COMP241
Software Engineering Development
Lecture 17: Usability and Tools 3 (Javadoc)

Mark Hall

- Usability
  - Impact
  - Poor usability in interative software
  - Software usability guidelines
- Javadoc
  - Main description
  - HTML Tags
  - Summary sentence
  - Tags
  - Using with ANT

THE UNIVERSITY OF WAIKATO
DEPARTMENT OF COMPUTER SCIENCE
TARI ROROHIKO

Why Consider Usability?

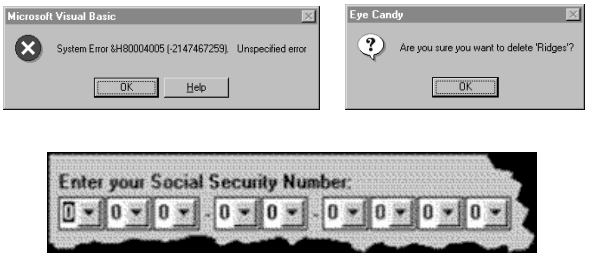
Did you ever …

- try to push/pull a door that opened the other way?
- turn on the wrong light switch in a room?
- use the windscreen wipers instead of the indicators in a car?
- have difficulty setting a microwave correctly?
- forget how to do something in a software application?

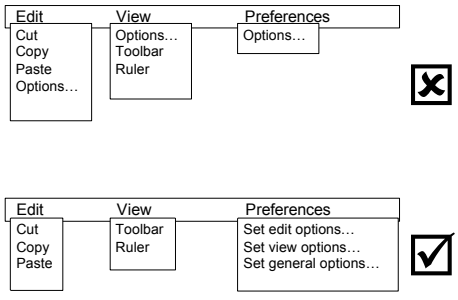
The Impact of Software Usability

- Usability issues pervade our everyday lives from the life-threatening to the mundane.
- You will be responsible for the experience of people using the software that you design and develop.
- Small usability issues for one person (you the designer) can scale up to enormous problems for users.

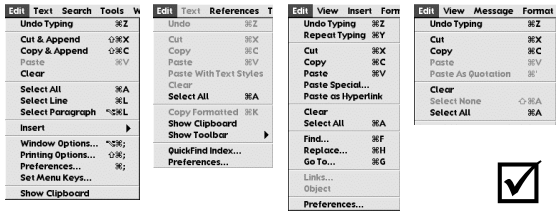
Poor Usability in Interactive Software



Do not Duplicate Menu Items



Arrange Menus in a Standard Way



- Edit menus from different applications
- Note the similarity for standard tasks

When Duplication is Acceptable

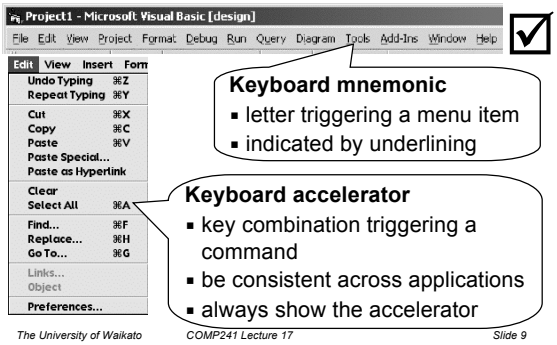
Duplication can occur ***between*** access mechanisms, *e.g.,*

- a menu item
- a toolbar icon
- a keyboard shortcut
- a right-button popup menu

Use Keyboard Equivalents

- GUIs are often centred around pointing, clicking and dragging.
- BUT many software applications also require text or data entry using a keyboard.
- Unfortunately people do not have three hands.

