Technical Specification Document

Addressing Critical Design Tool Pain Points

Consumer Requirements & Implementation Strategy

Product Engineering Specification

January 2026
Version 1.0

Abstract

This specification document provides a comprehensive analysis of the eight most critical pain points identified through extensive user research across design communities, forums, and social platforms. For each issue, we document the specific user complaints, articulate what consumers actually want, and propose detailed implementation strategies. The goal is to create a design tool experience that addresses these fundamental gaps while maintaining stability and professional-grade performance.

Contents

1 Executive Summary

1.1 Document Purpose

This specification serves as the authoritative reference for addressing critical user experience gaps in modern design tooling. Through analysis of thousands of user complaints, forum discussions, and feature requests, we have identified eight fundamental areas requiring immediate attention.

1.2 Identified Pain Points

#	Pain Point	Severity	Priority
1	UI/UX Redesign Backlash	Critical	P0
2	Performance Degradation with Large Files	Critical	P0
3	Pricing Model Frustrations	High	P1
4	AI Features Perceived as Gimmicky	Medium	P2
5	Lack of Proper Offline Mode	Critical	P0
6	No Native Linux Desktop Application	High	P1
7	Inadequate Version Control / Branching	High	P1
8	Annotation Visibility & Collaboration Gaps	Medium	P1

Table 1: Pain Point Severity and Priority Matrix

1.3 Design Philosophy

Our implementation strategy adheres to the following core principles:

1. **User Agency**: Provide options, not mandates. Users choose their experience.
2. **Progressive Enhancement**: New features enhance, never diminish, existing workflows.
3. **Performance First**: Every feature must pass performance benchmarks before release.
4. **Professional Focus**: Prioritize power users who drive adoption within organizations.
5. **Stability Guarantee**: Rock-solid reliability is non-negotiable.

2 Issue #1: UI/UX Redesign Backlash

2.1 Problem Statement

Current State

The UI3 redesign has generated significant negative response from the professional design community. Core complaints include:

- Essential tools relocated to hidden menus requiring additional clicks
- Toolbar moved from top (established convention) to bottom of screen
- Nested component instances collapsed by default, breaking inspection workflows
- Floating panels overlap canvas content, obscuring work
- Interface perceived as "dumbed down" for non-professionals
- No option to revert to previous UI version

User Voice

"You got rid of one of the most intuitive designs and replaced it with an awful version without giving users an option to switch back."
"The interface is so horrifyingly oversimplified that it makes Microsoft Paint look like feature-rich professional software. Essential features are buried so deep within cryptic menus that finding a tool feels like trying to find the minotaur at the end of a labyrinth."
"Nested instances collapsed by default is the most frustrating one for me. I have to manually open every accordion for every instance I need to adjust. It's a huge time-waster."

2.2 What Consumers Want

User Requirements

1. **UI Version Toggle**: Ability to switch between UI versions (classic/modern)
2. **Toolbar Position Choice**: Option to place toolbar at top, bottom, or side
3. **Panel Behavior Control**: Configure whether panels float or dock
4. **Instance Expansion Memory**: Remember expansion state per component
5. **Quick Access Customization**: User-defined toolbar with frequently used tools
6. **Gradual Transition**: Phased rollout with opt-in, not forced migration

2.3 Implementation Plan

2.3.1 Phase 1: UI Preference System (Weeks 1–3)

```typescript
// types/ui-preferences.ts

export type ToolbarPosition = 'top' | 'bottom' | 'left' | 'right';
export type PanelBehavior = 'floating' | 'docked' | 'auto';
export type UITheme = 'classic' | 'modern' | 'custom';

export interface UIPreferences {
  readonly theme: UITheme;
  readonly toolbarPosition: ToolbarPosition;
  readonly panelBehavior: PanelBehavior;
  readonly instanceExpansion: 'collapsed' | 'expanded' | 'remember';
  readonly quickAccessTools: readonly string[];
  readonly showAdvancedOptions: boolean;
  readonly compactMode: boolean;
  readonly keyboardShortcutProfile: 'default' | 'vim' | 'custom';
}

export interface UIPreferenceStore {
  preferences: UIPreferences;
  setPreference<K extends keyof UIPreferences>(
    key: K,
    value: UIPreferences[K]
  ): void;
  resetToDefaults(): void;
  exportPreferences(): string;
  importPreferences(json: string): void;
}
```

2.3.2 Phase 2: Customizable Toolbar (Weeks 4–6)

```tsx
// components/CustomizableToolbar.tsx

interface ToolbarConfig {
  readonly position: ToolbarPosition;
  readonly items: readonly ToolbarItem[];
  readonly showLabels: boolean;
  readonly iconSize: 'small' | 'medium' | 'large';
  readonly grouping: 'flat' | 'grouped' | 'dropdown';
}

interface ToolbarItem {
  readonly id: string;
  readonly icon: string;
  readonly label: string;
  readonly shortcut?: string;
  readonly action: () => void;
  readonly isVisible: boolean;
  readonly order: number;
}

export const CustomizableToolbar: React.FC<{
  config: ToolbarConfig;
  onConfigChange: (config: ToolbarConfig) => void;
}> = ({ config, onConfigChange }) => {
  // Drag-and-drop reordering
  // Right-click to add/remove items
  // Position snapping to screen edges
};
```

2.3.3 Phase 3: Classic UI Mode (Weeks 7–10)

Technical Approach

1. Create abstraction layer for UI component rendering
2. Implement "Classic" theme with UI2-equivalent layouts
3. Ensure feature parity between themes (no functionality loss)
4. Store preference in user profile for cross-device sync
5. Provide migration assistant for users transitioning between modes

3 Issue #2: Performance Degradation

3.1 Problem Statement

Current State

Users report severe performance degradation when working with:

- Large files with many pages and components
- Complex nested component hierarchies
- Design systems with extensive variant libraries
- Files with numerous prototyping connections
- Real-time collaboration with multiple editors

Symptoms include: typing lag, canvas pan/zoom stuttering, multi-second delays when switching pages, library update freezes, and browser/app crashes.

