Confluence Search

**1. YAML Configuration File**

The YAML configuration file (crew\_config.yml) defines the complete multi-agent system with all the components:

* **Tools**: The Confluence search tool with configuration parameters
* **Agents**: All four specialized agents with their roles and settings
* **Tasks**: Sequential tasks with detailed descriptions and dependencies
* **Crew**: The overall configuration that ties everything together

Key advantages of the YAML approach:

* **Cleaner separation** between configuration and code
* **Easier to modify** without touching Python code
* **More declarative** representation of your agent system
* **Explicit dependencies** between tasks using the depends\_on field

**2. Python Loader Script**

The Python script loads and runs the YAML configuration:

python

*# Load the crew*

loader = YamlLoader("crew\_config.yml")

crew = loader.load()

*# Run with input*

result = crew.kickoff(inputs={"requirement\_text": "Your requirement here"})

I've also included a commented example of how you would implement the actual SearchConfluenceTool class in a separate file that gets referenced from the YAML configuration.

**How to Use This Approach**

1. Save the YAML content to crew\_config.yml
2. Create the tool implementation in tools/confluence\_tools.py
3. Use the Python loader script to run the system

This approach gives you the best of both worlds:

* Configuration details in YAML for easy editing
* Tool implementation in Python for functionality
* Clean separation of concerns

# Comprehensive Requirements Document: Confluence-Powered Requirements Bot

## Project Overview

This project implements a sophisticated multi-agent AI system using CrewAI that processes new product requirements by analyzing them against historical knowledge stored in Confluence. The system performs detailed analysis of requirements, identifies gaps, conducts comprehensive impact assessments across all potentially affected systems, and generates standardized documentation that follows organizational best practices.

## Business Need

Organizations face significant challenges in requirements management:

1. **Inconsistent Documentation**: Requirements often vary in format, detail level, and quality depending on the author
2. **Knowledge Silos**: Critical information about past implementations remains trapped in Confluence pages
3. **Incomplete Impact Analysis**: Teams often miss affected systems or underestimate implementation complexity
4. **Reinventing Solutions**: Similar requirements are often approached differently across teams
5. **Requirements Drift**: Changes in requirements aren't properly tracked against original specifications
6. **Insufficient Historical Context**: New team members lack understanding of why past decisions were made

This system addresses these challenges through AI-powered analysis of historical knowledge and standardized documentation.

## Project Scope

### In Scope

* Development of a multi-agent AI system using CrewAI framework
* Vector database integration with Confluence content
* Semantic search functionality with filtering capabilities
* Automated requirements analysis against historical context
* Gap identification between new and previous requirements
* Comprehensive impact assessment across all systems
* Risk evaluation and complexity estimation
* Generation of standardized requirements documents
* API for requirement submission and document retrieval

### Out of Scope

* UI/frontend development (initial implementation will be API-based)
* Direct integration with JIRA or other project management tools
* Automated implementation of requirements
* Training/fine-tuning of custom AI models
* Automated testing of implemented requirements
* Code generation from requirements

## System Components

### 1. Vector Database Integration

#### Description

System to convert Confluence pages into vector embeddings for semantic search

#### Requirements

* **Automated Indexing**: Automatically index and update Confluence content on schedule
* **Change Detection**: Identify and process only changed Confluence pages during updates
* **Embedding Storage**: Store embeddings in a vector database (e.g., Pinecone, Weaviate)
* **Metadata Capture**: Include relevant metadata with embeddings (page author, date, tags, etc.)
* **Search API**: Support semantic search via API with filtering options
* **Versioning**: Track changes to Confluence pages over time
* **Error Handling**: Robust error handling for failed indexing attempts

#### Technical Details

* Use of sentence transformers for embedding generation
* Chunking strategies for long documents
* Metadata extraction from Confluence API
* Vector database schema design

### 2. Custom CrewAI Tools

#### Description

Tools that enable AI agents to interact with the vector database and process information

#### Requirements

* **Search Tool**: Retrieve relevant Confluence pages based on semantic similarity
* **Filtering**: Filter search results based on configurable criteria
* **Result Processing**: Format retrieved content for agent consumption
* **Content Extraction**: Extract specific sections or information from retrieved documents
* **Quality Control**: Provide confidence scores and relevance metrics for search results
* **Rate Limiting**: Prevent excessive API calls or resource consumption
* **Caching**: Cache frequently accessed content to improve performance

