Lecture 5:

What is "User Interface Software", Overview of UI Software and Tools

05-431/631 Software Structures for User Interfaces (SSUI)

Fall, 2022

Logistics

- Homework 1 due today!
  - Late policy: https://www.cs.cmu.edu/~bam/uicourse/05631fall2022/homeworks.html#latepolicy
- Homework 2 is posted, due in 2 weeks: 9/27/2022
  - https://www.cs.cmu.edu/~bam/uicourse/05631fall2022/HW2/index.html

What does UI Software encompass?

- "User Interface Software"
  - All the software that implements the user interface
  - "User Interface" = The part of an application that a person (user) can see or interact with (look + feel)
    - Often distinguished from the "functionality" (back-end) implementation
    - Some ambiguity if a tool is specifically for "UI" part
- User Interface Software Tools
  - Ways to help programmers create user interface software
- Tools used by UI Designers and UI Builders

Names

- Tools = UI Frameworks, Toolkits, Development Kits = SDK, Libraries
  - UI APIs = Application Programming Interfaces (APIs)
  - Old names = User Interface Management Systems (UIMS), User Interface Development Environments (UIDE)
- Micro-Service Architecture for web UIs
- Also interactive tools: Resource Editors, Interface Builders, Prototypers, UI Builders

Examples of UI Software Tools

- We created a list in 2017 = https://docs.google.com/document/d/1qeBkQaoIBmwK9Z7LmWP4L_4GKgddzNn7YmvSQuPukGo
  - Over 100!
  - Lots of new ones that should be added
- APIs for UI development:
  - JavaScript/Web: ReactJS, Vue, AngularJS
  - Apple Cocoa, Carbon
  - Microsoft Foundation Classes, .Net, wx-Python
  - Java AWT, Swing, Android UI classes
- Interactive tools
  - Visual Basic .Net
  - Adobe Flash, Adobe Catalyst
  - Prototypers like Axure, Balsamiq, Adobe XD
- Research systems:
  - Garnet
  - Amulet
  - ConstraintJS
  - InterState

Why is This Important?

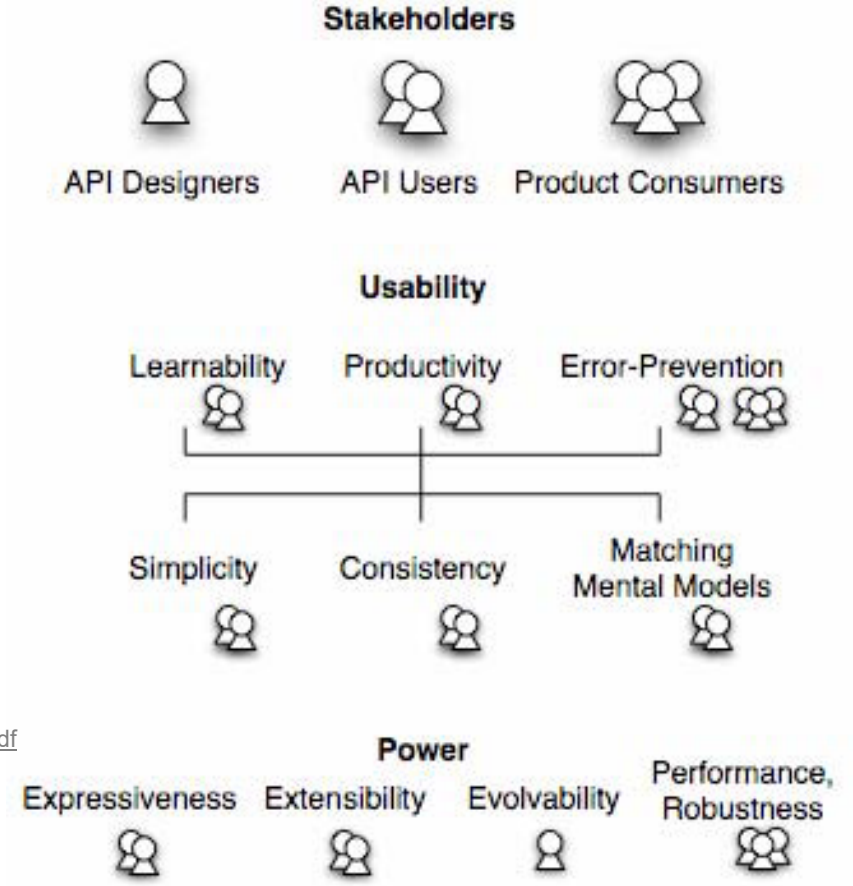
- Virtually all UIs are created using such Tools
- Previous research has influenced today's tools
  - Enormous impact!
- New tools are created all the time
  - E.g., Flutter for Dart language and mobile
  - Some are easier to use than others!
- Principles and architectures for good designs
  - Avoid "reinventing the wheel"
  - What are the "best practices" for these tools
- Modern UIs and Tools for web, phones, wearables, Smart TVs, etc. all use similar designs
  - Speech and conversational interfaces are different
- Research topic in ACM UIST, CHI
  - Also ICSE, SPLASH, PLATEAU, CHASE, many others

