# 140509\_30.md - AI-Powered Design Generation Platform

## README

**Summary:** Create a platform that generates creative designs for various media (logos, posters, web layouts) based on user requirements and brand guidelines.

**Problem Statement:** Creative design requires balancing aesthetic appeal with functional requirements and brand consistency. Your task is to build an AI platform that generates designs across multiple formats, considers brand guidelines and user preferences, and provides iterative refinement capabilities. The system should understand design principles, maintain style consistency, and enable collaborative design workflows.

**Steps:** - Design generative models for different design formats (logos, layouts, illustrations) - Implement style transfer and brand guideline adherence mechanisms - Create user preference learning and iterative refinement systems - Build collaborative design tools with version control and feedback integration - Develop design principle evaluation and quality assessment frameworks - Include export capabilities for various formats and design software integration

**Suggested Data Requirements:** - Design portfolio datasets across different styles and formats - Brand guideline examples and style specifications - User feedback and preference data - Design principle documentation and best practices

**Themes:** AI for creative, GenAI & its techniques

## PRD (Product Requirements Document)

### Product Vision

Create an AI-powered design generation platform that democratizes creative design by enabling users to generate professional-quality designs across multiple formats while maintaining brand consistency and design principles.

### Target Users

* **Primary:** Small to medium businesses, marketing teams, freelance designers
* **Secondary:** Large enterprises, design agencies, non-design professionals
* **Tertiary:** Educational institutions, students learning design

### Core Value Propositions

1. **Speed:** Generate designs in minutes instead of hours
2. **Consistency:** Maintain brand guidelines across all generated content
3. **Accessibility:** Enable non-designers to create professional designs
4. **Scalability:** Generate multiple design variations efficiently
5. **Collaboration:** Support team-based design workflows

### Key Features

1. **Multi-Format Generation:** Support for logos, posters, web layouts, social media graphics, business cards
2. **Brand Guideline Integration:** Upload and enforce brand colors, fonts, style guidelines
3. **Intelligent Design Assistance:** AI-powered suggestions based on design principles
4. **Iterative Refinement:** Progressive improvement through user feedback
5. **Template Library:** Extensive collection of customizable design templates
6. **Collaboration Tools:** Real-time editing, commenting, and version control
7. **Export Capabilities:** Support for multiple file formats (SVG, PNG, PDF, AI, PSD)

### Success Metrics

* Design generation time reduction: >80%
* User satisfaction score: >4.2/5.0
* Brand compliance rate: >95%
* Design iteration cycles: <5 per project
* Platform adoption rate: 40% within 6 months

## FRD (Functional Requirements Document)

### Core Functional Requirements

#### F1: Multi-Format Design Generation

* **F1.1:** Generate logos with scalable vector formats
* **F1.2:** Create poster designs with customizable dimensions
* **F1.3:** Produce web layouts responsive to different screen sizes
* **F1.4:** Generate social media graphics for major platforms
* **F1.5:** Create print materials (business cards, brochures, flyers)

#### F2: Brand Guideline Management

* **F2.1:** Upload and parse brand guideline documents
* **F2.2:** Extract brand colors, fonts, and style elements
* **F2.3:** Enforce brand compliance during design generation
* **F2.4:** Validate designs against brand standards
* **F2.5:** Suggest brand-compliant alternatives

#### F3: User Preference Learning

* **F3.1:** Track user interactions and preferences
* **F3.2:** Learn from user feedback and ratings
* **F3.3:** Personalize design suggestions
* **F3.4:** Adapt style recommendations over time
* **F3.5:** Provide preference-based filtering

#### F4: Iterative Design Refinement

* **F4.1:** Enable real-time design modifications
* **F4.2:** Support text, color, and layout adjustments
* **F4.3:** Provide variation generation
* **F4.4:** Implement feedback incorporation mechanisms
* **F4.5:** Track design evolution history

#### F5: Collaboration Features

* **F5.1:** Support multi-user project sharing
* **F5.2:** Implement real-time collaborative editing
* **F5.3:** Provide commenting and annotation tools
* **F5.4:** Enable version control and rollback
* **F5.5:** Manage user permissions and access levels

#### F6: Quality Assessment

* **F6.1:** Evaluate designs against design principles
* **F6.2:** Assess visual hierarchy and balance
* **F6.3:** Check color harmony and contrast
* **F6.4:** Validate typography and readability
* **F6.5:** Generate design quality scores

