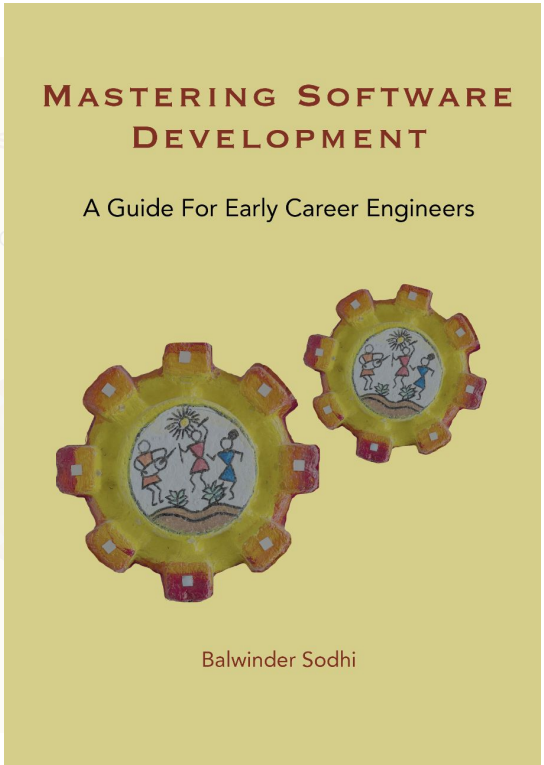
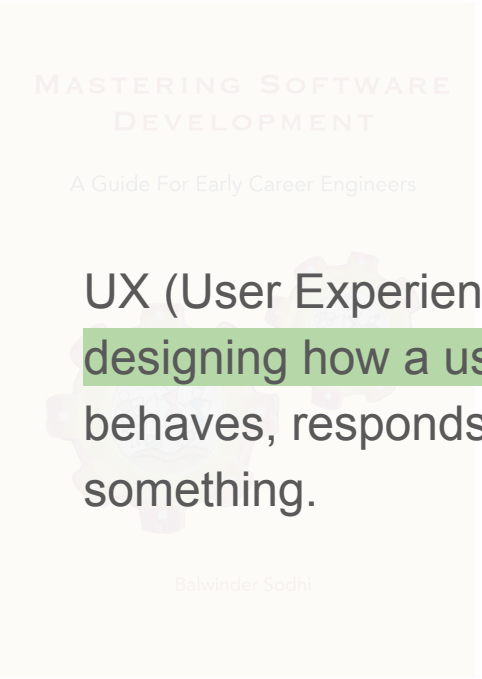


Basics of UX-UE

Balwinder Sodhi

What exactly is UX/UE? 1/2



UX (User Experience) or UE (User Experience Engineering) is the discipline of designing how a user *experiences* a product — not just how it looks, but how it behaves, responds, guides, and supports the user while they try to achieve something.

What exactly is UX/UE? 2/2

- What UX Really Is
  - UX is the end-to-end quality of interaction between a user and a system.
  - It covers everything that affects how someone feels and how effectively they can use the product.
  - UX = usefulness + ease of use + clarity + efficiency + emotion

  …all working together.

- What UX Is Not
  - It's not just UI (colors, buttons, layouts).
  - It's not only usability testing.
  - It's not only wireframes or visuals.
  - It's not limited to front-end developers or designers.
- UX includes engineering decisions, architecture, API ergonomics, error handling, performance, accessibility, and workflows.

A Good Mental Model

Think of UX as answering:

- Can the user understand what to do?
- Can they do it without friction or confusion?
- Does the system behave in a predictable, forgiving, and fast way?
- Does it feel good to use?
- Does it help them complete their goal efficiently?
- If any of these fail, the user's experience suffers.