Analogy with OS or Compilers

- In CS curriculums, often teach Operating Systems, Compilers, Networking
  - Even though most people will just *use* these
  - Useful to know the *key principles*
  - Lasts longer than the specifics of one current tool
- Some people will need to know how to *build* the next generation of tools
- The algorithms, architectures, design patterns, and principles are useful in *other* situations as well

Stakeholders

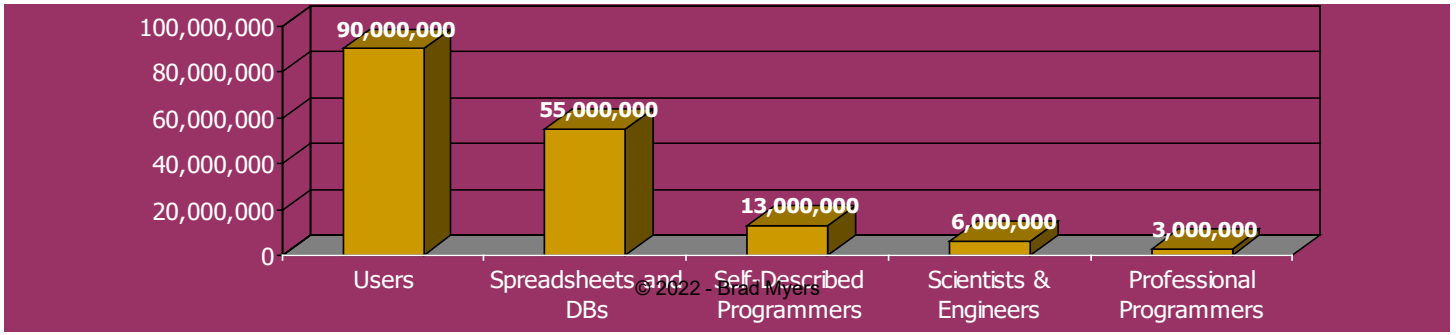
- Many groups of people
- Need unambiguous names
  - Too many “users”
- Tool Designers
- Tool Users (are programmers)
- Users of products created with the tools = consumers or end-users