Use Keyboard Equivalents



Keyboard mnemonic
- letter triggering a menu item
- indicated by underlining

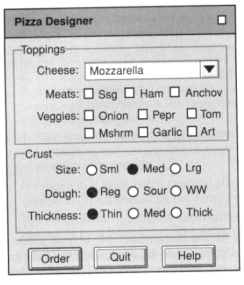
Keyboard accelerator
- key combination triggering a command
- be consistent across applications
- always show the accelerator

Two Types of Windows

Primary windows
- generally visible all of the time
- functionality is accessed through menu items or toolbars

Dialog windows
- appear when necessary
- often contain functionality PLUS
- buttons to apply actions, say OK, cancel *etc.*

Unconventional Application Windows

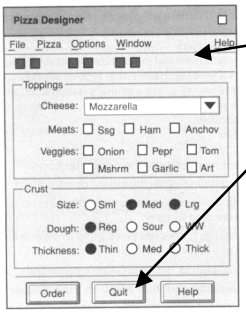


- Main window looks like dialog box
- Unexpected button behaviour

Unconventional Application Windows

- Options should be in a menu bar and as toolbar buttons

Avoid Hybrid Windows



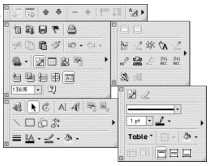
- Menu bar and toolbar make window look like a primary window.
- Control buttons make it look like a dialog.

Primary Window vs. Dialog Box

Feature	Primary Window	Dialog Box
Display duration	usually long	usually short
Modal	no	if needed
Menubar	yes	no
Toolbar	yes	no
Help provided	rightmost menubar item	help button if needed
Resizable	yes	usually not
Iconifiable	yes	no
Closing the window	menu item: Exit or Close	button: OK, Close, Cancel

Toolbars

- Toolbars can provide quick access to commonly used functions.



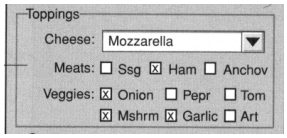
However …

- Do NOT have commands that are only on a toolbar.
- Do NOT put all commands on toolbars.

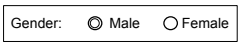
Choice Controls

Two common situations

1. Zero or more from *N* selection
   ⇒ **Checkboxes**



2. One from *N* selection
   ⇒ **Radio buttons**



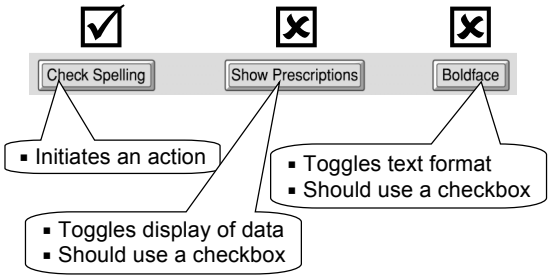
Choosing Choice Controls

Don't use checkboxes instead of radio buttons!

- Use radio buttons when
  - one from *N* selection required
  - number of options is between 2 and 8
  - there is enough space
  - radio buttons should have an initial value
- Use checkboxes when
  - zero or more from *N* options required
  - each setting is an ON/OFF value
  - each option is independent

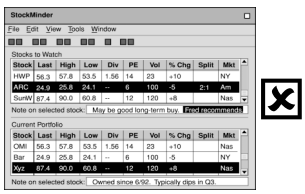
Misusing Choice Controls

Do not use command buttons as toggles.



Feedback

- Tell the user when the software is busy.
- Make selections consistent in different views.



Javadoc

- "Javadoc is a great tool, and should be used with feelings of unbridled joy."
  - —`www.javapractices.com`
- A program that automatically extracts documentation from your Java sources
  - Can be viewed with a web browser

Why Document Code?

- To allow understanding of what services are offered ("What is this supposed to do?"), without having to look at its implementation ("How does it do it?")
- To provide additional information that is not obvious like boundary conditions, parameter ranges, etc.

What is Extracted?

