# Software Design Document

Version 1.0

February 28, 2025

# **Web Publishing System**

Suraj Sahu - 20232742

Rahul Gupta - 20232757

# Table of Contents

# *1.0 Introduction*

## *1.1 Purpose*

This document explains the Web Publishing System—its purpose, features, interfaces, functionality, constraints, and reactions to external factors. It's intended for stakeholders and developers and will be proposed to the Regional Historical Society for approval.

## *1.2 Scope*

The Web Publishing System is designed for a local editor of a regional historical society to streamline the article review and publishing process. It aims to boost productivity by automating tasks and simplifying communication between the editor, reviewers, and authors. Key features include email-based communication, preformatted reply forms for consistent reviews, and a database to manage authors, reviewers, and articles. The system is user-friendly and meets the editor’s needs efficiently.

## *1.3 Overview*

The next chapter, "Overall Description," gives a general idea of the product’s functionality and sets the stage for the technical details in the following chapter. The third chapter, "Requirements Specification," is aimed at developers and explains the product’s features in technical terms.

Both chapters describe the same product but are written for different audiences with different styles.

# *2.0 Overall Description*

## *2.1 Overall Description*

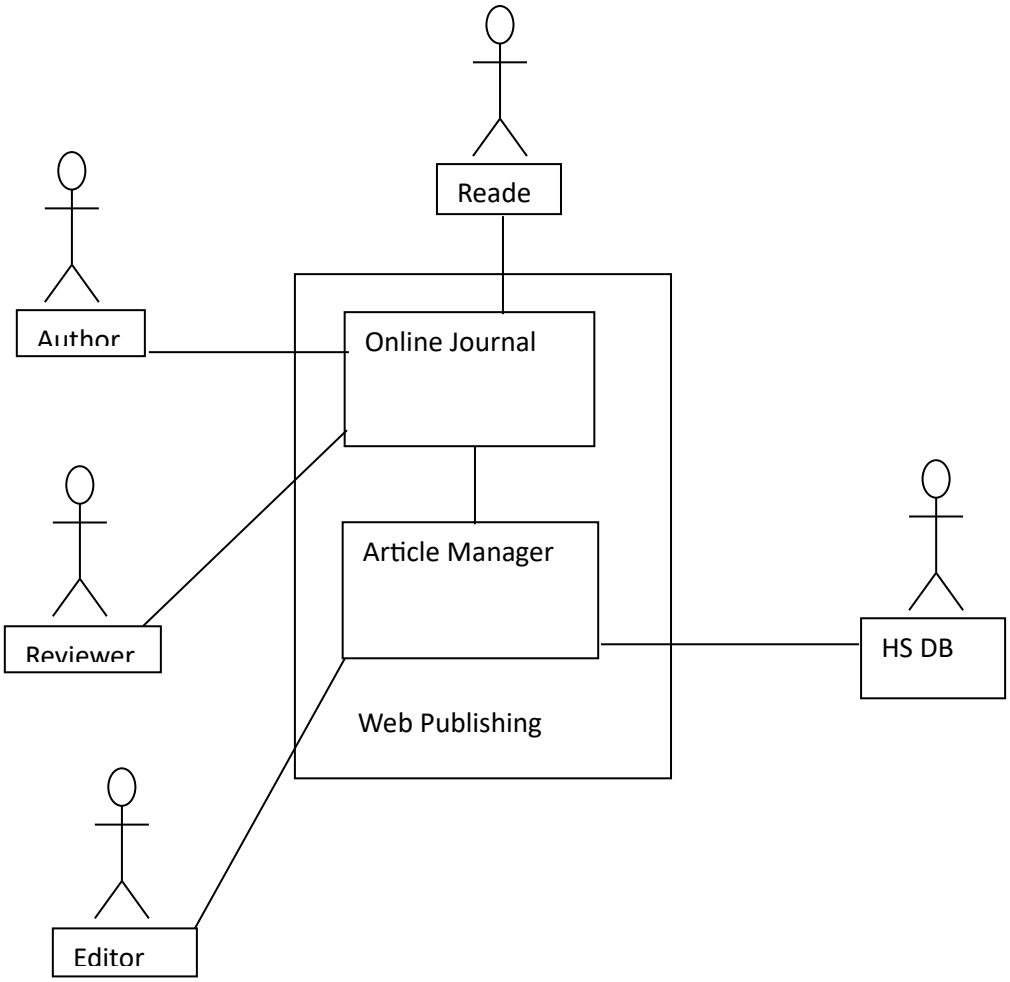



**Figure - System Environment**

The Web Publishing System involves four main roles and one linked system:

- **Author, Reader, and Reviewer**: They use the Online Journal through the Internet. Authors and Reviewers communicate via email.
- **Editor**: Has direct access to the entire system.
- **Historical Society**: The system is connected to the society's existing database.

The system is split into two parts—Online Journal and Article Manager—to make things clearer using domain classes.