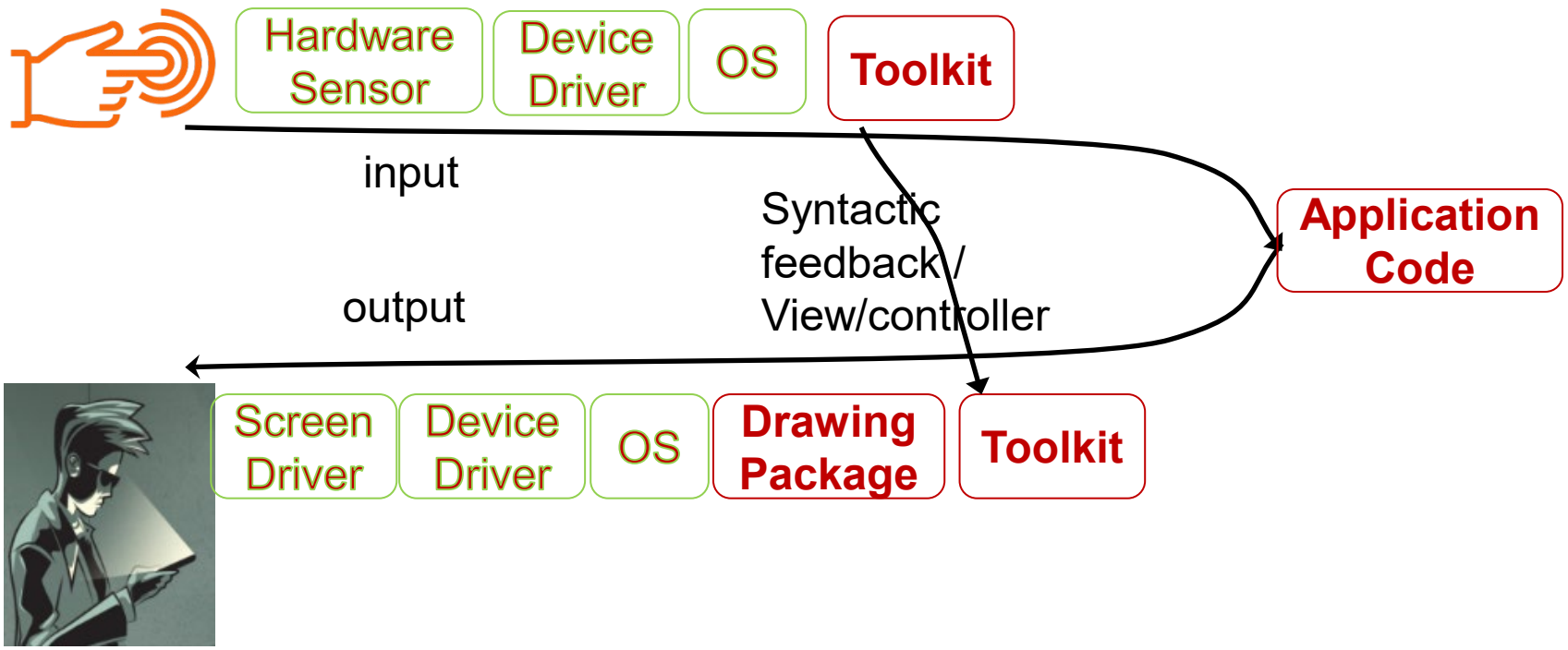
- Source: Jeffrey Stylos and Brad Myers, "Mapping the Space of API Design Decisions," *2007 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'07*. Sept 23-27, 2007, Coeur d'Alene, Idaho. pp. 50-57. pdf

Who Are Developers?

