Introduction to Designing for Error Prevention and Recovery

- Error handling is a core design skill, not an afterthought
- Preventable errors cause abandonment, support tickets, and eroded trust
- Two pillars: prevention stops errors; recovery helps users get back on track
- Focus: predictable user mistakes in forms, workflows, and transactional interfaces
- 67.9% form abandonment rate; 27% from validation friction; 35.2% of sessions impacted by frustration

The Business Case: Why Error Handling Drives Revenue and Retention

- 88% of users less likely to return after a poor UX experience
- 70% of online businesses fail due to bad usability, not competition
- Conversion drops from 23% at 3 fields to 11.4% at 7 fields
- Each removed field reduces form abandonment by 7-11%
- Preventable errors directly generate support tickets and erode trust
- Error handling quality directly affects NPS and customer lifetime value
- UX investment yields 141-379% ROI over three years

Understanding the Error-Prone Moment: Psychology and Triggers

- Slips (automatic actions gone wrong) vs. mistakes (conscious wrong decisions)
- Common triggers: high cognitive load, interruptions, unfamiliar patterns
- Frequent errors: format mismatches, missing fields, duplicate submissions
- Map error-prone moments in forms, checkouts, and transactional interfaces

Prevention Pillar: Designing Errors Out Before They Happen

- Invest in prevention: the best error message is never seen
- Constrain inputs so the right action is obvious, the wrong one impossible
- Smart defaults, autofill, and pre-fill reduce typing and format errors
- Accept multiple formats; normalize on the server, guide with masks
- Reveal complexity only when needed; confirm dialogs as a last resort

Input Formatting and Field Design: Prevention at the Micro-Level

- Masks guide, not control; paste must always work
- Auto-format phone numbers, dates, credit cards, and IBANs
- Normalize while typing, format on blur
- Use correct mobile keyboards and auto-advance for short fields
- Document display vs. stored format to eliminate handoff friction
- 72% of sites format expiry dates differently from physical cards—a simple fix