## *2.2 Functional Requirements Specification*

**Use Cases Overview**

This section describes the roles and their specific use cases in the Online Journal System. The Reader, Author, and Reviewer each have one primary function, while the Editor plays a central role in managing articles.

### *2.2.1 Reader Use Case*

**Use case:** Search Article
**Description:** The Reader searches for an article using author name, category, or keyword, views its abstract, and downloads it.

**Steps:**

1. Search by author, category, or keyword.
2. System displays search results.
3. Reader selects an article.
4. System presents the abstract.
5. Reader downloads the article.
6. System provides the article.

### *2.2.2 Author Use Case*

**Use case:** Submit Article
**Description:** The Author submits an original or revised article via email.

**Steps:**

1. Click "Email Editor."

2. System opens email with pre-filled address.
3. Author fills in subject, attaches files, and sends.
4. System sends confirmation email.

### *2.2.3 Reviewer Use Case*

**Use case:** Submit Review
**Description:** The Reviewer submits feedback on an article via email.

**Steps:**

1. Click "Email Editor."
2. System opens email with pre-filled address.
3. Reviewer fills in subject, attaches review, and sends.
4. System sends confirmation email.

### *2.2.4 Editor Use Cases*

The Editor manages authors, reviewers, and articles through multiple functions.

**Managing Authors and Reviewers**

- **Update Author:** Add or edit author information.
- **Update Reviewer:** Add or edit reviewer details.

**Managing Articles**

- **Update Article:** Modify article details.
- **Receive Article:** Add new or revised articles.
- **Assign Reviewer:** Assign reviewers to articles.
- **Receive Review:** Enter received reviews into the system.
- **Check Status:** View the status of all active articles.
- **Send Response:** Email authors about their submission status.

- **Send Copyright:** Email authors a copyright form.
- **Remove Article:** Remove declined articles.
- **Publish Article:** Transfer accepted articles to the Online Journal.