#### Technical Details

* Tool implementation using LangChain's BaseTool
* Input validation using Pydantic models
* Structured output formatting
* Error handling and retry logic

### 3. Agent System

#### Description

Four specialized AI agents that collaborate on requirement analysis with distinct responsibilities

#### Requirements

##### 3.1 Requirements Analyzer Agent

* **Historical Analysis**: Find similar historical requirements in Confluence
* **Pattern Recognition**: Identify common patterns in similar requirements
* **Terminology Extraction**: Extract domain-specific terminology and definitions
* **Module Identification**: Determine modules historically affected by similar requirements
* **Knowledge Contextualization**: Understand the organizational context of requirements
* **Content Summarization**: Distill key information from lengthy Confluence pages
* **Evidence Collection**: Gather supporting evidence for analyses

##### 3.2 Gap Identifier Agent

* **Differential Analysis**: Compare new requirements against historical data
* **Innovation Detection**: Identify novel aspects not seen in previous requirements
* **Modification Recognition**: Highlight differences from existing functionality
* **Challenge Identification**: Flag potential implementation challenges
* **Expectation Analysis**: Detect shifts in user or stakeholder expectations
* **Constraint Recognition**: Identify new constraints or limitations
* **Risk Assessment**: Evaluate innovation risks based on previous experience

##### 3.3 Impact Assessment Specialist Agent

* **System Coverage**: Analyze impacts across all potentially affected systems
* **Component Analysis**: Identify specific components requiring modification
* **Dependency Mapping**: Trace dependencies between affected components
* **Database Impact**: Assess changes needed to database schemas or data
* **API Changes**: Identify affected APIs and required modifications
* **Service Impact**: Evaluate effects on microservices or other service components
* **Integration Points**: Analyze impacts on third-party integrations
* **Security Implications**: Assess security implications of proposed changes
* **Performance Considerations**: Evaluate potential performance impacts
* **Scalability Effects**: Consider how changes might affect system scalability
* **Backward Compatibility**: Assess effects on backward compatibility
* **Testing Requirements**: Identify specific testing needs for affected systems
* **Deployment Complexity**: Evaluate deployment challenges for each impacted system
* **Rollback Strategy**: Consider rollback implications per system

##### 3.4 Document Generator Agent

* **Format Standardization**: Generate documents matching organizational standards
* **Comprehensive Inclusion**: Include all relevant analyses and findings
* **Visual Elements**: Create appropriate diagrams or visual aids where helpful
* **Reference Linking**: Properly link to referenced Confluence pages
* **Audience Awareness**: Adjust content based on intended audience
* **Clear Organization**: Structure content logically and consistently
* **Implementation Guidance**: Provide clear next steps for implementation teams
* **Risk Mitigation**: Suggest approaches to address identified risks
* **Timeline Estimation**: Provide rough implementation timeline estimates

#### Technical Details

* Agent configuration using YAML
* Model selection with appropriate temperature settings
* Tool access control per agent
* Inter-agent communication protocols

### 4. Task Pipeline

#### Description

Sequential workflow that processes requirements through multiple agents with clear dependencies

#### Requirements

* **Task Sequencing**: Execute tasks in logical order with dependencies
* **Data Transformation**: Format outputs for consumption by subsequent tasks
* **Progress Tracking**: Monitor and report on task completion status
* **Error Recovery**: Handle failures and provide recovery options
* **Logging**: Maintain detailed logs of task execution
* **Performance Monitoring**: Track timing and resource usage per task
* **Result Validation**: Validate task outputs against expected schemas
* **Pipeline Visualization**: Provide visual representation of workflow state

#### Technical Details

* Sequential task execution model
* Input/output schema validation
* Dependency handling
* Timeout and retry mechanisms

