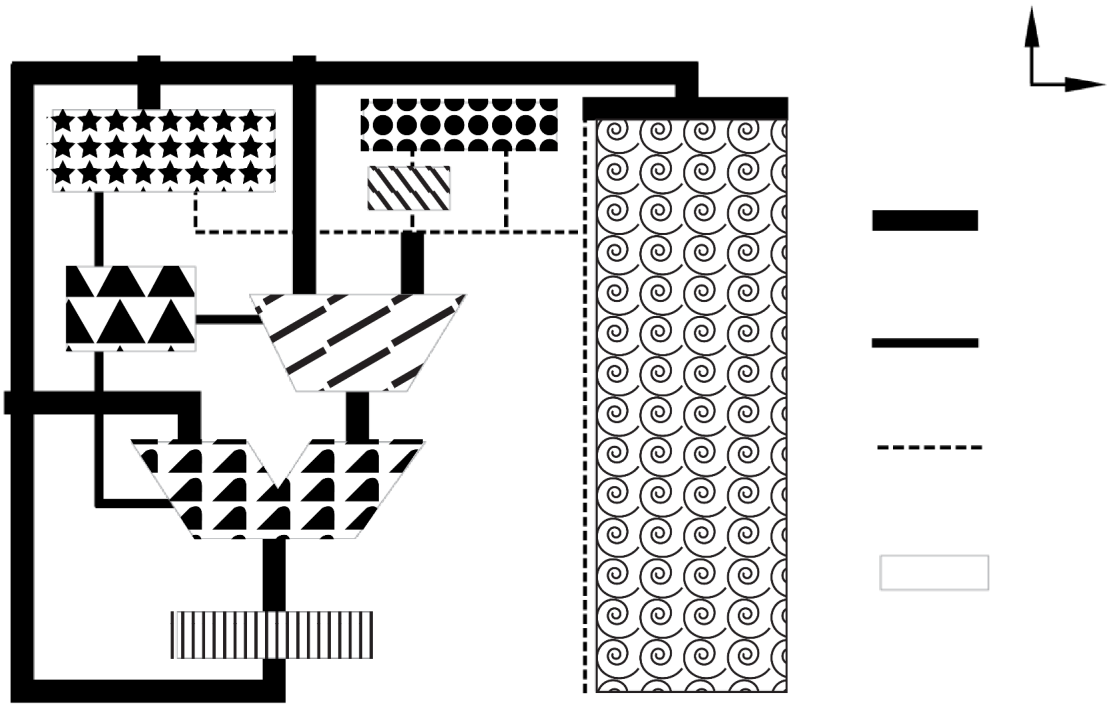


2.2.3.2.2. A CPU circuit with Braille tactile graphics

Below is a block diagram of a CPU:



Here is a tactile version of the graphic:



In this example, line weight and style (dashed vs. solid) have been used to tactilely distinguish between the different busses. Different tactile textures have also been added to each of the functional components of the CPU to help the student distinguish between them. Ideally, the student would also be given a legend that matched these textures to a Braille description of the component.

2.2.4. Braille and Braille Displays

Braille is a writing system that maps characters to rectangular blocks of dots (three dots high by two dots wide) called "cells." In Grade 1 Braille, each character is represented by one cell. In Grade 2 Braille, which is more advanced, hundreds of common abbreviations and contractions are added. Grade 2 Braille is most commonly used in publications, and is generally what is meant by the phrase "English Braille." Grade 3 Braille refers to personal shorthands that a Braille user might develop, and is almost never used in publications. These patterns of dots can be printed as embossed bumps on a page, and a blind individual can read them with their sense of touch. Pages of embossed Braille text can be produced using Braille typewriters or by

using embossing printers.

Refreshable Braille displays are electro-mechanical devices that include the ability to generate raised bumps dynamically. Commercially available Braille displays can generally display one line of between 12 and 80 cells at a time. In some models, vibration is used on particular cells to indicate the location of a cursor in the text. Prototypes for multi-line Braille displays have been developed, but no multi-line Braille displays are commercially available at this time.

Refreshable Braille displays may be the most popular option among professional programmers and others who need to closely proofread their work. However, they are quite expensive and screen reader software or printed Braille are more commonly used. Part of the reason for this is that only 13% of individuals with impairments (including all types of impairments, not just vision impairments) are aware of Braille embossers and printers, and only 8% are aware of refreshable Braille displays (see Accessible Technology in Computing - Examining Awareness, Use, and Future Potential, a study Commissioned by Microsoft, Conducted by Forrester Research, Inc. in 2004). In addition, only 10% of blind school children are taught Braille, even though 80% of all employed blind people read and write Braille fluently.

2.2.5. Narration

Audio narration of video media or animated graphical content provides an accessible alternative for blind students. It consists of an audio track of a narrator describing the action in the video or animation, often during natural pauses in the audio, and sometimes during dialog (if necessary). In some media players, audio narration can consist of a separate audio track to the video which includes descriptive text. When using players that don't support multiple audio tracks, the audio content can be included as a separate audio file. It is also helpful to provide a textual version of the narration that is screen-reader accessible, which may also be useful to students with hearing impairments (in addition to captioning of the video).

For more information on creating descriptive narration, please see Description Key. There is also a handy tip sheet on creating descriptive audio. Finally, see Providing Video for some examples of making video content accessible for programming labs.

2.2.6. Other Assistive Technologies

Some recent technologies are currently being investigated for their value in accessibility. Many of these technologies are promising, but may be challenging to apply, require specialized hardware that isn't commercially available, or are still being investigated. Three such technologies are described below.

2.2.6.1. Sonification

Sonification is the presentation of information using non-speech audio. For example, the pitch of a tone might rise and fall to convey the shape of a curve on a graph. Some forms of sonification are built into many applications: for example, a development environment might play an alert tone when the program encounters an error. Other forms of sonification are built into the screen reader program. For example, when a Windows progress bar is displayed, the NVDA screen reader will play a series of rising tones as the progress bar advances. Sonification is a powerful technique for assisting the blind, however making good use of sonification in your course materials may require a certain amount of creativity.

2.2.6.2. Haptics

Haptics is the presentation of information through the sense of touch. Tactile graphics and Braille displays are both examples of haptic technology. Other examples include having a touch screen vibrate to indicate the outline of a shape as the student traces their finger along the outline, or to "feel" the location of user interface elements on the screen.

Haptic technologies also include the use of "force feedback," where the student interacts with special purpose devices that convey the sense of interacting with a real object by producing pressure or resistance. For example, a force feedback joystick can be used to indicate the weight of an object by producing a force against the student's hand. Some force feedback devices are commercially available (for example, rumble pad video game controllers and force feedback joysticks), but other devices are more experimental and are not commercially available at the present time.

2.2.6.3. 3D Printing

3D printing is a promising new technology that enables the user to create physical objects from models on the computer. Many of these models are available for free, online (for example, see the Thingiverse web site). For example, a model of a human heart can be printed in plastic and given to a blind student in a biology course, or an abstract model representing a linked list data structure can be printed and given to a student in a Computer Science course. Consumer-level 3D printers are available, and are generally much less expensive than many other assistive technologies, because they are targeted at a mass market. Generally, these printers create physical objects (refered to as "parts") out of PLA or ABS plastic. PCC recently launched a "MakerSpace" in HP202 at the Sylvania campus where a variety of 3D printers and other "rapid manufacturing" equipment can be found. Generally, if a model is already available online, then printing it is a simple matter (though, it might take several hours depending on the size and level of detail of the model). However, creating a new model from scratch is fairly complicated technically, and requires familiarity with CAD packages.

3. Making Programming Labs Accessible

Creating a programming lab that is accessible to blind students is a complex task, and requires a certain amount of specialized knowledge. In order to aid in this process, we have identified several characteristics of a programming lab that can play a role in accessibility. These characteristics have been documented as a set of standards, and used to create a rubric for assessing whether a programming lab will be accessible to a blind student. The rubric is designed to be useful for two separate purposes:

- During design of new course materials, it can be used to avoid accessibility pitfalls that may be difficult to correct when a student with an accommodation signs up for the class. It can also be used to suggest supplemental materials that can improve the accessibility of programming labs.
- When a student with an accommodation signs up for a programming class, it can be used by Disability Services and Distance Learning to find accessibility obstacles and to suggest ways of addressing them.

Many of the standards listed here also apply to computer-related activities (other than programming) that a student may be asked to perform. For example, the student may be asked to create a document using Microsoft Word, or to create a web site using Dreamweaver. Standards such as keyboard accessibility and providing detailed instructions which are part of the programming lab rubric, will also play a role in the accessibility of these learning activites. A subset of the standards presented in this rubric can serve as the basis for creating rubrics for assessing other types of computer-related learning activities. For example, a rubric that covers document creation assignments may include many of the standards listed in this document.

It is important to note that the rubric is intended to support our ongoing efforts to provide accessible content to our diverse student body, and to support the development of accessibility plans moving forward. It is not intended as a set of new and additional requirements for determining whether a course meets PCC's existing accessibility standards. In particular, it is possible for an existing course to meet Standard 8 of PCC's Quality Matters Rubric without meeting all the standards listed below (however, some requirements, such as the requirement for accessibly formatted documents, are already a part of our Standard 8 review, and will continue to be required before the course can be recommended for online teaching).

3.1. Programming Lab Accessibility Rubric for Visually Impaired Students

The following rubric has been developed to assess whether a programming lab is accessible to blind students. The rubric lists a set of standards that can play a role in making programming labs accessible. Each standard has been assigned one of three levels of importance: Essential, Important, and Helpful. The importance level relates to accessibility in the following ways:

1. **Essential**: Each essential standard must be met for a blind student to complete the programming lab substantially on their own. If an essential standard is not met, then an Assistive Aid or the course instructor will be required through the entire process of completing the lab, or the student will need to be provided with an equally effective, equally integrated alternative.
2. **Important**: If all essential standards are met, but one or more important standard is unmet, then the student will be able to complete most of the lab on their own, but will require help from an Assistive Aid or the course instructor to overcome some accessibility obstacles.
3. **Helpful**: The student will be able to complete the lab substantially on their own if all essential and important standards are met, but will be able to work more effectively if helpful standards are also met.

The rubric consists of the following standards:

Requirement	Importance	Accessible	Not Accessible
Programming Language	Essential	The programming language is text based. Languages that are insensitive to capitalization errors and indentation are comparatively more accessible than other languages[1].	The language is primarily graphical or uses a drag-and-drop metaphor for programming.[2]
Keyboard Shortcuts	Essential	The development environment provides keyboard shortcuts for all commonly used actions. Workarounds[3] are available for less commonly used actions that don't have keyboard shortcuts.	Keyboard shortcuts are unavailable and no workarounds exist.
Screen Reader Compatibility	Essential	Screen reader software will provide descriptive text for most user interface elements (such as buttons, menus, text labels, etc.) in the development environment. Interface elements without tab stops or other keyboard shortcuts are readable with the JAWS cursor or NVDA Object Navigation, Screen Review and Document Review.	Screen reader software does not read the majority of user interface elements, or required actions can only be performed using visual information.
Document Formatting	Important	Assignment documents are properly formatted for maximum compatibility with screen reader software.	Assignment documents are improperly structured and will appear as a "blob of text" to screen reader software[4].
Diagrams	Important	If information is provided in the form of a system diagram, UML Class diagram, Use-case diagram, or other visual format, then descriptive text or tactile graphics[5] are provided as an alternative to this information.	Information required to complete the lab is provided in graphical format without alternative text or descriptions.
Video	Important	If a video walkthrough or lecture is provided, then narration describes all of the actions that the student must perform to complete the lab.	Actions are shown in the video but not described in the narration.
Example Code	Important	If example code or code fragments are given, they should be presented in text format or as a screen capture with descriptive text.	Lab requires the student to manually type in text from a screen capture or other graphical format.
Debugging	Important	Compiler or runtime interpreter should provide errors in textual format. Breakpoints, stepping, variable values, etc. should all be compatible with keyboard shortcuts and screen readers.	Error information relies on highlighting or other visual presentation of information that is

			not compatible with screen readers.
Interface Design	Important	If the assignment requires designing and laying out user interface elements, the student will be able to do so using code.	The student is required to use a GUI-based interface design tool, which requires the use of the mouse and the screen.
Output Requirements	Important	Output is primarily static, textual, and is screen-reader accessible.	Output is primarily graphical or animated[6].
Sample Output	Important	If sample output is provided, it is given in text format or as a screen capture with descriptive text that fully specifies the output requirements.	A screen capture of sample output is provided without an accompanying description or suitable alternative text.
Shortcuts List	Helpful	Frequently used keyboard shortcuts are provided as a separate document and are suitable for printing in Braille.	No documentation on keyboard shortcuts is provided, or an exhaustive list is provided that requires the student to search for each shortcut.
Instructions	Helpful	If a series of actions must be performed by the student, these instructions should be provided as a numbered list. It is also helpful to provide the keyboard shortcuts in these instructions[7].	Instructions are provided in paragraph format or other unstructured block of text.