UX vs GUI

Aspect	UX	GUI
Focus	How it works, how it feels	How it looks on screen
Includes	Workflows, performance, error handling, guidance, accessibility	Layouts, typography, color, visual hierarchy
Goal	Help users achieve tasks smoothly	Present information clearly

Most UX problems are engineering problems

- Slow systems → bad UX
- Poor error messages → bad UX
- Inconsistent data states → bad UX
- Complex workflows → bad UX
- Backend failures without recovery → bad UX
- APIs that are hard for developers → bad UX for integrators
- Lack of accessibility → excludes users

UX is about how the product works in the real world, not just how it is drawn on a screen.

Real-World Examples

- Bad UX
  - A form that clears everything after a validation error.
  - A slow dashboard with no loading indicators.
  - An app where the “Back” button behaves differently on different screens.
  - A complex enterprise workflow with 12 steps when 4 would do.
- Good UX
  - Gmail auto-saves drafts in real time.
  - VS Code lets beginners use menus and experts use shortcuts.
  - Stripe Checkout handles errors gracefully and shows recovery paths.
  - Slack’s search bar exposes shortcuts with hints as you type.

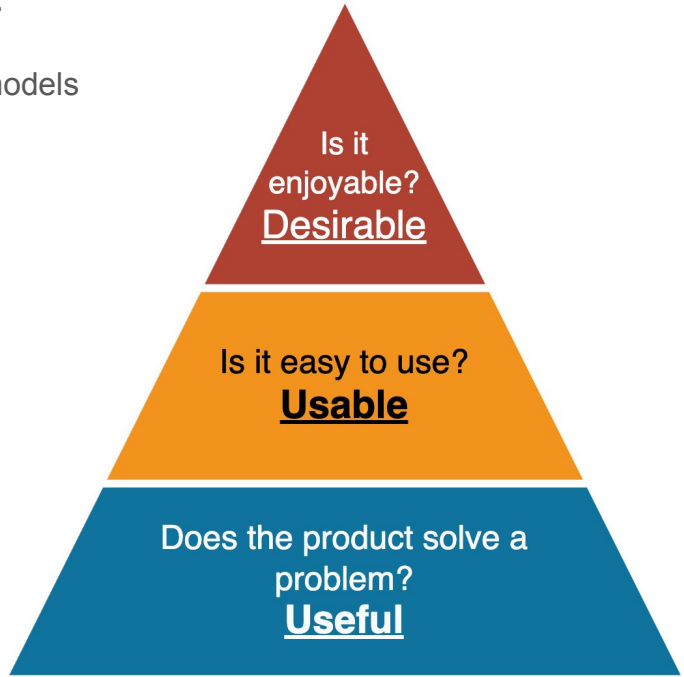
Why Should a Developer Care?

