Vibration Analyzer Pro

Complete Technical Documentation

Version 1.0.0
Generated: August 11, 2025
Author: Development Team
Student: Professor with Industrial & Systems Engineering Background

Table of Contents

Executive Summary

A professional-grade vibration analysis application that demonstrates industry-standard software engineering practices while solving real industrial problems. This isn't just a "student project" - it's a foundation for production software.

Every design decision was made to reflect real-world industrial software development:

- Maintainability: Easy to modify and extend

- Scalability: Can grow from prototype to enterprise software

- Reliability: Robust error handling and data validation

- Professionalism: Industry-standard patterns and practices

Software Architecture Philosophy

We built this software using layered architecture - each layer has a specific responsibility and doesn't interfere with others:

■■

■ GUI Layer ■ ← User Interface Only

■■

■ Business Logic Layer ■ ← Signal Processing & Analysis

■■

■ Data Access Layer ■ ← File I/O & Data Management

■■

Why this matters in industrial settings:

- GUI changes don't affect signal processing algorithms
- Analysis improvements don't break the user interface
- Data format changes don't require GUI modifications
- Multiple interfaces can use the same analysis engine

Directory Structure Deep Dive

VibrationAnalyzer/

■■■ main.py # Application Entry Point

■■■ requirements.txt # Dependency Management

■■■ src/ # Source Code Organization

■ ■■■ __init__.py # Python Package Marker

■ ■■■ analysis/ # Signal Processing Domain

■ ■ ■■■ __init__.py

■ ■ ■■■ signal_processor.py

■ ■■■ data/ # Data Management Domain

■ ■ ■■■ __init__.py

■ ■ ■■■ mock_generator.py

■ ■■■ gui/ # User Interface Domain

■ ■■■ __init__.py

■ ■■■ main_window.py

■■■ test_data/ # Generated Test Files

■■■ exports/ # Analysis Results Output

Code Architecture Patterns

Model: VibrationProcessor class (business logic)

View: VibrationAnalyzerGUI class (user interface)

Controller: Event handlers connecting GUI to processing

python

```
def run_analysis(self):
# 1. Get data from View (GUI)
sampling_rate = self.sampling_rate_spin.value()
# 2. Process with Model (Analysis Engine)
self.analysis_worker = AnalysisWorker(filepath, sampling_rate)
# 3. Update View with results
self.analysis_worker.analysis_completed.connect(self.on_analysis_completed)
```

Component-by-Component Analysis

Purpose: Core signal processing algorithms for vibration analysis

Key Methods Deep Dive:

python

def __init__(self, sampling_rate: float):

self.fs = sampling_rate

self.nyquist = sampling_rate / 2

Why store sampling rate and Nyquist frequency?

- Nyquist Frequency: Maximum analyzable frequency (fs/2)

- Anti-Aliasing: Prevents frequency analysis errors

- Filter Design: Many filters need sampling rate parameter

python

Technical Concepts Explained

The Analysis Chain:

Raw Data → Time Domain → Frequency Domain → Envelope → Order Tracking

Why this sequence?

1. Time Domain: Basic statistics, data quality check

2. Frequency Domain: Identify frequency components

3. Envelope: Reveal hidden periodicities (bearing faults)

4. Order Tracking: Separate machine speed effects

Common Real-World Issues:

- Flipped time arrays: Data acquisition errors

- Non-uniform sampling: Hardware timing issues

- Missing data points: Sensor dropouts

Design Decisions and Rationale

Why Python over C++/MATLAB/LabVIEW?

Advantages:

■ Rapid Development: Faster prototype to production

■ Scientific Libraries: NumPy, SciPy, Matplotlib ecosystem

■ Maintainability: Readable code, easy to modify

■ Cross-Platform: Works on Windows, Linux, macOS

■ Learning Curve: Easier for non-programmers to understand

Trade-offs:

■ Speed: Slower than C++ for intensive computations

■ Memory: Higher memory usage than compiled languages

Our Verdict: For vibration analysis, the productivity gains outweigh performance costs. Critical loops can be optimized later if needed.

Professional Development Practices

Why Git from Day 1?

- Change Tracking: Every modification is recorded
- Backup: Remote repository protects against data loss
- Collaboration: Multiple developers can work simultaneously
- Branching: Experiment with features without breaking main code

Professional Workflow:

bash

git add . # Stage changes

git commit -m "Add feature" # Local save with description

git push # Backup to remote repository

Types of Documentation:

1. Code Comments: Explain complex algorithms

Future Extensibility

Current Architecture Supports:

python

```
class DatabaseReader:
def load_vibration_data(self, machine_id, date_range):
# Connect to industrial database
return time_data, signal_data
processor = VibrationProcessor(sampling_rate)
time_data, signal_data = database_reader.load_vibration_data(...)
```

Threading Architecture Enables:

python

Learning Outcomes

1. Software Architecture
- Layered design principles
- Separation of concerns
- Object-oriented programming patterns
2. Signal Processing
- FFT analysis and windowing
- Time-domain statistical analysis
- Envelope analysis for fault detection
- Order tracking for rotating machinery
3. GUI Development
- Professional interface design
- Threading for responsiveness
- Event-driven programming
4. Industrial Practices

Conclusion

This isn't just a "toy application" - it demonstrates real-world software engineering practices:

■ Scalable Architecture: Can grow from prototype to enterprise software

■ Industry Standards: Uses established patterns and practices

■ Quality Engineering: Robust error handling and validation

■ Professional Tools: Git, professional GUI framework, comprehensive documentation

■ Real Applications: Solves actual industrial vibration analysis problems

1. Separation of Concerns: Each component has a single, well-defined responsibility
2. DRY (Don't Repeat Yourself): Common functionality is centralized and reused
3. SOLID Principles: Objects are well-designed and extensible
4. User-Centered Design: Interface designed for actual industrial users
5. Defensive Programming: Assumes inputs may be invalid and handles gracefully