Notes:

1. Textual languages that rely on indentation level for meaning, such as Python, can be used by visually impaired students with refreshable Braille displays, or using a screen reader to read character-by-character. However, this does provide an additional obstacle to accessibility.
2. Some examples of primarily graphical programming languages in use at PCC include Scratch, LabVIEW and GameMaker. These languages will be extremely difficult to use for visually impaired students.
3. The exact form of the workaround depends on the type of assignment and the development environment. Please see the Case Studies section for some examples.
4. Document formatting is listed as "Important" in the rubric rather than as "Essential," because if everything in the lab is accessible except the document formatting, the student will likely be able to complete most of the lab on their own, with some help from an Assistive Aid. However, the Quality Matters Rubric that PCC uses to evaluate online course shells includes additional requirements in Standard 8 that must be met for a course to be considered "ready to teach online."
5. Please see the section on Tactile Graphics for more information on this technology and its limitations.
6. Examples of primarily graphical or animated output include the output from a Scratch or CeeBot program, or an interactive game written in GameMaker.
7. An example of providing keyboard shortcuts in the instructions would be to specify "Type Alt+F followed by S in order to save the document," instead of "Select Save from the File menu."

3.2. Programming Language

The Lab Rubric specifies that the programming language is text based, and that languages that are insensitive to capitalization errors and indentation are comparatively more accessible than other languages.

The table below presents information on whether selected programming languages are accessible to blind students. The selected languages represent the most commonly taught languages for the CAS, CIS and CS departments.

Note that, in some cases, the programming language itself might be text based and (in theory) accessible, however there my be no accessible development environment. For example, CeeBot is a text-based language similar to the C programming language, but can only be written and run within the CeeBot development environment, which is inaccessible and cannot produce accessible output. In such cases, we list the level of programming language accessibility as "partial," but would list the development environment as inaccessible due to keyboard shortcut requirements or screen-reader compatibility requirements. Generally, a blind student will not be able to complete those labs without substantial help from an Assistive Aid or the Instructor.

Language	Accessible?	Notes
ActionScript	**Partially**	ActionScript is a scripting language inside of Adobe Flash. The scripting language is accessible, but the rest of Flash may not be, and there may be no way to generate accessible output.
bash Script	Yes	
C#	Yes	The language is entirely text-based and accessible, however some tools like the Visual Studio Forms Designer may not be accessible (see Keyboard Shortcuts).
C/C++	Yes	
CeeBot	**Partially**	The language itself is text based, but there is no accessible development environment, and no way to generate accessible output.

CSS	Yes	
GML	**Partially**	GML is a scripting language inside of GameMaker. The scripting language is accessible, but the rest of GameMaker uses drag-and-drap programming, and is inaccessible (see Keyboard Shortcuts).
HTML/XHTML	Yes	
Java	Yes	
Javascript	Yes	
LabVIEW	**No**	LabVIEW uses drag-and-drop programming, and cannot be programmed without using the mouse and the screen.
PHP	Yes	
Python	Yes	Note that while the language is text-based and accessible, it also relies on proper indentation, which may require the use of a refreshable Braille display for maximum ease of use.
QBasic	Yes	
Scratch	**No**	Scratch uses drag-and-drop programming, and cannot be programmed without using the mouse and screen.
SQL (Transact-SQL, PL/SQL)	Yes	
UnityScript	**Partially**	UnityScript is a scripting language inside of Unity. The scripting language is accessible, but the rest of Unity uses a complex GUI, and is inaccessible (see Keyboard Shortcuts).
VBA	Yes	The language is entirely text-based and accessible, however some tools like the Forms Designer in the Visual Basic Developer's Window may not be accessible (see Keyboard Shortcuts).
Visual Basic.NET	Yes	The language is entirely text-based and accessible, however some tools like the Visual Studio Forms Designer may not be accessible (see Keyboard Shortcuts).

3.3. Keyboard Shortcuts

The Lab Rubric specifies that the development environment should provide keyboard shortcuts for all commonly used actions, and that workarounds are available for less commonly used actions that don't have keyboard shortcuts.

Development Environment	Languages	Accessible?	Notes
bash shell	bash script, C/C++ (via gcc)	Yes	bash is a text-based program that runs in a terminal window, such as xterm, GNOME terminal, or PuTTY. These terminal window programs are primarily textual and keyboard accessible.
Bloodshed	C/C++	Yes	The Dev-C++ "Orwell" development environment includes keyboard shortcuts. Users can also add their own customized keyboard shortcuts. However, there doesn't seem to be a list of default shortcuts online.
BlueJ	Java	Yes	Most operations have keyboard shortcuts. Some operations, such as creating "Uses" and "Inheritance" arrows, are not fully keyboard accessible. See The BlueJ Environment Reference Manual for available shortcuts, particularly section 3-10: "Use BlueJ using only the keyboard." There are workarounds for all critical actions.
CeeBot	CeeBot	**No**	The CeeBot development environment does not provide keyboard shortcuts.
Chrome	Javascript	Yes	Documentation is available on Windows keyboard shortcuts and Mac keyboard shortcuts.
Dreamweaver	HTML/CSS/Javascript	Yes	Dreamweaver has the ability to customize keyboard shortcuts as well as to create a reference sheet for the current shortcuts. Documentation is at Dreamweaver / Keyboard shortcuts.
Eclipse	Java, C/C++, PHP	Yes	A list of Eclipse keyboard shortcuts (for the default configuration) is available. The user may also create custom shortcuts.
Flash	ActionScript	**Partially**	A list of Flash keyboard shortcuts is available online. Some activities (such as designing animations) will require the student to interact with Flash using the mouse and screen.
GameMaker	GML, GameMaker	**Partially**	Some commands are keyboard accessible (see the global user interface document. However, several critical actions require the use of the mouse and screen.
gcc/gdb	C/C++	Yes	gcc and gdb are commands that are executed from a shell (such as the bash shell) running in a terminal program (such as xterm or PuTTY). Most terminal programs are keyboard accessible, so the student will be able to use gcc and gdb with the keyboard.
Internet Explorer	Javascript	Yes	Keyboard shortcuts for Internet Explorer vary from version to version. Most versions are fully keyboard accessible, and are documented online at Internet Explorer keyboard shortcuts.

LabVIEW	LabVIEW	**No**	LabVIEW is a drag-and-drop programming language, and is not keyboard accessible.
LaTeX	LaTeX	Yes	LaTeX is a text-based typesetting language. It can be used with most text editors. When used with a keyboard-accessible text editor, LaTeX will be keyboard-accessible as well.
MS Access	Access	Yes	Keyboard shortcuts for MS Access vary from version to version. Shortcuts for Microsoft Access 2013, the latest version, are available online. Separate documents listing shortcuts for previous versions are also available online.
MS SQL Server Management Studio	SQL	Yes	Keyboard shortcuts for MS SQL Server Management Studio 2012 are available online. This document also links to keyboard shortcuts for previous versions.
Notepad	HTML, CSS, Javascript, Java, PHP, etc.	Yes	Notepad can be used to edit most text-based programming languages. Lists of keyboard shortcuts for Notepad are available online.
Notepad++	HTML, CSS, Javascript, Java, PHP, etc.	Yes	Notepad++ is an open-source replacement for Notepad that includes plug-ins and support for a variety of text-based programming languages. A list of keyboard shortcuts for Notepad++ is available online.
Oracle SQL Developer	SQL	Yes	Keyboard shortcuts (Oracle calls them "accelerators" can be found at SQL Developer Accelerator Hotkeys, or by viewing the Accelerators Pane (documented on page 1-48) in SQL Developer.
PuTTY	bash script, C/C++	Yes	PuTTY provides a terminal window to a Unix or Linux system. It is primarily text-based and keyboard driven. The exact keyboard shortcuts will vary depending on which program is running in the PuTTY window.
Scratch	Scratch	**No**	Scratch is a drag-and-drop programming language, and is not keyboard accessible.
Unity	UnityScript, Unity	**Partially**	A list of keyboard shortcuts (hotkeys) for Unity are available online. Many tools will require the user to interact with the mouse and screen and are not keyboard accessible.
Visio	Visio	**Partially**	Keyboard shortcuts for some Visio commands are available online. However, designing a viewable diagram will require the student to interact with the mouse and screen.
Visual Studio	C/C++, C#, Visual Basic.NET	Yes	Keyboard shortcuts for Visual Studio 2013, the current version, are available online. Keyboard shortcuts for previous versions differ somewhat, and are also available online. Some tools, such as the Forms Designer, may not be entirely keyboard accessible.
XCode	C/C++	Yes	XCode supports both gestures and keyboard shortcuts. Some tools, such as the Interface Builder for iOS apps may not be completely keyboard accessible.
xterm	bash script, C/C++	Yes	xterm provides a terminal window to a Unix or Linux system which is compatible with a window-oriented desktop environment. It is primarily text-based and keyboard driven. The exact keyboard shortcuts will vary depending on which program is running in the xterm window, and which desktop environment is running on the computer (e.g., Windows, OSX, GNOME, KDE, etc.).

3.4. Screen Reader Compatibility

The Lab Rubric specifies that screen reader software should provide descriptive text for most user interface elements in the development environment, and that interface elements without tab stops or other keyboard shortcuts are still readable.

There is additional information available on screen reader software. Most of the below tests were run using the NVDA screen reader.

Development Environment	Languages	Accessible?	Notes
bash shell	bash script, C/C++ (via gcc)	Yes	Some experimentation will be necessary to determine the best solution for a particular student. bash is a text-based program that runs in a terminal window, such as xterm, GNOME terminal, PuTTY terminal window, and several others. GNOME terminal in Linux is compatible with the Orca screen reader, and MinGW under Windows is compatible with the NVDA and JAWS screen readers. xterm under Linux and cygwin under Windows are not compatible with ORCA or NVDA, and require the use of Linux Speakup. PuTTY running on a Linux system requires Speakup, but PuTTY running on a Windows system will work with JAWS or NVDA.
Bloodshed	C/C++	Yes	
BlueJ	Java	Yes	BlueJ is compatible with screen reader software, but requires Java Access Bridge to be enabled (it comes packaged with Java, but is disabled by default).

CeeBot	CeeBot	**No**	
Chrome	Javascript	Partial	Chrome is compatible with some screen readers but not all. Note that the linked page claims that Chrome is compatible with NVDA, but I had some problems accessing Chrome content with NVDA that I didn't have with Firefox or Internet Explorer. It is also not compatible with Orca under Linux.
Dreamweaver	HTML/CSS/Javascript	Yes	Adobe claims screen reader support for DreamWeaver. However, this hasn't been thoroughly tested at PCC at the present time.
Eclipse	Java, C/C++, PHP	Yes	
Flash	ActionScript	**Partially**	Some user interface elements in the Flash development environment are screen-reader compatible, but many actions require the use of the mouse and screen, and making the requested output accessible requires significant additional work.
GameMaker	GML, GameMaker	**Partially**	Some user interface elements in GameMaker are screen-reader compatible, but many actions require the use of the mouse and screen, and making the requested output accessible requires significant additional work.
gcc/gdb	C/C++	Yes	When used with a compatible terminal window (see the notes for "bash shell" above), gcc and gdb will be accessible.
Internet Explorer	Javascript	Yes	
LabVIEW	LabVIEW	**No**	Not compatible with JAWS or NVDA.
LaTeX	LaTeX	Yes	If used with a screen-reader compatible text editor, then LaTex will be screen-reader compatible. Please see the Accessible Content Creation in Mathematics Subject Area Study for more details.
MS Access	Access	Yes	
MS SQL Server Management Studio	SQL	Yes	
Notepad	HTML, CSS, Javascript, Java, PHP, etc.	Yes	
Notepad++	HTML, CSS, Javascript, Java, PHP, etc.	Yes	
Oracle SQL Developer	SQL	Yes	Oracle SQL Developer is compatible with screen reader software, but requires Java Access Bridge to be enabled (it comes packaged with Java, but is disabled by default).
PuTTY	bash script, C/C++	Partial	Compatible with JAWS or NVDA under Windows, but not compatible with Orca under Linux.
Scratch	Scratch	**No**	
Unity	UnityScript, Unity	**Partially**	Some user interface elements will be screen reader accessible, but many critical activities require the use of the mouse and screen.
Visio	Visio	**Partially**	Some user interface elements will be screen reader accessible, but many critical activities require the use of the mouse and screen.
Visual Studio	C/C++, C#, Visual Basic.NET	Yes	Some tools, such as the Forms Designer tool, may not be fully compatible with screen reader software.
XCode	C/C++	Yes	Some tools, such as the Interface Builder tool, may not be fully compatible with screen reader software.
xterm	bash script, C/C++	No	xterm is not compatible with Orca. It may work with Speakup under Linux. I have tested running MobaXterm on my Windows machine, and using PuTTY to open an xterm window on my Windows machine, but NVDA doesn't read the contents of that window either.

3.5. Document Formatting

The Lab Rubric specifies that the assignment documents are properly formatted for maximum compatibility with screen reader software.