FIELD GUIDE ENTRY

INPUT MASKING

CORRECT MASK BEHAVIOR

Card number

4242 4242 4242 4242

Guides entry with spaces for readability

PASTE MUST ALWAYS WORK

Card number

4242424242424242

Pasted values are accepted and formatted automatically

MISMATCHED PHYSICAL CARD FORMAT

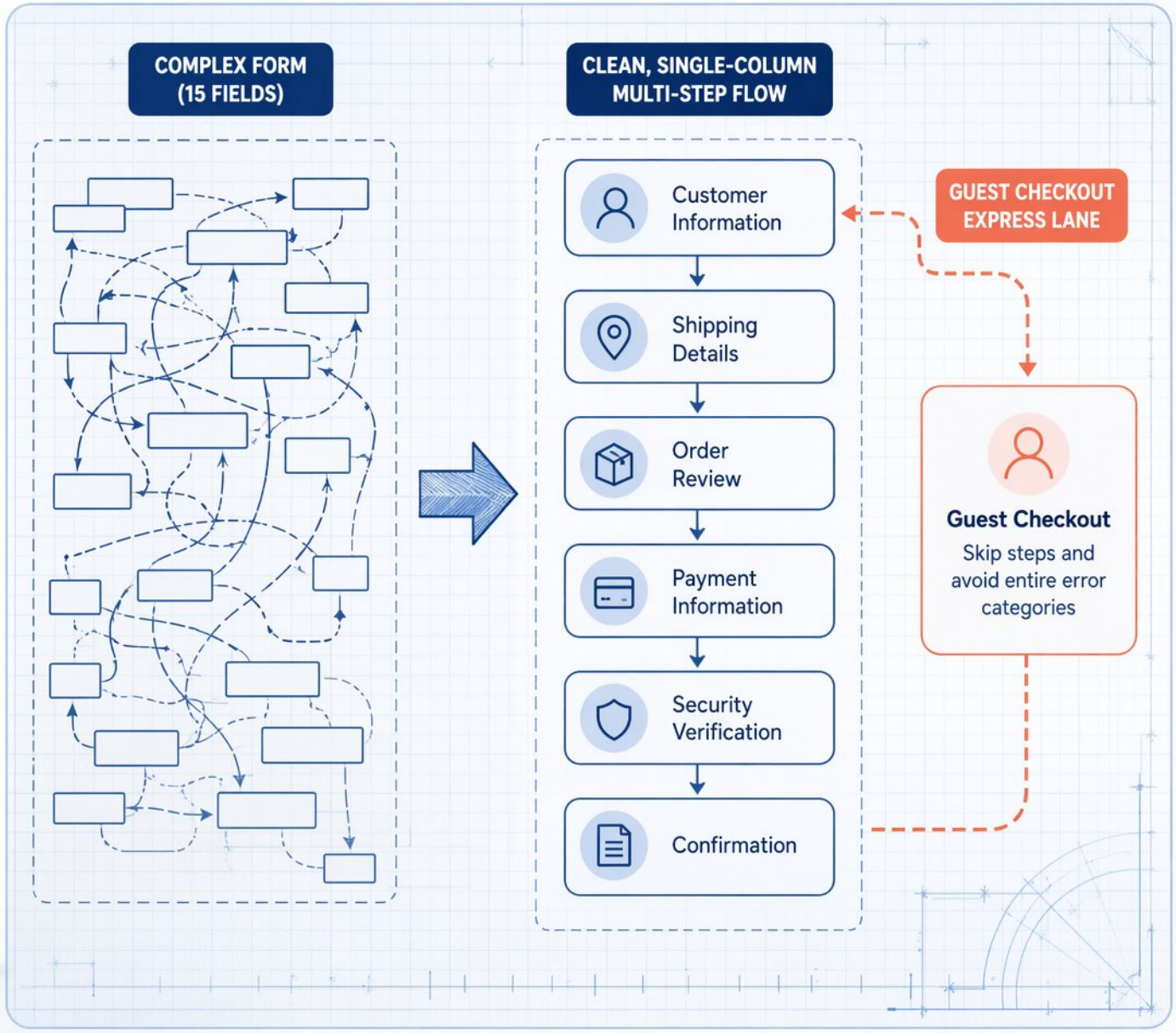
Physical card shows:

Physical cards often show MM/YY, but many sites expect different formats—causing user friction

Align formats with what users see on their cards

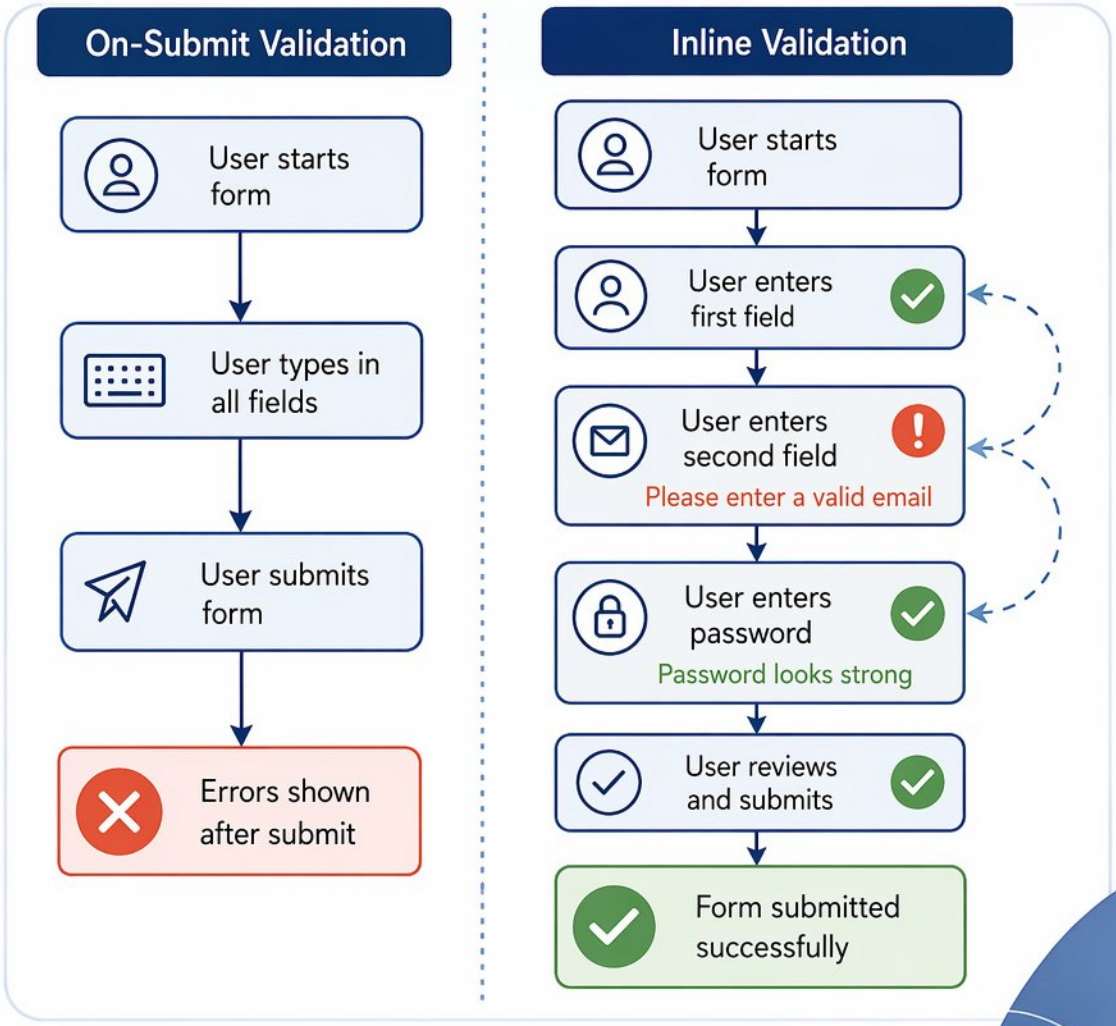
Prevention Through Architecture: Forms, Checkouts, and Multi-Step Flows