### 5. Integration API

#### Description

Interface for submitting requirements and receiving generated documents

#### Requirements

* **RESTful Endpoints**: Standard REST API for submitting new requirements
* **Authentication**: Secure authentication and authorization
* **Status Tracking**: Endpoints to check analysis status
* **Document Retrieval**: APIs to retrieve completed documents
* **Format Options**: Support for different output formats (Markdown, HTML, PDF)
* **Batch Processing**: Support for submitting multiple requirements
* **Webhook Support**: Notifications when analyses complete
* **Rate Limiting**: Protection against excessive API usage
* **Error Reporting**: Clear error messages and problem reporting

#### Technical Details

* API framework (FastAPI/Flask)
* Authentication implementation (OAuth 2.0)
* Asynchronous task handling
* Documentation with OpenAPI

## Technical Requirements

### Vector Database

#### Performance

* Query accuracy >90% using cosine similarity
* Support for at least 100,000 document embeddings
* Sub-second query response time for standard queries
* <3 second response time for complex filtered queries

#### Functionality

* Support for metadata filtering on fields including:
  + Page creation date
  + Last updated date
  + Author
  + Tags/labels
  + Space/project
  + Page type
* Support for hybrid search (vector + keyword)
* Multi-vector search capabilities
* Support for semantic filtering

#### Integration

* REST API access
* Authentication using API keys
* Rate limiting protection
* Monitoring and alerting capabilities

### AI Model Requirements

#### Base Models

* Use of GPT-4o or equivalent for agent reasoning
* Alternative options for cost optimization:
  + Requirements Analyzer: GPT-4o
  + Gap Identifier: GPT-4o
  + Impact Assessment: GPT-4o
  + Document Generator: Claude 3 Sonnet or equivalent

#### Performance

* Context window supporting at least 16,000 tokens
* Agent-specific temperature settings:
  + Requirements Analyzer: 0.2 (more precise)
  + Gap Identifier: 0.3 (balanced)
  + Impact Assessment: 0.3 (balanced)
  + Document Generator: 0.4 (more creative)

#### Integration

* API access with appropriate rate limiting
* Error handling for API failures
* Token usage tracking and optimization

### Performance Requirements

#### Speed

* Process requirement analyses within 3 minutes (simple requirements)
* Process complex requirements within 5 minutes
* Document generation within 1 minute

#### Scale

* Support concurrent processing of up to 10 requirements
* Handle up to 100 requirements per day
* Support Confluence repositories with up to 10,000 pages

#### Availability

* 99.5% system availability during business hours
* 24/7 operation with maintenance windows
* Maximum downtime of 4 hours per month

### Security Requirements

#### Authentication and Authorization

* OAuth 2.0 authentication for API access
* Role-based access control with at least three roles:
  + Admin: Full system access
  + Analyst: Submit requirements and view results
  + Viewer: View results only

#### Data Protection

* Encryption of data in transit (TLS 1.3)
* Encryption of data at rest (AES-256)
* Secure handling of access credentials
* Regular security audits

#### Compliance

* Audit logging of all system operations
* User activity tracking
* Compliance with organizational data handling policies
* Regular penetration testing

## Implementation Details

### Tool Implementation

The search\_confluence\_tool is a critical component that provides:

* Semantic search across vectorized Confluence content
* Configurable similarity threshold
* Metadata filtering options
* Structured response format

class SearchConfluenceTool(BaseTool):

name = "search\_confluence\_tool"

description = "Search for relevant information in Confluence pages based on semantic similarity"

args\_schema = SearchConfluenceInput

def \_\_init\_\_(self, vector\_db\_connection\_string: str, confidence\_threshold: float = 0.7):

super().\_\_init\_\_()

self.vector\_db\_connection\_string = vector\_db\_connection\_string

self.confidence\_threshold = confidence\_threshold

# Initialize vector database connection

def \_run(self, query: str, top\_k: int = 5, filters: Optional[Dict[str, Any]] = None) -> List[Dict[str, Any]]:

"""

Execute semantic search on vectorized Confluence pages.