User Voice

"I'm still having serious performance issues, especially when a file becomes larger with complex components. The main thing that can't be handled well right now is Components inside Components."
"It's very laggy when doing anything—typing, moving elements, double-clicking layers. It's slowing my production and lags during stakeholder presentations which is causing issues with my clients."
"Excruciatingly slow, and the number of times I have devs saying they can't open it or it just gets stuck loading is crazy."

3.2 What Consumers Want

User Requirements

1. **Consistent 60fps**: Smooth canvas interaction regardless of file complexity
2. **Instant Page Switching**: Sub-100ms page navigation
3. **Background Processing**: Library updates without blocking UI
4. **Predictable Memory Usage**: Clear limits with graceful degradation
5. **Performance Monitoring**: Real-time metrics for optimization
6. **Large File Support**: Handle 500+ page files without degradation

3.3 Implementation Plan

3.3.1 Architecture: GPU-Accelerated Rendering Pipeline

```
// rendering/GPURenderer.ts

export class GPUCanvasRenderer {
  private device: GPUDevice;
  private context: GPUCanvasContext;
  private renderPipeline: GPURenderPipeline;
  private layerCache: Map<string, GPUTexture> = new Map();

  // Hybrid rendering: GPU for canvas, CPU for UI
  private readonly BATCH_SIZE = 1000; // Elements per draw call
  private readonly CACHE_THRESHOLD = 100; // Cache layers with >100
      children

  async initialize(canvas: HTMLCanvasElement): Promise<void> {
    const adapter = await navigator.gpu.requestAdapter({
      powerPreference: 'high-performance'
    });

    this.device = await adapter!.requestDevice({
      requiredFeatures: ['texture-compression-bc'],
      requiredLimits: {
        maxTextureDimension2D: 16384,
        maxBufferSize: 256 * 1024 * 1024, // 256MB
      }
    });

    this.context = canvas.getContext('webgpu')!;
    this.setupRenderPipeline();
  }

  // Frustum culling: only render visible elements
  private cullInvisibleElements(
    elements: readonly CanvasElement[],
    viewport: Viewport
  ): CanvasElement[] {
    return elements.filter(el =>
      this.intersectsViewport(el.bounds, viewport)
    );
  }

  // Level-of-detail: simplify distant elements
  private applyLOD(
    element: CanvasElement,
    zoomLevel: number
  ): RenderableElement {
    if (zoomLevel < 0.1) {
      return this.createBoundingBoxProxy(element);
    }
    if (zoomLevel < 0.25) {
      return this.createSimplifiedProxy(element);
    }
    return element;
  }

  // Incremental rendering for large operations
  async renderLargeFile(
    file: DesignFile,
    onProgress: (percent: number) => void
  ): Promise<void> {
    const totalElements = file.getAllElements().length;
    let rendered = 0;

    for (const page of file.pages) {
      for (const chunk of this.chunkElements(page.elements, this.
          BATCH_SIZE)) {
        await this.renderBatch(chunk);
        rendered += chunk.length;
        onProgress((rendered / totalElements) * 100);

        // Yield to main thread to keep UI responsive
        await new Promise(r => requestAnimationFrame(r));
      }
    }
  }
}
```

3.3.2 Virtual Scrolling for Layer Panel

```
// components/VirtualLayerTree.tsx

interface VirtualTreeProps {
  nodes: readonly LayerNode[];
  estimatedNodeHeight: number;
  overscanCount: number;
  onNodeExpand: (nodeId: string) => void;
}

export const VirtualLayerTree: React.FC<VirtualTreeProps> = ({
  nodes,
  estimatedNodeHeight = 32,
  overscanCount = 5,
}) => {
  const flattenedNodes = useMemo(() =>
    flattenTree(nodes), [nodes]
  );

  const rowVirtualizer = useVirtualizer({
    count: flattenedNodes.length,
    getScrollElement: () => containerRef.current,
    estimateSize: () => estimatedNodeHeight,
    overscan: overscanCount,
  });

  return (
    <div ref={containerRef} className="layer-tree-container">
      <div style={{ height: rowVirtualizer.getTotalSize() }}>
        {rowVirtualizer.getVirtualItems().map(virtualRow => (
          <LayerNodeRow
            key={flattenedNodes[virtualRow.index].id}
            node={flattenedNodes[virtualRow.index]}
            style={{
              position: 'absolute',
              top: virtualRow.start,
              height: virtualRow.size,
            }}
          />
        ))}
      </div>
    </div>
  );
};
```

