

# **StressSpec Week 2 Progress Report**

## **Requirements Stress Tester - MVP Implementation**

Individual Project – Jeffrey Perdue

Week 2: Foundation & Core Functionality

9/15-9/21



## Week 2 Milestones - 100% Complete

All 10 Week 2 Requirements Delivered

Milestone	Status	Completion
Python Environment Setup	Complete	100%
CLI Entry Point	Complete	100%
Input Ingestion (.txt/.md)	Complete	100%
Requirement Parsing & Labeling	Complete	100%
Error Handling	Complete	100%
Sample Data Files	Complete	100%
Unit Testing	Complete	100%
Integration Testing	Complete	100%



# Key Metrics & Numbers pt 1

## Code Statistics

- Total Files Created: 15
- Lines of Code: 800+
- Test Coverage: 27 passing tests
- Documentation: 100% annotated
- Error Handling: 6 different error types covered



## Key Metrics & Numbers pt 2

### Functionality Delivered

- File Formats Supported: 2 (.txt, .md)
- Input Validation: 4 validation checks
- Output Formats: 1 (structured console output)
- CLI Options: 2 (--file, --verbose)



# Architecture & Design

## SOLID Principles Applied

-  **Single Responsibility:** Each class has one clear purpose
-  **Open/Closed:** Extensible design for future features
-  **Liskov Substitution:** Proper inheritance patterns
-  **Interface Segregation:** Clean, focused interfaces
-  **Dependency Inversion:** Loose coupling between modules

## Design Patterns Implemented

-  **Factory Pattern:** Requirement object creation
-  **Strategy Pattern:** Extensible file loading
-  **Data Class Pattern:** Clean data structures

# 📁 Project Structure Delivered

```
StressSpec/
├── main.py          # CLI entry point (135 lines)
├── requirements.txt # Dependencies (14 lines)
├── README.md        # Documentation (130 lines)
└── ANNOTATION_GUIDE.md # Learning guide (183 lines)
src/
├── __init__.py      # Package metadata
├── file_loader.py   # File operations (135 lines)
├── requirement_parser.py # Text processing (104 lines)
└── models/
    ├── __init__.py  # Model exports
    └── requirement.py # Data model (82 lines)
data/
└── sample_requirements.txt # Test data
└── sample_requirements.md # Test data
tests/                # 27 passing tests
├── test_requirement.py
├── test_file_loader.py
└── test_requirement_parser.py
└── test_integration.py
```



# Testing Results

## Test Suite Performance

- Total Tests: 27
- Passing Tests: 27 (100%)
- Test Categories: 4
- Coverage Areas: Unit, Integration, Error Handling, Edge Cases

## Test Breakdown

Test Category	Tests	Status
Requirement Model	9	 All Pass
File Loader	9	 All Pass
Requirement Parser	6	 All Pass
Integration	3	 All Pass



# Functionality Demonstrated

## Input Processing

- **File Loading:** .txt and .md files
- **Text Cleaning:** Whitespace removal, comment filtering
- **Validation:** File existence, extension checking
- **Error Handling:** Graceful failure with clear messages

## Requirement Processing

- **ID Assignment:** Automatic R001, R002, R003... generation
- **Line Tracking:** Traceability to original file location
- **Data Validation:** Input sanitization and verification
- **Object Creation:** Structured Requirement objects

## Output Generation

-  **Console Display:** Clean, readable format
-  **Verbose Mode:** Detailed processing information
-  **Error Reporting:** Clear, actionable error messages



# Quality Metrics

## Code Quality

-  **Type Hints:** 100% coverage
-  **Documentation:** Comprehensive docstrings
-  **Error Handling:** 6 different exception types
-  **Code Style:** PEP 8 compliant

## User Experience

-  **CLI Interface:** Professional command-line tool
-  **Help System:** Built-in documentation
-  **Error Messages:** Clear, actionable feedback
-  **Sample Data:** Ready-to-use test files



## Week 2 Deliverable Status

 **MVP End-to-End Flow Complete**

**Input → Processing → Output**

1.  Accept .txt/.md files via CLI
2.  Load and validate files
3.  Parse requirements with IDs and line numbers
4.  Display structured results
5.  Handle errors gracefully

## Sample Output Achieved

```
Successfully parsed 10 requirements:
```

```
-----  
R001: Line 1
```

```
    The system shall allow users to login with email and password
```

```
R002: Line 2
```

```
    The system shall display user dashboard after successful login
```

```
...
```



## Ready for Week 3

### Solid Foundation Established

- **Modular Architecture:** Easy to extend
- **Comprehensive Testing:** Reliable base
- **Clean Code:** Well-documented and maintainable
- **Error Handling:** Robust error management

### Week 3 Preparation

- **Risk Detection Modules:** Architecture ready
- **Configurable Rules:** Extension points identified
- **Reporting System:** Output framework in place
- **Severity Scoring:** Data structures prepared



## Summary Statistics

Metric	Count	Percentage
Week 2 Milestones	10/10	100%
Test Coverage	27/27	100%
Files Created	15	100%
Documentation	Complete	100%
Error Handling	6 types	100%
Design Patterns	3 applied	100%



## Week 2 Success



### All Objectives Met

- **Functional MVP:** Complete end-to-end workflow
- **Quality Assurance:** Comprehensive testing
- **Documentation:** Beginner-friendly annotations
- **Architecture:** SOLID principles applied
- **User Experience:** Professional CLI interface



### Ready for Week 3

The foundation is solid, tested, and documented. Week 2 can focus on adding risk detection capabilities without worrying about core infrastructure.

Week 2 Status: COMPLETE 