- Reduce field count: aim for 6–8 fields; every field above 8 costs ~2.8% conversion
- Single-column layout, logical grouping, and multi-step flows for complex forms
- Guest checkout and express wallets (Apple Pay, Google Pay) skip entire error categories
- Use HTML autocomplete attributes on every field — the most powerful prevention tool
- Show fields conditionally to reduce cognitive load and error opportunities
- Display costs early and explain why sensitive data is needed



Validation and Feedback Patterns: Timing and Technique

- Validate on blur, not on every keystroke — wait for the user to finish typing
- Inline validation lifts completion rates by ~22% over on-submit only
- Use password strength indicators, character counters, and format confirmations
- Client-side validation helps, but the server is the final authority
- Never wipe user input on error — user input is sacred



Recovery Pillar: Helping Users After an Error Occurs

- The four-step recovery pattern: detect, explain, resolve, reassure
- The Avoid, Explain, Resolve framework: say what to check, what happened, what to do
- Diagnose the problem, don't blame the user: be specific, actionable, constructive
- Three-part error structure: what went wrong, why it matters, how to fix it
- Undo, redo, and version history let users reverse mistakes without consequence

Writing Error Messages That Users Actually Read and Understand

VAGUE & UNHELPFUL

Email

name@example

Invalid Input

Doesn't explain the problem.
Doesn't help the user fix it.

VS.