3.3.3 Background Processing with Web Workers

```
// workers/LibraryUpdateWorker.ts

// Runs in dedicated Web Worker thread
self.onmessage = async (event: MessageEvent<LibraryUpdateMessage>) =>
    {
  const { libraryId, components, currentInstances } = event.data;

  // Heavy computation happens off main thread
  const updates = await computeLibraryDiff(components,
      currentInstances);
  const patches = generateMinimalPatches(updates);

  // Stream results back incrementally
  for (const patch of patches) {
    self.postMessage({
      type: 'PATCH_READY',
      patch,
      progress: patches.indexOf(patch) / patches.length
    });
  }

  self.postMessage({ type: 'COMPLETE' });
};

// Main thread usage
const libraryWorker = new Worker(
  new URL('./workers/LibraryUpdateWorker.ts', import.meta.url)
);

libraryWorker.onmessage = (event) => {
  if (event.data.type === 'PATCH_READY') {
    // Apply patches incrementally without blocking UI
    requestIdleCallback(() => applyPatch(event.data.patch));
  }
};
```

3.3.4 Performance Metrics Dashboard

```
// services/PerformanceMonitor.ts

export interface PerformanceMetrics {
  fps: number;
  frameTime: number;
  memoryUsage: number;
  memoryLimit: number;
  layerCount: number;
  componentInstanceCount: number;
  renderTime: number;
  networkLatency: number;
}

export class PerformanceMonitor {
  private metrics: PerformanceMetrics;
  private observers: Set<(metrics: PerformanceMetrics) => void> = new
      Set();

  startMonitoring(): void {
    // Frame timing
    let lastFrameTime = performance.now();
    const measureFrame = () => {
      const now = performance.now();
      const delta = now - lastFrameTime;
      this.metrics.frameTime = delta;
      this.metrics.fps = Math.round(1000 / delta);
      lastFrameTime = now;
      requestAnimationFrame(measureFrame);
    };
    requestAnimationFrame(measureFrame);

    // Memory monitoring
    if ('memory' in performance) {
      setInterval(() => {
        const memory = (performance as any).memory;
        this.metrics.memoryUsage = memory.usedJSHeapSize;
        this.metrics.memoryLimit = memory.jsHeapSizeLimit;
        this.notifyObservers();
      }, 1000);
    }
  }

  // Display warning when approaching limits
  checkThresholds(): PerformanceWarning[] {
    const warnings: PerformanceWarning[] = [];

    if (this.metrics.fps < 30) {
      warnings.push({
        level: 'critical',
        message: 'Frame rate below 30fps',
        suggestion: 'Consider splitting file or reducing component
            complexity'
      });
    }

    if (this.metrics.memoryUsage > this.metrics.memoryLimit * 0.8) {
      warnings.push({
        level: 'warning',
        message: 'Memory usage at 80%',
        suggestion: 'Close unused pages or reduce image resolution'
      });
    }

    return warnings;
  }
}
```

4 Issue #3: Pricing Model Frustrations

4.1 Problem Statement

Current State

Pricing concerns center on:

- Freelancers needing additional seats for each client engagement
- Dev Mode becoming a paid feature, multiplying costs for dev-heavy teams
- Confusing seat management leading to unexpected charges
- Branching limited to expensive Organization/Enterprise tiers
- View-only users still requiring licenses for certain features
- Price increases affecting small teams disproportionately

User Voice

"For freelancers the use can quickly become very costly, as freelancers need to book additional seats for each client they work with."
"Dev Mode becoming a paid feature racks up the price for teams, since teams usually have

more developers than designers."
"We get constantly overcharged. I switch seats, get charged. I remove my seat, then the teammate can't access anything."

4.2 What Consumers Want

User Requirements

1. **Freelancer-Friendly Model**: Client collaboration without per-client seats
2. **Developer Access Tiers**: Read-only dev mode at lower cost
3. **Transparent Billing**: Clear invoices, no surprise charges
4. **Feature Unbundling**: Pay for features used, not bundled tiers
5. **Branching for All**: Core version control at all paid tiers
6. **Viewer Annotations**: View annotations without full license

4.3 Proposed Pricing Structure

Tier	Features	Price
Free	3 files, unlimited viewers, basic prototyping, community access	$0
Maker	Unlimited files, branching, offline mode, 10 client guests	$12/mo
Team	Everything in Maker + team libraries, advanced prototyping	$18/seat/mo
Dev Access	Inspect mode, code export, specs (no edit)	$8/seat/mo
Business	SSO, advanced permissions, analytics, priority support	$30/seat/mo

Table 2: Proposed User-Centric Pricing Tiers

4.3.1 Freelancer Guest System

```
// models/ClientGuest.ts

export interface ClientGuest {
  readonly id: string;
  readonly email: string;
  readonly name: string;
  readonly projectAccess: readonly ProjectAccess[];
  readonly expiresAt: Date | null;
  readonly permissions: GuestPermissions;
}

export interface GuestPermissions {
  readonly canView: boolean;
  readonly canComment: boolean;
  readonly canInspect: boolean;
  readonly canExport: boolean;
  readonly canPrototype: boolean;
  // Explicitly NO edit permissions
  readonly canEdit: false;
}

export interface ProjectAccess {
  readonly projectId: string;
  readonly grantedAt: Date;
  readonly grantedBy: string;
}

// Freelancer can invite up to N clients without additional seats
export const MAKER_TIER_GUEST_LIMIT = 10;
export const TEAM_TIER_GUEST_LIMIT = 25;
```

5 Issue #4: AI Features Perceived as Gimmicky

5.1 Problem Statement

Current State

AI features are seen as:

- Focused on trendy content generation rather than productivity
- Unreliable in following instructions or respecting selections
- Prioritized over fixing core UX issues
- Aimed at non-designers rather than professionals
- Lacking integration with actual design workflows

User Voice

“AI features are gimmicky and useless in their current state. It’s clear that the application is trying to appeal to non-designers.”
“As soon as I use the ’make a change’ menu, it’s really bad. Most of the time it doesn’t stick to the selection I gave and changes something else.”
“90% of their efforts seem to be spent on something that’s ’cool’ but not necessarily useful. The auto layer renaming feature was a highlight, but it’s not enough.”