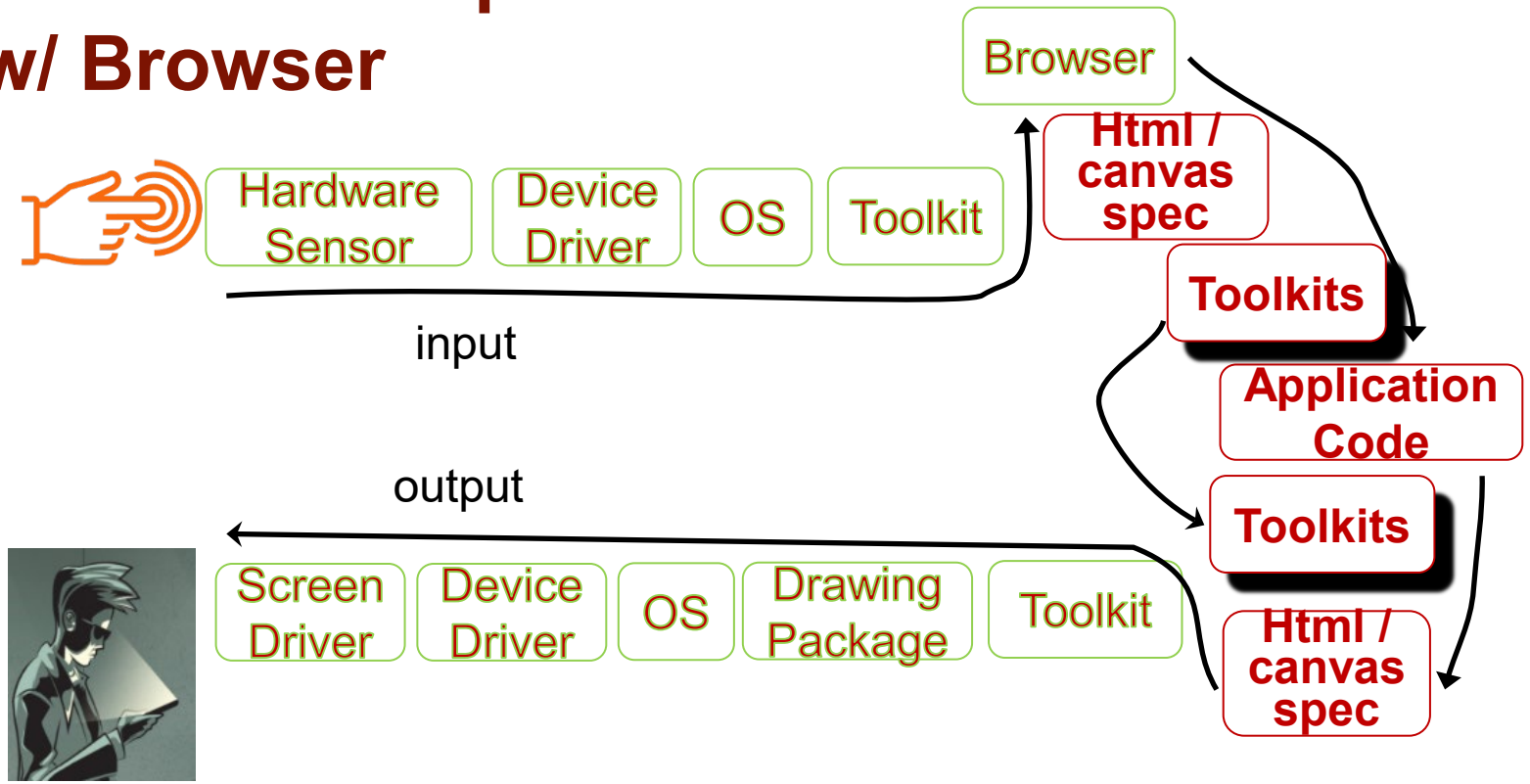
- Programming tools are not just used by highly-trained professional programmers
- End-User Programmers = People whose primary job is *not* programming
- In 2012 in USA at work: — *[Scaffidi, Shaw and Myers 2005]*
  - 3 million professional programmers
  - 6 million scientists & engineers
  - 13 million will describe themselves as programmers
  - 55 million will use spreadsheets or databases at work
  - 90 million computer users at work in US
- All these may be the tool users!



