I want you to design and implement a modular, reusable software tool that can repeatedly analyze project files in my workspace and support three operation modes. Supported file formats:

Markdown (\*.md)

Excel (\*.xlsx)

Microsoft Project (\*.mpp)

Operation modes:

Document Check: Scan the configured project directory, verify the presence of required documents (e.g., Project Charter, Scope Statement, Risk Management Plan, Work Breakdown Structure, Roadmap, Stakeholder Register), and output a report listing missing files or mismatched formats.

Status Analysis: Extract key data from available files:

Risks and their status from Risk Management Plan

Deliverables from Work Breakdown Structure

Milestones and deadlines from Roadmap or .mpp files

Stakeholders and roles from the Stakeholder Register Compile this into a consolidated status report (Markdown & Excel).

Learning Module Presentation: Present short predefined learning units on project management topics (e.g., risk management, stakeholder analysis, scheduling) based on stored Markdown content.

Technical requirements:

Write the program in Python 3, production-ready, following PEP8 style guidelines.

Implement a modular architecture with separate modules for:

File handling (Markdown, Excel, MPP)

Data extraction logic for each document type

Learning module presentation

Report generation (Markdown & Excel)

Configuration management & logging

Use a configuration file (YAML or JSON) to define:

Required documents & expected file names

Paths to project directories

Operation mode selection

Output format options

Automatically detect available files and process them according to the selected mode.

Include comprehensive comments and docstrings for maintainability.

Include error handling for missing files, unsupported formats, or parse errors.

For .mpp files, use a Python library capable of reading Microsoft Project files (e.g., mpparser, pywin32 if Windows-only, or a suitable open-source alternative).

Output:

Provide a complete Python project folder structure with all modules.

Include an example config.json or config.yaml with placeholders.

Include a README.md with setup instructions and usage examples for all three modes.

Do not run the analysis — only generate the reusable code and example configuration.

Future-proofing:

The tool must be easy to extend for new file types (e.g., CSV, PDF) and new learning modules.

The learning module component should load content dynamically from Markdown files, so adding new topics doesn’t require code changes.

**Kiro**

I'll help you create a spec for this modular project management analysis tool. Based on your detailed requirements, I'll call this feature "pm-analysis-tool" and start by creating the requirements document.

Created

requirements.md

I've created the initial requirements document for your PM Analysis Tool. The requirements are structured with clear user stories and detailed acceptance criteria in EARS format, covering all the key aspects you mentioned:

* Document verification and compliance checking
* Status analysis with data extraction from multiple file types
* Learning module presentation system
* Modular architecture with separate concerns
* Configuration-driven behavior
* Automatic file detection and processing
* Comprehensive error handling and logging
* Python best practices and code quality
* Future extensibility for new file types and learning content

**Do the requirements look good? If so, we can move on to the design.**

Please add, that the software system shoul automaticall detect, which mode makes sense. Only if the project manager wants to do a specific thing it should (for example) execute a status analysis without all docments provided.

I'll add that intelligent mode detection requirement to the requirements document.

Accepted edits to

requirements.md

Perfect! I've added Requirement 9 that addresses the intelligent mode detection you requested. The system will now automatically analyze available files and suggest the most appropriate operation mode, while still allowing explicit mode selection when needed.

**Do the requirements look good now? If so, we can move on to the design.**

Great! Now I'll move on to creating the design document. Let me first read the requirements to ensure the design addresses all aspects properly.

Read file(s)

requirements.md

Now I'll research the key technologies and libraries needed for this project to inform the design decisions.

Searched workspace

Searching for existing project files to understand the current file structure and formats

Searched workspace

Examining the Bahnprojekt directory to understand the current file types and structure

Perfect! I can see the actual project structure with the exact file types we need to support. Now I'll create the comprehensive design document.

Created

design.md

I've created a comprehensive design document that addresses all the requirements from the requirements document. The design includes:

**Key Design Decisions:**

1. **Modular Architecture**: Clear separation of concerns with dedicated modules for file handling, data extraction, processing, and reporting
2. **Plugin-Based System**: Extensible file handlers and processors that can be easily extended for new formats
3. **Intelligent Mode Detection**: Automatic analysis of available files to recommend the most appropriate operation mode
4. **Comprehensive Error Handling**: Graceful degradation with detailed logging and user-friendly error messages
5. **Configuration-Driven**: YAML-based configuration for easy customization without code changes

**Technical Highlights:**

* Factory pattern for processors and file handlers
* Abstract base classes ensuring consistent interfaces
* Structured data models for all domain objects
* Multiple fallback options for Microsoft Project file handling
* Performance optimizations including lazy loading and caching
* Comprehensive testing strategy with unit and integration tests

The design leverages modern Python practices and includes detailed class interfaces, data models, and configuration schemas that will guide the implementation phase.

**Does the design look good? If so, we can move on to the implementation plan.**

Excellent! Now I'll create the implementation plan based on the design document. Let me first read the design to ensure the tasks align properly with the architecture.

Read file(s)

design.md

Now I'll create the implementation plan with detailed coding tasks based on the design document.

Created

tasks.md

I've created a comprehensive implementation plan with 16 main tasks and several sub-tasks that systematically build the PM Analysis Tool. The plan follows a logical progression:

**Key Implementation Strategy:**

1. **Foundation First**: Start with project structure, data models, and utilities
2. **Core Infrastructure**: Build configuration management and file scanning
3. **Extensible Architecture**: Create abstract base classes for future extensibility
4. **File Processing**: Implement handlers for each supported file format
5. **Data Extraction**: Build specialized extractors for each document type
6. **Intelligence Layer**: Add mode detection and processing logic
7. **Reporting System**: Create output generation for multiple formats
8. **Integration**: Wire everything together with the core engine and CLI
9. **Quality Assurance**: Add comprehensive testing and documentation

**Incremental Development Approach:**

* Each task builds on previous tasks and can be tested independently
* Sub-tasks allow for focused development of complex components
* Early testing ensures quality throughout development
* Clear requirement references link each task to specific user needs

The plan ensures all requirements are addressed while maintaining the modular, extensible architecture outlined in the design document.

**Do the tasks look good?**

Perfect! The spec for your PM Analysis Tool is now complete. I've successfully created:

1. **Requirements Document** - Detailed user stories and acceptance criteria covering all functionality
2. **Design Document** - Comprehensive architecture with modular components, data models, and technical specifications
3. **Implementation Plan** - 16 structured tasks that build the system incrementally with clear dependencies

The spec covers your complete vision for a modular, reusable tool that can:

* Automatically detect the best operation mode based on available files
* Perform document verification, status analysis, and learning module presentation
* Handle multiple file formats (Markdown, Excel, Microsoft Project)
* Provide extensible architecture for future enhancements
* Follow Python best practices with comprehensive error handling

**Next Steps:** You can now begin executing the implementation tasks. To start development, open the tasks.md file in .kiro/specs/pm-analysis-tool/ and click "Start task" next to any task item to begin implementation. I recommend starting with Task 1 to establish the project foundation.

The spec provides all the context needed for systematic development of your production-ready PM analysis tool!