Auto ProjectManagement System

Intelligent project management with automated monitoring and GitHub integration

Python 3.8+ | MIT License | 85%+ Test Coverage | Complete Docs

24
Core Modules

15K+
Lines of Code

25+
API Endpoints

50+
Config Options

Auto Sync

Live Monitoring

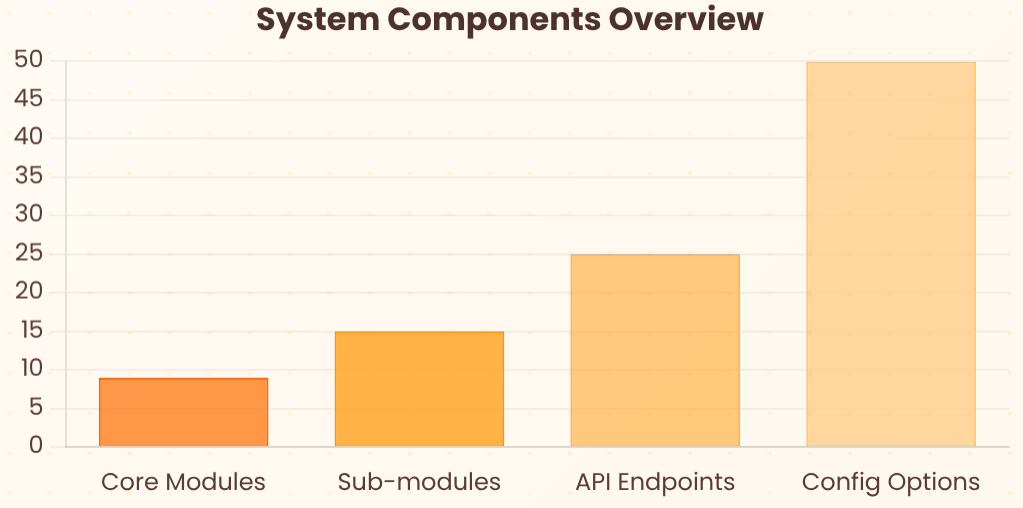
System Overview

A comprehensive Python-based automated project management system

Key Features

- 100% Automated Project Management
- Real-time Risk Assessment
- Intelligent Resource Allocation
- Continuous Progress Tracking
- GitHub-Native Integration
- Automatic Wiki Documentation
- JSON-Driven Configuration
- CLI-Based Interface

System Statistics

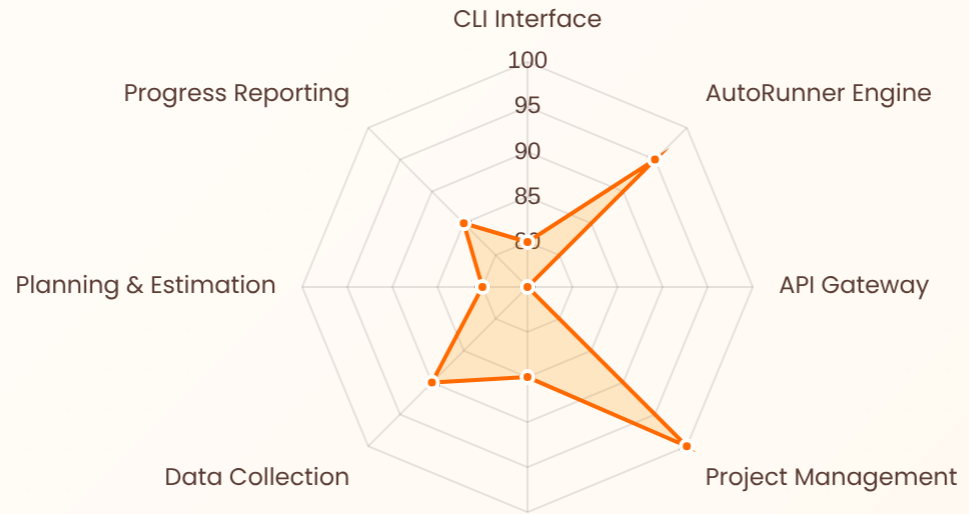


24	15K+	85%+
Core Modules	Lines of Code	Test Coverage

Architecture & Design

System structure and data flow

High-Level System Architecture



Core Modules

- **PMS** - Project Management System
- Communication Risk Module
- Data Collection & Processing
- Planning & Estimation

