Individual Project - StressSpec

A.I Option: Vibe-Coding

Project Status: SPRINT 1 & SPRINT 2 COMPLETED ✅

Current Project Status

✅ Sprint 1 (4 weeks) - COMPLETED

All core MVP features have been successfully implemented and tested.

Key Achievements:

- Comprehensive risk detection system
- Multi-format reporting
- Configurable rules engine
- Production-ready web application

✅ Sprint 2 (5 weeks) - COMPLETED

All Sprint 2 features successfully implemented and tested.

Key Achievements:

- Complete 8-category risk detection
- Advanced scoring with Top 5 Riskiest Requirements
- Enhanced HTML reporting
- Comprehensive test coverage (241+ tests across 4 test types)

Features Implemented (Sprint 1)

1. ✅ **Input Ingestion** - Accepts .txt or .md requirements
2. ✅ **Requirement Parsing & Labeling** - Unique IDs (R001, R002, etc.)
3. ✅ **Risk Detection Modules** - 6 detector categories
4. ✅ **Configurable Rules** - JSON-driven rules with severity levels

Features Implemented (Sprint 1 - Continued)

5. ✅ **Reporting** - Multi-format output (Markdown, CSV, JSON)
6. ✅ **Severity Scoring** - 5-level severity system
7. ✅ **Testing** - Comprehensive test suite (241+ test cases)
8. ✅ **CLI Interface** - Full command-line interface
9. ✅ **Web UI** - Complete FastAPI web application

Features Completed (Sprint 2)

1. ✅ **Comprehensive Test Suite** - 241+ tests across 4 test types
2. ✅ **Complete 8-Category Risk Detection** - Added Traceability and Scope detectors
3. ✅ **Enhanced Severity Scoring** - Top 5 Riskiest Requirements analysis
4. ✅ **Enhanced HTML Reporting** - Professional styling for stakeholders

Architecture Overview

The project follows **SOLID principles** and implements several **design patterns**:

- **Factory Method Pattern** - `RiskDetectorFactory` for creating detector instances
- **Strategy Pattern** - Extensible risk detection algorithms
- **Template Method Pattern** - Shared workflow in `BaseRiskDetector`
- **Data Classes** - Clean, immutable data structures

Project Structure Overview

```
StressSpec/
├── main.py              # CLI entry point
├── requirements.txt     # Dependencies
├── src/                 # Source code
├── data/                # Configuration & samples
└── tests/               # Test suite (241+ tests)
```

Project Structure - Source Code

```
src/
├── file_loader.py       # File loading and processing
├── requirement_parser.py # Requirement parsing logic
├── analyzer.py          # Runs detectors and aggregates risks
├── models/              # Requirement and Risk data models
├── detectors/           # 8 implemented risk detectors
├── factories/           # Factory Method for detectors
└── reporting/           # Multi-format reporting system
```

Project Structure - Test Suite

```
tests/
├── unit/                # Unit tests (21 files, ~176 tests)
├── integration/         # Integration tests (2 files, ~10 tests)
├── regression/          # Regression tests (1 file, ~30 tests)
└── acceptance/          # Acceptance tests (1 file, ~35 tests)
```

Total: 241+ comprehensive tests

Requirements

Epic User Story

As a project manager,
I want to analyze requirement documents for hidden risks,
so that I can improve requirement quality and reduce project failures before development begins.

User Stories - Input & Parsing

Input Ingestion
As a developer,
I want to upload a .txt or .md file containing requirements,
so that the tool can process them automatically.

Requirement Parsing & Labeling
As a QA engineer,
I want to see each requirement assigned an ID and line number,
so that I can trace flagged risks back to the original text.

User Stories - Risk Detection (1)

Risk Detection - Ambiguity
As a business analyst,
I want to detect vague or ambiguous language in requirements,
so that unclear requirements are highlighted for revision.

Risk Detection - Missing Details
As a business analyst,
I want to identify incomplete requirements,
so that requirements can be completed before development begins.

User Stories - Risk Detection (2)

Risk Detection - Security
As a security engineer,
I want to detect missing authentication and authorization requirements,
so that security gaps are identified early.

Risk Detection - Conflicts
As a project manager,
I want to identify duplicate or contradictory requirements,
so that conflicts can be resolved before development starts.

