

Lecture 2:

Why are User Interfaces Hard to Design and Implement?



Brad Myers

05-830

Advanced User Interface Software
Spring, 2020

Who are “Users”?

- People who will use a product or web site.
- As opposed to the “Designers”
 - People who *create* the system or web site
- Designers \neq Users
- You are the *designer*
- Have to make an effort to ***Know The User***

What is the “User Interface”?

- Everything the user encounters
 - Functionality & Usefulness
 - Content
 - Labels
 - Presentation
 - Layout
 - Navigation
 - Speed of response
 - Emotional Impact
 - Documentation & Help

What is “Usability”?

- = Quality!
- Learnability
- Efficiency
 - Productivity
- Memorability
 - Little “re-learning” required
- Errors
- Satisfaction
 - Pleasurable

User “Experience” (UX)

- Even more than “usability”
 - Usability focuses on performance
- User **Experience**
 - Emotion, Heritage
 - Fun, Style, Art
 - Branding, Reputation
 - Political, social personal connections
 - Beyond just the device itself – “Service Design”
- Blends: usability engineering, software engineering, ergonomics, hardware engineering, marketing, graphic design

Why Hard to Design UIs?

“It is easy to make things hard. It is hard to make things easy.”

- No silver bullet
- Seems easy, common sense, but seldom done right
 - Once done right, however, seems “obvious”
- User Interface design is a creative process
- Designers have difficulty thinking like users
 - Often need to understand task domain
 - Can’t “unlearn” something

Can't Unlearn Something



Why Difficult, 2

- Specifications are always wrong:
 - "Only slightly more than 30% of the code developed in application software development ever gets used as intended by end-users. The reason for this statistic may be a result of developers not understanding what their users need."
 - Hugh Beyer and Karen Holtzblatt, "Contextual Design: A Customer-Centric Approach to Systems Design," *ACM Interactions*, Sep+Oct, 1997, iv.5, p. 62.
- Need for prototyping and iteration

Why Difficult, 3

- Tasks and domains are complex
 - Word 1 (100 commands) vs. Word 2013 (>2000)
 - MacDraw 1 vs. Illustrator
 - BMW iDrive adjusts over 700 functions
- Adding graphics can make *worse*
 - Pretty ≠ Easy to use
- Can't necessarily just copy other designs
 - Legal issues
- All design/development involves *tradeoffs*
 - Add Features
 - Test/Fix Bugs
 - Test/Fix usability
 - **Time-to-market**

Why are User Interfaces Difficult to *Implement*?



What are the most difficult kinds of programs, in general?

- What properties make a task difficult to program?

What are the most difficult kinds of programs, in general?

- What properties make a task difficult to program?
- *GUI programming has most of them!*

Why Are User Interfaces Hard to Implement?



- They are hard to design, requiring iterative implementation
 - Not the waterfall model: specify, design, implement, test, deliver
- They are reactive and are programmed from the "inside-out"
 - Event based programming
 - More difficult to modularize
- They generally require multi-processing
 - To deal with user typing; aborts
 - Window refresh
 - Window system as a different process
 - Multiple input devices

Why Hard to Implement? cont.

- There are real-time requirements for handling input events
 - Output 60 times a second
 - Keep up with mouse tracking
 - Video, sound, multi-media
- Need for robustness
 - No crashing, on any input
 - Helpful error messages and recover gracefully
 - Aborts
 - Undo

Why Hard to Implement? cont.

- Lower testability
 - Few tools for regression testing
- Little language support
 - Primitives in computer languages make bad user interfaces
 - Enormous, complex libraries
 - Features like object-oriented, constraints, multi-processing
- Complexity of the tools
 - Full bookshelf for documentation of user interface frameworks
 - MFC, Java Swing, VB .Net, etc.
- Difficulty of Modularization

Examples

- Difference between displaying “hello” in console and displaying a blue rectangle in a window
- Difficulty to read a file name
 - Readln() in Pascal, Java, C++, etc.
 - Vs. tool in modern toolkits
 - Complexity of the file dialog itself
 - You must deal with aborting, undo, etc.