5.2 What Consumers Want

User Requirements

1. **Workflow Automation**: AI that accelerates repetitive tasks
2. **Intelligent Layer Management**: Auto-naming, organization, cleanup
3. **Design System Assistance**: Component suggestions, consistency checks
4. **Context Awareness**: AI understands selection and respects boundaries
5. **Developer Handoff**: Auto-generate specs, documentation, code snippets
6. **Optional Integration**: AI features opt-in, not forced into UI

5.3 Implementation Plan: Productivity-First AI

```
// ai/ProductivityAI.ts

export interface AICapability {
  readonly id: string;
  readonly name: string;
  readonly description: string;
  readonly category: 'organization' | 'generation' | 'analysis' | '
     handoff';
  readonly requiresSelection: boolean;
  readonly isDestructive: boolean;
}

export const PRODUCTIVITY_AI_CAPABILITIES: AICapability[] = [
  {
    id: 'auto-name-layers',
    name: 'Smart Layer Naming',
    description: 'Rename layers based on content and hierarchy',
    category: 'organization',
    requiresSelection: false,
    isDestructive: false,
  },
  {
    id: 'component-suggestions',
    name: 'Component Suggestions',
    description: 'Identify repeated patterns that could be components
        ',
    category: 'analysis',
    requiresSelection: false,
    isDestructive: false,
  },
  {
    id: 'accessibility-audit',
    name: 'Accessibility Audit',
    description: 'Check contrast, touch targets, and WCAG compliance'
        ,
    category: 'analysis',
    requiresSelection: true,
    isDestructive: false,
  },
  {
    id: 'generate-specs',
    name: 'Generate Dev Specs',
    description: 'Create technical specifications for selected
        elements',
    category: 'handoff',
    requiresSelection: true,
    isDestructive: false,
  },
  {
    id: 'cleanup-layers',
    name: 'Layer Cleanup',
    description: 'Remove hidden layers, flatten groups, optimize
        structure',
    category: 'organization',
    requiresSelection: true,
    isDestructive: true,
  },
];

export class ProductivityAIService {
  // AI strictly respects selection boundaries
  async executeOnSelection(
    capability: AICapability,
    selection: readonly SceneNode[]
  ): Promise<AIResult> {
    if (selection.length === 0 && capability.requiresSelection) {
      return {
        success: false,
        error: 'This action requires a selection'
      };
    }

    // Preview changes before applying
    const preview = await this.generatePreview(capability, selection)
       ;

    return {
      success: true,
      preview,
      apply: () => this.applyChanges(preview),
      dismiss: () => this.discardPreview(preview),
    };
  }

  // Non-destructive by default
  private async generatePreview(
    capability: AICapability,
    selection: readonly SceneNode[]
  ): Promise<ChangePreview> {
    const changes = await this.computeChanges(capability, selection);

    return {
      changes,
      affectedNodes: changes.map(c => c.nodeId),
      // Visual diff for user review
      beforeAfterComparison: await this.renderComparison(changes),
    };
  }
}
```

6 Issue #5: Lack of Proper Offline Mode

6.1 Problem Statement

Current State

Offline functionality is severely limited:

- Cannot open existing files when starting offline
- Only current page available; other pages inaccessible
- No access to linked libraries or external components
- Desktop app is essentially a browser wrapper with no offline advantage
- Autosave is not a substitute for proper offline mode
- Users on airplanes, trains, or in areas with poor connectivity cannot work

User Voice

"I am frustrated that this application cannot be used offline! I made a request three years ago, and they mentioned plans to create an offline version."
"I wanted to work on my designs on an airplane without internet... impossible."
"I do a lot of my work while on my daily commute. Unfortunately, the signal is scarce

along most of the route."
"As a UI/UX Director, this remains the reason why I haven't made the jump and don't use it in my team."

6.2 What Consumers Want

User Requirements

1. **Full Offline Editing**: Work on complete files without connection
2. **Selective Sync**: Choose which files to make available offline
3. **Library Caching**: Offline access to linked design system components
4. **Conflict Resolution**: Intelligent merge when reconnecting
5. **Offline Prototyping**: Test prototypes without network
6. **Desktop Advantage**: Native app should offer genuine offline capability