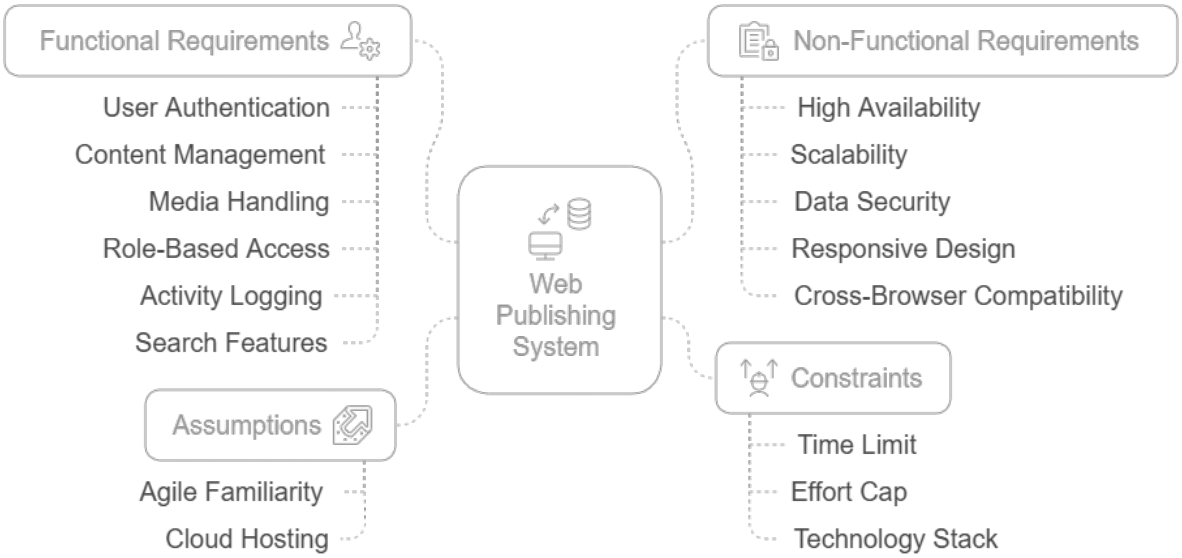
## *2.3 User Characteristics*

- **Reader:** Must be able to search and download articles online.
- **Author/Reviewer:** Should be familiar with email and attachments.
- **Editor:** Must be comfortable with Windows, buttons, and menus.

## *2.4 Non-Functional Requirements*

- Hosted on a high-speed server managed by the Historical Society.
- Uses Tomcat for database integration.
- Article Manager runs on the Editor's Windows PC using an Access database.

# *3.0 Requirements Specification*




## *3.1 External Interface Requirements*

The system connects to the Historical Society (HS) Database to check if a Reviewer is a member. The Editor requires Reviewers to be members, believing it improves quality. The system uses the HS Database to verify membership by checking the name, ID number, and email (optional). When assigning or updating a Reviewer, the system confirms membership and retrieves relevant details.

### *3.1.1 Functional Requirements*

- **Search Article**: Users can search by Author, Category, or Keyword. The system presents relevant results, allowing users to select and download articles.
- **Communicate**: Users can send emails via a provided mail link. Alternatively, they can use their own email system.
- **Add Author**: The Editor enters author details into a form. The system checks required fields and adds the author to the database.
- **Add Reviewer**: The system retrieves HS members, and the Editor selects a Reviewer. The information is transferred to the system, prompting for missing email addresses if needed.

- **Update Person**: The Editor selects and updates an Author or Reviewer's details.
- **Update Article Status**: The Editor updates an article's details, including categories and reviewer assignments.
- **Enter Communication**: The Editor attaches and updates article-related files.
- **Assign Reviewer**: The Editor assigns a reviewer, and the system emails them the article (excluding author info).
- **Check Status**: The system lists active articles with statuses like "Review Pending" or "Accepted but not Published."
- **Send Communication**: The Editor emails an Author directly from the system.
- **Publish Article**: The Editor transfers an accepted article to the Online Journal.
- **Remove Article**: The Editor can delete an article from the active database, confirming before removal.

### *3.1.2 Data Structure*

- **Author Data**: Includes name, email, and article references.
- **Reviewer Data**: Includes name, ID, email, assigned articles, number of pending reviews, and expertise.
- **Review Data**: Tracks articles, assigned reviewers, review dates, and content.
- **Article Data**: Stores the title, authors, abstract, content, category, and status (accepted, copyrighted, published).
- **Published Articles**: Stored in the Online Journal database with metadata for searchability.

### *3.1.3 Security*

- The Online Journal has server security for write/delete access; reading is unrestricted.
- The Article Manager is secured by limiting physical access to the Editor.
- Email communication is handled externally by the user's email system.

# *4.0 Risk Management*

Effective risk management ensures the Web Publishing System remains resilient against foreseeable and unforeseeable challenges. By proactively identifying, estimating, and mitigating risks, the project maximizes the probability of timely and quality delivery.