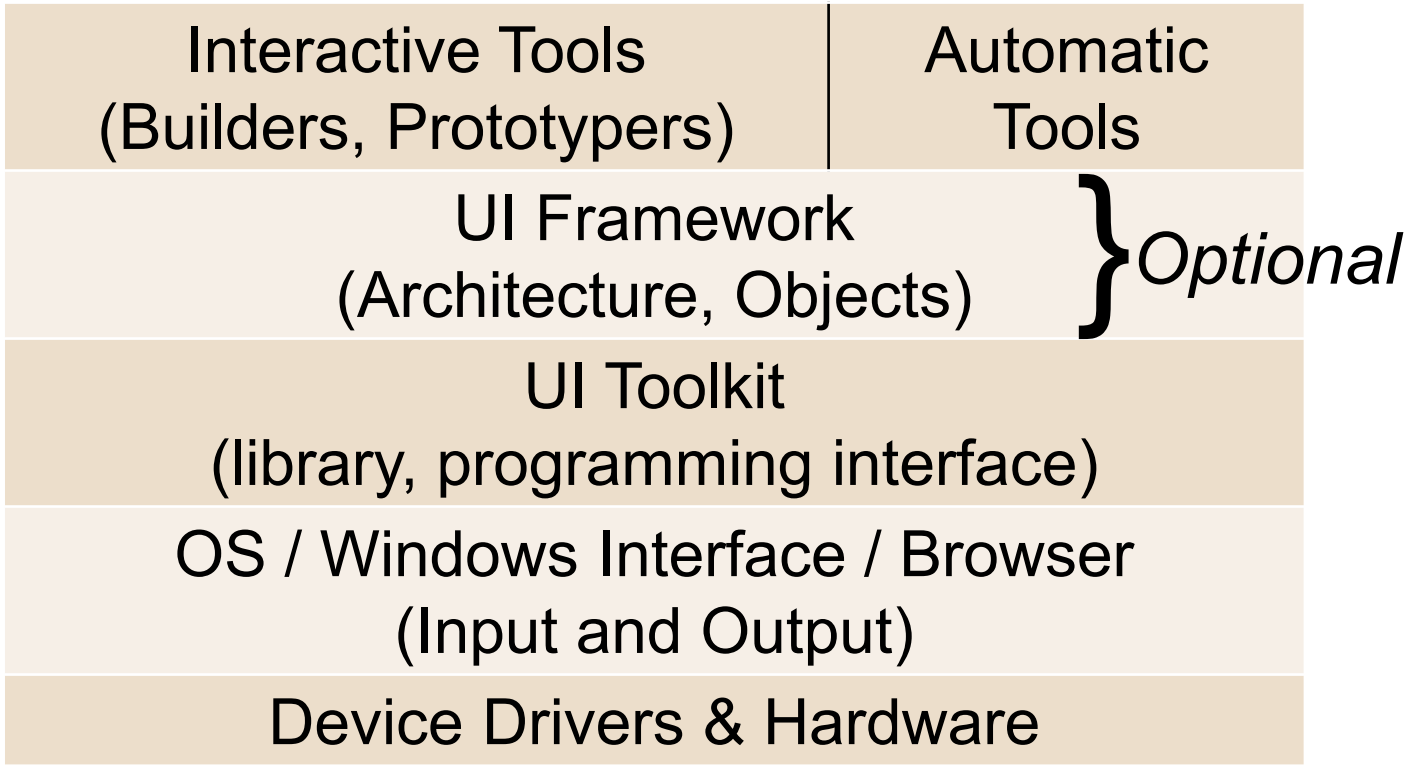
UI Runtime Pipeline



UI Runtime Pipeline w/ Browser



UI Tools stack



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
- Or the equivalent provided by the browser's "OS"
- Low level input events – keycodes, mouse position, values from accelerometers
- Low level graphics primitives
  - Draw Circle, Draw Line, set pixel color
  - Called "Imaging Model"
- Clipped to window boundaries

UI Toolkits

- (Specific meaning, one part of the tool set)
- A library of UI widgets that application programs can use
  - Only a programming interface
- Provides higher-level "widgets"
  - Also called "controls"
  - Scroll bars, buttons, text input fields
- Examples:
  - Html "input"
  - Java Swing, SWT, AWT
  - Win32, Macintosh "toolbox"

UI Frameworks

- Higher-level programming architecture
  - Generally all-encompassing – controls all of the ways apps are built
- Supports 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!)
  - Note: React says it is a "library" and not a "framework", but provides React.Component and other classes from which developers should inherit
- Other Examples:
  - Historical: Apple MacApp (invented in 1986), my Amulet
  - Current: Unity, AngularJS

Historical Notes

- OO
  - Early toolkits and Frameworks invented their own object systems
  - Before C++ was widely used
- Look and Feel
  - Early UI Toolkits and Frameworks tried not to enforce a look-and-feel
    - Microsoft DOS
    - Unix and "X window system" (1986) originally
      - "…the system must provide hooks (mechanism) rather than religion (policy)." [Scheifler, 1986]
      - "intrinsics" layer of the toolkit
        - How widgets would be built
      - Gave up on this approach as early as 1989
  - Windows, Mac, iOS, Android – quite constrained look and feel
  - Web – still pretty open, but built-in widgets (html "input") standardized