6.3 Implementation Plan: Hybrid Offline Architecture

6.3.1 Local Storage Layer

```typescript
// offline/LocalFileStore.ts

export class LocalFileStore {
  private db: IDBDatabase;
  private readonly DB_NAME = 'DesignToolOffline';
  private readonly STORES = {
    files: 'files',
    pages: 'pages',
    components: 'components',
    assets: 'assets',
    syncQueue: 'syncQueue'
  };

  async initialize(): Promise<void> {
    this.db = await this.openDatabase();
    await this.setupServiceWorker();
  }

  // Mark file for offline availability
  async makeAvailableOffline(fileId: string): Promise<OfflineStatus>
     {
    const file = await this.fetchFileWithDependencies(fileId);

    // Store file structure
    await this.storeFile(file);

    // Cache all pages
    for (const page of file.pages) {
      await this.storePage(page);
    }

    // Cache linked library components
    for (const libraryRef of file.linkedLibraries) {
      await this.cacheLibraryComponents(libraryRef);
    }

    // Cache image assets
    await this.cacheAssets(file.referencedAssets);

    return {
      fileId,
      cachedAt: new Date(),
      size: await this.calculateCacheSize(fileId),
      pagesCount: file.pages.length,
      componentsCount: file.components.length,
    };
  }

  // Track changes made offline
  async recordOfflineChange(change: DesignChange): Promise<void> {
    const queueEntry: SyncQueueEntry = {
      id: crypto.randomUUID(),
      change,
      timestamp: Date.now(),
      fileId: change.fileId,
      status: 'pending'
    };

    await this.addToSyncQueue(queueEntry);
  }

  // Get offline-available files
  async getOfflineFiles(): Promise<OfflineFileInfo[]> {
    const files = await this.getAllFromStore<CachedFile>(this.STORES.
      files);
    return files.map(f => ({
      id: f.id,
      name: f.name,
      cachedAt: f.cachedAt,
      size: f.size,
      pendingChanges: f.pendingChangesCount,
      lastModified: f.lastModified,
    }));
  }
}
```

6.3.2 Sync Engine with CRDT Conflict Resolution

```
// offline/SyncEngine.ts

export class SyncEngine {
  private localStore: LocalFileStore;
  private conflictResolver: CRDTResolver;
  private connectionMonitor: ConnectionMonitor;

  constructor() {
    this.conflictResolver = new CRDTResolver();
    this.connectionMonitor = new ConnectionMonitor();

    // Auto-sync when connection restored
    this.connectionMonitor.onOnline(() => this.syncAll());
  }

  async syncAll(): Promise<SyncResult> {
    const pendingChanges = await this.localStore.getPendingChanges();
    const results: SyncItemResult[] = [];

    for (const change of pendingChanges) {
      try {
        const serverState = await this.fetchServerState(change.fileId);
        const localState = await this.localStore.getFile(change.fileId);

        if (this.hasConflict(localState, serverState, change)) {
          // Use CRDT to resolve automatically where possible
          const resolution = await this.conflictResolver.resolve(
            localState,
            serverState,
            change
          );

          if (resolution.requiresUserInput) {
            results.push({
              changeId: change.id,
              status: 'conflict',
              conflictDetails: resolution.conflicts,
            });
          } else {
            await this.applyResolution(resolution);
            results.push({ changeId: change.id, status: 'resolved' });
          }
        } else {
          // No conflict, push directly
          await this.pushChange(change);
          results.push({ changeId: change.id, status: 'synced' });
        }
      } catch (error) {
        results.push({
          changeId: change.id,
          status: 'error',
          error: error.message,
        });
      }
    }

    return { results, timestamp: new Date() };
  }

  // Visual conflict resolution UI
  async presentConflictResolution(
    conflicts: Conflict[]
  ): Promise<ConflictResolution[]> {
    return new Promise((resolve) => {
      // Show side-by-side comparison
      // User chooses: keep local, keep remote, or merge
      this.showConflictModal(conflicts, resolve);
    });
  }
```

```
}
```

6.3.3 Service Worker for Offline-First

```
// sw/offline-service-worker.ts

const CACHE_VERSION = 'v1';
const STATIC_CACHE = `static-${CACHE_VERSION}`;
const DYNAMIC_CACHE = `dynamic-${CACHE_VERSION}`;

self.addEventListener('fetch', (event: FetchEvent) => {
  const { request } = event;

  // API requests: try network, fall back to cached response
  if (request.url.includes('/api/')) {
    event.respondWith(
      fetch(request)
        .then(response => {
          // Cache successful responses
          const clone = response.clone();
          caches.open(DYNAMIC_CACHE).then(cache => {
            cache.put(request, clone);
          });
          return response;
        })
        .catch(() => {
          // Offline: return cached version
          return caches.match(request);
        })
    );
    return;
  }

  // Asset requests: cache-first strategy
  if (request.url.includes('/assets/')) {
    event.respondWith(
      caches.match(request).then(cached => {
        return cached || fetch(request).then(response => {
          const clone = response.clone();
          caches.open(STATIC_CACHE).then(cache => {
            cache.put(request, clone);
          });
          return response;
        });
      })
    );
    return;
  }
});

// Background sync for offline changes
self.addEventListener('sync', (event: SyncEvent) => {
  if (event.tag === 'sync-changes') {
    event.waitUntil(syncPendingChanges());
  }
});
```

7 Issue #6: No Native Linux Desktop Application

7.1 Problem Statement

Current State

Linux users face:

- No official desktop application for Linux
- Unofficial community builds are unstable and lack features
- Browser-only access limits integration with Linux workflows
- Missing system-level features (file associations, notifications)
- No integration with Linux desktop environments (GNOME, KDE)

User Voice

"There are some unofficial desktop versions but they are 'BUGGY'. I request you guys to release an official desktop app for Linux."