CLEAR & HELPFUL

Email

name@example

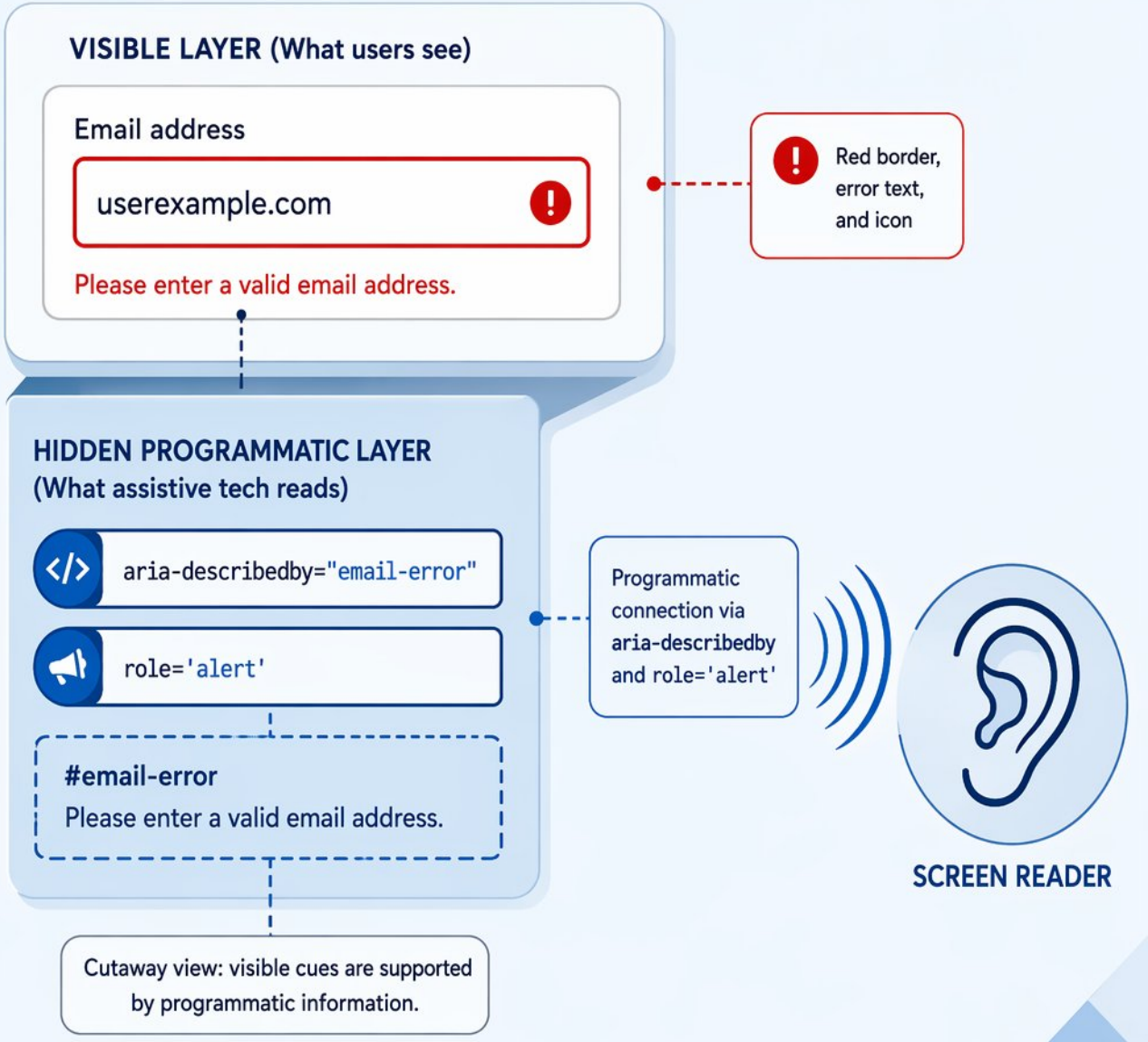
Enter a valid email, like name@example.com

Explains the issue and shows exactly what to do next.

- "Enter a valid email, like name@example.com" beats "Invalid input"
- Always pair the problem with a concrete action to fix it
- Be helpful, clear, and direct — not condescending or cute
- Avoid blame words: never say "illegal," "invalid," or "failed"
- Apologize for system errors, not user errors
- Place inline next to the field, not in a generic top banner