#### F7: Export and Integration

* **F7.1:** Export to multiple file formats
* **F7.2:** Integrate with Adobe Creative Suite
* **F7.3:** Support design software compatibility
* **F7.4:** Provide API access for third-party integrations
* **F7.5:** Enable batch export capabilities

## NFRD (Non-Functional Requirements Document)

### Performance Requirements

* **NFR-P1:** Design generation response time: <10 seconds for simple designs, <30 seconds for complex layouts
* **NFR-P2:** System availability: 99.9% uptime
* **NFR-P3:** Concurrent user support: 10,000+ simultaneous users
* **NFR-P4:** Design export time: <5 seconds for standard formats
* **NFR-P5:** Real-time collaboration latency: <200ms

### Scalability Requirements

* **NFR-S1:** Horizontal scaling capability to handle 100x load increase
* **NFR-S2:** Auto-scaling based on demand patterns
* **NFR-S3:** Database scaling for millions of design assets
* **NFR-S4:** CDN integration for global content delivery
* **NFR-S5:** Microservices architecture for independent scaling

### Security Requirements

* **NFR-SE1:** End-to-end encryption for user data and designs
* **NFR-SE2:** OAuth 2.0 and SAML integration for enterprise SSO
* **NFR-SE3:** GDPR and CCPA compliance for data privacy
* **NFR-SE4:** Regular security audits and penetration testing
* **NFR-SE5:** Secure API endpoints with rate limiting

### Usability Requirements

* **NFR-U1:** Intuitive interface requiring <5 minutes onboarding
* **NFR-U2:** Mobile-responsive design for tablet and phone access
* **NFR-U3:** Accessibility compliance (WCAG 2.1 AA)
* **NFR-U4:** Multi-language support for global markets
* **NFR-U5:** Offline capability for basic design editing

### Reliability Requirements

* **NFR-R1:** Data backup and recovery (RPO: 1 hour, RTO: 30 minutes)
* **NFR-R2:** Graceful degradation during high load
* **NFR-R3:** Error handling with user-friendly messages
* **NFR-R4:** Design version history preservation
* **NFR-R5:** Redundancy across multiple data centers

### Compatibility Requirements

* **NFR-C1:** Cross-browser compatibility (Chrome, Firefox, Safari, Edge)
* **NFR-C2:** Integration with popular design tools
* **NFR-C3:** API compatibility with existing marketing platforms
* **NFR-C4:** File format compatibility with industry standards
* **NFR-C5:** Operating system independence (web-based platform)

## AD (Architecture Diagram)

graph TB  
 subgraph "Client Layer"  
 WEB[Web Application]  
 MOB[Mobile App]  
 API\_CLIENT[API Clients]  
 end  
   
 subgraph "Load Balancer"  
 LB[Application Load Balancer]  
 end  
   
 subgraph "API Gateway"  
 GW[API Gateway]  
 AUTH[Authentication Service]  
 RATE[Rate Limiter]  
 end  
   
 subgraph "Core Services"  
 DESIGN[Design Generation Service]  
 BRAND[Brand Management Service]  
 COLLAB[Collaboration Service]  
 QUAL[Quality Assessment Service]  
 EXPORT[Export Service]  
 USER[User Preference Service]  
 end  
   
 subgraph "AI/ML Layer"  
 GAN[GAN Models]  
 STYLE[Style Transfer Models]  
 QUAL\_AI[Quality Assessment AI]  
 PREF[Preference Learning ML]  
 NLP[Text Processing NLP]  
 end  
   
 subgraph "Data Layer"  
 POSTGRES[PostgreSQL]  
 MONGO[MongoDB]  
 REDIS[Redis Cache]  
 S3[Object Storage]  
 ELASTIC[Elasticsearch]  
 end  
   
 subgraph "External Services"  
 CDN[Content Delivery Network]  
 FONTS[Font APIs]  
 STOCK[Stock Image APIs]  
 ANALYTICS[Analytics Service]  
 end  
   