User Stories - Risk Detection (3)

Risk Detection - Performance
As a performance engineer,
I want to detect missing performance specifications,
so that performance requirements are clearly defined.

Risk Detection - Availability
As a system administrator,
I want to detect missing uptime and reliability requirements,
so that availability expectations are explicitly stated.

User Stories - Configuration & Reporting

Configurable Rules
As a compliance officer,
I want to edit a rules.json file to add or change detection terms,
so that the tool adapts to different domains.

Multi-Format Reporting
As a project manager,
I want to generate reports in Markdown, CSV, and JSON formats,
so that I can share results with the team in multiple ways.

User Stories - Scoring & Interfaces

Severity Scoring
As a team lead,
I want to see each flagged risk assigned a severity level,
so that I can prioritize which requirements need attention first.

CLI Interface
As a developer,
I want to use command-line options to specify input files and output formats,
so that I can integrate the tool into automated workflows.

User Stories - Testing

Testing & Quality Assurance
As a developer,
I want to have comprehensive test coverage for all components,
so that I can ensure the tool works reliably and can be maintained safely.

The Problem

What is the Problem?

Most software project failures stem from **unclear, unrealistic, or incomplete requirements**.

Key Statistics:

- Fixing requirement defects late costs **5–10x more** than early detection
- Around **37%** of enterprise project failures are linked to poor requirements

The Problem (Continued)

Current tools help write or clarify requirements, but they **don't stress-test them** for hidden risks like:

- Ambiguity
- Conflicts
- Compliance gaps
- Scalability issues

Teams often only discover these problems **after coding begins**, when fixing them is expensive and disruptive.

Why is it Important?

Catching requirement problems early:

- **Saves time and money** by preventing costly rework later in development
- **Improves quality** by ensuring requirements are testable, realistic, and aligned with regulations

Why is it Important? (Continued)

- **Supports collaboration** between project managers, analysts, developers, and QA by providing traceable, prioritized risk reports
- **Recruiter/industry relevance**: Demonstrates practical application of AI/rule-based analysis to real-world software engineering challenges

The Solution

How Will You Solve It?

The solution is a Python-based **Requirements Stress Tester** that acts like a **"wind tunnel"** for requirements:

- **Input Ingestion**: Accept .txt or .md files with one requirement per line
- **Requirement Parsing & Labeling**: Assign each requirement an ID (e.g., R001) and line number

Solution Components (1)

- **Risk Detection Modules**: Run checks in categories such as ambiguity, availability, performance, security, conflicts, and scope
- Each check is modular, keyword/regex-driven, and returns flags
- **Configurable Rules**: Store detection rules in rules.json so users can update keywords/conditions without editing code

Solution Components (2)

- **Severity Scoring**: Assign each flag a severity and calculate totals to rank risky requirements
- **Reporting**: Generate outputs in Markdown (human-readable), CSV (sortable), and JSON (machine-readable)
- Reports link each flag back to its requirement ID and evidence

Milestones

Sprint 1 (4 Weeks) → MVP COMPLETED ✅

✅ Feature #1: Input Ingestion

Requirement: The system shall accept .txt or .md files with one requirement per line or bullet.

Status: **COMPLETED** — CLI interface, file loader, and comprehensive error handling implemented.

Sprint 1 - Feature #2 & #3

✅ Feature #2: Requirement Parsing & Labeling

Requirement: The system shall parse lines into requirement objects with IDs (R001...) and line numbers.

Status: **COMPLETED** — Parser module built with ID assignment and line number tracking.

✅ Feature #3: Risk Detection Modules (expanded)

Status: **COMPLETED** — **6 detector categories implemented**: Ambiguity, Missing Detail, Security, Conflict, Performance, Availability.

Sprint 1 - Feature #4 & #5

✅ Feature #4: Configurable Rules (advanced)

Status: **COMPLETED** — JSON-driven configuration with severity levels, enable/disable switches, and comprehensive rule sets.

✅ Feature #5: Reporting (Markdown + CSV + JSON)

Status: **COMPLETED** — Multi-format reporting system with CLI format selection and custom output paths.