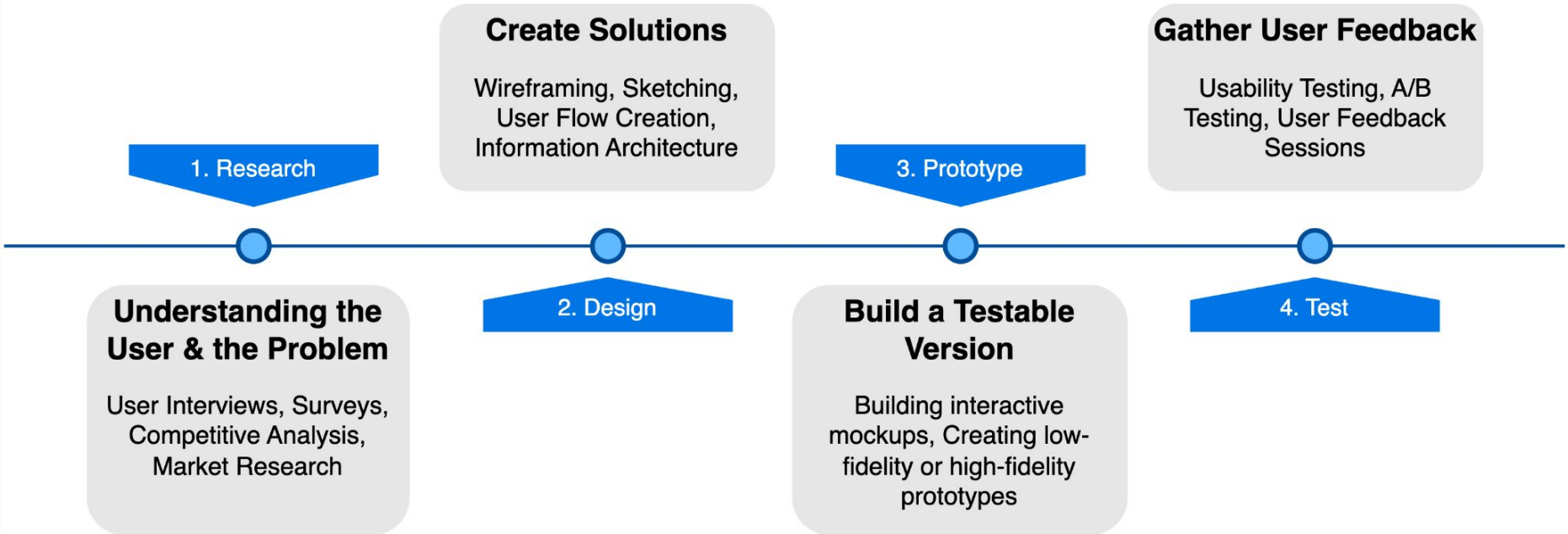
- Users judge software in seconds — UX issues often matter more than feature depth.
- Poor UX creates hidden costs: support tickets, training overhead, churn, and operational risk.
- Good UX accelerates adoption and reduces cognitive load.
- Real-world examples:
  - AWS Console: powerful but often criticized for complexity.
  - Stripe Dashboard: praised for clarity and intuitive flows despite domain complexity.

Key Principles

Usability

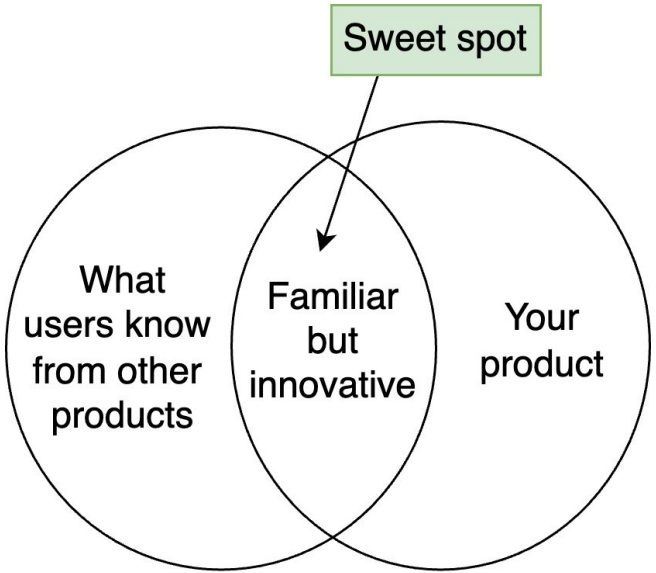
- It means users can accomplish tasks efficiently and without confusion.
- Focuses on clarity, intuitiveness, and ease of learning.
- What engineers influence
  - Well-structured data models that map cleanly to user mental models
  - Clear defaults and fallbacks
  - Predictable behavior under latency or failure
  - Minimizing “surprise” in the workflow
- Signs of poor usability
  - Users need documentation for basic tasks
  - Frequent support tickets for “simple” actions
  - Confusion around terminology or data state
- Patterns that help
  - Inline guidance
  - Smart defaults
  - Progressive disclosure (show complexity only when needed)
- Example
  - Gmail auto-saves drafts → avoids accidental data loss.