This standard applies to learning resources in general. An excellent collection of resources on making learning resources accessible is available on PCC's Instructional Support Resources on Accessibility for Online Course Content. Rather than repeating all of that information here, some of the

important themes include:

- Choose fonts that are readable using assistive technology, or use the standard template pages provided in your course development shell in Desire2Learn.
- Use properly formatted headings to structure your documents (<h1>, <h2>, <h3>, etc. for web pages; Heading 1, Heading 2, Heading 3, etc. for Word documents).
- Provide alternative text for images and graphics.
- Write meaningful link text that indicates a link's destination. For example:
  - **Good:** A rubric for evaluating whether programming labs are accessible
  - **Poor:** Click here for a rubric on evaluating whether programming labs are accessible
  - **Poor:** A page on evaluating whether programming labs are accessible is available at [http://www.pcc.edu/access/studies/cs_cis_cas/rubric.php].
- Use column headers on data tables.
- Use properly formatted bulleted and numbered lists rather than blocks of text with line breaks.
- Avoid using color alone to convey meaning (for example, don't use red text as the sole method for marking that an instruction is important).

As mentioned above, for more information on creating accessible web pages, Word documents, Powerpoint presentations, PDFs, and other learning resources, please visit PCC's Instructional Support Resources on Accessibility for Online Course Content. It is also worth noting that Karen Sorensen, PCC's Accessibility Advocate, offers frequent online training sessions on document accessibility. For example, here is a recording of one of Karen's sessions on making Microsoft Word 2010 documents accessible.

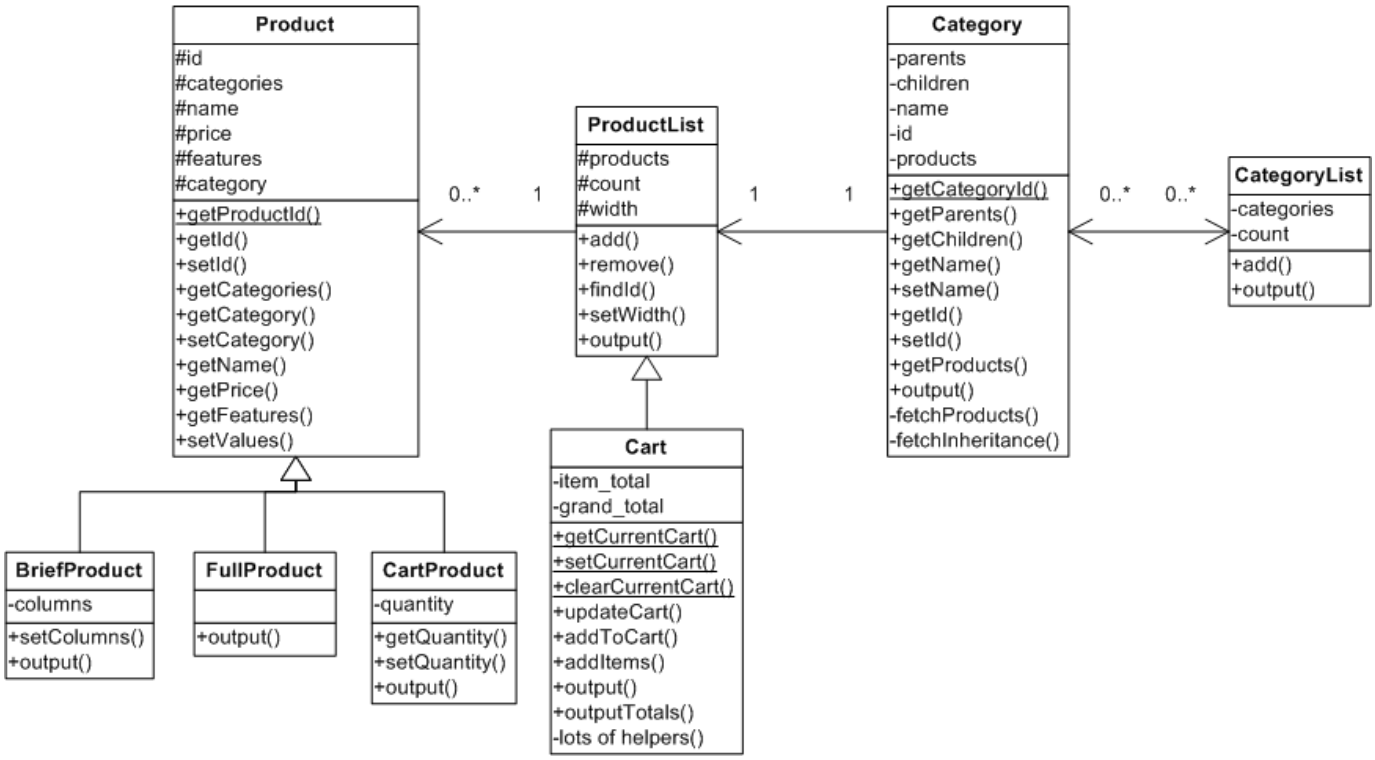
You might also find these videos helpful. They were developed for use in the Online Instructor Orientation in Spring, 2014. In the first video, we look at some resources for creating accessible documents, and talk about some general guidelines:

Here, we create an accessible document based on the Lecture template in Desire2Learn, using the HTML templates provided as part of all development shells (if you don't already have a copy, contact dlhelp@pcc.edu and ask for a copy of the standard course development shell.

3.6. Providing Diagrams

The Lab Rubric specifies that if information is provided in the form of a system diagram, UML Class diagram, Use-case diagram, or other visual format, then descriptive text or tactile graphics should be provided as an alternative to this information.

Here is a UML Class diagram from a CIS 195P course shell:



This diagram shows important relationships between classes, and between classes and their methods and properties. Students can use such diagrams to understand the overall design of an existing system, as well as to implement and extend the functionality of a system. Much of this information is presented visually, and will be inaccessible to a blind student. Two methods for providing accessible alternatives for this diagram are:

1. Provide a text-based alternative that describes the important information in the graphic.
2. Provide a tactile version of this graphic.

Please see the section on tactile graphics for more information on this technology and its suitability for various types of content. A text-based alternative for a subset of the above graphic (the Product class and its subclasses) might include the following information:

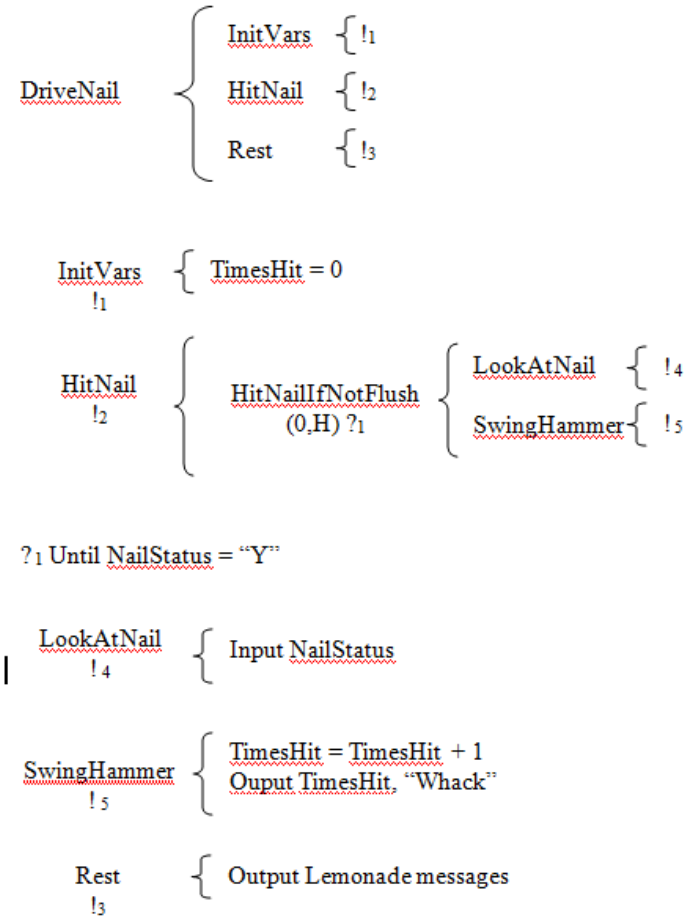
Product Class
- Has the following subclasses:
  - BriefProduct Class
  - FullProduct Class
  - CartProduct Class
- Is contained by:
  - 0 or more ProductList objects
- Has the following properties:
  - id (protected)
  - categories (protected)
  - name (protected)
  - price (protected)
  - features (protected)
  - category (protected)
- Has the following methods:
  - getProductId (static and public)
  - getId (public)
  - setId (public)
  - getCategories (public)
  - getCategory (public)
  - setCategory (public)
  - getName (public)
  - getPrice (public)
  - getFeatures (public)
  - getValues (public)

- BriefProduct Class
  - Is a subclass of:
    - Product Class
  - Has the following properties:
    - columns (private)
  - Has the following methods:
    - setColumns (public)
    - output (public)
- FullProduct Class
  - Is a subclass of:
    - Product Class
  - Has the following properties:
    - none
  - Has the following methods:
    - output (public)
- CartProduct Class
  - Is a subclass of:
    - Product Class
  - Has the following properties:
    - quantity (private)
  - Has the following methods:
    - getQuantity (public)
    - setQuantity (public)
    - output (public)

Note that while the text alternative might not be as easy to grasp by the student, all of the important relationships have been captured, and the information is structured in such a way as to be compatible with screen reader software.

Another example is this Warnier-Orr diagram that is used to document a demo program in some of our CIS 122 course shells:

Warnier/Orr:



This demo meets the accessibility standard by providing equivalent pseudocode, following the class pseudocode standard:

```
DriveNail Module
        INITVARS
        HITNAIL
        REST
End Module

INITVARS Module
        TimesHit = 0
End Module

HITNAIL Module
        LOOKATNAIL
        Do PreTest Until NailStatus = "Y"
                TimesHit = TimesHit + 1
                SWINGHAMMER
                LOOKATNAIL
        End PreTest
End Module

LOOKATNAIL Module
        Input NailStatus
End Module

SWINGHAMMER Module
        Ouput TimesHit, "Whack"
End Module

REST Module
        Output Lemonade messages
End Module
```

As with many of the strategies described in this guide, providing multiple alternative versions of the content will also help students with different learning strategies to absorb this material, in addition to providing accessible alternatives for blind students.

3.7. Providing Video

The Lab Rubric specifies that if a video walkthrough or lecture is provided, then narration describes all of the actions that the student must perform to complete the lab.

Very few of our instructional videos currently meet this standard. For example, here's a short excerpt of a video that Marc Goodman (one of the authors of this report) created as a lab walkthrough for his CIS 122 course shell (the actual video used in the course shell includes captions for the hearing

impaired, which are not reproduced in this excerpt. Here is a link to the full video used in the course):

Your browser does not support the video tag.

As you can see from this video, very few of the things that Marc says in his narration match what Marc types in the video exactly. In addition, some of the code that Marc types has no associated narration at all. Marc also clicks on a line to insert text, without describing which line he is clicking on, and refers to a "yellow box in our dialog." All of these issues will make the video far less useful for a blind student who is attempting to complete the lab. Three methods for making this video more accessible are:

- Create a separate narration as a stand-alone audio recording or as a text-based document that is accessible to screen readers, in which the actions performed onscreen are described to the student.
- Add a separate document that matches narration from the existing video to what is typed onscreen. Note that if the video has already been captioned, the existing narration can often be exported and used as the basis for this new document.
- Provide an equally effective, equally integrated set of instructions that can be used as an alternative to the video (see the section on providing instructions).

An example of matching the existing narration to the actions performed onscreen is given in the table below:

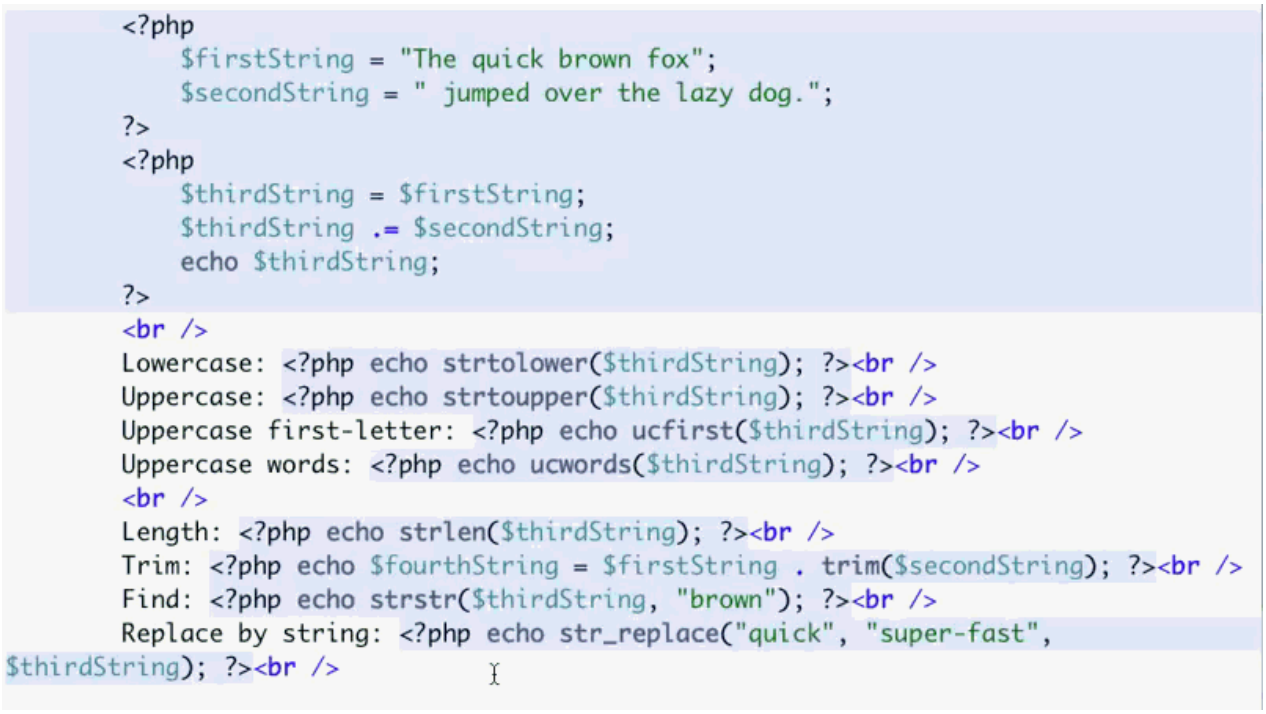
What Marc Says	What Marc Types
"So I Dim Number as Single precision floating point number"	Click on the line after the line that reads "'Variables: Place your Dim statement here." Type: Dim Number As Single [Carriage Return]
"And Dim Square as Single precision floating point number"	Type: Dim Square As Single [Carriage Return]
"And then, our Module Name goes here"	Click on the line that says 'Driver Module Name: <--- Enter the driver module name... and type in the module name "Square" after the ":".
"So, the first thing we want to do is input our number"	Click on the line after the line that reads "'Place your code below here." Type: Number = InputBox("Please enter a number.") [Carriage Return]
"Now we have our number and we want to calculate the square. In Visual Basic, you can raise a number to a power like this."	Type: Square = Number ^ 2 [Carriage Return]
"And finally, we want to output our line, and the way we're outputting our line is we're going to change the caption on that yellow box in our dialog. So, we say LabelOut.Caption is equal to"	Type: LabelOut.Caption =
"and we want to add our new line to the existing output. So, we get the existing caption"	Continue typing on the same line: LabelOut.Caption
"and we add ..."	Continue typing on the same line: & "The square of " & Number & " is " & Square
"and now we want to end the line. I'm running out of space, so I'm going to put an underscore to continue"	Continue typing on the same line: _, [Carriage Return followed by indentation, followed by] &
"So, first we want to break the line and we want to output a blank line after that. So that each of our outputs is separated, just like in the lab specification."	Continue typing on the same line: vbNewLine
"So we write ..."	Continue typing on the same line: & vbNewLine [Carriage Return]
"So, that's it!"	Nothing

Note that this content might also benefit students who are not native speakers of English, have hearing impairments, or who have cognitive impairments that make following along with the video more challenging. For a closely related standard, please see the section on providing instructions.

3.8. Providing Example Code

The Lab Rubric specifies that if example code or code fragments are given, they should be presented in text format or as a screen capture with descriptive text.

Instructors ofter provide example code or code fragments as screen captures. Often, this is done because it requires that the student actually type in the code themselves instead of copying and pasting it. This helps the student to develop their attention to detail, and provides opportunities for the student to practice debugging when they make an error. For example, the following image shows a screen capture from a Lynda.com video with some PHP code:



This image is used in a CIS 195P course shell, with the alt text "string function examples," and directions state that the student should "Be sure to type and test all of the code shown here." However, the content of this image will not be compatible with screen reader software, and a blind student will be unable to comply with these directions without the help of an Assistive Aid. In such cases, the instructor should carefully weigh the benefit of forcing sighted students to type this code, instead of copying and pasting, against the impossibility of a blind student being able to complete this assignment without help. Generally speaking, an accessible form of this content should be provided to students.

Here is a version of the above code that will be compatible with screen reader software:

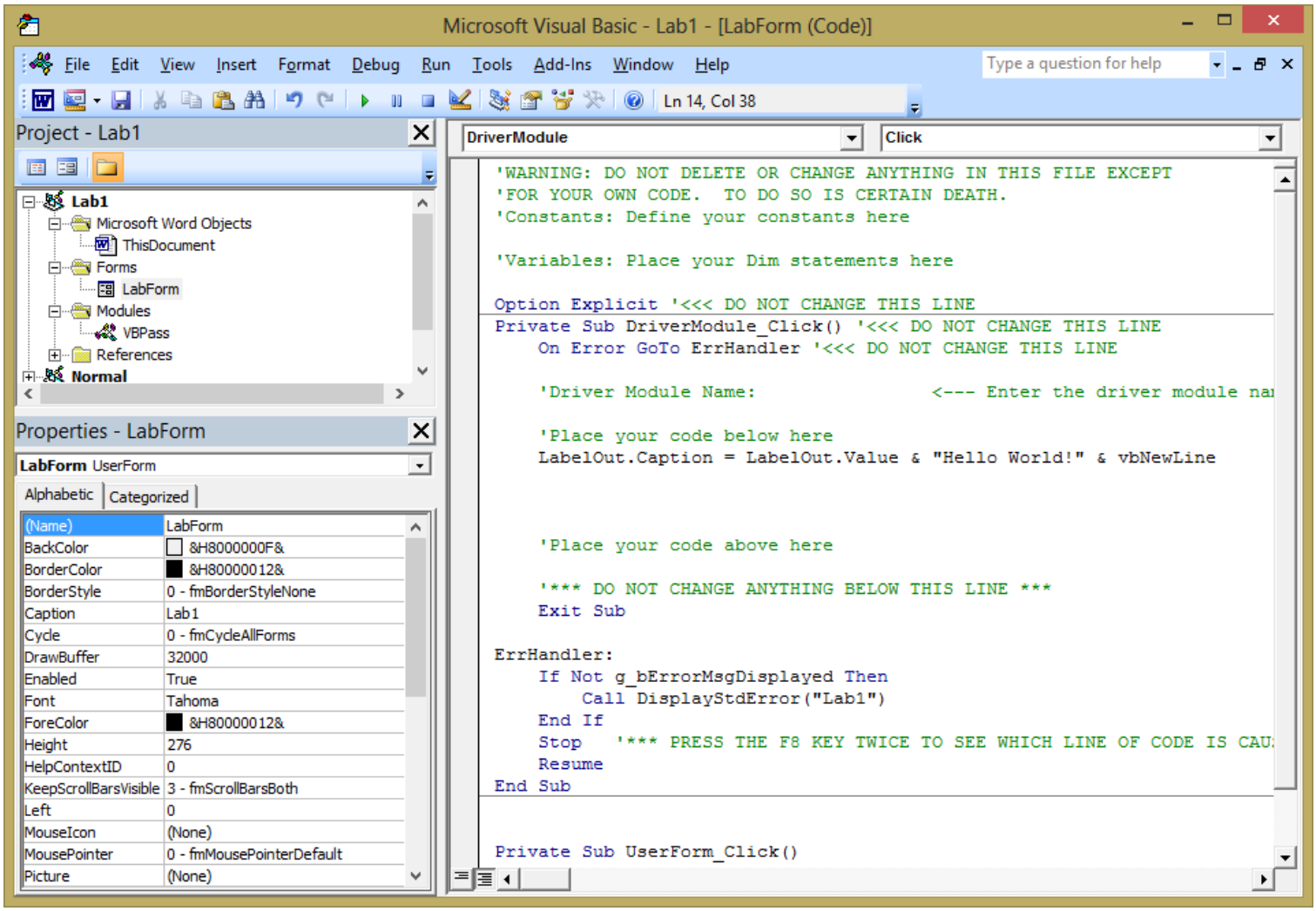
```
<?php
        $firstString = "The quick brown fox";
        $secondString = " jumped over the lazy dog.";
?>
<?php
        $thirdString = $firstString;
        $thirdString .= $secondString;
        echo $thirdString;
?>
<br />
Lowercase: <?php echo strtolower($thirdString); ?><br />
Uppercase: <?php echo strtoupper($thirdString); ?><br />
Uppercase first-letter: <?php echo ucfirst($thirdString); ?><br />
Uppercase words: <?php echo ucwords($thirdString); ?><br />
<br />
Length: <?php echo strlen($thirdString); ?><br />
Trim: <?php echo $fourthString = $firstString . trim($secondString); ?><br />
Find: <?php echo strstr($thirdString, "brown"); ?><br />
Replace by string: <?php echo str_replace("quick", "super-fast", $thirdString); ?><br />
```

At a minimum, it would be useful to create text versions of these images and to store them in the course shell under a "draft" accessibility module. Then, if students who need an accommodation register for the class, they can be provided with the accessible alternative to these images, while still providing the opportunity for "hands on" practice for your sighted students. Please also see the section on Sample Output for a closely related standard.

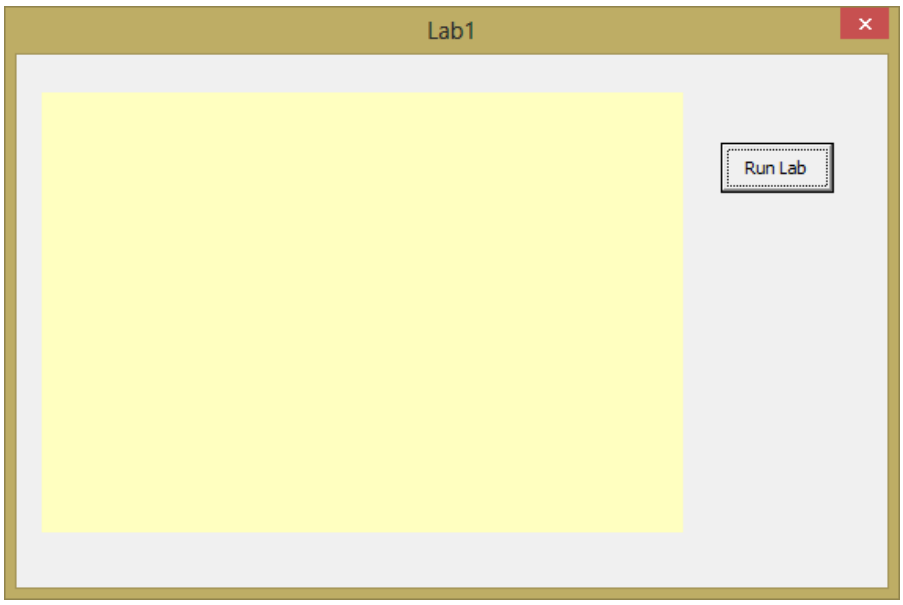
3.9. Debugging

The Lab Rubric specifies that the compiler or runtime interpreter should provide errors in textual format, and that breakpoints, stepping, variable names, etc. should all be compatible with keyboard shortcuts and screen readers.

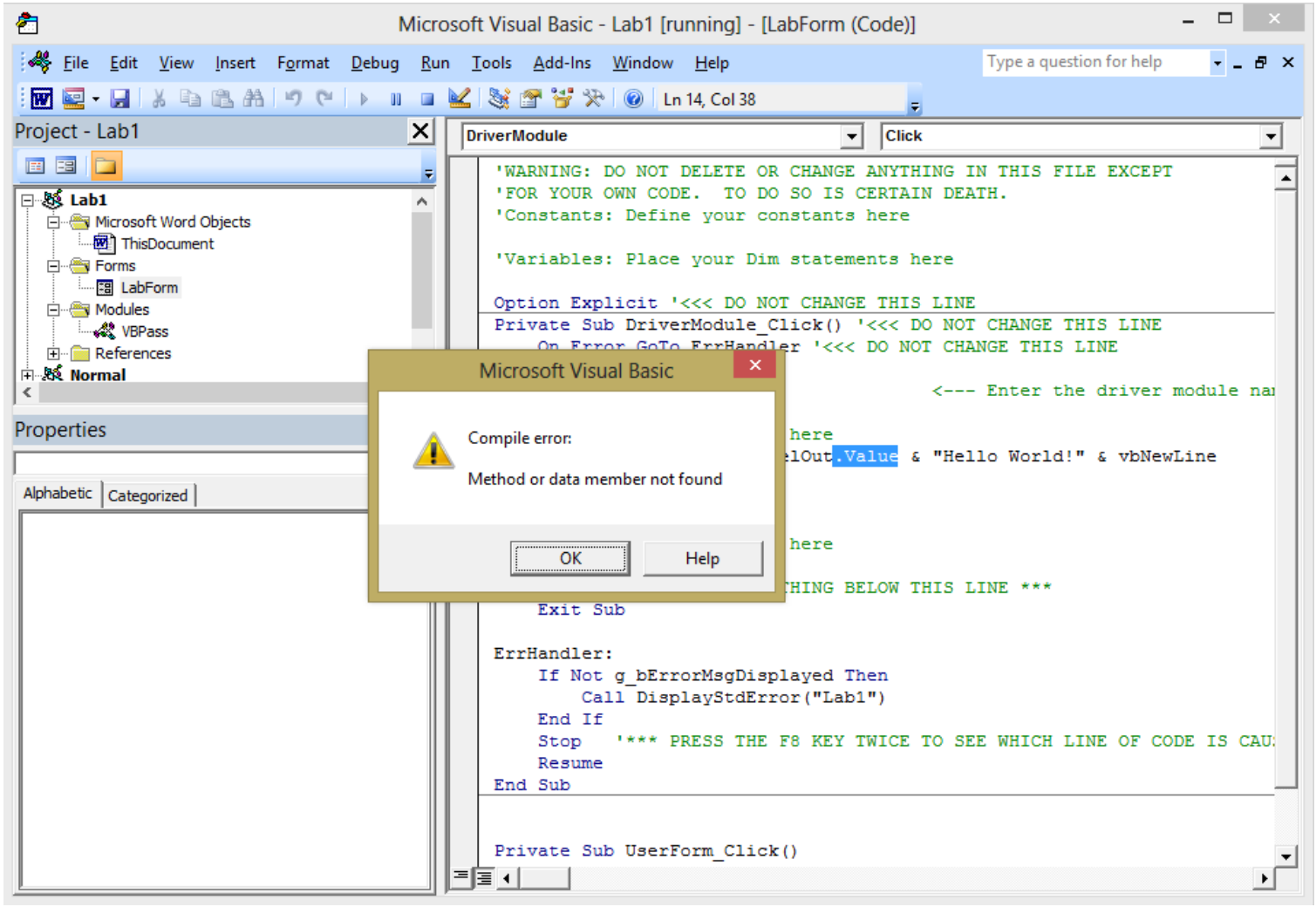
The ability to test and debug code is a learning outcome in many of our programming classes. It is also one of the most difficult skills for many of our students to master (both students with impairments and students without impairments). In addition, debugging is likely to provide special accessibility challenges in many IDEs. Most debuggers provide the ability to set breakpoints, watch variable values, and to step thorough the code. In addition, highlighting errors is common. Even in development environments that are mostly accessible, much of this information may not be compatible with screen reader programs. For example, here is a screen capture of a VBA program with a bug:



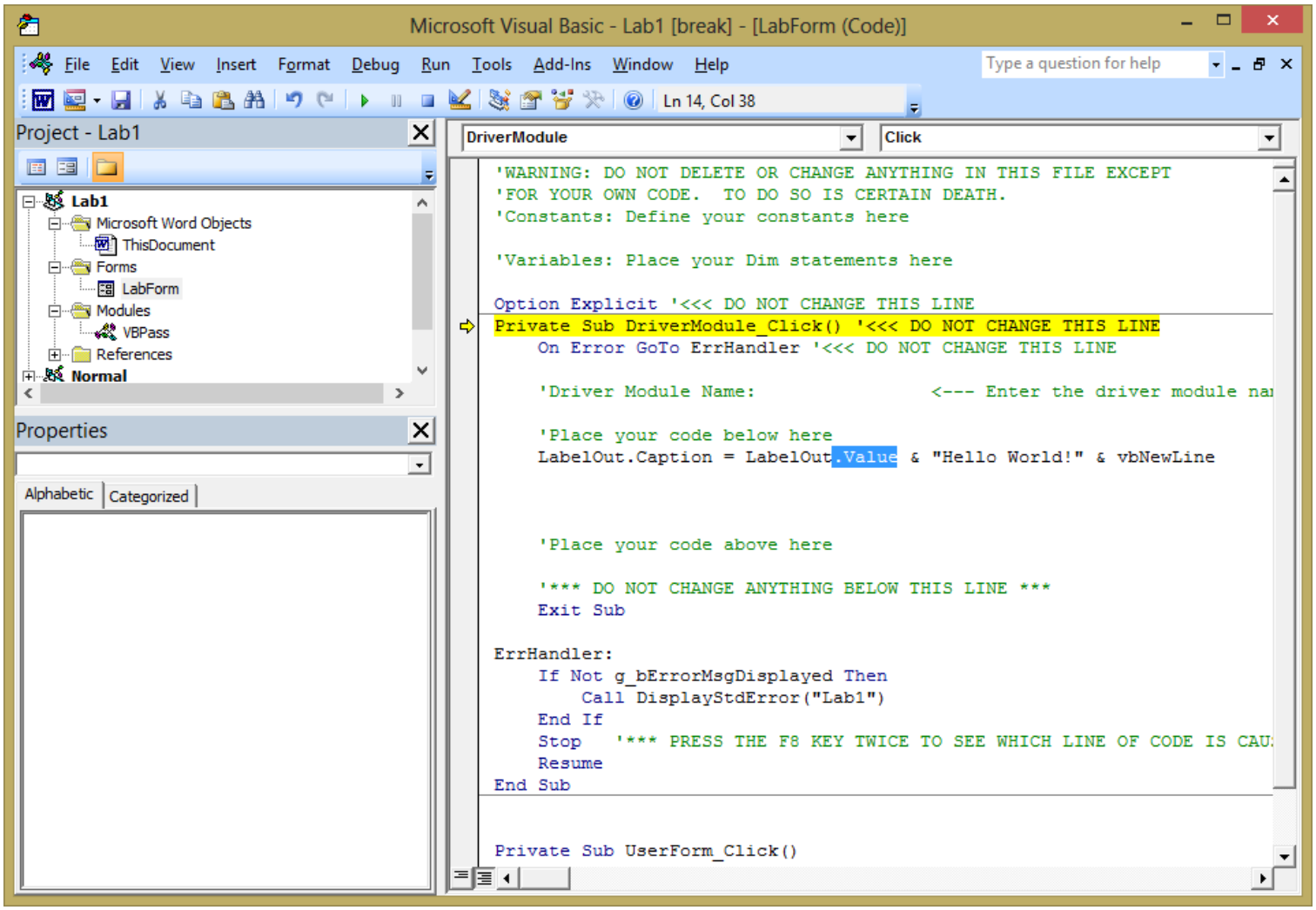
When I press F5 to run this code, the Lab Form dialog pops up:



When I press the Run Lab button, I get the following error message:



This error dialog is compatible with the screen reader, and I hear that I have a compile error, "Method or data member not found." When I click OK, I am returned to the Visual Basic Developer's Window. What I see is that ".Value" has been highlighted:



however, that isn't what the screen reader reads. Instead, it reads the entirety of the following two lines:

```
Private Sub DriverModule_Click() '<<< DO NOT CHANGE THIS LINE
    LabelOut.Caption = LabelOut.Value & "Hello World!" & vbNewLine
```

As you can imagine, this presents additional difficulty for a student who is unable to see the highlighting. However, it is possible for the student to navigate through the code using the arrow keys, toggle breakpoints, and step through the code using keyboard shortcuts. Generally speaking, it is likely that a blind student will need some help from an Assistive Aid to debug their code, and will likely need your help as the instructor as well (the Assistive Aid is unlikely to have the necessary subject matter expertise to help the student debug their code fully). There are three techniques you can use to make this process somewhat easier for the student:

- Provide detailed instructions for using the debugger to debug simple issues, using the appropriate keyboard shortcuts and the screen reader.
- Provide a link to common error messages that your compiler or runtime environment might produce. The more guidance you can give the student on how these error messages relate to the student's code, the more useful the error messages will be for debugging.
- Recommend that the student use an incremental development process where they implement at most a few lines of code, and test that code thoroughly before moving on to the next few lines. This will help the student to more quickly locate the errors when they occur, and will also minimize the amount of context that the student must remember during the debugging process. If at all possible, instructions provided to the student should include frequent testing points where work can be verified before proceeding.

It is worth noting that providing detailed examples of using the debugger and using an incremental development process may also help your sighted students to gain proficiency when debugging their code.

Whenever possible, avoid development environments where:

- Errors are not displayed in textual format (for example, some IDEs display error messages as stop or yield icons next to the code, or using text coloring or highlighting).
- The debugger is incompatible with screen reader software. When the user hits an error or breakpoint, does the screen reader read the current line? Can the student navigate around with the arrow keys to read line-by-line and character-by-character?
- The debugger does not have appropriate shortcut keys. Many debuggers provide buttons to pause, stop, play, step into, step out of, step over, and so on. Are there shortcut keys for all of these buttons? Alternatively, can the student navigate through these buttons using the keyboard, and will the screen reader read the buttons appropriately?

Note that if the debugger is not accessible, then the student may still be able to complete the lab with your help, but will probably not be able to satisfy a learning outcome that specifies that the student will be able to test and debug their code.

3.10. Interface Design

The Lab Rubric specifies that if the assignment requires designing and laying out user interface elements, the student will be able to do so using code.

In some programming classes (for example, CIS 133B and CIS 233B), students are required to design and build a GUI for their programs. In most programming languages and development environments, there are two ways to accomplish this:

1. Many development environments include WYSIWYG interface builders where the programmer can drag and drop user interface elements onto a form, drag handles on those elements to resize them, change properties such as color, label text, and so on, and connect these interface elements to actions that the program will perform when the program is running.
2. Most programming languages include user interface libraries that programs can use to create instances of windows, dialogs, buttons, labels, and other user interface elements. Programs written using these libraries can often explicitly control the size, location, appearance, and in some cases the layout of these elements through code.

Generally speaking, a blind student will be unable to construct a user interface using a WYSIWYG interface building tool. Even for development environments that are mostly accessible, it is often the case that interface building tools will lack keyboard shortcuts and will be incompatible with screen reading software. In addition, students with motor control impairments may also find it difficult to use these tools with sufficient accuracy. Whenever possible, if an existing lab requires the use of a WYSIWYG interface building tool, consider whether any of these other approaches can be used as an alternative:

- Can an alternative version of the lab use code to control user interface libraries to produce the same result as an interface builder?
- Can an alternative version of the interface requirements be developed that would allow for a primarily text-based interface to be developed?
- Can the student be provided with an existing interface layout in some format, and then asked to hook this interface to the rest of the program, instead of having to do the layout themselves?
- Can you (the instructor) or an Assistive Aid work with the student to complete portions of the interface layout that require the interface builder tool?

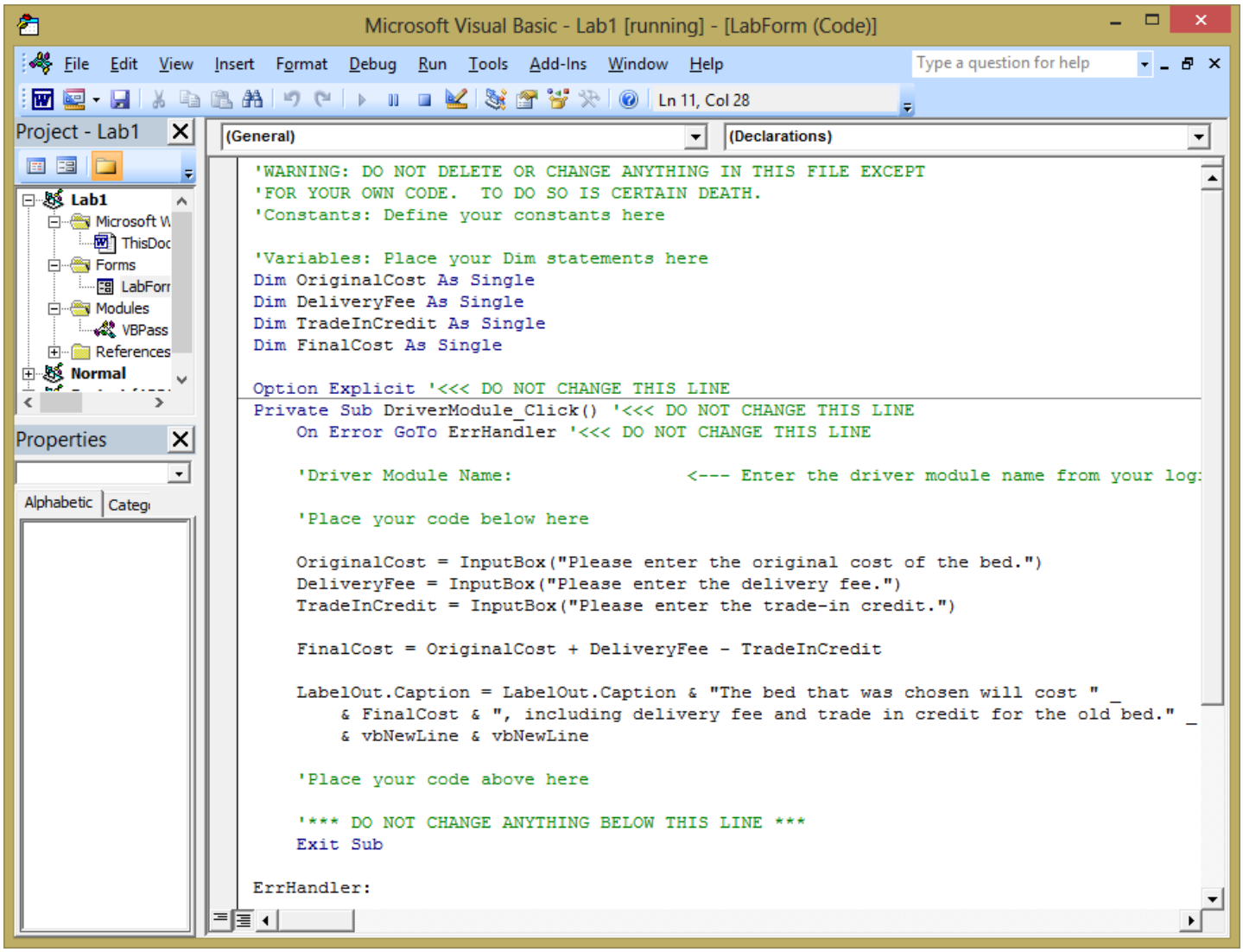
Any of the above alternatives may enable the student to complete the remainder of the lab. However, it is worth noting that accommodations should not require a fundamental change to the learning outcomes for the course. If one or more learning outcomes are centered around interface design, and the course is dedicated to learning a particular programming language or development environment for which no accessible alternative exists, then it may be more appropriate for Disability Services to warn the student that there will be accessibility issues in the class. However, it is the student's right to make the ultimate decision as to whether to sign up for the course.