Accessibility and Error Handling: Designing for All Users

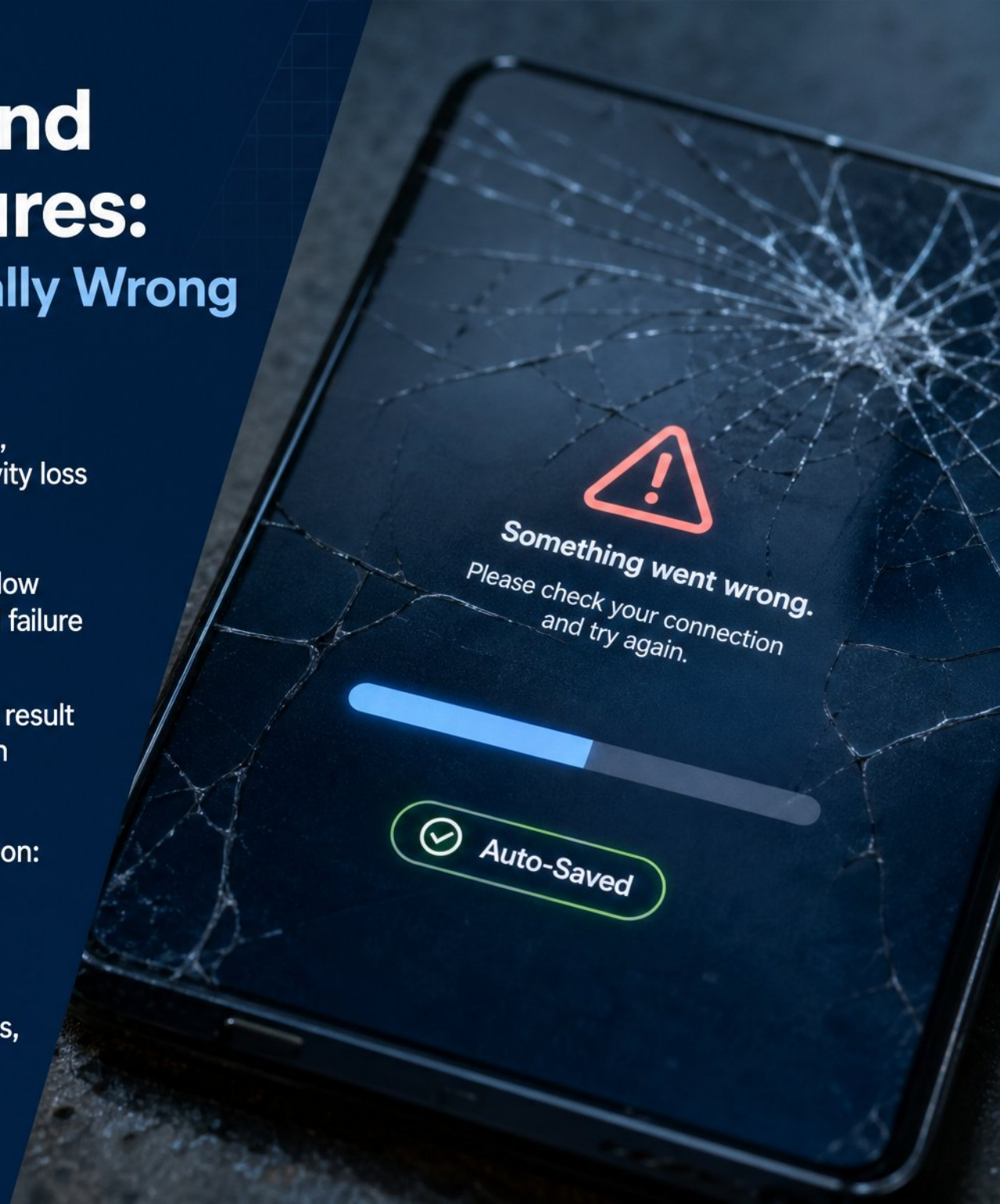
- WCAG 2.2 requires text error IDs, suggestions, and status announcements (SC 3.3.1, 3.3.3, 4.1.3)
- Programmatic association: use aria-describedby, aria-invalid, and role="alert" for screen readers
- Move focus to the first invalid field or error summary after failed submission
- Never rely on color alone; pair with text, icons, and programmatic cues
- Error summary pattern: top-level list with anchor links plus inline field errors
- Mobile: minimum 44x44px touch targets, correct input types, and working autofill



Edge Cases and Graceful Failures:

When Things Go Really Wrong

- Anticipate empty states, timeouts, and connectivity loss
- Auto-save drafts and allow resumption after partial failure
- Use optimistic UI: show result immediately, rollback on server rejection
- Double-submit prevention: disable buttons during pending state
- Network errors: clear messaging, retry options, offline support



Error States as Design System Components

- Build shared error pattern libraries: error summary, inline, toast, empty states
- Classify by severity: warning, blocking error, system failure—different UI patterns
- Use templates for consistent tone and structure across all error message types
- Document fields: display format, validation timing, error behavior, accessibility specs
- Integrate error states into design reviews as a first-class design concern