Args:

query: The search query

top\_k: Number of results to return

filters: Optional metadata filters

Returns:

A list of dictionaries containing document content and metadata

"""

# Implementation details

# Connect to vector database

# Execute search with filters

# Process and format results

# Apply confidence threshold

# Return structured results

### Impact Assessment Process

The Impact Assessment Specialist Agent conducts a comprehensive analysis across all potentially affected systems, including:

#### 1. System Identification

* Identifies all potentially impacted systems based on:
  + Direct mentions in requirements
  + Historical patterns from similar requirements
  + Known dependencies between systems
  + Data flow analysis

#### 2. Per-System Analysis

For each identified system, the agent evaluates:

* **Code Impact**
  + Classes/modules requiring changes
  + Estimated lines of code affected
  + New components needed
  + Deprecated components
* **Database Impact**
  + Schema changes required
  + Data migration needs
  + Query performance implications
  + Storage requirements
* **API Impact**
  + New endpoints needed
  + Modified endpoints
  + Deprecated endpoints
  + Versioning requirements
  + Contract breaking changes
* **Infrastructure Impact**
  + Compute resource changes
  + Storage requirements
  + Network configuration changes
  + Deployment pipeline modifications
* **Integration Impact**
  + Effects on third-party integrations
  + Service mesh configuration changes
  + Event bus modifications
  + Authentication/authorization changes

#### 3. Risk Assessment

For each system, risks are evaluated across dimensions:

* **Technical Risk**
  + Implementation complexity
  + Technical debt implications
  + Testing challenges
  + Performance concerns
* **Operational Risk**
  + Deployment complexity
  + Rollback difficulty
  + Monitoring implications
  + Support requirements
* **Business Risk**
  + User-facing impact
  + Downtime requirements
  + Revenue implications
  + Compliance concerns

#### 4. Implementation Complexity

Each system receives an implementation complexity score based on:

* **Time Estimation**
  + Developer days required
  + QA effort needed
  + Documentation requirements
* **Expertise Requirements**
  + Specialized knowledge needed
  + Team capability assessment
  + Training requirements
* **Dependencies**
  + External dependencies
  + Internal team dependencies
  + Sequencing requirements

#### 5. Structured Output

The impact assessment is provided in a structured format:

{

"impacted\_systems": [

{

"name": "User Authentication Service",

"impact\_level": "high",

"impact\_details": {

"code\_impact": {

"components": ["AuthController", "TokenService"],

"effort\_estimate": "5 developer days",

"description": "Significant changes to token validation logic"

},

"database\_impact": {

"schema\_changes": true,

"migration\_required": true,

"description": "New fields needed in user\_tokens table"

},

"api\_impact": {

"breaking\_changes": true,

"new\_endpoints": 1,

"modified\_endpoints": 2,

"description": "Token refresh endpoint needs modification"

},

"infrastructure\_impact": {

"changes\_required": false,

"description": "No infrastructure changes needed"

},

"integration\_impact": {

"affected\_integrations": ["Mobile App", "Partner API"],

"description": "Token format changes affect all clients"

}

},

"risk\_factors": [

{

"category": "Technical",

"severity": "high",

"description": "Complex changes to critical authentication logic"

},

{

"category": "Operational",

"severity": "medium",

"description": "Requires careful deployment to avoid user lockouts"

},

{

"category": "Business",

"severity": "low",

"description": "Minimal user-visible changes"

}

],

"implementation\_complexity": "high",

"testing\_requirements": "Comprehensive integration testing needed",

"suggested\_approach": "Phase deployment by user segment"

},

{

"name": "Analytics Dashboard",

"impact\_level": "medium",

// Similar detailed structure for each system

}

],

"overall\_assessment": "This requirement primarily impacts authentication services with moderate changes to dependent systems. The high technical risk in the User Authentication Service requires careful planning and testing.",

"suggested\_implementation\_sequence": [

"Update Database Schema",

"Modify Authentication Service",

"Update Client Libraries",

"Deploy Dashboard Changes"

],

"estimated\_total\_effort": "15-20 developer days"

}

### Agent Configuration

The four specialized agents are configured with distinct roles, using a YAML configuration approach:

agents:

- id: requirements\_analyzer

role: "Requirements Analyzer"

goal: "Analyze new requirements against historical data in Confluence"

backstory: >

You are an expert requirements analyst with deep knowledge of the organization's

systems and previous requirements. Your job is to find patterns and similarities between new

requirements and past documentation.

verbose: true

allow\_delegation: true

tools:

- search\_confluence\_tool

llm:

provider: "openai"

model: "gpt-4o"

temperature: 0.2

- id: gap\_identifier

role: "Gap Identifier"

goal: "Identify differences between new requirements and historical data"

# Additional configuration...

- id: impact\_assessment

role: "Impact Assessment Specialist"

goal: "Determine which systems or modules will be affected by the new requirements"

# Additional configuration...

- id: document\_generator

role: "Requirements Document Generator"

goal: "Create comprehensive, well-structured requirements documents"

# Additional configuration...

## Task Implementation

The task pipeline is defined with specific instructions and dependencies:

tasks:

- id: analysis\_task

description: >

Analyze the user-submitted requirement against existing Confluence content.