1. Research
Understanding the User & the Problem
User Interviews, Surveys, Competitive Analysis, Market Research
2. Design
Create Solutions
Wireframing, Sketching, User Flow Creation, Information Architecture
3. Prototype
Build a Testable Version
Building interactive mockups, Creating low-fidelity or high-fidelity prototypes
4. Test
Gather User Feedback
Usability Testing, A/B Testing, User Feedback Sessions

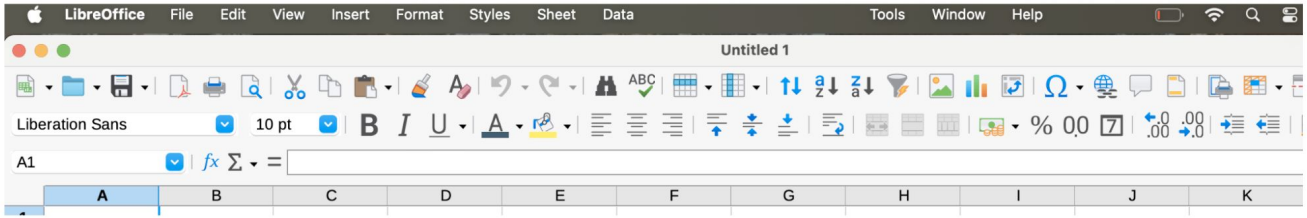
Consistency

- Predictability reduces cognitive load.
- Enables transfer learning from one part of the system to another.
- Types of consistency
  - Visual (colors, typography, spacing)
  - Behavioral (navigation patterns, button actions)
  - Terminological (same labels everywhere)
- Engineering considerations
  - Component reuse in shared libraries
  - Consistent API responses
  - Consistent error codes and messages
  - Strict typing to prevent accidental variations
- Example
  - GitHub uses consistent naming ("Issues", "Pull Requests", "Actions") across repos.

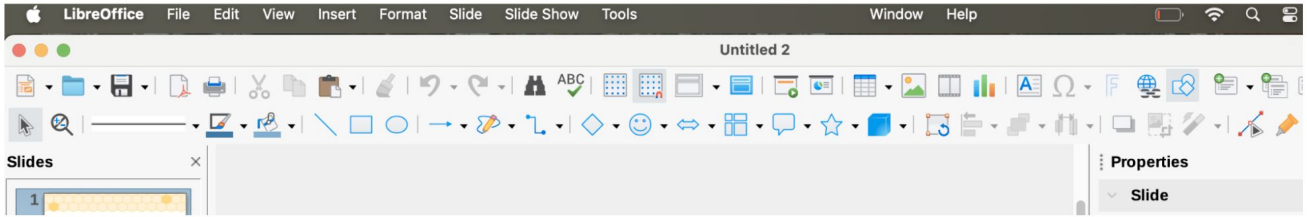


Consistency example

(a) LibreOffice Writer Toolbar



(b) LibreOffice Calc Toolbar



(c) LibreOffice Impress Toolbar

Responsiveness & Performance

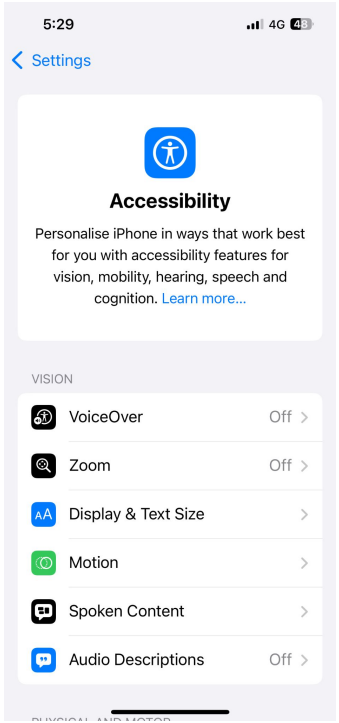
- Key ideas
  - Fast response times reduce user frustration and increase task completion rates.
  - Perceived performance == real performance for UX.
- Users perceive speed as part of quality. A system that “feels” fast is fast from the user’s perspective.
- What matters
  - Time-to-interactive (initial render)
  - Smooth transitions
  - Avoiding blocking interactions
  - Predictable loading states
- Engineering responsibilities
  - Async processing with clear UI feedback
  - Cache aggressively with clear invalidation semantics and lazy-loading
  - Using skeleton screens instead of blank UI
  - Optimizing queries and over-fetching
- UX cost of slow systems
  - Abandonment
  - Double submissions
  - Task context loss
- Example
  - YouTube loads video frame and metadata early; heavy features load later.