7.2 What Consumers Want

User Requirements

1. **Official Linux App**: First-class support, not community workaround
2. **Distribution Packages**: .deb, .rpm, Flatpak, Snap, AppImage
3. **Desktop Integration**: File associations, system notifications, tray icon
4. **Performance Parity**: Same performance as macOS/Windows builds
5. **GPU Acceleration**: Full hardware acceleration on Linux
6. **Wayland Support**: Native Wayland rendering, not XWayland

7.3 Implementation Plan: Cross-Platform Native Application

7.3.1 Technology Stack

Recommended Approach: Tauri + WebView

- **Framework**: Tauri 2.0 with system WebView (WebKitGTK on Linux)
- **Backend**: Rust for native performance and system integration
- **Frontend**: Existing web codebase with platform-specific enhancements
- **Rendering**: WebGPU/WebGL with GPU acceleration
- **Distribution**: Flatpak primary, with .deb/.rpm/AppImage alternatives

```rust
// src-tauri/src/main.rs

use tauri::{Manager, SystemTray, SystemTrayEvent};
use tauri_plugin_store::StoreBuilder;

fn main() {
    tauri::Builder::default()
        // Offline file storage
        .plugin(tauri_plugin_store::Builder::default().build())
        // System tray integration
        .system_tray(create_system_tray())
        .on_system_tray_event(handle_tray_event)
        // File associations
        .plugin(tauri_plugin_deep_link::init())
        // GPU acceleration
        .invoke_handler(tauri::generate_handler![
            open_file,
            save_file,
            check_for_updates,
            get_system_info,
        ])
        // Linux-specific: handle DBus for notifications
        #[cfg(target_os = "linux")]
        .plugin(tauri_plugin_notification::init())
        .run(tauri::generate_context!())
        .expect("error while running application");
}

#[tauri::command]
async fn get_system_info() -> Result<SystemInfo, String> {
    Ok(SystemInfo {
        os: std::env::consts::OS.to_string(),
        gpu: detect_gpu()?,
        display_server: detect_display_server(), // X11 or Wayland
        desktop_environment: detect_de(),
    })
}

fn detect_display_server() -> String {
    std::env::var("XDG_SESSION_TYPE")
        .unwrap_or_else(|_| "x11".to_string())
}
```

7.3.2 Flatpak Manifest

```
# com.designtool.DesignApp.yaml
app-id: com.designtool.DesignApp
runtime: org.gnome.Platform
runtime-version: '45'
sdk: org.gnome.Sdk
command: design-app

finish-args:
  - --share=ipc
  - --share=network
  - --socket=fallback-x11
  - --socket=wayland
  - --device=dri  # GPU access
  - --filesystem=home
  - --talk-name=org.freedesktop.Notifications

modules:
  - name: design-app
    buildsystem: simple
    build-commands:
      - install -Dm755 design-app /app/bin/design-app
      - install -Dm644 design-app.desktop /app/share/applications/
      - install -Dm644 design-app.svg /app/share/icons/hicolor/
        scalable/apps/
    sources:
      - type: archive
        url: https://releases.designtool.com/linux/latest.tar.gz
```

8 Issue #7: Inadequate Version Control / Branching

8.1 Problem Statement

Current State

Version control limitations include:

- Branching only available on expensive Organization/Enterprise plans
- Cannot create branches of branches (no feature branch workflow)
- No way to lock main file to enforce branch-only editing
- Visual diff tool limited; hard to identify specific changes
- No integration with Git or other VCS tools
- Version history lacks semantic versioning support

User Voice

"The branching feature should be available on the Professional plan. The price difference between professional and org plans is just too big."
"Allow branching of branches to allow for feature branches from the development branch. This would be closer to what we currently have in Git."
"I wish I could lock the main 'prod' file so that all editors must use branches—keeping autosaves of other team members from affecting other branches."
"It's still difficult to VISUALLY identify changes for them to be highlighted."

8.2 What Consumers Want

User Requirements

1. **Branching for All Paid Tiers**: Core feature, not enterprise upsell
2. **Nested Branches**: Branch from branches (feature/develop/main workflow)
3. **Protected Branches**: Lock main, require reviews for merge
4. **Visual Diff**: Side-by-side and overlay comparison with change highlighting
5. **Git Integration**: Export/sync with Git repositories
6. **Semantic Versioning**: Major.minor.patch with changelogs