Measuring Error Prevention and Recovery: Metrics That Matter

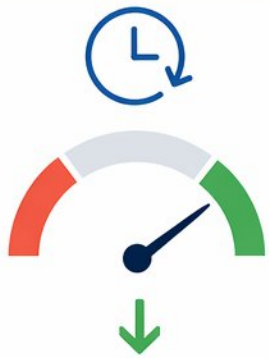
- Average error rate: 0.7 errors per task, with 2 in 3 users making at least one error
- 67.9% form abandonment rate; 27% of abandonment caused by validation friction alone
- Inline validation lifts completion rates by ~22% over on-submit-only validation
- Track: error rate per field, recovery time, form abandonment rate, dedupe hit rate
- The Single Usability Metric (SUM) combines completion, time, satisfaction, and errors into one score

Single Usability Metric (SUM)

Combines completion, time, satisfaction, and errors into one score

Error Rate per Field

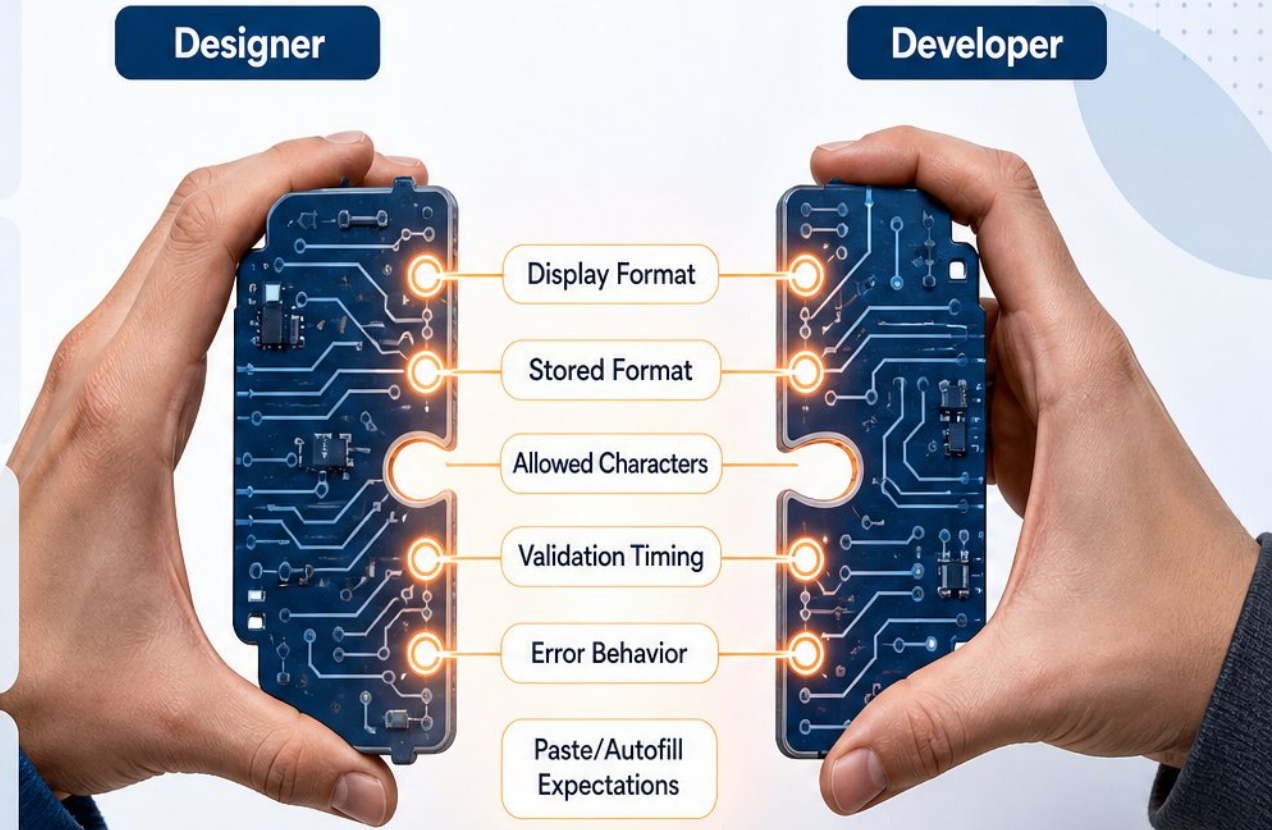
Recovery Time



Abandonment Rate

Collaboration: Working with Developers on Error Strategy

- Define the client-server validation contract: client helps, server is the authority; map field errors predictably
- **The six-point handoff** for every input: display format, stored format, allowed characters, validation timing, error behavior, paste/autofill expectations
- **Error message governance:** shared content repository, version control, localization workflows, and review processes
- **Collaborate on optimistic UI:** apply it for low-stakes toggles; avoid it for payments, irreversible side-effects, and branching wizards
- **Integrate error prevention into sprints:** make error states part of the definition of done