1. Use the search\_confluence\_tool to find similar historical requirements

2. Identify patterns, terminology, and common characteristics

3. Extract information about previously impacted modules

4. Prepare a structured summary of your findings

Input: {requirement\_text}

Output format:

{

"similar\_requirements": [list of similar requirements with relevance scores],

"common\_terminology": [key terms that appear in historical requirements],

"previously\_impacted\_modules": [modules affected in similar past requirements],

"patterns": [identified patterns in requirement structure or implementation]

}

expected\_output: "A JSON-formatted analysis of the requirement compared to historical data"

agent: requirements\_analyzer

- id: gap\_identification\_task

description: >

Compare the new requirement with retrieved historical data to identify gaps and differences.

# Additional task details...

depends\_on: analysis\_task

# Additional tasks...

## Output Document Structure

The final requirements document follows a standardized structure:

### 1. Requirement Overview

* **Title**: Clear, concise title of the requirement
* **ID**: Unique identifier for the requirement
* **Description**: Detailed description of the requirement
* **Business Value**: Expected business outcomes
* **Priority**: Implementation priority (High/Medium/Low)
* **Requested By**: Stakeholder name and department
* **Date Submitted**: Submission date

### 2. Historical Context

* **Similar Requirements**: References to similar past requirements with Confluence links
* **Common Patterns**: Identified patterns from historical implementations
* **Relevant Terminology**: Glossary of domain-specific terms
* **Previous Approaches**: Summary of how similar requirements were implemented

### 3. Gap Analysis

* **New Elements**: Aspects not found in historical requirements
* **Modified Functionality**: Changes to existing functionality
* **Innovations**: Novel approaches or technologies introduced
* **Potential Challenges**: Areas requiring special attention

### 4. Impact Assessment

* **System Impact Matrix**: Table of all affected systems with impact levels
* **Detailed System Analysis**: Per-system breakdown of impacts (as described above)
* **Cross-System Dependencies**: Relationships between affected systems
* **Implementation Risks**: Identified risks by category
* **Effort Estimation**: Total estimated effort and breakdown by system

### 5. Implementation Guidance

* **Suggested Approach**: Recommended implementation strategy
* **Sequencing Recommendations**: Suggested order of implementation tasks
* **Testing Requirements**: Key testing considerations
* **Deployment Considerations**: Special deployment notes
* **Rollback Strategy**: Approach for handling implementation issues

### 6. Next Steps

* **Required Approvals**: Stakeholders who need to approve the requirement
* **Planning Recommendations**: Suggestions for planning the implementation
* **Resource Recommendations**: Suggested team composition
* **Timeline Considerations**: Factors affecting implementation timeline

## Integration Approach

The system uses a YAML-based configuration approach for CrewAI to simplify deployment and maintenance:

# Tools, agents, and tasks defined in YAML

tools:

- name: search\_confluence\_tool

# Configuration...

agents:

- id: requirements\_analyzer

# Configuration...

tasks:

- id: analysis\_task

# Configuration...

### Loading and Execution

def load\_confluence\_crew() -> Crew:

"""Load the CrewAI configuration from YAML file."""

yaml\_path = "crew\_config.yml"

loader = YamlLoader(yaml\_path)

crew = loader.load()

return crew

def process\_new\_requirement(requirement\_text: str) -> str:

"""Process a new requirement through the CrewAI system."""