Common Patterns are Hard to Use

- Our research shows that software that uses *design patterns* is usually hard to learn and to use
- Most UI software uses the *listener pattern* (also called the *observer pattern*)
 - Set up methods (often *call-backs procedures*) when events happen, like `mouse-down`, `mouse-move`
 - Hard to trace the control flow
 - Static analysis and control flow graphs are less useful
 - Debugging is difficult when *not* called

PhD research of Stephen Oney:

```
var isDragLocked = false,
    mm_listener = function(mm_event) {
        draggable.attr({ x: mm_ev.x, y: mm_ev.y });
    },
    mu_listener = function(mu_event) {
        removeEventListener("mousemove", mm_listener);
        removeEventListener("mouseup", mu_listener);
    };

draggable.mousedown(function(md_ev) {
    draggable.attr({ x: md_ev.x, y: md_ev.y });
    addEventListener("mousemove", mm_listener);
    addEventListener("mouseup", mu_listener);
}).dblclick(function(md_event) {
    if(isDragLocked) {
        removeEventListener("mousemove", mm_listener);
    } else {
        addEventListener("mousemove", mm_listener);
    }
    isDragLocked = !isDragLocked;
});
```


PhD research of Stephen Oney:



```
var isDragLocked = false,
    mm_listener = function(mm_event) {
        draggable.attr({ x: mm_ev.x, y: mm_ev.y });
    },
    mu_listener = function(mu_event) {
        removeEventListener("mousemove", mm_listener);
        removeEventListener("mouseup", mu_listener);
    };

draggable.mousedown(function(md_ev) {
    draggable.attr({ x: md_ev.x, y: md_ev.y });
    addEventListener("mousemove", mm_listener);
    addEventListener("mouseup", mu_listener);
}).dblclick(function(md_event) {
    if(isDragLocked) {
        removeEventListener("mousemove", mm_listener);
    } else {
        addEventListener("mousemove", mm_listener);
    }
    isDragLocked = !isDragLocked;
});
```

PhD research of Stephen Oney:



```
var isDragLocked = false,
    mm_listener = function(mm_event) {
        draggable.attr({ x: mm_ev.x, y: mm_ev.y });
    },
    mu_listener = function(mu_event) {
        removeEventListener("mousemove", mm_listener);
        removeEventListener("mouseup", mu_listener);
    };


draggable.mousedown(function(md_ev) {
    draggable.attr({ x: md_ev.x, y: md_ev.y });
    addEventListener("mousemove", mm_listener);
    addEventListener("mouseup", mu_listener);
}).dblclick(function(md_event) {
    if(isDragLocked) {
        removeEventListener("mousemove", mm_listener);
    } else {
        addEventListener("mousemove", mm_listener);
    }
    isDragLocked = !isDragLocked;
});
```

PhD research of Stephen Oney:



```
var isDragLocked = false,
    mm_listener = function(mm_event) {
        draggable.attr({ x: mm_ev.x, y: mm_ev.y });
    },
    mu_listener = function(mu_event) {
        removeEventListener("mousemove", mm_listener);
        removeEventListener("mouseup", mu_listener);
    };

draggable.mousedown(function(md_ev) {
    draggable.attr({ x: md_ev.x, y: md_ev.y });
    addEventListener("mousemove", mm_listener);
    addEventListener("mouseup", mu_listener);
}).dblclick(function(md_event) {
    if(isDragLocked) {
        removeEventListener("mousemove", mm_listener);
    } else {
        addEventListener("mousemove", mm_listener);
    }
    isDragLocked = !isDragLocked;
});
```