## *4.1 Function Point Analysis (FPA)*

Function Point Analysis provides a structured, quantifiable method to assess the functionality delivered to the user, independent of the technology or programming language used. For the Web Publishing System:

| **Component** | **Count** | **Weight (Avg)** | **FP (Count × Weight)** |
|---|---|---|---|
| External Inputs (EI) | 4 | 4 | 16 |
| External Outputs (EO) | 3 | 5 | 15 |
| External Inquiries (EQ) | 2 | 4 | 8 |
| Internal Logical Files (ILF) | 3 | 7 | 21 |
| External Interface Files (EIF) | 2 | 5 | 10 |

Total Function Points: 70 FP

This quantification serves as the baseline for effort estimation and schedule planning. It reflects a moderately complex system, balancing input handling, storage, external interfacing, and outputs crucial for a publishing platform.

## *4.2. Effort Estimation (COCOMO - Basic Model)*

Using the Basic COCOMO Model, the effort for developing the Web Publishing System is estimated based on the derived size:

- **1 Function Point ≈ 100 LOC**
- **Total LOC ≈ 7000 LOC (7 KLOC)**

Applying the Basic COCOMO Equation:

**Effort (person-months) = 2.4 × (KLOC)^1.05**

**Effort = 2.4 × (7)^1.05 ≈ 18.2 person-months**

This estimation underlines the moderate effort requirement for the system and emphasizes the necessity of disciplined project management to ensure resource optimization.

## *4.3. Schedule Table*

A pragmatic and structured schedule is vital for ensuring the project's success. Here's the proposed breakdown:

| **Phase** | **Duration (Weeks)** | **Description** |
|---|---|---|
| Requirements | 2 | Gather functional and non-functional requirements |
| Design | 2 | UI/UX + Architecture |
| Development | 6 | Backend + Frontend |
| Testing | 2 | Unit, Integration, System |
| Deployment | 1 | Server setup, Go-live |
| Documentation | 1 | User manual + Technical docs |

## *4.4. Risk Table*

The Web Publishing System development is exposed to several risks. Proactive mitigation strategies are crucial:

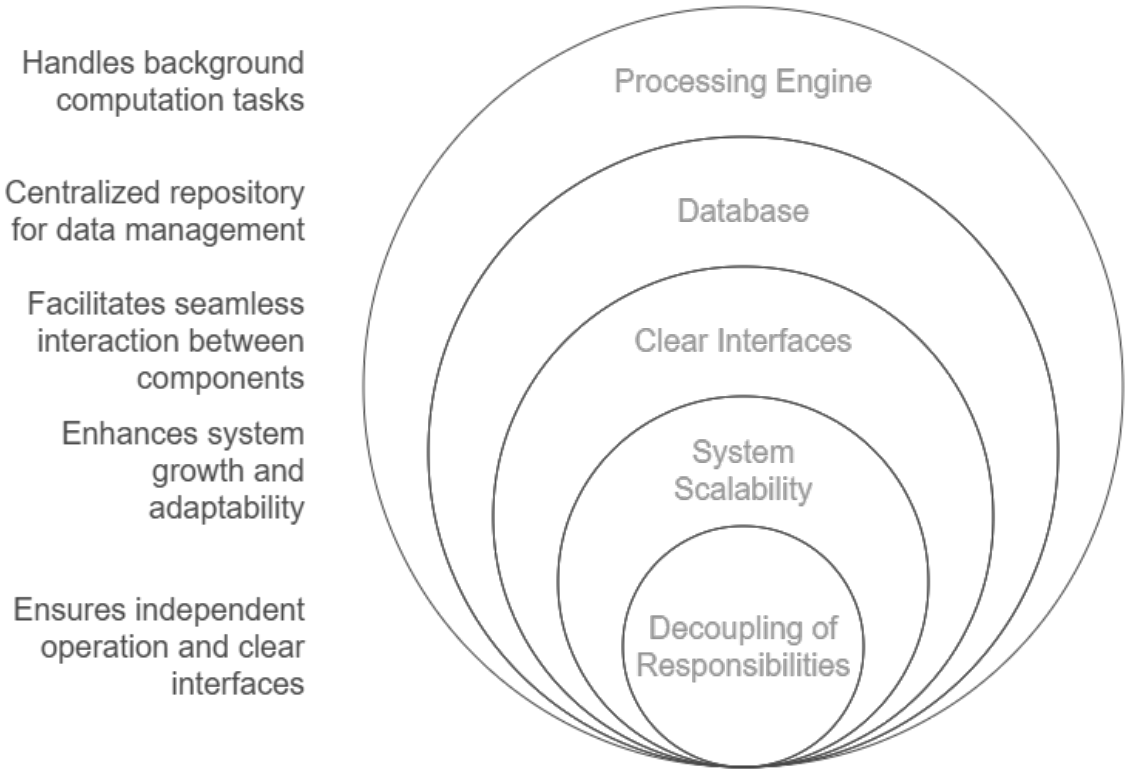
| **Risk** | **Probability** | **Impact** | **Mitigation Plan** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Requirement changes | Medium | High | Freeze scope early, use Moscow |
| Developer leaves project | Low | High | Documentation, backups |
| Delay in deployment | Medium | Medium | Buffer time, prepare staging env |
| Security vulnerabilities | Medium | High | Use HTTPS, input validation |
| Hosting/server failure | Low | High | Use reliable provider, regular backup |