 WEB --> LB  
 MOB --> LB  
 API\_CLIENT --> LB  
   
 LB --> GW  
 GW --> AUTH  
 GW --> RATE  
   
 GW --> DESIGN  
 GW --> BRAND  
 GW --> COLLAB  
 GW --> QUAL  
 GW --> EXPORT  
 GW --> USER  
   
 DESIGN --> GAN  
 DESIGN --> STYLE  
 BRAND --> NLP  
 QUAL --> QUAL\_AI  
 USER --> PREF  
   
 DESIGN --> POSTGRES  
 BRAND --> MONGO  
 COLLAB --> REDIS  
 USER --> POSTGRES  
   
 EXPORT --> S3  
 DESIGN --> S3  
 QUAL --> ELASTIC  
   
 DESIGN --> CDN  
 EXPORT --> CDN  
 DESIGN --> FONTS  
 DESIGN --> STOCK  
 USER --> ANALYTICS

## HLD (High Level Design)

### System Architecture Overview

The AI-Powered Design Generation Platform follows a microservices architecture with clear separation of concerns:

#### 1. Client Layer

* **Web Application:** React-based responsive interface
* **Mobile Application:** React Native for iOS/Android
* **API Clients:** RESTful APIs for third-party integrations

#### 2. API Gateway Layer

* **Authentication:** JWT-based authentication with OAuth2 support
* **Rate Limiting:** Prevents API abuse and ensures fair usage
* **Request Routing:** Directs requests to appropriate microservices
* **API Versioning:** Supports multiple API versions for backward compatibility

#### 3. Core Microservices

##### Design Generation Service

* **Responsibilities:** Orchestrates AI model inference for design creation
* **Components:**
  + Design Request Handler
  + Model Selection Engine
  + Generation Coordinator
  + Post-processing Pipeline
* **Technologies:** Python, FastAPI, Celery for async processing

##### Brand Management Service

* **Responsibilities:** Manages brand guidelines and enforcement
* **Components:**
  + Brand Parser (extracts guidelines from documents)
  + Style Validator
  + Brand Asset Manager
  + Compliance Checker
* **Technologies:** Python, OpenCV, PIL for image processing

##### Collaboration Service

* **Responsibilities:** Enables real-time collaboration features
* **Components:**
  + WebSocket Manager
  + Version Control System
  + Comment/Annotation Handler
  + Permission Manager
* **Technologies:** Node.js, Socket.io, Redis for real-time features

##### Quality Assessment Service

* **Responsibilities:** Evaluates design quality using AI models
* **Components:**
  + Design Analyzer
  + Principle Checker
  + Quality Scorer
  + Improvement Suggester
* **Technologies:** Python, TensorFlow, custom evaluation models

#### 4. AI/ML Layer

* **Generative Models:** StyleGAN, CLIP-based models for design generation
* **Style Transfer:** Neural style transfer for brand consistency
* **NLP Models:** BERT/GPT for text understanding and generation
* **Recommendation System:** Collaborative filtering for personalization

#### 5. Data Architecture

* **PostgreSQL:** Relational data (users, projects, metadata)
* **MongoDB:** Document storage (brand guidelines, design templates)
* **Redis:** Caching and session management
* **Object Storage:** Design assets, images, generated files
* **Elasticsearch:** Search functionality and analytics

### Data Flow

1. **Design Request:** User submits design requirements through UI
2. **Authentication:** Request authenticated via API Gateway
3. **Brand Processing:** Brand guidelines parsed and validated
4. **AI Generation:** Appropriate models selected and executed
5. **Quality Check:** Generated design evaluated for quality
6. **User Feedback:** Iterative refinement based on user input
7. **Export:** Final design exported in requested formats

### Scalability Strategy

* **Horizontal Scaling:** Microservices can scale independently
* **Async Processing:** Heavy AI computations handled asynchronously
* **Caching Strategy:** Multi-layer caching for performance optimization
* **CDN Integration:** Global content distribution for assets

## LLD (Low Level Design)

### Design Generation Service - Detailed Implementation

#### Core Components

##### 1. Design Request Handler

class DesignRequestHandler:  
 def \_\_init\_\_(self):  
 self.validator = RequestValidator()  
 self.model\_selector = ModelSelector()  
   
 async def handle\_request(self, request: DesignRequest) -> DesignResponse:  
 # Validate request parameters  
 validation\_result = self.validator.validate(request)  
 if not validation\_result.is\_valid:  
 raise ValidationError(validation\_result.errors)  
   
 # Select appropriate AI models  
 models = self.model\_selector.select\_models(request.design\_type)  
   
 # Queue generation task  
 task = await self.queue\_generation\_task(request, models)  
 return DesignResponse(task\_id=task.id, status="queued")