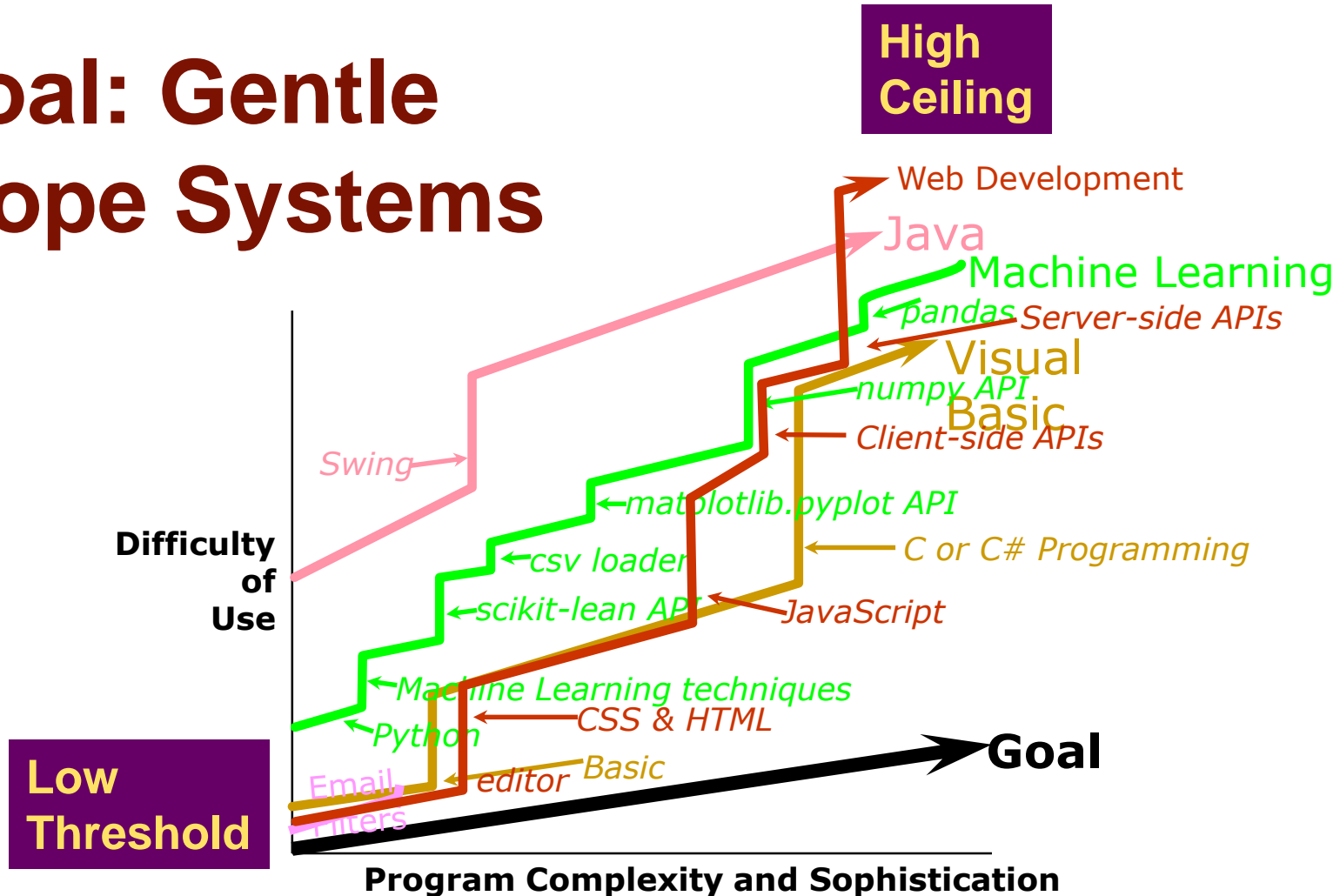
```
var isDragLocked = false,
    mm_listener = function(mm_event) {
        draggable.attr({ x: mm_ev.x, y: mm_ev.y });
    },
    mu_listener = function(mu_event) {
        removeEventListener("mousemove", mm_listener);
        removeEventListener("mouseup", mu_listener);
    };

draggable.mousedown(function(md_ev) {
    draggable.attr({ x: md_ev.x, y: md_ev.y });
    addEventListener("mousemove", mm_listener);
    addEventListener("mouseup", mu_listener);
}).dblclick(function(md_event) {
    if(isDragLocked) {
        removeEventListener("mousemove", mm_listener);
    } else {
        addEventListener("mousemove", mm_listener);
    }
    isDragLocked = !isDragLocked;
});
```

We called this “the Spaghetti of Call-Backs” back in 1991!