3.11. Output Requirements

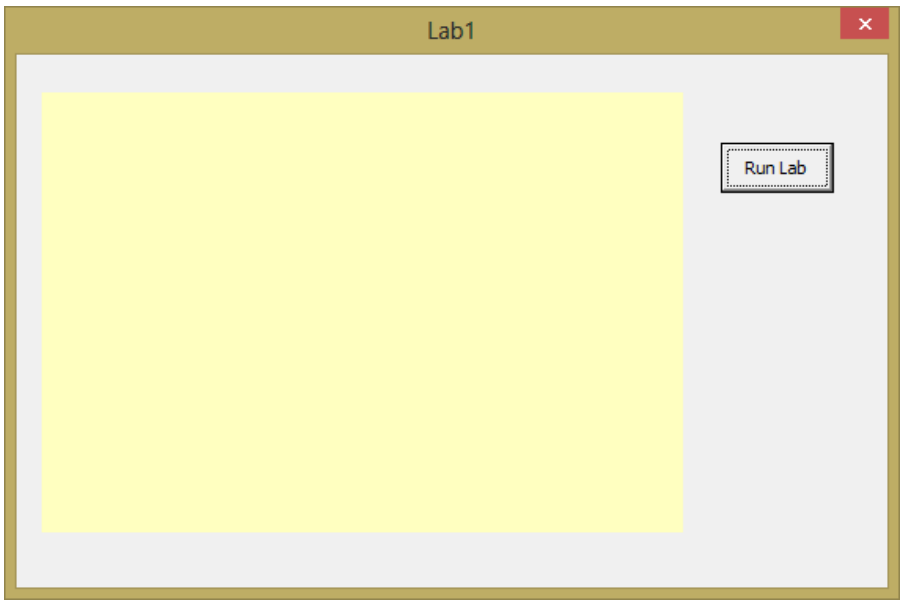
The Lab Rubric specifies that output should be primarily static, textual, and screen-reader accessible.

The student must be able to run their program, input data (if appropriate), and review the results. This requires that the interface to their program be keyboard accessible, and that the output from their program be compatible with a screen reader. In particular, output that displays primarily as graphics or animation will not be compatible with screen reader software, and will be difficult or impossible for the student to review without an Assistive Aid.

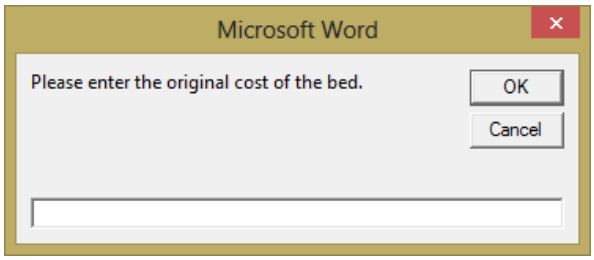
An example of an assignment that meets this standard is Lab 1 (from many of our CIS 122 course shells) in VBA. The student is provided with a lab form and asked to fill out the code for the assignment. Here is a correctly completed example:



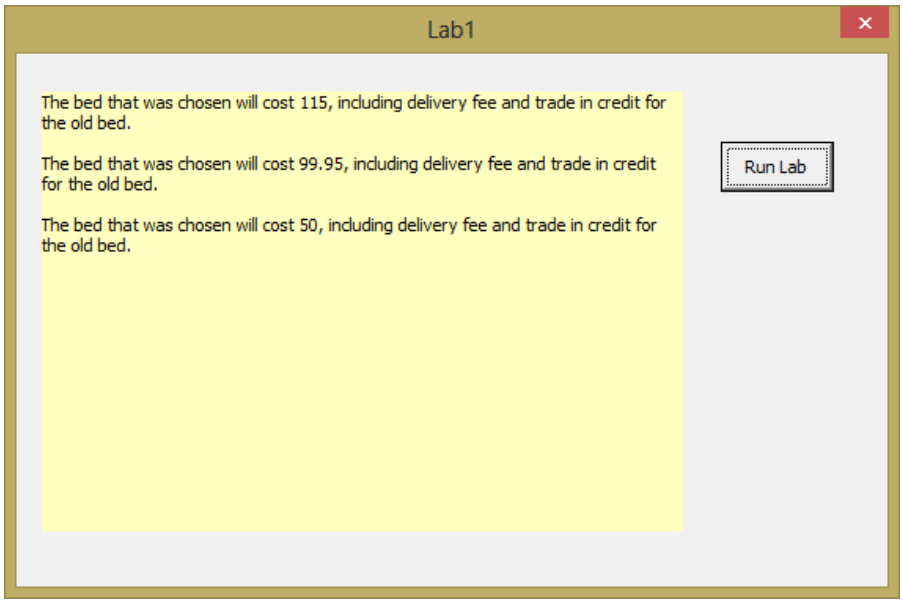
When the program is run, a Lab Form dialog appears with the following interface:



The student is able to navigate to the Run Lab button and press "enter" to click it. The program displays a series of input boxes, such as:



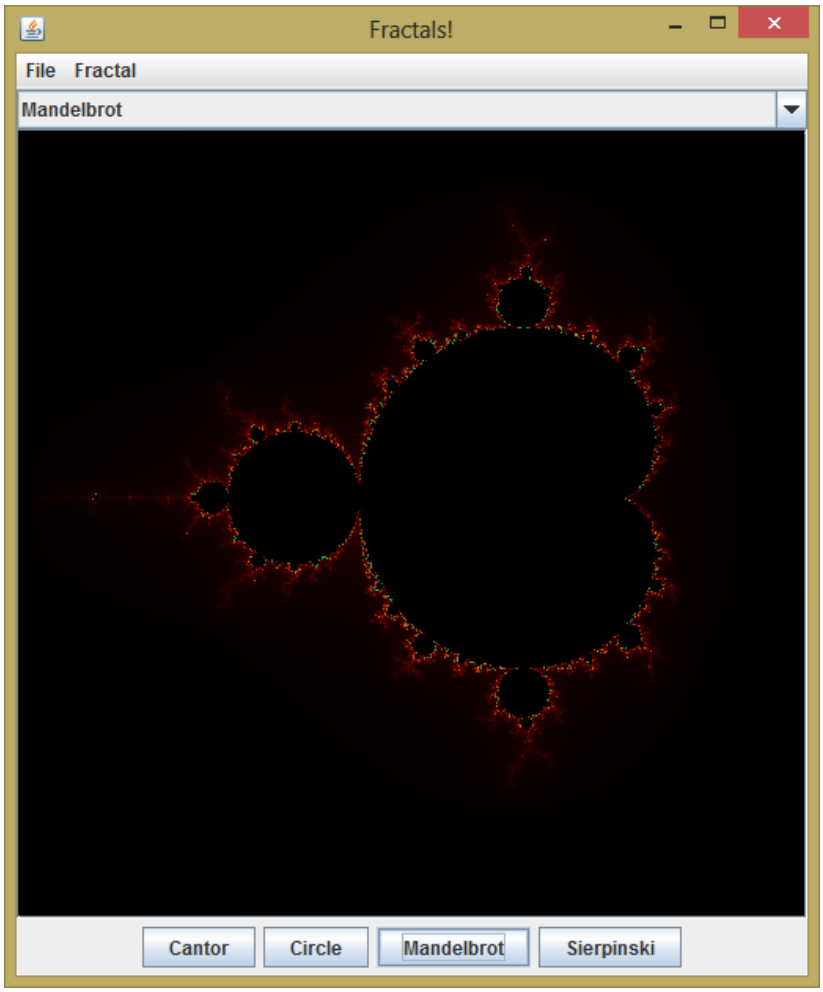
The screen reader reads that there is a Microsoft Word dialog and reads the prompt "Please enter the original cost of the bed." The student is able to type in the requested input and to use the tab key to navigate to the OK button. They can then press "enter" to input the information. After entering the remaining requested input and running the program for three test cases, the lab form dialog contains this output:



The screen reader is able to read the output, and the student is able to verify that the program is working correctly. However, some of our labs have output that isn't primarily textual and will not be accessible to a screen reader program. For example, in CS 133G (Introduction to Computer Games), the first lab asks the student to write a "Catch the Clown" video game. Here, the output from the program will be an image of a room with a clown sprite moving around in a zig-zag fashion. When the student clicks on the clown sprite, their score is increased. A short video of the completed assignment is below:

Your browser does not support the video tag.

A blind student will not be able to complete this assignment and verify that their program works correctly without significant help. In addition to the output being primarily graphical and animated, the student will not be able to use keyboard shortcuts to provide input to the program. Other labs might allow for accessible interaction with the running program, but still have output that's not compatible with a screen reader. For example, Assignment 7 in a CIS 233J (Java Programming II) course has the student complete the following application:



Here, the student can navigate using the Fractal menu, the dropdown menu, or the buttons along the bottom, but will not be able to see that the content pane has changed, and will not be able to verify that the program is running correctly. Also, Java Access Bridge must be enabled to make even the navigable interface elements accessible to a screen reader (this setting only needs to be changed once, but the student may need to be told to make this change). Another example of inaccessible program output is a CeeBot program in which a simulated robot grips and moves a titanium cube. The output from successfully running this program is a graphical animation.

It is important to note that accommodations should not require a fundamental change to the learning outcomes for the course. For example, in CS 133G, which is a class on video games, it is an important learning activity to write a functioning video game. Rather than searching for an equally effective alternative for these activities which may not exist, it may be more appropriate for Disability Services to warn the student that there will be accessibility issues in the class. However, it is the student's right to make the ultimate decision as to whether to sign up for the course. In other cases, where it is possible to provide an accessible alternative that still satisfies learning outcomes, this is something that should be seriously considered.

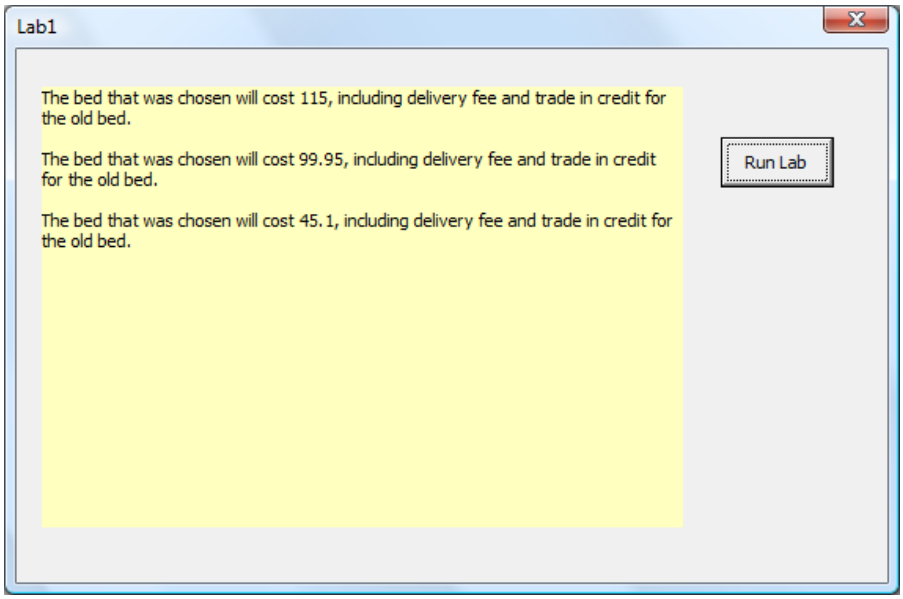
3.12. Providing Sample Output

The Lab Rubric specifies that if sample output is provided, it is given in text format or as a screen capture with descriptive text that fully specifies the output requirements.