Sprint 1 - Feature #6 & #7

✅ Feature #6: Severity Scoring (advanced)

Status: **COMPLETED** — **5-level severity system**: Low, Medium, High, Critical, Blocker with numeric scoring.

✅ Feature #7: Testing & Quality Assurance

Status: **COMPLETED** — Comprehensive test suite with unit, integration, regression, and acceptance tests covering all components with 241+ test cases.

Sprint 1 - Feature #8 & Milestone

✅ Feature #8: Web UI Implementation

Status: **COMPLETED** — Production-ready FastAPI web application with modern UI, responsive design, file upload system, real-time analysis, and comprehensive API.

Milestone Deliverable (End of Sprint 1)

✅ **COMPLETED** — Full MVP end-to-end flow from input → risk detection → multi-format reports with severity scoring + complete web application exceeding original scope.

Sprint 2 (5 Weeks) → Complete 8-Category System ✅

Status: COMPLETED - All Sprint 2 features successfully implemented and tested.

Sprint 2 - Feature #1

✅ Feature #1: Comprehensive Test Suite Implementation

Requirement: The system shall have comprehensive test coverage across unit, integration, regression, and acceptance test types.

Status: **COMPLETED** — Complete test suite implemented with 241+ test cases organized into 4 test categories (unit: 21 files, integration: 2 files, regression: 1 file, acceptance: 1 file). All tests passing with 100% reliability.

Sprint 2 - Feature #2

✅ Feature #2: Risk Detection Modules (Complete 8-Category System)

Requirement: The system shall add Traceability and Scope detection modules to complete the original 8-category plan.

Note: Privacy detector was planned but not implemented. 8 categories achieved via Traceability + Scope detectors.

Status: **COMPLETED** — Traceability and Scope detectors implemented in Week 7. All 8 categories now functional.

Sprint 2 - Feature #3 & #4

✅ Feature #3: Advanced Severity Scoring

Status: **COMPLETED** — Scoring aggregation engine implemented in Week 8. Top 5 Riskiest Requirements available in all report formats.

✅ Feature #4: Enhanced HTML Reporting

Status: **COMPLETED** — Standalone HTML reports with professional styling implemented in Week 9. Includes executive summary and Top 5 section.

Sprint 2 - Feature #5 & Milestone

✅ Feature #5: Documentation & Polish

Status: **COMPLETED** — Comprehensive testing completed in Week 10. All deprecation warnings fixed. Documentation updated.

Milestone Deliverable (End of Sprint 2)

✅ **COMPLETED** — Tool now includes all 8 risk detection categories, advanced scoring with "Top 5 riskiest requirements," professional HTML reporting, and comprehensive test suite with 241+ tests across 4 test types.

Current Implementation Details

Risk Detection Modules (8 Implemented) ✅

All 8 Categories Implemented:

1. **AmbiguityDetector** - Detects vague language and imprecise terms
2. **MissingDetailDetector** - Identifies incomplete requirements and unspecified actors
3. **SecurityDetector** - Flags missing authentication, authorization, and data protection
4. **ConflictDetector** - Finds duplicate and contradictory requirements

Risk Detection Modules (Continued)

5. **PerformanceDetector** - Identifies missing performance specifications
6. **AvailabilityDetector** - Detects missing uptime and reliability requirements
7. **TraceabilityDetector** - Identifies missing requirement IDs and test coverage references (Sprint 2)
8. **ScopeDetector** - Flags scope creep and boundary violations (Sprint 2)

Note: Privacy detector was originally planned but not implemented. However, the 8-category goal was achieved through Traceability and Scope detectors.

Configuration System

- **rules.json**: Comprehensive configuration with 8 detector categories (all categories implemented)
- **Severity Levels**: 5-level system (Low=1, Medium=2, High=3, Critical=4, Blocker=5)
- **Enable/Disable**: Each detector can be individually enabled or disabled
- **Customizable Rules**: Keywords, patterns, and thresholds configurable per detector
- **Future Enhancement**: Multi-domain configuration profiles planned for Sprint 3+

Reporting System

- **Markdown Reporter**: Human-readable reports with risk summaries and Top 5 Riskiest Requirements
- **CSV Reporter**: Structured data for analysis and sorting with score columns and Top 5 summary
- **JSON Reporter**: Machine-readable format for integration with top_5_riskiest array
- **HTML Reporter**: Standalone HTML reports with professional styling for stakeholder presentations (Sprint 2)