8.3 Implementation Plan: Git-Inspired Design Version Control

```typescript
// vcs/DesignVersionControl.ts

export interface Branch {
  readonly id: string;
  readonly name: string;
  readonly parentBranchId: string | null; // null = main branch
  readonly createdFrom: string; // commit hash
  readonly createdAt: Date;
  readonly createdBy: string;
  readonly isProtected: boolean;
  readonly requiresReview: boolean;
  readonly status: 'active' | 'merged' | 'archived';
}

export interface Commit {
  readonly hash: string;
  readonly branchId: string;
  readonly parentHash: string | null;
  readonly message: string;
  readonly author: string;
  readonly timestamp: Date;
  readonly changes: readonly DesignChange[];
}

export interface ProtectionRules {
  readonly branchPattern: string; // e.g., "main", "release/*"
  readonly requirePullRequest: boolean;
  readonly requiredReviewers: number;
  readonly allowedMergers: readonly string[]; // user IDs or "
     maintainers"
  readonly preventDirectPush: boolean;
  readonly requireLinearHistory: boolean;
}

export class DesignVersionControl {
  // Create branch from any other branch
  async createBranch(
    name: string,
    fromBranchId: string
  ): Promise<Branch> {
    const parentBranch = await this.getBranch(fromBranchId);
    const latestCommit = await this.getLatestCommit(fromBranchId);

    const branch: Branch = {
      id: crypto.randomUUID(),
      name,
      parentBranchId: fromBranchId,
      createdFrom: latestCommit.hash,
      createdAt: new Date(),
      createdBy: this.currentUser.id,
      isProtected: false,
      requiresReview: false,
      status: 'active',
    };

    await this.saveBranch(branch);
    return branch;
  }

  // Git-style semantic versioning
  async createRelease(
    branchId: string,
    version: SemanticVersion,
    changelog: string
  ): Promise<Release> {
    const commit = await this.getLatestCommit(branchId);

    return {
      version: `${version.major}.${version.minor}.${version.patch}`,
      commitHash: commit.hash,
      changelog,
      createdAt: new Date(),
      tag: `v${version.major}.${version.minor}.${version.patch}`,
    };
  }

  // Visual diff between any two commits/branches
  async visualDiff(
    fromRef: string,
    toRef: string
  ): Promise<VisualDiff> {
    const fromSnapshot = await this.getSnapshot(fromRef);
    const toSnapshot = await this.getSnapshot(toRef);

    const changes = this.computeVisualChanges(fromSnapshot,
       toSnapshot);

    return {
      added: changes.filter(c => c.type === 'added'),
      removed: changes.filter(c => c.type === 'removed'),
      modified: changes.filter(c => c.type === 'modified'),
      moved: changes.filter(c => c.type === 'moved'),
      // Generate overlay/side-by-side images
      overlayImage: await this.renderOverlay(fromSnapshot, toSnapshot
         , changes),
      sideBySideImages: await this.renderSideBySide(fromSnapshot,
         toSnapshot),
    };
  }
}
```

8.3.1 Protected Branch Configuration

```
// components/BranchProtectionSettings.tsx

export const BranchProtectionSettings: React.FC<{
  branch: Branch;
  onUpdate: (rules: ProtectionRules) => void;
}> = ({ branch, onUpdate }) => {
  const [rules, setRules] = useState<ProtectionRules>({
    branchPattern: branch.name,
    requirePullRequest: true,
    requiredReviewers: 1,
    allowedMergers: ['maintainers'],
    preventDirectPush: true,
    requireLinearHistory: false,
  });

  return (
    <div className="protection-settings">
      <h3>Branch Protection: {branch.name}</h3>

      <Toggle
```

```
          label="Require pull request before merging"
          checked={rules.requirePullRequest}
          onChange={(checked) => setRules(r => ({
            ...r, requirePullRequest: checked
          }))}
        />

        <NumberInput
          label="Required reviewers"
          value={rules.requiredReviewers}
          min={0}
          max={5}
          onChange={(value) => setRules(r => ({
            ...r, requiredReviewers: value
          }))}
        />

        <Toggle
          label="Prevent direct pushes (all changes via branches)"
          checked={rules.preventDirectPush}
          onChange={(checked) => setRules(r => ({
            ...r, preventDirectPush: checked
          }))}
        />

        <UserSelect
          label="Users who can merge"
          selected={rules.allowedMergers}
          onChange={(users) => setRules(r => ({
            ...r, allowedMergers: users
          }))}
        />

        <Button onClick={() => onUpdate(rules)}>
          Save Protection Rules
        </Button>
      </div>
    );
};
```

9 Issue #8: Annotation Visibility & Collaboration Gaps

9.1 Problem Statement

Current State

Annotation and collaboration issues:

- Viewing annotations requires a full license
- Business analysts and POs cannot see design specs without paying
- Comments are not persistently visible; easily missed
- Flow descriptions cannot be edited in prototype/present mode
- No structured way to communicate design decisions
- Developer handoff requires expensive seats

User Voice

"It's beyond frustrating that to view annotations you have to have a license. People like BAs and POs just need to see annotations but do not need to edit."
"I understand the licensing aspect for editing, but to view an annotation makes no sense that you would need to purchase another license."
"In Present mode, it is not possible to edit the flow description. I have to go back to design mode to do that."

9.2 What Consumers Want

User Requirements

1. **Free Annotation Viewing**: Anyone with link can view annotations
2. **Persistent Comments**: Always-visible option for critical notes
3. **Flow Documentation**: Edit descriptions while presenting
4. **Structured Specs**: Formalized design decision documentation
5. **Export Options**: Generate PDF/HTML specs for offline viewing
6. **Role-Based Access**: Granular permissions (view specs, comment, edit)

