Bloopers: Definition
► Mistakes that system developers frequently make when designing graphical user interfaces
  • Not just specific examples but mistakes that developers make over and over
  • Johnson proposed simple basic principles (BP) that can avoid those bloopers
► The book is v.2 (many things had changed since the original one)

GUI Control Bloopers
► Two categories of control bloopers
  • Using the wrong GUI Control
  • Using a control incorrectly
► Control bloopers harm usability and give customers an impression of a shoddy, unprofessional product

Blooper 1: Confusing checkboxes and radio buttons
► Radio buttons when only one is selectable
► Checkboxes when many selectable

Blooper 2: Checkboxes for non-ON/OFF Setting
► Checkboxes should be used for on/off not for a selection of items.

Blooper 3: Command Buttons as Toggles
► Saves space on the screen but toggling meaning of a button can be missed by the user
► Misleads users; can’t predict by looking at them how they’ll behave, have to try them
  • “Mystery Meat Navigation”
► Use two buttons and disable the inactive one, or use a toggle switch style control
Blooper 4: Using tabs as radio buttons
- Misuse of tabs is to use them as if they are radio buttons to present choices that affect what the application will do rather than just which controls are displayed.
- Some users will not realize the last tab selected is the one that is used – users expect tabs just for switching between panels.

Blooper 5: Too Many Tabs
- Intended to save space but too many uses more space – usually doesn’t scale beyond a handful.
- Never use dancing tabs; change position based upon which tab is selected.
- Unavoidable with multi-rows of tabs.

Blooper 6: Using input controls for display-only data
- Don’t use input controls (textboxes, radio buttons, checkboxes, etc.) to present data users cannot change.
  - This refers to controls that are never editable, not to ones that are temporarily inactive (grayed out).

Blooper 7: Overusing text fields for constrained input
- Text fields are too unstructured for constrained data.
  - Dates, postal codes, volume levels, monetary amounts, etc.
  - Especially occurs in paper to GUI conversion.
- Use structured controls to allow only valid data.

Take advantage of the GUI
- Use structured controls to allow only valid data.
Blooper 8: Dynamic Menus
► Menu item that changes depending upon the context
  ▪ Might seem to help; removes commands one shouldn’t be able to execute at that time
  ▪ But users end up wondering where commands went

Dynamic Menus: Better
► Add and remove menus, not menu items

<table>
<thead>
<tr>
<th>File</th>
<th>Edit</th>
<th>View</th>
<th>Format</th>
<th>Window</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Edit</td>
<td>View</td>
<td>Format</td>
<td>Window</td>
<td>Help</td>
</tr>
</tbody>
</table>

Table menu appears if a table is selected.
► Exception: quick lists
  ▪ E.g. recently opened files, bookmarks, opened documents

Blooper 9 : Intolerant Data Fields
► To be friendly and helpful your text fields should tolerate reasonable variations in what people type
  ▪ E.g. filter out spaces (common with cut and paste), dashes, period, tab, etc.
  ▪ Be aware of different formats (zip/post code number only?)

Avoiding Blooper 9
► Match field length to data
  ▪ Visible length suggests how much to type
► Accept common formats
► Beware of rejecting legitimate data
► Make case irrelevant
► Provide a pattern (e.g. draw dashes in QP-00-3412)
► Structure text fields
  ▪ Use pull-down menus or combo-boxes

Blooper 10 : Input fields and controls with no default
► Defaults should be set up with the most likely values; users only need to scan the settings, change a few, and proceed
Blooper 10 Examples
► Drop-down menu with no default and poor labeling

Avoiding Blooper 10
► Use likely default values
► Always add initial value for radio buttons, or “None” as an explicit choice
► When no default can be assumed (e.g. asking a user’s gender), menus are better than radio buttons

Blooper 11 : Poor Defaults
► A default value that is unlikely to be what users want is more harmful than no default value

Avoiding Blooper 11: Choose default values using
- Common sense
- Business logic
- Experience and site data
- Individual user’s data
- Arbitrary choice: only if any value equally likely or makes sense (e.g., gender)

Blooper 12: Negative Checkboxes
► Negative checkboxes turn a feature or attribute OFF when checked and ON when unchecked

Avoiding Blooper 12
- All checkboxes should be positive

BP-1: Focus on the users and their tasks, not on the technology
► Answering these questions:
  - For whom is the system being designed?
  - What is the system for?
  - What problems do the users have now?
  - What skills and knowledge do the users have?
  - How do users conceptualize the data?
  - What are the users’ preferred ways of working?
► Requires collaboration with the user ➔ participatory design
► “Systems should be designed neither for users nor by them, but rather with them.”