Testing and Iterating on Error Handling

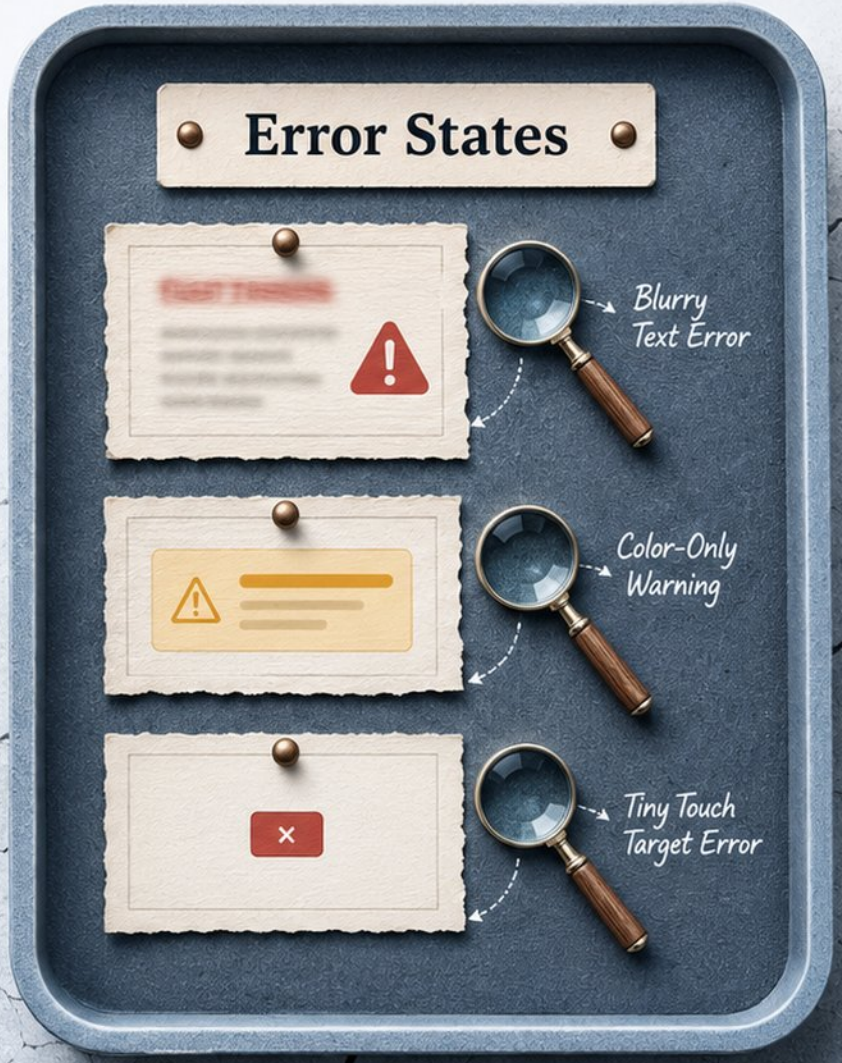
- **Error-focused usability testing:** observe comprehension, recovery time, and emotional response

- **Prototype the unhappy path:** include error messages in clickable prototypes

- **Accessibility testing:** keyboard-only, screen reader, and high-contrast passes

- **A/B test error copy:** compare headlines and bodies to reduce time-to-recover

- **Session replay and analytics:** identify where users slow down, abandon, or trigger repeated errors



Building an Error-Prevention Mindset on Your Team

- Make error prevention a standard design review question: 'What could go wrong here?'
- Create a shared error severity taxonomy: warning, validation, workflow interruption, system failure
- Build a centralized error message library with templates for common error types
- Run error-focused design critiques and pre-mortems to anticipate failures before shipping
- Shift team culture: errors are a primary interface, not edge cases

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/aa1569f3/public