Feedback & Affordances

- Feedback = the system's response to a user action.
- Affordances = signals that show what actions are possible.
- Good feedback examples
  - Button enters “loading” state
  - Form fields show live validation
  - Snackbar/alert confirms save
  - Drag-and-drop highlights drop zones
- Poor feedback examples
  - Clicking “Submit” does nothing for 3 seconds
  - Errors appear only after submit and at top of form
  - Interactions that look like buttons but are not clickable
- Why it matters
  - Feedback is about system observability for humans.

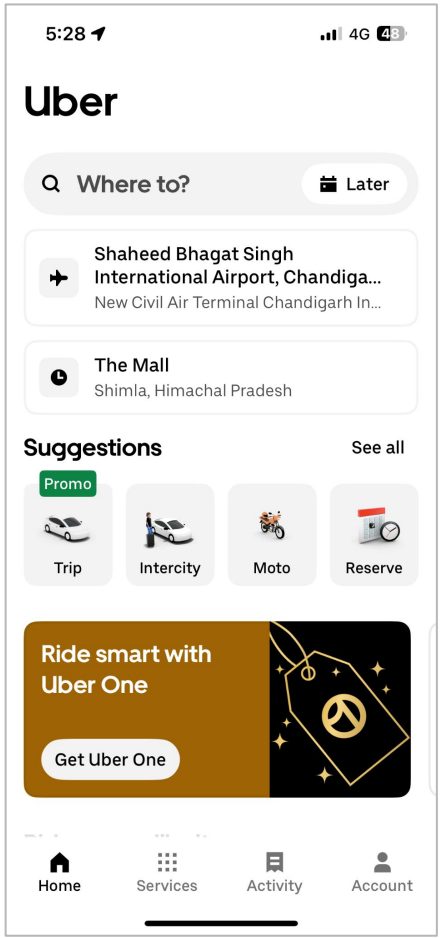
Accessibility

- Expands reach and ensures legal compliance (WCAG, Section 508).
- Good accessibility improves usability for everyone.
- Principles
  - High contrast text
  - Keyboard navigation, tab order
  - Screen reader compatible structure and semantics
  - Captions for audio/video content
  - Avoiding hover-only interactions
- Example
  - Apple's system-wide accessibility features integrate seamlessly with apps.



Contextual Design

- Design that understands user goals, constraints, environment, and workflow.
- Engineering considerations
  - System architecture must match real workflows
  - Context-specific defaults
  - Reducing context switching
  - Personalization based on role or past behavior
- Examples
  - CRM showing relevant fields based on contact type
  - Code editor surfaces refactoring options only when relevant
  - Contextual design is the bridge between “what users do” and “what software needs to support.”



Simplicity (KISS)

- Simplicity is not about lack of features—it's about reducing cognitive load.
- Strategies
  - Remove unnecessary choices
  - Hide complexity until required (advanced settings)
  - Show only essential actions at each step
- Engineering aspects
  - Avoid overloaded screens
  - Avoid over-engineered workflows
  - Use feature flags to keep UI clean for most users
- Anti-pattern
  - "More settings" to solve every edge case → complexity explosion.