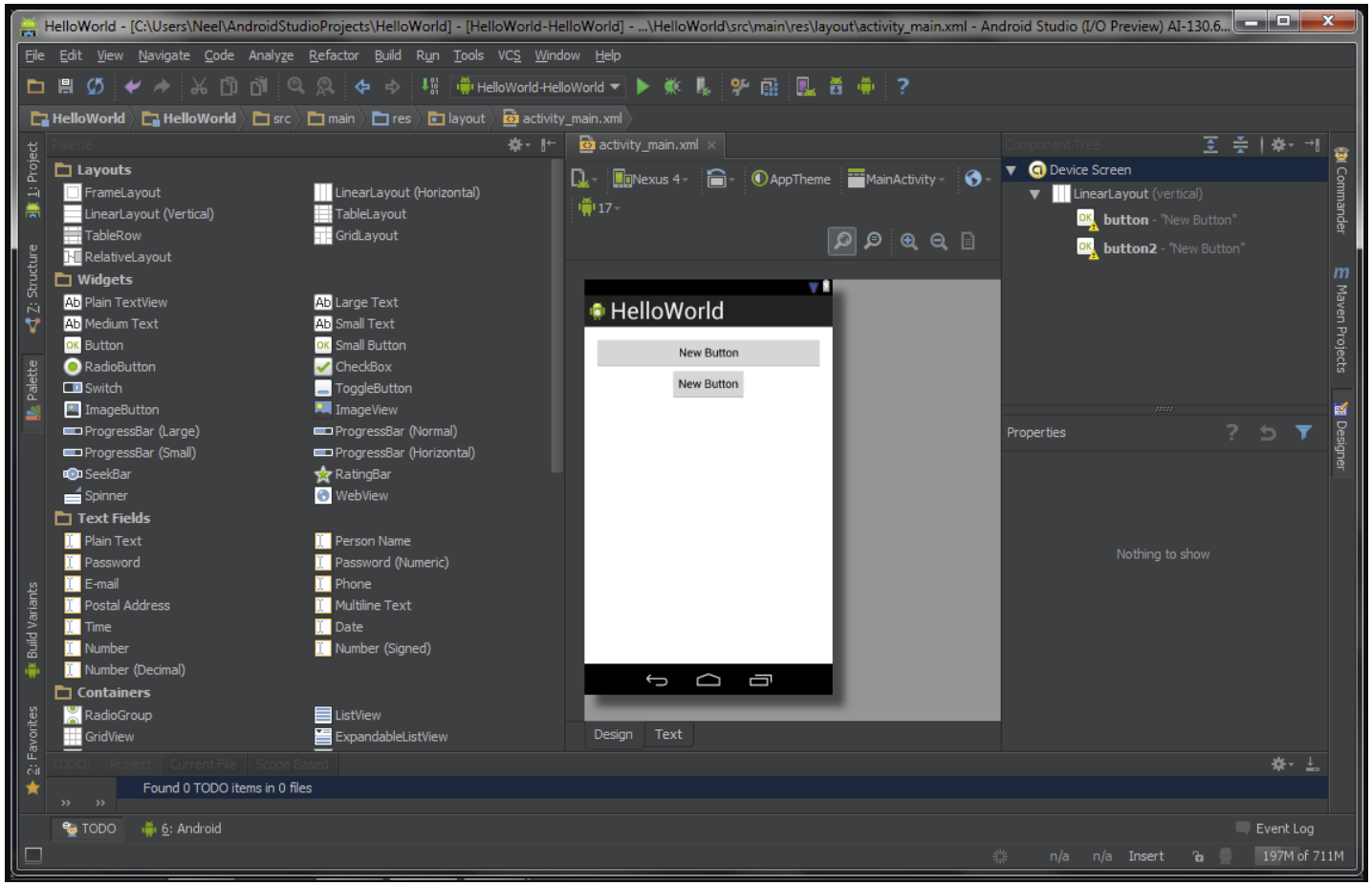
APIs

- Application Programming Interface (API)
  - “the collection of software utilized without change to create other software”
- Covers both UI Toolkits and UI Frameworks
- Plus all the other libraries programmers need to use

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
- Two categories:
  - GUI Tools – create representations used by the real code
    - Often built into IDEs
  - Prototypers – just to work out look and feel, and must be re-implemented
- Examples:
  - Adobe Dreamweaver for web pages
  - Resource editors & builders: Eclipse, Xcode IB, Android studio, Microsoft Visual Basic IDE
  - Prototypers: Figma, Balsamiq, Axure, etc.