requirements\_crew = load\_confluence\_crew()

initial\_context = {"requirement\_text": requirement\_text}

result = requirements\_crew.kickoff(inputs=initial\_context)

return result

## Success Criteria

### Business Metrics

1. **Efficiency Improvement**:
   * 30% reduction in time spent on requirements gathering
   * 25% reduction in requirements clarification cycles
   * 20% faster implementation time for requirements
2. **Quality Improvement**:
   * 40% improvement in requirements consistency
   * 25% reduction in missed dependencies or impacts
   * 35% reduction in requirement-related bugs
   * 30% improvement in requirements completeness
3. **Knowledge Utilization**:
   * 50% increase in references to existing Confluence content
   * 40% reduction in duplicate solutions
   * 30% improvement in cross-team knowledge sharing
4. **Stakeholder Satisfaction**:
   * 85% stakeholder satisfaction with requirements quality
   * 75% developer satisfaction with requirements clarity
   * 80% implementation team satisfaction with impact assessment

### Technical Metrics

1. **Accuracy**:
   * 95% accuracy in identifying similar historical requirements
   * 90% accuracy in system impact identification
   * 85% accuracy in effort estimation
2. **Performance**:
   * <3 minute average processing time for standard requirements
   * <5 minute average processing time for complex requirements
   * <1 second search response time
3. **Reliability**:
   * 99.5% system availability during business hours
   * <1% error rate in document generation
   * 100% data integrity preservation
4. **Adoption**:
   * 80% of teams using the system within 6 months
   * 70% of requirements processed through the system
   * 90% of generated documents accepted without major revision

## Deployment Plan

### Phase 1: Development and Testing (6 weeks)

1. **Week 1-2**: Vector Database Implementation
   * Configure vector database
   * Develop Confluence indexing process
   * Create test dataset and validation process
2. **Week 3-4**: Agent and Tool Development
   * Implement search tool and validate accuracy
   * Configure and test individual agents
   * Develop initial task pipeline
3. **Week 5-6**: Integration Testing
   * End-to-end system testing
   * Performance optimization
   * Error handling improvements

### Phase 2: Limited Pilot (4 weeks)

1. **Week 1**: Deployment Preparation
   * Create documentation
   * Train pilot team
   * Set up monitoring
2. **Week 2-4**: Pilot Execution
   * Deploy with 3-5 selected teams
   * Process 50-100 requirements
   * Gather detailed feedback
   * Iterate on agent prompts and tool functionality

### Phase 3: Full Deployment (6 weeks)

1. **Week 1-2**: System Refinement
   * Implement improvements from pilot feedback
   * Optimize performance
   * Enhance documentation
2. **Week 3-4**: Rollout Preparation
   * Create training materials
   * Develop adoption strategy
   * Prepare support processes
3. **Week 5-6**: Organization-wide Deployment
   * Phased rollout to all teams
   * Ongoing support and training
   * Establish feedback mechanisms

### Phase 4: Continuous Improvement (Ongoing)

1. **Monthly**: System Performance Review
   * Analyze usage metrics
   * Review error logs
   * Implement minor improvements
2. **Quarterly**: Major Review
   * Comprehensive quality assessment
   * Major functionality enhancements
   * Model and prompt improvements

## Maintenance and Support

### Vector Database Maintenance

* **Daily**: Automated health checks
* **Weekly**: Incremental updates of Confluence content
* **Monthly**: Full reindexing of all content
* **Quarterly**: Embedding model evaluation

### Agent System Maintenance

* **Weekly**: Log analysis and error correction
* **Monthly**: Review of agent outputs for quality
* **Quarterly**: Updates to agent prompts based on feedback
* **Semi-annual**: Evaluation of newer LLM options

### Support Process

* **Tier 1**: Self-service troubleshooting via documentation
* **Tier 2**: Dedicated Slack channel for issues
* **Tier 3**: Direct support from implementation team
* **Monitoring**: System health dashboard with alerts

## Conclusion

This Confluence-Powered Requirements Bot represents a significant advancement in requirements management by leveraging AI and organizational knowledge. The system's multi-agent approach ensures comprehensive analysis, accurate impact assessment, and standardized documentation.

The detailed impact assessment process provides unprecedented visibility into how requirements affect various systems, enabling better planning, resource allocation, and risk management. By codifying organizational knowledge and applying it systematically, the system helps ensure consistency, completeness, and quality in the requirements process.

The YAML-based configuration approach ensures flexibility and maintainability, allowing for easy updates as organizational needs evolve or as new capabilities become available in the underlying AI models.