BP-2: Consider function, then presentation
► Function means determining the requirements and basic functions of the systems. It doesn’t mean writing actual codes.
► Before layout we must decide what data the users can create, view, or manipulate.
► Develop a conceptual model
  - Not a user interface
  - It is expressed in terms of how users manipulate data, how data is organized, abstract functions of the software and what concepts people need to use it
► For example, a checkbook system might:
  - Organize checks by payees
  - Organize checks by date and check number
BP-3: Conform to the user’s view of the task
► Strive for naturalness  
  • E.g. in chess, drag and drop piece or enter coordinates?  
► Don’t impose arbitrary restrictions  
  • E.g. maximum of 255 entries  
► Use user’s vocabulary, not your own  
► Keep program internals inside the program  
  • Includes error messages

BP-4: Design for the common case
► Ever create a new “object” in a program and find yourself having to change its default properties all the time?  
► Strive to make common tasks easy  
► Sensible defaults, templates or “canned” solutions, wizards, customizability

BP-5: Don’t distract users from their goals
► People are good at multi-tasking, but not for problem solving and stuff we don’t do all the time. Software shouldn’t distract users from their own tasks and goals.  
► Operate in the background, not the foreground of user’s consciousness

BP5 examples
► Image software only displays JPG, not GIF  
  • Forces user to convert to JPG  
► Burying a common operation in a menu and forcing the user to find it by the process of elimination  
► “Hmmm. This checkbox is labeled “Align icons horizontally.” If I uncheck it will my icons be aligned vertically or simply not aligned?  
► “The email sent me an ID number but the website asks for a PIN, is it the same thing?”

BP-6: Facilitate Learning
► System is often hard to learn  
  • Clear to developers but may not be clear to users  
► Example: graphical ambiguity  
  • With lots of icons it is difficult to make them meaningful  
  • Test with users what each icon & control mean  
► The right way to design: Think outside-in  
  • Make sure that the UI makes sense to people who don’t know everything you know about it  
  • Be consistent across the application  
  • Test prototypes on representative users  
► Provide a low-risk environment  
  • If someone makes a mistake it should be easy to correct it

BP-7: Deliver information, not just data
► App to determine if someone is in their office; only requires one bit of information  
  • Lots of ways to get data: phone call, send email, view webcam, knock on door, etc.  
► Don’t treat the data like information, focus on the important data and extract necessary information  
► To present information on a display:  
  • Minimize contrast to focus attention to a specific location  
  • Match the medium, e.g. porting a phone app to a computer might retain the phone UI  
  • Design for scannability  
  • Pay attention to detail

BP-7: Deliver information, not just data
► Don’t take over the screen, e.g. to unilaterally move controls and data around on the screen  
  • Jump or “warp” the mouse to new positions  
  • Move something to the mouse location  
  • Reposition windows  
  • Automatically rearrange data for the user  
► Controlling the mouse for the user violates the hand-eye coordination a user has with the machine  
► When software changes a display to show the effect of a user’s actions, it should try to minimize what it changes  
  • Small local changes should produce small, local changes on the display  
  • Attempt to keep as much of the display unchanged as possible
BP-8: Design for responsiveness

► A system’s ability to keep up with users and not make them wait
  ▪ The most important factor in determining user satisfaction
  ▪ Users hate waiting more than anything else

► Desire for speed is perceived, not actual
  ▪ Game vs. formatting a disk

► Systems can be slow but still responsive
  ▪ Might queue requests but never lock-up or force users to wait for the system to catch up

► Examples of poor responsiveness
  ▪ Delayed feedback for button-press or mouse click
  ▪ Operations that block activity
  ▪ No visual feedback as to how long an operation takes
  ▪ Jerky animation

BP-8: Try it out on users, then fix it!

► Test early and often, results may surprise even experienced designers

► Schedule time to correct problems found by tests

► Tests have two goals
  ▪ Information on aspects of the UI that cause difficulty
  ▪ Socially it convinces developers that there are design problems that need correcting. Some developers need to see users have problems for themselves.

BP-8: Design for responsiveness

► Small things go a long way
  ▪ Acknowledge user actions instantly, even if returning the answer will take time
  ▪ Let users know when it is busy and when it isn’t
  ▪ Free users to do other things while waiting for something to finish
  ▪ Animate movement smoothly
  ▪ Allow users to abort lengthy operations
  ▪ Allow users to accurately judge how long something will take

Effect of response time on user productivity