Error Handling & Prevention

- Errors should be prevented; when they occur, recovery should be simple.
- Prevention
  - Constraints and validation (client + server)
  - Smart defaults
  - Confirmation for destructive actions
  - Autosave
- Recovery
  - Clear error messages (actionable, specific)
  - Undo/rollback
  - Retry mechanisms
- Engineering mindset
  - Treat UX errors like API errors: predictable, recoverable, explicit.

User Control & Flexibility

- Good UX balances guided workflows with freedom for experienced users.
- User control means
  - Undo
  - Cancel
  - Redo
  - Manual overrides
  - Granular permissions
- Flexibility means
  - Keyboard shortcuts
  - Custom views
  - Bulk actions
  - Configurable defaults
- Examples
  - Photoshop: toolbar for beginners, shortcuts for experts
  - VS Code: settings JSON + GUI settings
- Flexibility prevents “tool fatigue” for power users.

Aesthetic & Minimalist Design

- A clean interface improves comprehension and reduces visual noise.
- Engineering influence
  - Design tokens: spacing, color, typography
  - Consistent grid/layout system
  - Avoiding unnecessary decoration (shadows, gradients)
- Why minimalism matters
  - Essential information stands out
  - Reduces cognitive load
  - Improves scannability
- Good UI design serves clarity, not decoration.

Task Efficiency & Workflow Optimization

- UX must optimize the cost of completing high-frequency tasks.
- Techniques
  - Predictive inputs (autocomplete)
  - Pre-filled forms
  - Macro-like bulk actions
  - One-click workflows
  - Reducing navigation hops
- Engineering work
  - Event-driven architecture
  - Fast API responses
  - Debouncing and throttling
  - Avoid reloading whole screens unnecessarily
- Workflow optimization = reducing total interaction cost.

Onboarding & Learnability

- Users should reach value quickly and learn the system as they go.
- Techniques
  - First-time experience (FTX) flows
  - Guided tours
  - Tooltips explaining features
  - Demo data
  - Progressive disclosure
- It matters because onboarding affects adoption more than functionality.

Mobile Responsiveness

- Mobile UX has different constraints: touch input, limited screen, bandwidth, battery.
- Engineering considerations
  - Adaptive layout
  - Larger tap targets
  - Avoiding hover interactions
  - Compressing images
  - Lazy-loading heavy features
  - Offline + intermittent connectivity handling
- Real-world example
  - Google Maps adjusts layout, gestures, and information density based on screen size.

Emotional Design & Engagement

- Users respond emotionally to software—delight, trust, confidence.
- Key elements
  - Micro-interactions (subtle animations, haptics)
  - Friendly copywriting
  - Progress indicators
  - Positive reinforcement
  - Personality in the interface
- Engineering support
  - Smooth animations (60 FPS)
  - Quick state transitions
  - Stable interface even under network issues
  - Consistent, human-readable messages
- Emotion is not fluff — it increases retention and decreases frustration.

How Does It All Fit Together

- Core ideas
  - UX is not visual design — it is how the product behaves for real users.
  - Most UX problems are engineering problems: poor defaults, unnecessary complexity, unclear state, or slow responses.
  - Good UX blends:
  - Usability and simplicity
  - Predictability and flexibility
  - Performance and accessibility
  - Emotional engagement and learnability
- Outcomes
  - Faster adoption
  - Fewer support issues
  - Happier users
  - More effective software

Final Takeaways for Engineers

- UX is a shared responsibility, not a design-only concern.
- Think in terms of user tasks, not screens.
- Reduce friction relentlessly.
- Make the software feel simple, stable, and responsive.
- All UX principles reinforce one another:
  - Usability + consistency → predictable interactions
  - Performance + feedback → smooth experience
  - Simplicity + contextual design → less cognitive load
  - Accessibility + control → inclusive and trustworthy systems
  - Emotional design + efficiency → a product people enjoy using
- UX/UE is fundamentally about making the product work for users under real constraints — not just designing screens.