# *5.0 Design Engineering*

Design Engineering focuses on structuring the Web Publishing System to meet both the functional and non-functional requirements while ensuring scalability, maintainability, and performance. It encapsulates three major elements: Architectural Design, Data Design, and Component-Level Design.

## *5.1 Architectural Design*



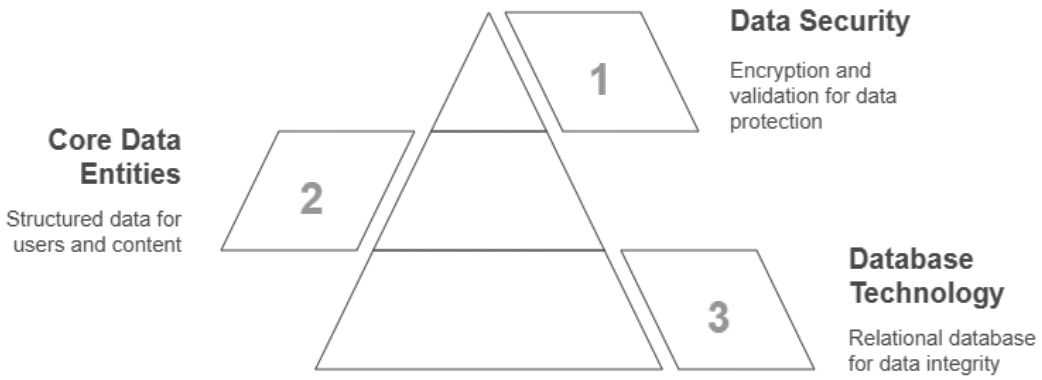

The architecture of the Web Publishing System is deliberately designed as a modular, layered structure composed of three main subsystems: **Web Application**, **Processing Engine**, and **Database**.

- **Web Application**:
  Acts as the primary interface for end-users (publishers, editors, and readers). It allows content creation, review, publishing, and management. The web application is designed to function independently, meaning it can operate even without the backend computation engine if needed.

- **Processing Engine**:
  Handles all background tasks that require computation, such as content analytics, auto-tagging, media optimization, and distribution scoring. The engine processes data asynchronously, following a "fire-and-forget" job subsystem architecture to prevent the web interface from being blocked or delayed.
- **Database**:
  Serves as the centralized repository for content, user data, system logs, and configuration settings. The database layer supports both the web application and engine through secure, efficient stored procedures accessed via Dapper ORM to ensure lightweight, performant interactions.

This architectural separation ensures **decoupling of responsibilities**, enhances **system scalability**, and provides **clear interfaces** between the user-facing components and the backend computation modules.