##### 2. Model Selection Engine

class ModelSelector:  
 def \_\_init\_\_(self):  
 self.model\_registry = ModelRegistry()  
 self.performance\_tracker = PerformanceTracker()  
   
 def select\_models(self, design\_type: str) -> List[ModelConfig]:  
 available\_models = self.model\_registry.get\_models\_for\_type(design\_type)  
   
 # Select best performing models based on historical data  
 selected = []  
 for category in ['generation', 'style\_transfer', 'quality\_check']:  
 best\_model = self.performance\_tracker.get\_best\_model(  
 design\_type, category  
 )  
 selected.append(best\_model)  
   
 return selected

##### 3. Generation Pipeline

class GenerationPipeline:  
 def \_\_init\_\_(self):  
 self.generators = {  
 'logo': LogoGenerator(),  
 'poster': PosterGenerator(),  
 'layout': LayoutGenerator(),  
 }  
 self.style\_transfer = StyleTransferModel()  
 self.post\_processor = PostProcessor()  
   
 async def generate\_design(self, request: DesignRequest) -> GeneratedDesign:  
 # Primary generation  
 generator = self.generators[request.design\_type]  
 base\_design = await generator.generate(request.specifications)  
   
 # Apply brand guidelines  
 if request.brand\_guidelines:  
 styled\_design = await self.style\_transfer.apply\_brand\_style(  
 base\_design, request.brand\_guidelines  
 )  
 else:  
 styled\_design = base\_design  
   
 # Post-processing and optimization  
 final\_design = self.post\_processor.optimize(styled\_design)  
   
 return GeneratedDesign(  
 design\_data=final\_design,  
 metadata=self.extract\_metadata(final\_design),  
 quality\_score=await self.assess\_quality(final\_design)  
 )

#### AI Model Integration

##### Logo Generation Model

class LogoGenerator(BaseGenerator):  
 def \_\_init\_\_(self):  
 self.model = self.load\_model('logo\_stylegan\_v2')  
 self.text\_encoder = CLIPTextEncoder()  
   
 async def generate(self, specs: LogoSpecs) -> LogoDesign:  
 # Encode text description  
 text\_embedding = self.text\_encoder.encode(specs.description)  
   
 # Generate logo variations  
 latent\_codes = self.generate\_latent\_codes(  
 text\_embedding,   
 num\_variations=specs.num\_variations  
 )  
   
 logos = []  
 for code in latent\_codes:  
 logo\_image = self.model.generate(code)  
 logo\_svg = self.convert\_to\_svg(logo\_image)  
 logos.append(LogoDesign(image=logo\_image, svg=logo\_svg))  
   
 return self.select\_best\_logo(logos, specs.criteria)

##### Style Transfer Implementation

class StyleTransferModel:  
 def \_\_init\_\_(self):  
 self.neural\_style\_model = self.load\_neural\_style\_model()  
 self.color\_transfer = ColorTransferModel()  
   
 async def apply\_brand\_style(self, design: Design, guidelines: BrandGuidelines) -> Design:  
 styled\_design = design.copy()  
   
 # Apply color palette  
 if guidelines.color\_palette:  
 styled\_design = self.color\_transfer.apply\_palette(  
 styled\_design, guidelines.color\_palette  
 )  
   
 # Apply typography  
 if guidelines.fonts:  
 styled\_design = self.apply\_typography(  
 styled\_design, guidelines.fonts  
 )  
   
 # Apply style patterns  
 if guidelines.style\_elements:  
 styled\_design = self.neural\_style\_model.transfer(  
 styled\_design, guidelines.style\_elements  
 )  
   
 return styled\_design

### Database Schema Design

#### PostgreSQL Tables

-- Users table  
CREATE TABLE users (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 email VARCHAR(255) UNIQUE NOT NULL,  
 username VARCHAR(100) UNIQUE NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 subscription\_tier VARCHAR(50) DEFAULT 'free'  
);  
  
-- Projects table  
CREATE TABLE projects (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES users(id),  
 name VARCHAR(255) NOT NULL,  
 design\_type VARCHAR(100) NOT NULL,  
 status VARCHAR(50) DEFAULT 'draft',  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
-- Designs table  
CREATE TABLE designs (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 project\_id UUID REFERENCES projects(id),  
 version\_number INTEGER DEFAULT 1,  
 design\_data JSONB NOT NULL,  
 quality\_score FLOAT,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 is\_final BOOLEAN DEFAULT FALSE  
);  
  