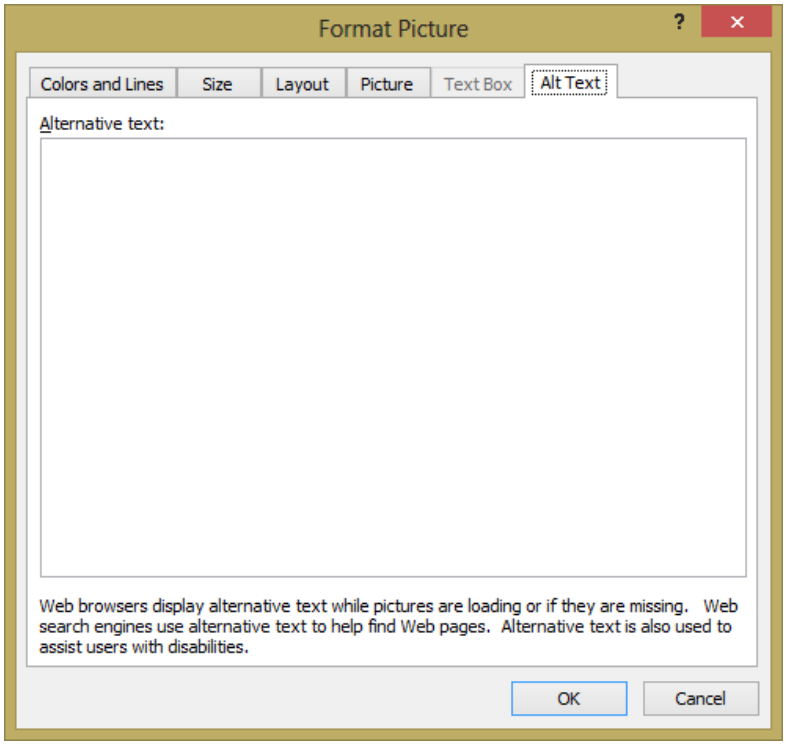
Here is a link to a Lab 1 assignment document in Microsoft Word format that is used in many of our CIS 122 course shells. This assignment document meets accessibility requirements by providing sample output in text format:

> The application should output the following sentence to the user's screen, with a blank line between each output:
>
> *The bed that was chosen will cost _____, including delivery fee and trade in credit for the old bed.*

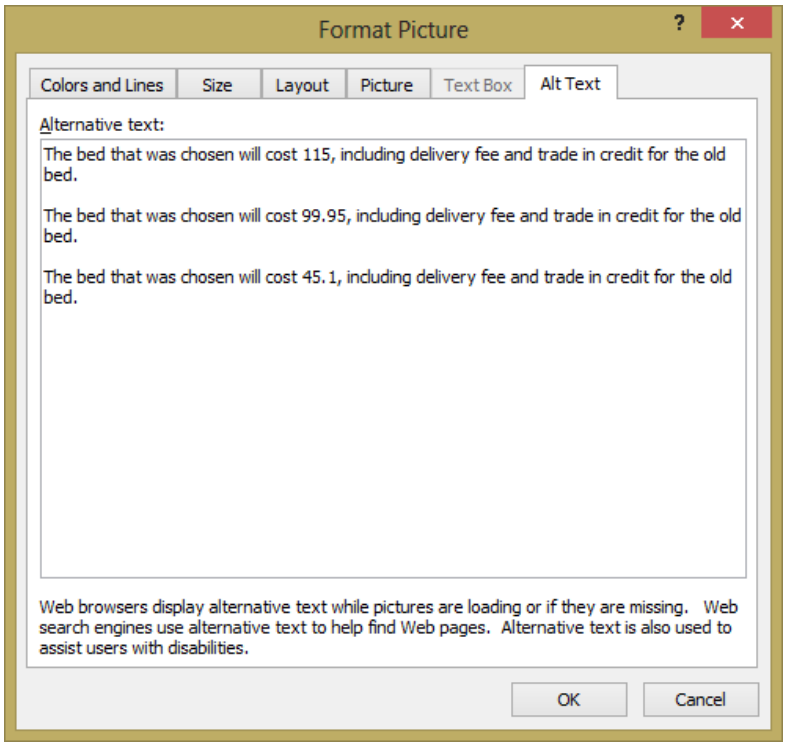
However, the assignment document also includes the following screen capture:



If we right click on this picture and select "Format Picture...", we can see that this image has no Alt Text:



Therefore, the text of the screen capture will be inaccessible to a blind student. We can easily improve this image by typing the text contents of this image into the alt text and saving:



This adds valuable information for a blind student, since the exact output in the screen capture matches the expected output from the Sample Input that is provided in the assignment document:

Sample input:

Original Price	Delivery Fee	Trade In Amount
100.00	20.00	5.00
89.95	10.00	0.00
50.00	0.00	4.90

After adding the alt text, screen readers will be able to read this text in line, as part of the document. For example, here's an audio recording of the NVDA screen reader reading this Word document, starting from a few lines before the screen capture:

Your browser does not support the audio element.

We also note that the accessibility of the table of sample input can be improved by adding header rows, which will make it more compatible with screen reading software. This issue has been addressed in the HTML table above.

Please see the section on Example Code for a closely related standard.

3.13. Providing a List of Commonly Used Shortcuts for Blind Students

The Lab Rubric specifies that frequently used keyboard shortcuts should be provided as a separate document and are suitable for printing in Braille.

A complex development environment might have hundreds of keyboard shortcuts. For example, here is the list of keyboard shortcuts for Microsoft Word, 2010. Visit this page and click on the link labelled "+ Show All" near the top of the page to expand the full list. The student must simultaneously search through the shortcut keys while attempting to follow a list of instructions for completing the lab. For example, here are instructions for completing a lab using VBA in Microsoft Word. Trying to follow all of the instructions in this document while simultaneously looking up shortcut keys in a separate Window can be extremely challenging. In addition to the Word keyboard shortcuts, the student must also refer to Windows shortcuts, Windows Explorer shortcuts, and application-specific shortcuts for the program they are writing. This can result in the student needing to switch between several different development environment windows, application windows, and browser windows with shortcut lists for different applications.

This process can be greatly simplified by creating a summary list of commonly used shortcut keys. Such a document will be easier to search through in online form, and can also be printed as a Braille document and read by the student without needing to switch Windows. This makes it much easier for the student to keep context when following the list of instructions. To illustrate this, here are the keyboard shortcuts needed to complete the VBA lab linked above. These shortcuts have also been organized to specify which applications they apply to, and also organized roughly in the order that the student will need to use them.

3.13.1. Windows Explorer

Shortcut	Action
Alt + Up Arrow	Up one level
Up and Down Arrows	Previous and next selected file
Return	Open selected file or folder
Shift + F10	Right click (brings up context menu)

3.13.2. Microsoft Word

Shortcut	Action
F6	Cycle through task panes (including notifications)
Alt + E	Select the "Enable this content" button in the Microsoft Office Security Options dialog.
Alt + F11	Open the Visual Basic Developer's Window
Alt	Activate Word ribbon toolbar
Left and Right Arrows	Cycle through tools when the Word ribbon toolbar is active
Alt then F then I	Open Word Options dialog
Up and Down Arrows	Previous and next options when the Word Options dialog is open
Alt then L then V	Open the Visual Basic Developer's Window
Ctrl + S	Save the document
Alt + F4	Close the document

3.13.3. Visual Basic Developer's Window

Shortcut	Action
Ctrl + R	Select the Project Explorer pane
Up and Down Arrows	Navigate through project folders when the Project Explorer pane is selected
Right Arrow	Open the currently selected folder when the Project Explorer pane is selected
F7	Open the Code Window for the currently selected document in the Project Explorer pane
F5	Launch the Lab Form Dialog for the code in the currently selected Code Window
Alt + F4	Close the Visual Basic Developer's Window

3.13.4. Lab Form Dialog

Shortcut	Action
Run Lab button	Run the lab form code
JAWS cursor	Read output from program (JAWS)
NVDA + /	Read output from the program (NVDA)
Alt + F4	Close the lab form dialog
Alt + Print Screen	Copy the topmost window to the clipboard

3.14. Providing Instructions for Blind Students

The Lab Rubric specifies that if a series of actions must be performed by the student, these instructions should be provided as a numbered list, and that it is also helpful to provide the keyboard shortcuts in these instructions.

Here is an example of a list of instructions for guiding a severely visually impaired or blind student through the process of creating a simple "Hello World!" program in VBA, using Microsoft Word. These instructions assume that the student is familiar with either JAWS or NVDA, and knows how to log into Desire2Learn. Ideally, this material would be covered in earlier course work, or with the help of an Assistive Aid. In order to make the instructions as useful as possible, please note that:

- The instructions are broken down into task-oriented subsections, with numbered lists of instructions for each task.
- Whenever possible, keyboard shortcuts are provided in the instructions, and information is provided on auditory cues that the student can use to complete the tasks.
- It is also highly recommended that the student be provided with a short list of commonly used keyboard shortcuts as a separate document that can be printed out in Braille. This will reduce the number of window changes the student will need to make in order to complete the lab.
- It is important to note that keyboard shortcuts change periodically as software is updated. It's a good idea to state which version of the development tools have been tested with these instructions.
- Try to avoid mentioning specific keyboard shortcuts for the screen reader program itself. This is because students who are using one screen reader will have trouble if you give instructions that are specific to a different screen reader. When it is absolutely necessary to give screen-reader specific instructions, try to provide a couple of alternatives for different screen readers, or to point out that the exact shortcuts may be different for different screen readers.

Here are the instructions that would be provided to the student:

3.14.1. Instructions for Completing Lab 1

Please follow these instructions to write the program for Lab 1. Note: these instructions have been tested with Microsoft Word 2007 using Windows 8 and should also work with Microsoft Word 2010 using Windows XP, Vista, Windows 7, or Windows 8. There may be some differences in keyboard shortcuts for other versions of Word or other operating systems (such as Mac OSX). If you are having trouble with these instructions, please contact the instructor or your Assistive Aid for help.

3.14.1.1. Download the Lab

1. Go to the Content tab in Desire2Learn
2. In the table of contents, there is an item for "Lesson 2: Sequences". Open this item.
3. Under "Lesson 2: Sequences" there is an item for Lab 1.
4. Under the content for Lab 1, there is a link for "lab 1 (download and unzip)". Select this link.
5. There is a Download button on this page. Click the button to download the Lab1.zip file.

3.14.1.2. Unzip the Lab Download

1. Lab1.zip should be stored in your Downloads folder or on your desktop. Locate this document in Windows Explorer.
2. Right-click on this document using Shift + F10. A context menu will pop up.
3. Use the up and down arrow keys to locate "Extract All..." or press the T shortcut.
4. Press return. A dialog will pop up titled "Extract Compressed (Zipped) Folders". There is an "Extract" button on this dialog. Use TAB to navigate to this button.
5. Press return to select the "Extract" button.
6. A new Windows Explorer folder will open with a Lab1 folder inside of it.

3.14.1.3. Open the Lab Document

1. Inside the Lab1 folder should be a file named Lab1.doc. If there is only a folder named Lab1 inside of your Lab1 folder, then open the Lab1 subfolder and Lab1.doc should be inside there.
2. Open Lab1.doc. It should open in Microsoft Word.
3. There may be a security warning that says "toolbar, property page, Protected View This file originated from an Internet location and might be unsafe. Click for more details.", along with a button labeled "Enable Editing" If you press F6 a few times, you might hear this button. If so, press enter to select it.
4. There may be a security warning that says "toolbar, property page, Security Warning Macros have been disabled", along with a button labeled "Options..." If you press F6 a few times, you might hear this button. If so, press enter to select it.
5. If you selected "Options..." the Microsoft Office Security Options dialog will open. There is a radio button labeled "Enable this content". Press Alt+E to select this button and press enter for OK. The dialog should go away.

3.14.1.4. Bring up the Visual Basic Developer's Window using a Keyboard Shortcut

1. Some versions of Microsoft Word use Alt + F11 to bring up the Visual Basic Developer's Window. Press Alt + F11 now. If a window opens titled "Microsoft Visual Basic - Lab1 - [LabForm (Code)]" then proceed to the section titled "Open the LabForm Code Pane" below.
2. If Alt + F11 doesn't work, then we'll need to use the Developer Tab in the Ribbon. Proceed to the next section, "Enable the Developer Tab in the Ribbon."

3.14.1.5. Enable the Developer Tab in the Ribbon

1. Press Alt to activate the Word ribbon toolbar. Use the left and right arrows to cycle through the tools on the ribbon (Home, Insert, Page Layout, References, etc.). If there is a Developer tab, proceed to the next section "Bring up the Visual Basic Developer's Window." If the Developer tab is not in the ribbon, we'll need to activate it using the following steps.
2. To activate the Developer tab, we need to open the Word Options dialog. This is under the file menu, and can be accessed by pressing Alt, followed by F, followed by I.
3. The Word Options dialog has a list of selections. One of these selections is Customize. Select Customize using the down arrow.

4. You should be on the Customize the Quick Access Toolbar and keyboard shortcuts pane. There are two lists on this pane. You want the one that is labeled "Customize the ribbon."
5. Under "Customize the ribbon" you should find Developer with a checkbox next to it. Check the checkbox (space should work if the checkbox is selected).
6. Press enter for OK. The Word Options dialog should go away.

3.14.1.6. Bring up the Visual Basic Developer's Window using the Developer Tool

1. Press Alt followed by L followed by V to bring up the Visual Basic developer's window.
2. A window titled "The Microsoft Visual Basic - Lab1 - [VBPass (Code)]" should pop up.

3.14.1.7. Open the LabForm Code Pane

1. Press Ctrl + R to select the Project - Lab1 pane.
2. Use the up and down arrows to find Forms in the list of folders.
3. Use the right arrow to open the Forms folder.
4. Use the right arrow again to select LabForm.
5. Press F7 to open the code window.