Integrated Services

- GitHub Integration
- JSON Database NoSQL
- Machine Learning Algorithms
- Report Generator

Data Flow

Input Files — Change Detection — Impact Analysis — Risk Calculation — Progress Update — Storage

Core Modules

Key components of the AutoProjectManagement System

Project Management System

Central orchestrator managing all project operations

- **ProjectManagementSystem** - Main controller
- **Project** - Project data structure
- **Task** - Task management

Communication Risk Module

Identifies and mitigates communication risks in projects

- **Code Conflicts** - Detected via merge frequency
- **Knowledge Silos** - Detected via commit distribution
- **Communication Gaps** - Issue response time

Data Collection & Processing

Collects and processes project data from multiple sources

- File system changes
- Git commit history
- GitHub issues and PRs

Progress Reporting

Generates comprehensive progress reports

- **Daily** - Every 24 hours (Markdown)
- **Weekly** - Every 7 days (PDF/HTML)
- **Monthly** - Every 30 days (PDF/HTML)

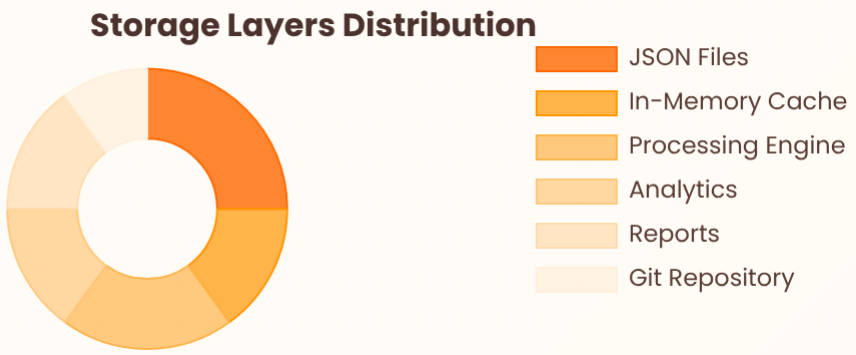
Other Key Modules

- Planning & Estimation
- Quality Commit Management
- Resource Management
- Task Workflow Management
- Utility Modules
- API Gateway

Data Management

Storage architecture and file organization

Storage Architecture



1. **JSON Files** - Primary data storage
2. **In-Memory Cache** - Fast data access
3. **Processing Engine** - Data analysis
4. **Git** - Version control & history

File Organization

Folder Structure

```
.auto_project/
├── config/
│   ├── auto_config.json
│   └── module_configs/
├── data/
│   ├── projects.json
│   ├── tasks.json
│   └── analytics.json
├── logs/
├── reports/
└── backups/
```

- **Configuration:** System & module settings in JSON files
- **Data:** Project info, tasks & analytics
- **Reports:** Daily, weekly & monthly reports
- **Backups:** Daily & weekly backup versions

JSON Configuration Schema

```
{ "project": { "id": "unique_identifier", "name": "Project Name",
"description": "Project description", "start_date": "2024-01-01",
"end_date": "2024-12-31", "status": "active|paused|completed", "priority":
"high|medium|low", "team_members": ["member1", "member2"], "milestones": [
{ "id": "milestone_1", "name": "Phase 1 Complete", "target_date": "2024-
06-01", "status": "pending" } ] } }
```

- **id** String
- **name** String
- **status** Enum
- **priority** Enum
- **team_members** Array
- **milestones** Object Array

Installation & Setup

Installation and initial configuration steps

Prerequisites

- Python 3.8 or higher
- Git installed & configured
- GitHub account with repo access

Installation Methods

PyPI Installation (Recommended)

```
pip install autoprojectmanagement
```

From Source

```
# Clone repository
git clone https://github.com/your-username/AutoProjectManagement.git
cd AutoProjectManagement

# Create virtual environment
python -m venv venv
source venv/bin/activate # Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Install package
pip install -e .
```

