Final Project Guidelines and Requirements

FTP Project for Computer Science Doctoral Students

1. PROJECT OVERVIEW

As part of the Fundamentals of Programming course, you are required to complete a comprehensive software development project that demonstrates your mastery of programming concepts and best practices covered throughout the course.

Your project must be a functional application, implementing solutions to computational problems within your specific domain of study.

2. PROJECT OBJECTIVES

The primary goals of this project are to:

- Design and implement a complete solution addressing a research-related problem
- Demonstrate proficiency in Python programming and software engineering principles
- Apply structured, modular, and object-oriented programming paradigms
- Follow industry-standard collaboration and development best practices
- Create reusable, maintainable code

3. PROJECT REQUIREMENTS

Your project must incorporate most of the 5 course components, demonstrating comprehensive understanding of the material.

4. PROJECT SCOPE AND TOPIC SELECTION

Topic Selection:

Your project must address a computational problem. Consider:

- What repetitive computational tasks could be automated in my research?
- What data processing or analysis tools would benefit my work?
- What simulations or modeling tools do I need?

5. PROJECT DELIVERABLES

Your final submission must include:

A. Application Implementation (50%)

Code Repository containing:

- All source code files (.py)
- Database schema files (if applicable)
- Configuration files
- Sample data files
- Requirements.txt or environment.yml
- .gitignore file

Quality Metrics:

- Clean, well-structured code
- Functional implementation meeting all requirements
- Passing all unit tests
- No critical bugs or errors

B. Technical Documentation (25%)

1. README.md (Primary documentation)

- Project title and description
- Problem statement and motivation
- Features and functionality
- Running instructions
- Usage guide with examples
- Dependencies and requirements
- Project structure overview
- Known issues and limitations
- Future enhancements
- Author information and license

2. Technical Report (5-10 pages) containing:

- **Introduction:** Research context and computational problem
- **Implementation Details:** Key algorithms, data structures, and design decisions
- **Database Design:** ER diagrams and schema (if applicable)

- **Testing Strategy:** Test cases and results
- **Challenges and Solutions:** Technical obstacles encountered
- **Performance Analysis:** Time/space complexity, benchmarks
- **Conclusion:** Project outcomes and contributions to thesis work

3. API Documentation:

- Docstrings for all modules, classes, and functions
- Generated documentation (e.g., using Sphinx) - optional but encouraged

C. Demonstration & Presentation (15%)

Live Demonstration (10-12 minutes):

- Problem overview and research relevance
- System architecture walkthrough
- Live demonstration of key features
- Code walkthrough of interesting implementation details
- Testing demonstration
- Q&A session (3-5 minutes)

Presentation Slides:

- Clear, professional slides
- Code snippets highlighting key implementations
- Visual diagrams (architecture, class diagrams, flowcharts)
- Demo screenshots or video

D. Version Control History (10%)

Git Repository Requirements:

- Minimum 5 meaningful commits
- Clear commit messages following conventions
- Logical development progression
- Proper use of .gitignore
- Branch usage (if applicable)
- Tagged releases (optional)

6. EVALUATION CRITERIA

Your project will be assessed based on:

Criterion	Weight	Description
Problem Relevance	10%	Clear connection to doctoral research and computational significance
Code Quality	20%	Clean, readable, maintainable code following best practices
OOP Design	15%	Effective use of classes, proper architecture, design patterns
Functionality	15%	Complete, bug-free implementation meeting requirements
Modularity	10%	Well-organized modules with clear separation of concerns
Database/File Handling	10%	Robust data persistence and retrieval
Testing & Error Handling	10%	Comprehensive tests, proper exception handling
Documentation	15%	Clear, complete documentation (README, docstrings, report)
Version Control	5%	Meaningful commit history, proper Git usage

7. TECHNICAL REQUIREMENTS

Programming Language:

- Python 3.8 or higher (required)
- Additional languages allowed for specific components (e.g., SQL for databases)

Minimum Project Complexity:

- Minimum 3 modules/files
- At least 3 custom classes
- Minimum 10 functions
- Database integration OR complex file handling
- At least 10 unit tests

Recommended Libraries:

You may use standard and third-party libraries including but not limited to:

- **Data Handling:** pandas, numpy, csv, json
- **Database:** sqlite3, SQLAlchemy, psycopg2

- **Testing:** unittest, pytest
- **Documentation:** sphinx (optional)
- **GUI (optional):** tkinter, PyQt, Streamlit
- **Web (optional):** Flask, FastAPI
- **Scientific Computing:** scipy, scikit-learn (if relevant)

7. DEVELOPMENT BEST PRACTICES

Follow these practices throughout development:

Code Organization:

```
project_name/
│
├── README.md
├── requirements.txt
├── setup.py (optional)
├── .gitignore
│
├── src/
│   ├── __init__.py
│   ├── main.py
│   ├── module1.py
│   ├── module2.py
│   └── utils.py
│
├── tests/
│   ├── __init__.py
│   ├── test_module1.py
│   └── test_module2.py
│
├── data/
│   ├── sample_data.csv
```

```
│   └── database_schema.sql
│
├── docs/
│   ├── technical_report.pdf
│   └── user_guide.md
│
└── examples/
    └── usage_example.py
```

Commit Message Convention:

feat: Add new feature

fix: Bug fix

docs: Documentation updates

test: Add tests

refactor: Code refactoring

style: Formatting changes

8. ACADEMIC INTEGRITY

- All code must be your original work
- Properly attribute any external code, libraries, or algorithms
- Document any AI assistance (e.g., GitHub Copilot, ChatGPT) and explain the code
- Collaboration is allowed for discussion, but implementation must be individual
- Plagiarism will result in project failure and academic consequences

Acceptable:

- Using standard libraries and frameworks
- Consulting official documentation
- Discussing design approaches with peers
- Seeking debugging help during office hours

Unacceptable:

- Copying code from classmates

- Submitting code written by others
- Using complete project solutions from online sources
- Having someone else write substantial portions of your code

9. SUBMISSION FORMAT

Submit via [GitHub & PADOC]:

1. **Git Repository URL** containing all code and documentation
2. **Technical Report** (PDF)
3. **Presentation Slides** (PDF or PPTX)

Repository Requirements:

- Public or instructor-accessible private repository
- All commits pushed before deadline
- Complete README.md in repository root
- All dependencies listed in requirements.txt

File Naming Convention:
Repository: lastname_firstname_ftp_project
Report: LastName_FirstName_Technical_Report.pdf
Slides: LastName_FirstName_Presentation.pdf

10. CONCLUSION

This project is your opportunity to apply everything you've learned in FTP to create a meaningful software tool that directly supports your doctoral research. By building a well-designed, fully functional application, you'll not only demonstrate programming proficiency but also create a reusable tool that can accelerate your thesis work.

Focus on writing clean, maintainable code that follows best practices. Remember: good software engineering is not just about making code work—it's about making code that others (including your future self) can understand, maintain, and extend.

Happy coding!