3.14.1.8. Write the Code

1. The Lab Form code includes some standard code that you will be adding your new code to. Use the arrow keys to Find the line that says "'Place your code below here".
2. On the line after this line, type this text. The capitalization and punctuation are very important here, so you'll want to have this text read character by character:

```
LabelOut.Caption = LabelOut.Caption & "Hello World!" & vbNewLine
```

3.14.1.9. Run the Code

1. Press F5 to run the program.
2. If there are no errors, a dialog will pop up with the title Lab1. If you do get errors, you'll probably get an error dialog. Exactly what the error will be is hard to predict, but it's most likely a spelling or punctuation error in the line you just typed, or that the line is not below the line that says "'Place your code below here". Check this line carefully and make sure it is in the right location. Visual Basic might also place your cursor next to where it believes the error has occurred, which might be helpful. If you are unable to find this problem on your own, you may need help from your instructor or from an Assistive Aid.
3. Assuming the Lab1 dialog did pop up, there should be a button named "Run Lab". Press this button. Once again, the program may run correctly or you might get an Error dialog.
4. If the program worked correctly, the dialog should now be displaying the text "Hello World!" in a text label to the left of the "Run Lab" button. Unfortunately, there is no keyboard shortcut for selecting the text label. Use the JAWS cursor or NVDA + / to cause the screen reader to read this text and verify it is correct.
5. If the above worked, congratulations! You have written your first program! Use Alt + F4 to close the Lab1 dialog and return to the Visual Basic Developer's Window. Use Alt + F4 again to return to Microsoft Word.

3.15. Case Studies

We started by looking at the most common development environments that instructors used to write programs (Visual Studio, BlueJ, Eclipse, CeeBot, gcc, gdb, and XCode). The important thing to check was to see if the editor was accessible by a screen reader like JAWS, and if all the commands required to compile and run a program were available using short cut keyboard commands. Most of the students write programs in C++, Java, or Javascript. However, irrespective of the language that was used, the editor and the short cut keyboard commands were the critical components.

Watch this space for additional case studies in the near future.

3.15.1. C/C++ in Visual Studio

We first tested the Visual Studio environment using Angel Chesimet, a blind student from PSU. We found that with a little bit of help, she was able to create a new project, create a file and open it in the editor. The editor was readable by the JAWS screen reader and there were short cut commands for most commands. She was able to compile and run a basic program in about an hour. She concluded that with practice the product was accessible and that she would be able to write programs using Visual Studio.

3.15.2. C/C++ in XCode

XCode is very accessible with VoiceOver as long as the user is doing text based programming. Since all our students use XCode to test text based code, the mac should be very accessible when used with XCode, and VoiceOver. Since VoiceOver comes with the mac, there is no extra cost for the student. The interface builder for iOS apps is completely inaccessible with VoiceOver. As a result it’s necessary to build the interface for any apps manually through code. Although this isn't very difficult, it is just time consuming! From the blogs and forums postings, users are generally pleased with XCode and VoiceOver.

3.15.3. CeeBot

Ceebot is a visual robotics program used in CS160 to teach basic programming concepts. Since it is visual and requires the use of a mouse, we concluded that there was no way it could be used by a blind student. An alternative programming language like Python would be an equally effective way to cover the outcomes of CS 160. Angel was once again very kind and patient to test some Python Interpreters for us which were unfortunately not readable by the JAWS screen reader. There is also another problem with python. The code blocks represented by indentation are a major stumbling block for the visually impaired. Research shows that a Braille display is the best solution for writing python programs. On the other hand, we could use something like visual basic for teaching basic programming concepts. The code can be written in visual studio, which is accessible, by keyboard short cut commands, and the language is easy for beginners to understand.

The theoretical part of this class is covered using materials from Virginia Tech's Computer Science department. This materials uses images and Java applets in every module that is not accessible by visually impaired students. A reasonable alternative to this web material is to use the book, "Computer Science, An Overview" by Brookshear. The materials are almost parallel, but the quizzes need to be modified to reflect the contents of the book, as opposed to the web site.

4. General Recommendations

In addition to accessibility standards discussed in the programming lab accessibility rubric, these recommendations will make it easier for Disability Services to provide an accommodation if a blind student signs up for your class.

1. Try to save all source files to one folder on D2L. This may include .doc, .pdf, .txt or any text based files (including code files which are text based), any videos, images, etc. This will make it easier for Disability Services to convert any of these files to an accessible format when needed.
2. If there are any images that can be converted to tactile images, and the student will benefit from that, this should be discussed with Disability Services in advance.
3. Instructors should prepare a syllabus early to allow students the option of beginning to read materials before the course begins and to allow adequate time to arrange for alternate formats, such as books in audio format or in Braille. Having at least a week's time will give Disability Services enough time to convert any documents and be prepared for the student.
4. Instructors should try to keep their content clean and uncluttered, avoid unnecessary jargon and complexity, and try to keep as much as possible in text-based format.
5. Narrated lectures and other media files should be introduced with a short text statement specifying what content they contain; for example, audio only, video and slides, or slides only. This description helps students know they have accessed all of the lecture components.
6. Instructors should consider extended deadlines for quizzes or programming assignments.
7. New textbook selections should consider accessible online content a top priority. Karen Sorensen has provided some details on the (in)accessibility of publisher content.

4.1. The (In)Accessibility of Publisher Content

Accessibility is required for all online course content, including materials and applications that are linked to on other websites, including publisher websites. ADA standards require that for all inaccessible content, instructors must also provide accessible content that is equally effective, equally integrated and that has substantially equivalent ease of use. (This should either be in place or be ready to put in place immediately).

While Disability Services can make sure a publisher's textbook is accessible for a student, they cannot ensure that the publisher's online materials are accessible.

1. Here are some initial questions to ask your publisher representative about online materials:
   A. Are the videos captioned?
   B. Can a student complete all assignments with a keyboard only (without using a mouse)?
   C. Do all of the images have alternative text descriptions? (Both online and in any materials supplied to you, such as PowerPoints)
   D. Do they use Flash? (Flash is usually inaccessible.)
2. Contact Karen Sorensen, Accessibility Advocate for Online Courses to help verify the accessibility of the publisher's platform.
3. If your publisher's product VPAT is not in the collection of VPATs on Spaces yet, Karen will make a request for it.
4. Questions regarding the VPAT will be sent to the publishing representative and if known, the publisher's accessibility manager.
5. When answers have been received, end user testing will be scheduled if requested by the instructor.

More information is available on the accessiblity of publishers used in CAS, CIS and CS.

4.2. Publishers used in CIS, CS and CAS

According to the survey of CIS, CS and CAS faculty, the following publishers are being used:

- Cengage
  - Cengage/Southwestern
  - Course Technology - part of Cengage
  - Thomson - part of Cengage
  - Publisher of the Malik C++ text book
- Labyrinth
- Lynda.com

- McGraw Hill
  - Glencoe - part of McGraw Hill
  - Gregg - part of McGraw Hill
- O'Reilly
- Prentice Hall / Pearson Education
  - Addison-Wesley - part of Pearson
- Paradigm
- Wiley
  - Sybex - part of Wiley

4.2.1. Cengage

All Cengage products come bundled with an Ebook and textbook. The MindTap ebook reader is screen reader accessible, but not keyboard-only accessible. We are hoping to see this change soon, because overall we were impressed with the MindTap ebook reader.

We have done some end-user testing with these Cengage products: CengageNow , SAM2010 and MindTap.

Cengage makes these claims:

- **Closed–Captioning**: "Audio–visual products © 2010 and later are closed–captioned with accompanying transcripts. If the product you are using is not closed–captioned, submit a request form for either permission to caption the material or for copies of any available transcripts."
- **Alt Formats**: "If you require an alternate version of homework or testing material that is on some of our less accessible platforms, we can coordinate with the product team to get a Word doc created of those items. You can contact my team at permissionrequest@cengage.com. We will work with the product team and Michele to get your students what they need to be successful. Turnaround varies (5-10 days to 4-6 weeks). If a lot of transcripts or if it's a big mapping project (online product homework questions to a word version of a testbank), the product team will deliver them to you in chunks. What they will need to know from you is when the professor plans to use the material in the class. Or if you could supply a copy of the professors syllabus that would help."

This information was supplied by:

Karen Lee
Sr. Mgr, IP Granting
500 Terry Francois Boulevard, Second Floor, San Francisco, CA 94158
(o) 415.839.2451| (e) karen.lee@cengage.com | www.cengage.com/permissions

4.2.2. Labyrinth

Labyrinth is an independent publisher with their own digital materials. We are in the process of contacting them for a VPAT.

4.2.3. Lynda.com

Videos are mostly captioned, but their website and the educational materials are not fully keyboard and screen reader accessible, at the present time. Here's Lynda.com's reply to an inquiry about this:

> "We are currently working towards being a fully keyboard-navigatable experience and have tabbed navigation for most areas of our site. We are aiming for a re-audit of our entire site for 508 compliance, including tabbed navigation by the end of June. This is an ongoing area of focus and something that we have in our designs for all new navigation as we continuously update the site. I hope this helps answer your question. Please let me know if I can be of any further assistance."

4.2.4. McGraw Hill

McGraw-Hill uses the Connect platform for most of their online courses. Currently Connect is not accessible. Here is their roadmap to accessibility (some deadlines which they've missed already.)

Aleks, McGraw Hill's math and science platform is fairly accessible. They don't offer a solution for graphical problems though. We've asked them to provide us with the images and their problems, so we could create tactile graphics for those problems. They have not responded to this request. Here's our more in-depth analysis of ALEKS.

4.2.5. O'Reilly:

The only digital materials they have are ebooks. If books are available as an Adobe Digital Edition, then they should be quite accessible. If not, we will need to do further testing.

4.2.6. Paradigm:

Paradigm is an independent publisher. The only digital materials they have are some ebooks which they distribute through EBSCO and ebrary.

The Ebrary app is overall pretty accessible, if the book is available for download as an Adobe Digital Edition. Student needs to create an account and set their preferences to accessibility mode

4.2.7. Pearson/Prentice Hall/ Addison-Wesley

Karen has multiple questions in to Pearson about their VPATs. Pearson claims they are very accessible now that they have HTML5 ebooks, but we have yet to find their claims are justifiable. According to Elaine Ober, the Pearson Associate Director of Accessibility and Compliance, you can find out exactly which titles are accessible for MyMathLab, MyStatLab, MyReadingLab and MyWritingLab, ticking the box for "Accessible ebook" under the Course Features section. She also sent us this list of accessible titles in Social Science, Business, Health Science , Spanish and Criminal Justice.

For titles that don't appear in these lists, please contact Karen Sorensen to verify accessibility.

4.2.8. Wiley

Wiley still has a ways to go before their online products are accessible. Here's a VPAT for WileyPlus. If you use a different Wiley product, please let Karen Sorensen know and she will request a VPAT.

5. Glossary

This glossary contains definitions for "jargon" terms that are used throughout this document.

- **Assistive Aid** - Job title for a person who helps a student with impairments. Often, this person will be a student who is doing this work on a part-time basis and may lack subject matter expertise.
- **CAD** - Computer Aided Design. Generally refers to a type of program in which 3D models are created (such as for use with 3D printing).
- **CCOG** - Course Content and Outcomes Guide. A statement of the course's description, learning outcomes, content, etc. A full set of CCOGs for courses at PCC is available online.
- **Development Environment** - The set of tools that are used to build a computer program. Often, these include the ability to edit text, design user interfaces, compile or build the application, and debug the program. When these development tools are bundled (integrated) into a single application, that application is often referred to as an IDE.
- **GUI** - Graphical User Interface.
- **IDE** - An Integrated Development Environment where a complete set of development tools is bundled (integrated) into a single application.
- **Learning Outcome** - What a student is expected to be able to do on completion of a course or activity. Course-level learning outcomes are stated in the course's CCOG.
- **UML** - Unified Modeling Language. A set of specifications for modeling various aspects of application and data processes and architectures.
- **VBA** - Visual Basic for Applications. A programming language embedded in many Microsoft Office applications, including Microsoft Word.
- **VPAT** - Voluntary Product Accessibility Template. A form that publishers fill out that describes the accessibility characteristics of their products.
- **WYSIWYG** - "What You See Is What You Get." Refers to editing applications that display the current document in the format it will ultimately appear. For example, a form layout tool which is used to design user interfaces may allow the user to construct a GUI in the same format it will appear when the application is run.

6. References

1. AccessComputing. 2009. Equal Access: Universal Design of Computing Departments. University of Washington, Seattle, WA. Available online at http://www.washington.edu/accesscomputing/equal_access_csd.html.
2. AHEAD. n.d. Universal design. Available online at http://www.ahead.org/resources/universal-design.
3. Universal Design: Process, Principles, and Applications. Available online at http://www.washington.edu/doit/Brochures/Programs/ud.html
4. Universal Design: Implications for Computing Education. Available online at http://staff.washington.edu/sherylb/ud_computing.html.
5. http://www.catea.gatech.edu/scitrain/kb/FullText_Articles/CompSc4visuallydisabled.pdf
6. "Designing Online Courses for Screen Reader Users", by Lorna R. Kearns, Barbara A. Frey, and Gabriel McMorland in Journal of Asynchronous Learning Networks, Volume 17: Issue 3. Available online at http://anitacrawley.net/Resources/DESIGNING%20ONLINE%20COURSES%20FOR%20SCREEN%20READER%20USERS.pdf.
7. AFB AccessWorld Magazine. Not What the Doctor Ordered: A Review of Apple's VoiceOver Screen Reader. Available online at http://www.afb.org/afbpress/pub.asp?DocID=aw060505.