-- Brand guidelines table  
CREATE TABLE brand\_guidelines (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES users(id),  
 name VARCHAR(255) NOT NULL,  
 guidelines\_data JSONB NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

#### MongoDB Collections

// Design templates collection  
{  
 "\_id": ObjectId,  
 "template\_name": String,  
 "category": String, // logo, poster, layout, etc.  
 "style\_tags": [String],  
 "template\_data": {  
 "layers": [Object],  
 "dimensions": Object,  
 "color\_scheme": [String],  
 "fonts": [String]  
 },  
 "usage\_count": Number,  
 "rating": Number,  
 "created\_at": Date,  
 "updated\_at": Date  
}  
  
// User preferences collection  
{  
 "\_id": ObjectId,  
 "user\_id": String,  
 "preferences": {  
 "color\_preferences": [String],  
 "style\_preferences": [String],  
 "layout\_preferences": Object,  
 "font\_preferences": [String]  
 },  
 "interaction\_history": [Object],  
 "learning\_model\_state": Object,  
 "updated\_at": Date  
}

### API Design

#### RESTful API Endpoints

# Design generation endpoints  
POST /api/v1/designs/generate  
GET /api/v1/designs/{design\_id}  
PUT /api/v1/designs/{design\_id}/refine  
DELETE /api/v1/designs/{design\_id}  
  
# Brand management endpoints  
POST /api/v1/brands  
GET /api/v1/brands/{brand\_id}  
PUT /api/v1/brands/{brand\_id}  
DELETE /api/v1/brands/{brand\_id}  
  
# Project management endpoints  
POST /api/v1/projects  
GET /api/v1/projects  
GET /api/v1/projects/{project\_id}  
PUT /api/v1/projects/{project\_id}  
DELETE /api/v1/projects/{project\_id}  
  
# Export endpoints  
POST /api/v1/exports/generate  
GET /api/v1/exports/{export\_id}/download

## Pseudocode

### Main Design Generation Workflow

ALGORITHM DesignGenerationWorkflow  
INPUT: design\_request (type, specifications, brand\_guidelines, user\_preferences)  
OUTPUT: generated\_design (design\_data, metadata, quality\_score)  
  
BEGIN  
 // Step 1: Validate and preprocess request  
 validation\_result = VALIDATE\_REQUEST(design\_request)  
 IF validation\_result.is\_invalid THEN  
 RETURN ERROR(validation\_result.errors)  
 END IF  
   
 // Step 2: Select appropriate AI models  
 models = SELECT\_MODELS(design\_request.type)  
   
 // Step 3: Load user preferences and brand guidelines  
 user\_prefs = LOAD\_USER\_PREFERENCES(design\_request.user\_id)  
 brand\_constraints = PROCESS\_BRAND\_GUIDELINES(design\_request.brand\_guidelines)  
   
 // Step 4: Generate base design  
 base\_design = GENERATE\_BASE\_DESIGN(  
 design\_request.specifications,  
 models.generation\_model,  
 user\_prefs  
 )  
   
 // Step 5: Apply brand styling  
 IF brand\_constraints.exists THEN  
 styled\_design = APPLY\_BRAND\_STYLE(base\_design, brand\_constraints)  
 ELSE  
 styled\_design = base\_design  
 END IF  
   
 // Step 6: Quality assessment and improvement  
 quality\_score = ASSESS\_DESIGN\_QUALITY(styled\_design)  
 IF quality\_score < MINIMUM\_THRESHOLD THEN  
 improved\_design = IMPROVE\_DESIGN(styled\_design, models.improvement\_model)  
 quality\_score = ASSESS\_DESIGN\_QUALITY(improved\_design)  
 styled\_design = improved\_design  
 END IF  
   
 // Step 7: Generate variations if requested  
 variations = []  
 IF design\_request.num\_variations > 1 THEN  
 FOR i = 1 TO design\_request.num\_variations - 1 DO  
 variation = GENERATE\_VARIATION(styled\_design, models.variation\_model)  
 variations.APPEND(variation)  
 END FOR  
 END IF  
   
 // Step 8: Prepare final output  
 final\_design = CREATE\_DESIGN\_PACKAGE(  
 primary\_design = styled\_design,  
 variations = variations,  
 metadata = EXTRACT\_METADATA(styled\_design),  
 quality\_score = quality\_score  
 )  
   