- Brad A. Myers. "Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs," *ACM Symposium on User Interface Software and Technology: **UIST'91***, Hilton Head, SC, Nov. 11-13, 1991. pp. 211-220. [pdf](#) or [YouTube video](#) or [local video](#)
- Need callbacks for all event types
 - Mouse down, up, move, double-tap, etc.
- Also for:
 - Data validation
 - Error handling
 - Final action (for “submit”, OK, Cancel)
- Still the way most UI toolkits work

Goal: Gentle Slope Systems



Why Tools?

- **The quality of the interfaces will be higher.** This is because:
 - Designs can be rapidly prototyped and implemented, possibly even before the application code is written.
 - It is easier to incorporate changes discovered through user testing.
 - More effort can be expended on the tool than may be practical on any single user interface since the tool will be used with many different applications.
 - Different applications are more likely to have consistent user interfaces if they are created using the same user interface tool.
 - A UI tool will make it easier for a variety of specialists to be involved in designing the user interface.

Why Tools, cont.

- **The user interface code will be easier and more economical to create and maintain.** This is because:
 - There will be less code to write, because much is supplied by the tools.
 - There will be better modularization due to the separation of the user interface component from the application.
 - The level of expertise of the interface designers and implementers might be able to be lower, because the tools hide much of the complexities of the underlying system.
 - The reliability of the user interface may be higher, since the code for the user interface is created automatically from a higher level specification.
 - It may be easier to port an application to different hardware and software environments since the device dependencies are isolated in the user interface tool.

What should tools do?

- Help **design** the interface given a specification of the tasks.
 - Help **implement** the interface given a design.
 - Help **evaluate** the interface after it is designed and propose improvements, or at least provide information to allow the designer to evaluate the interface.
-
- Create easy-to-use interfaces.
 - Allow the designer to rapidly investigate different designs.
 - Allow non-programmers to design and implement user interfaces.
 - Provide portability across different machines and devices.
 - Be easy to use themselves.

Tools might do:

- Provide sets of standard UI components
- Guide the implementation
- Help with screen layout and graphic design.
- Validate user inputs
- Handle user errors
- Handle aborting and undoing of operations
- Provide help and prompts
- Deal with field scrolling and editing
- Insulate the application from all device dependencies and the underlying software and hardware systems.
- Support features in the interface that allow the end user to customize the interface.

UI Tools stack

Interactive Tools
(Builders, Prototypers)

Framework
(Architecture, Objects)

Toolkit
(library, programming interface)

OS / Windows Interface
(Input and Output)

Device Drivers & Hardware

From the bottom: Windows & OS

- Window System + Operating System
 - Microsoft Windows, MacOS, Android, iOS, etc.
- Unix & older OS's separated OS, Windows
 - SunOS: X Windows or NeWS or SunTools
- Low level input events – keycodes, mouse position, values from accelerometers
- Low level graphics primitives
 - Draw Circle, Draw Line, set pixel color
- Clipped to window boundaries

Toolkits

- (Specific meaning, one part of the tool set)
- A library of procedures
 - Only a programming interface
- Provides higher-level “widgets”
 - Also called “controls”
 - Scroll bars, buttons, text input fields
- Examples:
 - Html, canvas, svg
 - Java Swing, SWT, AWT
 - Win32, Macintosh “toolbox”

Frameworks

- Higher-level programming architecture
- Common **design patterns**
 - Listener pattern, data bindings, etc.
- Significantly affects design of applications
- Often object-oriented
 - “Foundation Classes”
- Often cross-platform (iOS + Android)
 - React native, Flutter, Microsoft’s Xamarin, Titanium, ...
 - Electron (<https://electronjs.org/>): cross-platform toolkit for desktop apps
- Sometimes hard to distinguish from “toolkits”
 - (So we usually won’t!)
- Other Examples:
 - Historical: Apple MacApp, my Amulet
 - Current: Unity

Interactive Tools

- Not a programming interface
- Supports designers who might not be programmers
- Select widgets and place them
 - Layout, possibly with constraints
 - Specify properties of widgets
- Prototypes *or* real code
 - For real code, often built into IDEs
- Examples:
 - Adobe Dreamweaver for web pages
 - Prototypers: Balsamiq, Axure, etc.
 - Resource editors & builders: Eclipse, Xcode IB, Android studio