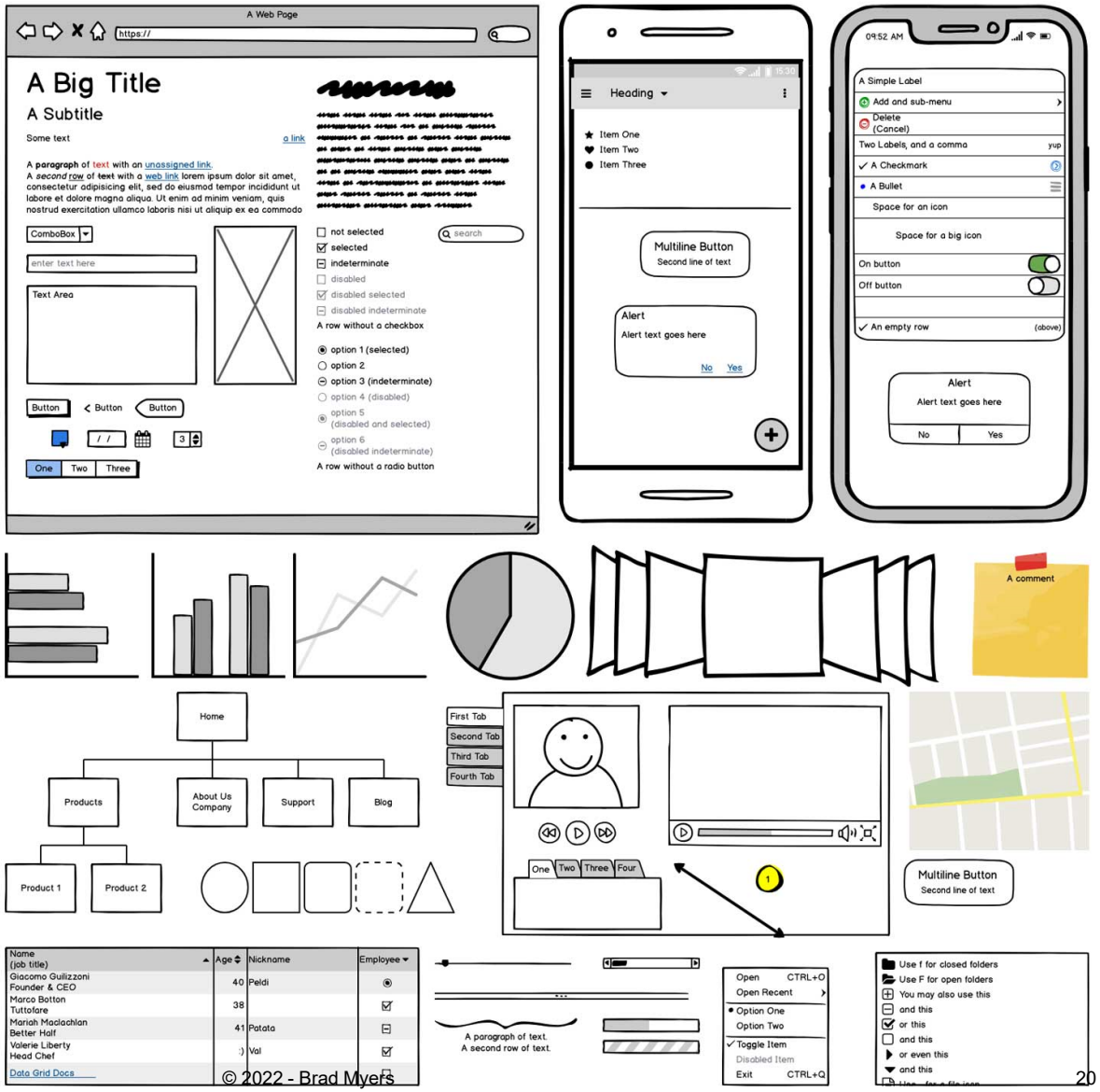
Android Studio Builder



- https://stacktips.com/tutorials/android/android-studio-features/attachment/android-studio-viewing-layout_builder

Balsamiq

- https://balsamiq.com/wireframes/



Automatic Tools

- Take a high-level specification of the UI
- Use algorithms to decide on such features as which specific widgets to use, the layout of where the widgets should go, and properties of the widgets
- Often AI-based
- Sometimes called “model-based design”
- Many research examples; few commercial successes
- Html “decides” on layout but not AI based, so doesn’t really “count”

Research Example

- Jeff Nichols' PhD system PUC took high-level spec in XML of a consumer electronics device and created personalized UI on a portable device
  - Jeffrey Nichols and Brad A. Myers. "Creating a Lightweight User Interface Description Language: An Overview and Analysis of the Personal Universal Controller Project". *ACM Transactions on Computer-Human Interaction*,. Vol. 16, no. 4, (November 2009). pp. 1-37. ACM DL

```
<state name="Channel" is-a="channel">
  <type type-name="ChannelType">
    <integer>
      <min>
        <constant value="2"/>
      </min>
      <max>
        <constant value="128"/>
      </max>
    </integer>
  </type>
  <labels>
    <label>Channel</label>
    <label>Chan</label>
  </labels>
</state>
```