## *5.2 Data Design*

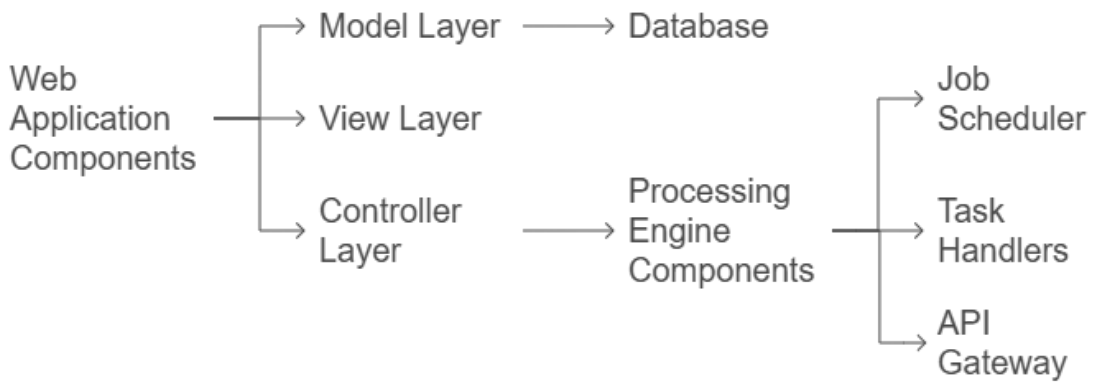



Data within the Web Publishing System is structured carefully to ensure high integrity, fast retrieval, and flexible future evolution.

- **Database Technology**:
  A relational database (e.g., PostgreSQL or SQL Server) is selected for its transactional integrity and strong support for structured queries.
    - Stored procedures are heavily employed to abstract complex queries and minimize SQL injection risks.
    - Dapper ORM is used to allow fast, simple mapping between database rows and domain objects without heavy overhead.
- **Core Data Entities**:
    - **Users**: Authentication credentials, roles, permissions.
    - **Content Items**: Title, body, tags, media attachments, status (draft, published, archived).
    - **Media Assets**: Images, videos, documents, linked to content items.
    - **Activity Logs**: Audit trails for user actions and system events.
- **Data Security**:
    - All sensitive data is encrypted at rest and in transit.
    - Input validation and sanitation occur at both API and database levels.

This structured approach allows quick access to frequently used data while keeping the system secure and maintainable.

## *5.3 Component-Level Design*



At the component level, the Web Publishing System follows the **Model-View-Controller (MVC)** paradigm, adhering to best practices for modular web applications.

**Web Application Components:**

- **Model Layer**:
    - Manages the business logic and validation rules.
    - Interfaces with the database via the provider layer and Dapper ORM.
- **View Layer**:
    - Dynamically renders UI elements based on user roles and permissions.
    - Fully responsive to accommodate mobile and desktop users.
- **Controller Layer**:
    - Handles incoming HTTP requests, orchestrates responses, and manages session states.
    - Maps specific user actions (e.g., "publish article") to corresponding service calls.

**Processing Engine Components:**

- **Job Scheduler**:
    - Manages queued tasks such as auto-tagging, content distribution scoring, and image optimization.
- **Task Handlers**:
    - Each job is isolated and processed independently to ensure transactional safety and system reliability.
- **API Gateway**:
    - Provides a clean, secure interface for the web application to interact with the engine.

**Design Decision Justifications**:

- **MVC Adoption**:
  Aligns with industry standards and facilitates easier onboarding, maintenance, and testing. It also ensures the Web Publishing System remains consistent with other systems used in enterprise environments, accelerating future integrations.

- **Use of Dapper**:
  Prioritizes speed and lightweight performance over heavier ORM solutions, enabling better control over database operations while maintaining simplicity for developers.

- **Fire-and-Forget Engine Jobs**:
  Enables non-blocking operations for long-running tasks, ensuring the user interface remains highly responsive even during complex backend processing.