 // Step 9: Save and return  
 SAVE\_DESIGN\_TO\_DATABASE(final\_design, design\_request.project\_id)  
 UPDATE\_USER\_PREFERENCES(design\_request.user\_id, final\_design)  
   
 RETURN final\_design  
END  
  
FUNCTION GENERATE\_BASE\_DESIGN(specifications, model, user\_prefs)  
BEGIN  
 // Encode text description using CLIP or similar  
 text\_embedding = ENCODE\_TEXT(specifications.description)  
   
 // Incorporate user preferences  
 preference\_vector = ENCODE\_PREFERENCES(user\_prefs)  
 combined\_embedding = COMBINE\_EMBEDDINGS(text\_embedding, preference\_vector)  
   
 // Generate latent code  
 latent\_code = SAMPLE\_LATENT\_SPACE(combined\_embedding, specifications.creativity\_level)  
   
 // Generate design using GAN or diffusion model  
 raw\_design = model.GENERATE(latent\_code)  
   
 // Post-process for format-specific requirements  
 processed\_design = POST\_PROCESS\_DESIGN(raw\_design, specifications.format)  
   
 RETURN processed\_design  
END  
  
FUNCTION APPLY\_BRAND\_STYLE(design, brand\_constraints)  
BEGIN  
 styled\_design = COPY(design)  
   
 // Apply color palette constraints  
 IF brand\_constraints.color\_palette.exists THEN  
 styled\_design = TRANSFER\_COLORS(styled\_design, brand\_constraints.color\_palette)  
 END IF  
   
 // Apply typography constraints  
 IF brand\_constraints.fonts.exists THEN  
 styled\_design = APPLY\_FONTS(styled\_design, brand\_constraints.fonts)  
 END IF  
   
 // Apply style pattern constraints  
 IF brand\_constraints.style\_patterns.exists THEN  
 styled\_design = NEURAL\_STYLE\_TRANSFER(  
 styled\_design,   
 brand\_constraints.style\_patterns  
 )  
 END IF  
   
 // Validate brand compliance  
 compliance\_score = CHECK\_BRAND\_COMPLIANCE(styled\_design, brand\_constraints)  
 IF compliance\_score < BRAND\_COMPLIANCE\_THRESHOLD THEN  
 styled\_design = ENFORCE\_BRAND\_COMPLIANCE(styled\_design, brand\_constraints)  
 END IF  
   
 RETURN styled\_design  
END  
  
FUNCTION ASSESS\_DESIGN\_QUALITY(design)  
BEGIN  
 quality\_metrics = {}  
   
 // Visual hierarchy assessment  
 quality\_metrics.hierarchy = ASSESS\_VISUAL\_HIERARCHY(design)  
   
 // Color harmony evaluation  
 quality\_metrics.color\_harmony = EVALUATE\_COLOR\_HARMONY(design)  
   
 // Typography quality  
 quality\_metrics.typography = ASSESS\_TYPOGRAPHY(design)  
   
 // Composition balance  
 quality\_metrics.balance = EVALUATE\_COMPOSITION\_BALANCE(design)  
   
 // Overall aesthetic appeal  
 quality\_metrics.aesthetic = NEURAL\_AESTHETIC\_SCORER(design)  
   
 // Compute weighted quality score  
 weights = {  
 hierarchy: 0.25,  
 color\_harmony: 0.20,  
 typography: 0.20,  
 balance: 0.20,  
 aesthetic: 0.15  
 }  
   
 quality\_score = 0  
 FOR metric, value IN quality\_metrics DO  
 quality\_score += weights[metric] \* value  
 END FOR  
   
 RETURN quality\_score  
END  
  
FUNCTION ITERATIVE\_REFINEMENT(design, user\_feedback, refinement\_model)  
BEGIN  
 current\_design = COPY(design)  
 iteration\_count = 0  
 MAX\_ITERATIONS = 5  
   
 WHILE iteration\_count < MAX\_ITERATIONS AND user\_feedback.satisfied = FALSE DO  
 // Process user feedback  
 feedback\_vector = ENCODE\_FEEDBACK(user\_feedback)  
   
 // Generate refinement suggestions  
 refinement\_params = refinement\_model.PREDICT\_REFINEMENTS(  
 current\_design,   
 feedback\_vector  
 )  
   
 // Apply refinements  
 refined\_design = APPLY\_REFINEMENTS(current\_design, refinement\_params)  
   