Docker

```
# Build Docker image
docker build -t autoprojectmanagement .

# Run container
docker run -v $(pwd)/my_project:/app/project autoprojectmanagement
```

Initial Configuration

1. **`autoproject init`**
   Initialize new project
2. **`autoproject config --github-token YOUR_TOKEN`**
   Configure GitHub integration
3. **`autoproject setup --project-name "MyProject"`**
   Set up project

Usage Guide

Basic commands and configuration options

Project Management Commands

`autoproject init --name "MyProject"`

Initialize new project

`autoproject task add --title "Feature X" --priority high`

Add new task

`autoproject status`

View project status

`autoproject report generate --type weekly`

Generate progress report

Wiki Synchronization

`autoproject wiki sync --repo-owner user --repo-name repo`

Sync docs to GitHub Wiki

`autoproject wiki sync --dry-run`

Preview changes (dry run)

`autoproject wiki sync --force`

Force full sync

Main Configuration (auto_config.json)

```
{ "system": { "check_interval": 300, "auto_commit_threshold": 5, "report_interval": 86400 }, "github": { "token": "your_github_token", "repo_owner": "your-username", "repo_name": "your-repo" }, "notifications": { "slack_webhook": "https://hooks.slack.com/...", "email_enabled": true } }
```

Project Configuration (project.json)

```
{ "project": { "name": "MyProject", "description": "Project description", "start_date": "2024-01-01", "end_date": "2024-12-31", "team_members": ["developer1", "developer2"] }, "tasks": [], "milestones": [] }
```

Advanced Usage

Custom Module Development

Create custom modules by inheriting from **BaseModule** and implementing the **process()** method

API Integration

Use **GitHubIntegration** class to interact with GitHub API and create issues and pull requests

Testing

How to run tests and test structure

Running Tests

`pip install -r requirements-dev.txt`
Install test dependencies

`pytest tests/`
Run all tests

`pytest tests/code_tests/UnitTests/`
Run specific test category

`pytest --cov=autoprojectmanagement tests/`
Run with coverage

`pytest tests/code_tests/IntegrationTests/`
Run integration tests

Test Types

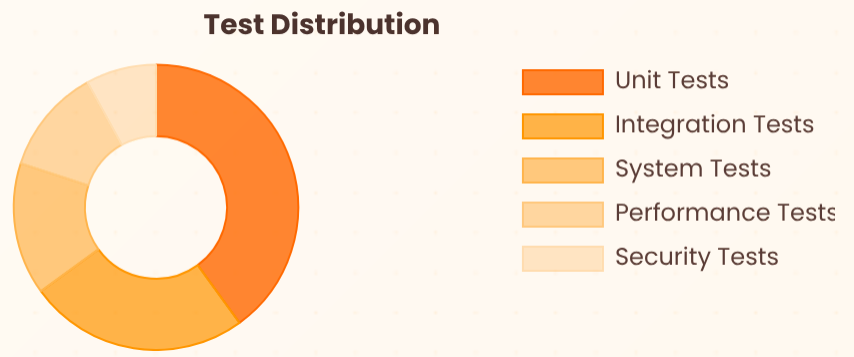
`UnitTests`
Individual component tests

`IntegrationTests`
Module interaction tests

`SystemTests`
Complete system tests

`PerformanceTests`
Speed and performance tests

`SecurityTests`
Security vulnerability tests




```
tests/
├── code_tests/
│   ├── 01_UnitTests/
│   ├── 02_IntegrationTests/
│   ├── 03_SystemTests/
│   ├── 04_PerformanceTests/
│   └── 05_SecurityTests/
├── conftest.py
├── pytest.ini
└── README.md
```

UnitTests: Test individual system components

IntegrationTests: Test module interactions

conftest.py: Shared fixtures and configurations

pytest.ini: Pytest configuration

Writing Tests

```python
import pytest
from autoprojectmanagement.main_modules import ProjectManagementSystem

def test_project_creation():
    pms = ProjectManagementSystem()
    project = pms.create_project("Test Project")
    assert project.name == "Test Project"
    assert project.status == "active"
```