Reporting System (Continued)

- **CLI Integration**: Format selection via `--report-format` parameter
- **Web UI Integration**: Interactive report viewing and export capabilities
- **Top 5 Riskiest Requirements**: Available in all 4 report formats (Sprint 2)

Testing & Quality Assurance

Test Coverage: 241+ comprehensive tests organized by test type (100% pass rate)

Test Organization: Tests organized into 4 categories with structured subdirectories:

- **Unit Tests** (`tests/unit/`): 21 test files covering individual components in isolation
- **Integration Tests** (`tests/integration/`): 2 test files for end-to-end workflow testing

Testing & Quality Assurance (Continued)

- **Regression Tests** (`tests/regression/`): 1 test file verifying previously fixed bugs don't reoccur
- **Acceptance Tests** (`tests/acceptance/`): 1 test file for user story and business requirement validation

Unit Test Coverage (21 files)

- Models: Risk, Requirement validation and behavior
- Core Components: FileLoader, RequirementParser, Analyzer, Scoring
- All 8 Detectors: Comprehensive tests for each detector
- Factories: DetectorFactory, ReporterFactory
- Services: StressSpecService orchestration
- Design Patterns: Observer, Chain of Responsibility
- Error Handling: Custom exceptions, error handlers, middleware

Integration Test Coverage (2 files)

- Complete workflow from file loading to report generation
- Cross-format report validation (MD, CSV, JSON, HTML)
- Top 5 Riskiest Requirements integration across all formats
- Multi-detector interaction testing

Regression Test Coverage (1 file)

- Previously fixed bugs (unicode handling, special characters, zero risks, etc.)
- Edge cases that caused issues (long text, whitespace, mixed line endings)
- Data integrity checks (unique IDs, line numbers, text preservation)
- Performance regressions (analysis time, memory usage)
- Version compatibility (report formats, config files)

Acceptance Test Coverage (1 file)

- User Story 1: Upload and Analyze Requirements
- User Story 2: View Risk Details
- User Story 3: Download Reports
- User Story 4: Business Requirements Validation (all 8 risk categories)
- User Story 5: Performance Requirements
- User Story 6: Error Handling from user perspective

Test Infrastructure

- Shared fixtures in `conftest.py` (accessible to all test types)
- TestDataFactory for consistent test data creation
- Comprehensive test documentation and organization
- Pytest best practices throughout

Sprint 2 Achievement: ✅ Comprehensive test suite implemented - 241+ tests across 4 test types, all passing reliably

Usage Examples

Web Interface (Primary Method)

Start the Development Server:

```
# Method 1: Using the run script (recommended)
python web_utils/run_web.py

# Method 2: Direct FastAPI
python -m uvicorn web.main:app --host 127.0.0.1 --port 8000 --reload
```

Web Interface - Access Points

Access Points:

- Main Web Interface: http://127.0.0.1:8000
- API Documentation: http://127.0.0.1:8000/api/docs
- Health Check: http://127.0.0.1:8000/health

Web Interface Features

- File upload with drag-and-drop support
- Real-time analysis progress
- Interactive results display with filtering
- Multi-format report downloads (HTML, Markdown, CSV, JSON)
- Sample file downloads for testing

CLI Usage (Advanced/Integration)

```
# Basic usage with Markdown output
python main.py --file data/sample_requirements.txt --verbose

# Generate CSV report with custom output path
python main.py --file requirements.txt --report-format csv --output analysis.csv
```

CLI Usage (Continued)

```
# Generate JSON report for integration
python main.py --file requirements.md --report-format json --output risks.json

# Generate HTML reports (Sprint 2 feature)
python main.py --file requirements.txt --report-format html --output analysis.html
```

CLI Options

- `--file` - Path to requirements file (required)
- `--report-format` - Output format: `md`, `csv`, `json`, or `html` (default: `md`)
- `--output` - Custom output file path (optional)
- `--verbose` - Show detailed processing information

Performance & Scalability

- **Efficient Processing**: Handles large requirement documents efficiently
- **Memory Management**: Streamlined data structures with dataclasses
- **Error Handling**: Comprehensive error messages and graceful failure handling
- **Extensibility**: Factory pattern allows easy addition of new detectors
- **Web UI Performance**: FastAPI async processing with responsive design