 // Present to user for feedback  
 user\_feedback = REQUEST\_USER\_FEEDBACK(refined\_design)  
   
 current\_design = refined\_design  
 iteration\_count += 1  
 END WHILE  
   
 RETURN current\_design  
END  
  
FUNCTION COLLABORATIVE\_WORKFLOW(project\_id, collaborators, design\_changes)  
BEGIN  
 // Lock project for concurrent access  
 project\_lock = ACQUIRE\_PROJECT\_LOCK(project\_id)  
   
 TRY  
 current\_version = GET\_LATEST\_DESIGN\_VERSION(project\_id)  
   
 // Apply changes from each collaborator  
 merged\_changes = MERGE\_DESIGN\_CHANGES(design\_changes)  
   
 // Check for conflicts  
 conflicts = DETECT\_CONFLICTS(current\_version, merged\_changes)  
   
 IF conflicts.exist THEN  
 resolved\_changes = RESOLVE\_CONFLICTS\_WITH\_USERS(conflicts, collaborators)  
 final\_changes = resolved\_changes  
 ELSE  
 final\_changes = merged\_changes  
 END IF  
   
 // Create new version  
 new\_version = APPLY\_CHANGES\_TO\_DESIGN(current\_version, final\_changes)  
 new\_version\_number = current\_version.version + 1  
   
 // Save new version  
 SAVE\_DESIGN\_VERSION(project\_id, new\_version, new\_version\_number)  
   
 // Notify all collaborators  
 NOTIFY\_COLLABORATORS(project\_id, collaborators, new\_version)  
   
 FINALLY  
 RELEASE\_PROJECT\_LOCK(project\_lock)  
 END TRY  
   
 RETURN new\_version  
END

### Brand Guideline Processing

ALGORITHM ProcessBrandGuidelines  
INPUT: brand\_document (file\_path, document\_type)  
OUTPUT: brand\_constraints (colors, fonts, styles, rules)  
  
BEGIN  
 brand\_constraints = INITIALIZE\_EMPTY\_CONSTRAINTS()  
   
 // Step 1: Document parsing  
 IF document\_type = "PDF" THEN  
 parsed\_content = EXTRACT\_PDF\_CONTENT(brand\_document)  
 ELSE IF document\_type = "DOCX" THEN  
 parsed\_content = EXTRACT\_DOCX\_CONTENT(brand\_document)  
 ELSE  
 parsed\_content = READ\_TEXT\_FILE(brand\_document)  
 END IF  
   
 // Step 2: Extract color palette  
 color\_mentions = EXTRACT\_COLOR\_REFERENCES(parsed\_content.text)  
 color\_images = EXTRACT\_IMAGES\_FROM\_DOCUMENT(brand\_document)  
   
 extracted\_colors = []  
 FOR color\_ref IN color\_mentions DO  
 rgb\_color = CONVERT\_COLOR\_TO\_RGB(color\_ref)  
 extracted\_colors.APPEND(rgb\_color)  
 END FOR  
   
 FOR image IN color\_images DO  
 dominant\_colors = EXTRACT\_DOMINANT\_COLORS(image)  
 extracted\_colors.EXTEND(dominant\_colors)  
 END FOR  
   
 brand\_constraints.color\_palette = DEDUPLICATE\_COLORS(extracted\_colors)  
   
 // Step 3: Extract typography information  
 font\_mentions = EXTRACT\_FONT\_REFERENCES(parsed\_content.text)  
 brand\_constraints.fonts = NORMALIZE\_FONT\_NAMES(font\_mentions)  
   
 // Step 4: Extract style patterns  
 style\_images = FILTER\_STYLE\_IMAGES(color\_images)  
 style\_patterns = []  
 FOR image IN style\_images DO  
 pattern\_features = EXTRACT\_STYLE\_FEATURES(image)  
 style\_patterns.APPEND(pattern\_features)  
 END FOR  
 brand\_constraints.style\_patterns = style\_patterns  
   
 // Step 5: Extract usage rules  
 rule\_text = EXTRACT\_RULE\_SECTIONS(parsed\_content.text)  
 brand\_constraints.usage\_rules = PARSE\_USAGE\_RULES(rule\_text)  
   
 RETURN brand\_constraints  
END

This comprehensive documentation for Problem Statement 30 provides a complete solution architecture from high-level requirements down to detailed implementation pseudocode. Each section builds upon the previous ones to create a cohesive and implementable solution for the AI-Powered Design Generation Platform.