9.3 Implementation Plan: Universal Annotation Access

```typescript
// annotations/AnnotationSystem.ts

export type ViewerPermission =
  | 'view_design'      // See canvas only
  | 'view_annotations' // See annotations (FREE)
  | 'view_specs'       // See dev specs (FREE)
  | 'comment'          // Add comments
  | 'annotate'         // Create annotations
  | 'edit';            // Full edit access

export interface ShareSettings {
  readonly linkAccess: 'restricted' | 'anyone_with_link';
  readonly defaultPermissions: readonly ViewerPermission[];
  readonly allowAnnotationViewing: boolean; // Always true for shared links
  readonly allowSpecsViewing: boolean;
  readonly allowCommenting: boolean;
  readonly expiresAt?: Date;
}

export interface Annotation {
  readonly id: string;
  readonly type: 'note' | 'spec' | 'decision' | 'todo' | 'question';
  readonly content: string;
  readonly attachedTo: string; // Node ID
  readonly position: { x: number; y: number };
  readonly author: string;
  readonly createdAt: Date;
  readonly visibility: 'always' | 'hover' | 'hidden';
  readonly category?: string;
  readonly priority?: 'low' | 'medium' | 'high' | 'critical';
}

```

```
// Annotation viewing is FREE for all shared links
export class AnnotationAccessControl {
  canViewAnnotations(user: User | null, file: DesignFile): boolean {
    // Always allow if file is shared
    if (file.shareSettings.linkAccess === 'anyone_with_link') {
      return true;
    }
    // Allow for any authenticated user with file access
    if (user && this.hasFileAccess(user, file)) {
      return true;
    }
    return false;
  }

  // Specs viewing also FREE
  canViewSpecs(user: User | null, file: DesignFile): boolean {
    return file.shareSettings.allowSpecsViewing &&
           this.canViewAnnotations(user, file);
  }

  // Only commenting and editing require seats
  canComment(user: User, file: DesignFile): boolean {
    return user.hasValidSeat &&
           file.shareSettings.allowCommenting &&
           this.hasFileAccess(user, file);
  }
}
```

9.3.1 Exportable Specification Generator

```
// specs/SpecificationExporter.ts

export interface ExportOptions {
  readonly format: 'pdf' | 'html' | 'markdown';
  readonly includeAnnotations: boolean;
  readonly includeSpecs: boolean;
  readonly includeComments: boolean;
  readonly includeVersionHistory: boolean;
  readonly pageRange: 'all' | readonly string[];
}

export class SpecificationExporter {
  async export(
    file: DesignFile,
    options: ExportOptions
  ): Promise<ExportedDocument> {
    const pages = options.pageRange === 'all'
      ? file.pages
      : file.pages.filter(p => options.pageRange.includes(p.id));

    const sections: DocumentSection[] = [];

    for (const page of pages) {
      sections.push({
        title: page.name,
        thumbnail: await this.renderPageThumbnail(page),
        content: await this.generatePageContent(page, options),
      });
    }

    switch (options.format) {
      case 'pdf':
        return this.generatePDF(sections);
      case 'html':
        return this.generateHTML(sections);
      case 'markdown':
        return this.generateMarkdown(sections);
    }
  }

  private async generatePageContent(
    page: Page,
    options: ExportOptions
  ): Promise<PageContent> {
    const content: PageContent = {
      frames: [],
      annotations: [],
      specs: [],
    };

    for (const frame of page.frames) {
      content.frames.push({
        name: frame.name,
        image: await this.renderFrame(frame),
        dimensions: { width: frame.width, height: frame.height },
      });

      if (options.includeAnnotations) {
        const annotations = await this.getAnnotationsForFrame(frame);
        content.annotations.push(...annotations);
      }

      if (options.includeSpecs) {
        content.specs.push({
          frameId: frame.id,
          colors: this.extractColors(frame),
          typography: this.extractTypography(frame),
          spacing: this.extractSpacing(frame),
          components: this.listComponents(frame),
        });
      }
    }

    return content;
  }
}
```

10 Implementation Timeline

10.1 Phased Rollout Plan

Phase	Deliverables	Duration	Target
1	UI Preference System, Classic Mode Toggle	6 weeks	Q1
2	Performance Optimizations, GPU Rendering	8 weeks	Q1-Q2
3	Offline Mode (Core), Local Storage	8 weeks	Q2
4	Version Control Improvements, Visual Diff	6 weeks	Q2-Q3
5	Linux Native App (Tauri)	6 weeks	Q3
6	Annotation Access, Free Specs Viewing	4 weeks	Q3
7	Pricing Restructure, Freelancer Features	4 weeks	Q3-Q4
8	Productivity AI, Workflow Automation	8 weeks	Q4

Table 3: Implementation Phase Timeline

10.2 Success Metrics

Key Performance Indicators

1. **UI Satisfaction**: >80% positive feedback on UI flexibility
2. **Performance**: 60fps maintained for files up to 500 pages
3. **Offline Adoption**: >40% of desktop users enable offline mode
4. **Linux Users**: 10,000+ monthly active Linux desktop users in Y1
5. **Branching Usage**: 3x increase in branch creation after tier change
6. **Annotation Views**: 5x increase in non-editor spec viewing
7. **NPS Score**: Improvement from current baseline by +15 points

11 Conclusion

This specification addresses the eight most critical pain points identified through extensive user research. By prioritizing user agency, performance, and professional workflows, we can recapture the trust of the design community while expanding accessibility.

The key principles guiding all implementations:

1. **Options Over Mandates**: Every major change offers user choice
2. **Performance is Features**: Speed and reliability are core features, not afterthoughts
3. **Professional First**: Power users drive organizational adoption
4. **Transparent Pricing**: Users understand exactly what they pay for
5. **Platform Parity**: All platforms receive first-class support

Implementation of these solutions positions the product as a true professional-grade design tool that respects its users while continuing to innovate responsibly.

Document prepared for engineering review and stakeholder approval.