Sprint 2 Enhancement: ✅ "Top 5 Riskiest Requirements" analysis implemented for prioritization

Web Interface Architecture

Technology Stack:

- **FastAPI** - Modern, fast web framework for building APIs
- **Jinja2** - Template engine for HTML rendering
- **HTMX** - Dynamic HTML interactions without complex JavaScript
- **Bootstrap 5** - Responsive UI framework
- **Uvicorn** - ASGI server for FastAPI

Web Interface - Key Components

- **File Upload System** - Handles .txt and .md file uploads with validation
- **Analysis Processing** - Background task processing for requirement analysis
- **Report Generation** - Multi-format report generation (HTML, Markdown, CSV, JSON)
- **API Endpoints** - RESTful API for programmatic access
- **Real-time Updates** - HTMX-powered dynamic UI updates

Web Interface - File Structure

```
web/
├── main.py                    # FastAPI app entry point
├── static/                    # Static files (CSS, JS, images)
├── templates/                 # Jinja2 HTML templates
└── api/                       # API endpoint modules
    ├── upload.py              # File upload endpoints
    ├── analysis.py            # Analysis processing endpoints
    └── reports.py             # Report generation endpoints
```

Setup & Configuration

Automated Setup:

```
python web_utils/setup_web.py
```

Manual Setup Steps:

1. Install dependencies: `pip install -r requirements.txt`
2. Create directories: `mkdir -p uploads logs`
3. Start server: `python web_utils/run_web.py`

Environment Configuration (.env file)

```
# Development settings
DEBUG=True
ENVIRONMENT=development

# Server configuration
HOST=127.0.0.1
PORT=8000
RELOAD=True

# File upload settings
MAX_FILE_SIZE=10485760  # 10MB
ALLOWED_EXTENSIONS=.txt,.md
UPLOAD_DIR=uploads
```

Environment Configuration (Continued)

```
# Analysis settings
ANALYSIS_TIMEOUT=300  # 5 minutes
MAX_CONCURRENT_ANALYSES=5
```

Sprint 2 Development Results

Sprint 2 Completed Successfully

- **Time Investment**: Completed within planned 15-20 hours over 5 weeks
- **Focus Areas**: ✅ Complete 8-category plan + enhanced reporting + advanced scoring
- **Development Strategy**: Followed existing patterns, incremental testing, quality focus
- **Quality Achievement**: ✅ 100% test coverage maintained throughout

Sprint 2 Deliverables - All Completed

1. ✅ **Comprehensive Test Suite** - 241+ tests across 4 test types all passing reliably
2. ✅ **8-Category Risk Detection** - Complete original plan implementation (Traceability + Scope added)
3. ✅ **Advanced Scoring** - Top 5 riskiest requirements identification across all formats

Sprint 2 Deliverables (Continued)

4. ✅ **Enhanced Reporting** - Standalone HTML report generation with professional styling
5. ✅ **Updated Documentation** - README, project docs, test documentation, and user guides updated

Sprint 2 Status: ✅ Complete - Ready for review and presentation

Technical Implementation Details

Design Principles Applied

SOLID Principles

- **Single Responsibility Principle**: Each class has one clear responsibility
  - `FileLoader`: Handles file operations only
  - `RequirementParser`: Handles parsing logic only
  - Each detector handles one specific risk category

SOLID Principles (Continued)

- **Open/Closed Principle**: Open for extension, closed for modification
  - New detectors can be added without modifying existing code
  - Factory pattern allows easy addition of new detector types

SOLID Principles (Continued)

- **Liskov Substitution Principle**: Derived classes can replace base classes
  - All detectors inherit from `BaseRiskDetector` and are interchangeable
- **Interface Segregation Principle**: Clients shouldn't depend on unused interfaces
  - Clean separation between detectors, parsers, and reporters

SOLID Principles (Continued)

- **Dependency Inversion Principle**: Depend on abstractions, not concretions
  - Factory pattern creates detectors based on configuration
  - Reporters implement common interface