- A description of packages, classes, methods, and data members

```
/**
* Main description: starts with summary sentence.
* More general information about the
* program, class, method or variable which
* follows the comment, using as many lines
* as necessary.
*
* Tag section:
* zero or more tags to specify more specific
* kinds of information, such as parameters
* and return values for a method
*/
```


Main Description

- Begins after the starting delimiter "/**", and continues until the tag section
- Written in HTML
- Leading "*" characters on each line (and blanks and tags preceding it) are discarded
- As of Java 1.4, if you omit the leading asterisk on a line, the leading white space is preserved

Including HTML Tags

`<P>…</P>`	Paragraph
` `	Line break

`<UL>`
`<LI>…</LI>`
`</UL>`
	List

`<I>…</I>`	*Italics*
`<B>…</B>`	**Boldface**
`<CODE>…</CODE>`	`Fixed width`

(Should not use <H1>, <H2>, etc)

Summary Sentence

- The first sentence of the main description should be an informative sentence or phrase that can stand on its own
- Concise but complete description of the declared entity
- The sentence ends at the first period that is followed by a blank, tab or line terminator; so be careful with abbreviations such as “e.g.”

Tags

- Block (or stand-alone) tags:
  `@tag`
  Must appear in the tag section at the beginning of a line (ignoring leading asterisks, white space, and tabs)
- In-line tags
  `{@tag}`
  Allowed and interpreted anywhere

Some Useful Tags

- Documenting method parameters
  `@param` <*name*> <*description*>
- Documenting method return values
  `@return` <*description*>
- Documenting exceptions
  `@exception` <*type*> <*description*>
- Adding cross references
  `@see` <*reference*>
  `{@link` <*link-target*> <*link-text*>`}`
- Including author’s names
  `@author` <*name*>

Example: Commenting a Class

```
/**
 * <code>ImageOrganizer</code> is the main interface to the image organization
 * library. It is used to obtain and manipulate <code>FileElement</code>s that
 * represent disk-based  folders and images. <code>ImageOrganizer</code> is
 * an abstract class that declares various abstract methods that
 * a concrete implementation should provide. You should subclass
 * <code>ImageOrganizer</code> and provide your own implementation for the
 * abstract methods. <strong>IMPORTANT: There is one method that you need to
 * provide an implementation for in this class. The static factory method
 * create() should return an object which is your concrete subclass of
 * <code>ImageOrganizer</code>. This is the only case where you will have to
 * directly change code in the <code>imageorg</code> package.</strong><p>
 *
 * <code>ImageOrganizer</code> provides a single <code>protected</code> constant
 * <code>REPOSITORY_HOME</code> defining the on disk root (or home) directory
 * that will contain all images and folders maintained by the
 * <code>ImageOrganizer</code> library. The <code>getHome()</code> method will
 * return a concrete implementation of the <code>Folder</code> interface that
 * encapsulates the home directory.<p>
 *
 * Users of the ImageOrganizer library will maniplulate images and folders using
 * methods defined in the {@link FileElement}, {@link Folder} and
 * {@link Image} interfaces.
 *
 * @author Mark Hall
 * @version 1.0
 */
```


Example: Commenting a Method

```
/**
 * Renames this <code>FileElement</code> to the the
 * supplied <code>newName</code>. Note that this method
 * should just change the name (as returned by <code>getName()</code>
 * of the <code>FileElement</code>. E.g. if this file element
 * represents the disk file /home/mhall/image_org_home/sunset.jpg, then
 * the call <code>rename("beach.jpg")</code> would result in
 * <code>getName()</code> returning the <code>String</code> "beach.jpg"
 * and the disk file becoming /home/mhall/image_org_home/beach.jpg.
 *
 * @param newName the new name of this <code>FileElement</code>.
 * <br>(Precondition: newName != null)
 * @return true if the rename operation was successful.
 * @exception Exception if something goes wrong during renaming
 */
boolean rename(String newName) throws Exception;
```


Using Javadoc in Ant

```
<javadoc source="1.5"
 destdir="${api}"
 private="true"
 author="true">
  <packageset dir="${src}"/>
  <link href=http://. . . />
  <classpath refid="classpath"/>
</javadoc>
```