Design Patterns

- **Factory Method Pattern**: `RiskDetectorFactory` for creating detector instances
- **Strategy Pattern**: Extensible risk detection algorithms
- **Template Method Pattern**: Shared workflow in `BaseRiskDetector`
- **Data Classes**: Clean, immutable data structures for requirements and risks
- **Observer Pattern**: Event-driven architecture for analysis progress

Code Quality Metrics

- **Test Coverage**: 241+ comprehensive tests across 4 test types (100% pass rate)
  - Unit Tests: 21 files covering all components in isolation
  - Integration Tests: 2 files for end-to-end workflows
  - Regression Tests: 1 file for bug prevention and data integrity
  - Acceptance Tests: 1 file for user story validation

Code Quality Metrics (Continued)

- **Code Organization**: Modular structure with clear separation of concerns
- **Test Organization**: Tests organized by type in structured subdirectories for maintainability
- **Error Handling**: Comprehensive error messages and graceful failure handling
- **Type Hints**: Extensive use of Python type hints for better code clarity
- **Documentation**: Inline comments and docstrings throughout codebase
- **PEP8 Compliance**: Code follows Python style guidelines

Testing Strategy - Unit Tests

1. Unit Tests (`tests/unit/` - 21 files):

- Individual component testing in isolation with mocked dependencies
- Models: Risk and Requirement validation, string representations, methods
- Core Components: FileLoader, RequirementParser, Analyzer, Scoring functions
- All 8 Detectors: Comprehensive tests for each detector
- Factories: DetectorFactory and ReporterFactory creation and configuration

Testing Strategy - Unit Tests (Continued)

- Services: StressSpecService workflow orchestration
- Design Patterns: Observer pattern, Chain of Responsibility pattern
- Error Handling: Custom exceptions, error handlers, middleware
- Edge case validation and boundary testing

Testing Strategy - Integration Tests

2. Integration Tests (`tests/integration/` - 2 files):

- End-to-end workflow testing from file upload to report generation
- Complete pipeline validation (load → parse → analyze → score → report)
- Cross-format report validation (MD, CSV, JSON, HTML)
- Top 5 Riskiest Requirements integration across all formats
- Multi-detector interaction testing
- Real file processing with temporary files

Testing Strategy - Regression Tests

3. Regression Tests (`tests/regression/` - 1 file):

- Previously fixed bugs verification (unicode, special characters, zero risks, etc.)
- Edge cases that caused issues in the past (long text, whitespace, mixed line endings)
- Data integrity checks (unique IDs, line numbers, text preservation, risk evidence accuracy)
- Performance regression prevention (analysis time benchmarks, memory usage)
- Version compatibility (report formats, configuration files)

Testing Strategy - Acceptance Tests

4. Acceptance Tests (`tests/acceptance/` - 1 file):

- User story validation from user perspective
- Business requirements verification (all 8 risk categories detected)
- Performance requirements (analysis completion time, file size handling)
- Error handling from user perspective (clear error messages, recovery)
- Complete user workflows (upload → analyze → download reports)

Test Infrastructure

- Shared fixtures in root `conftest.py` (accessible to all test types)
- TestDataFactory for consistent test data creation
- Organized structure for maintainability and clarity
- Pytest best practices with proper fixture usage

Configuration System

rules.json Structure:

- Detector enable/disable switches
- Severity level definitions
- Keyword and pattern configurations
- Category-specific thresholds
- Customizable detection rules

Configuration Features

- JSON-based configuration (no code changes needed)
- Per-detector customization
- Severity level assignment
- Pattern and keyword management
- Easy domain adaptation (finance, healthcare, etc.)

Reporting System Architecture

Reporter Interface:

- Common interface for all report formats
- Consistent data structure across formats
- Top 5 Riskiest Requirements in all formats
- Extensible design for future formats

Report Formats

- **Markdown**: Human-readable technical documentation
- **CSV**: Spreadsheet-compatible data with score columns
- **JSON**: Machine-readable format for integration
- **HTML**: Standalone reports with professional styling

Report Features

- Executive summary with statistics
- Top 5 Riskiest Requirements section
- Detailed requirements list with risk indicators
- Risk breakdown by category
- Severity color-coding (HTML reports)
- Print-friendly design (HTML reports)

Thank You

StressSpec - Requirements Stress Tester

Project Status: Complete ✅