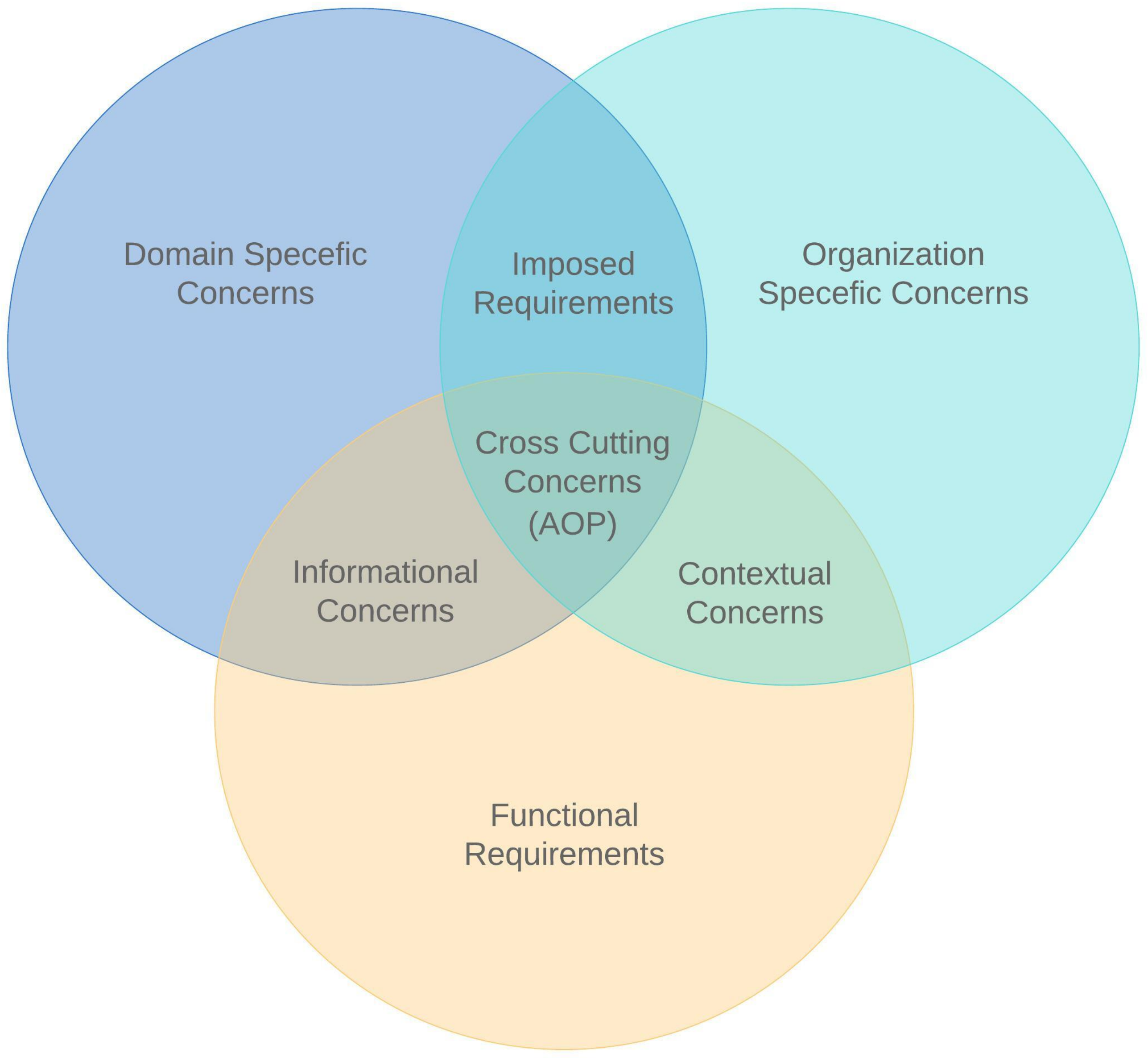
Software Requirement Specifications

1. Introduction

1.1 Purpose of the Project

The objective of this project is to create a comprehensive tool that offers advanced functionalities to address cross-cutting concerns, such as Logging, Profiling, Caching, and Parallelization, in software programs. By leveraging our tool, users can streamline their application's business logic and delegate other concerns to our tool, thereby reducing the development time and effort required. Our tool will provide a robust and efficient solution for handling cross-cutting concerns, allowing programmers to focus on their core application logic and benefit from improved productivity and code maintainability.



Aspect-Oriented Programming
Addressing Cross-Cutting Concerns Across Contexts and Requirements

1.2 Scope of the Project

The scope of the project encompasses the development of the AspectJ tool, including the following key deliverables.

1. **AspectJ Feature Development:** The project will involve designing aspects for various features that address cross-cutting concerns, such as Logging, Profiling, Caching, and Parallelization.
2. **Testing:** The project will thoroughly test and validate the tool to ensure its reliability, performance, and compatibility with various environments and use cases. This will involve unit testing to ensure that the tool meets the required quality standards.
3. **Deliverables:** The project's final deliverables will include the complete AspectJ tool, its documentation, test cases, and any other relevant artifacts. It will be packaged and distributed in a suitable format, such as a JAR file, for easy integration into users' projects.

1.3 Intended Audience

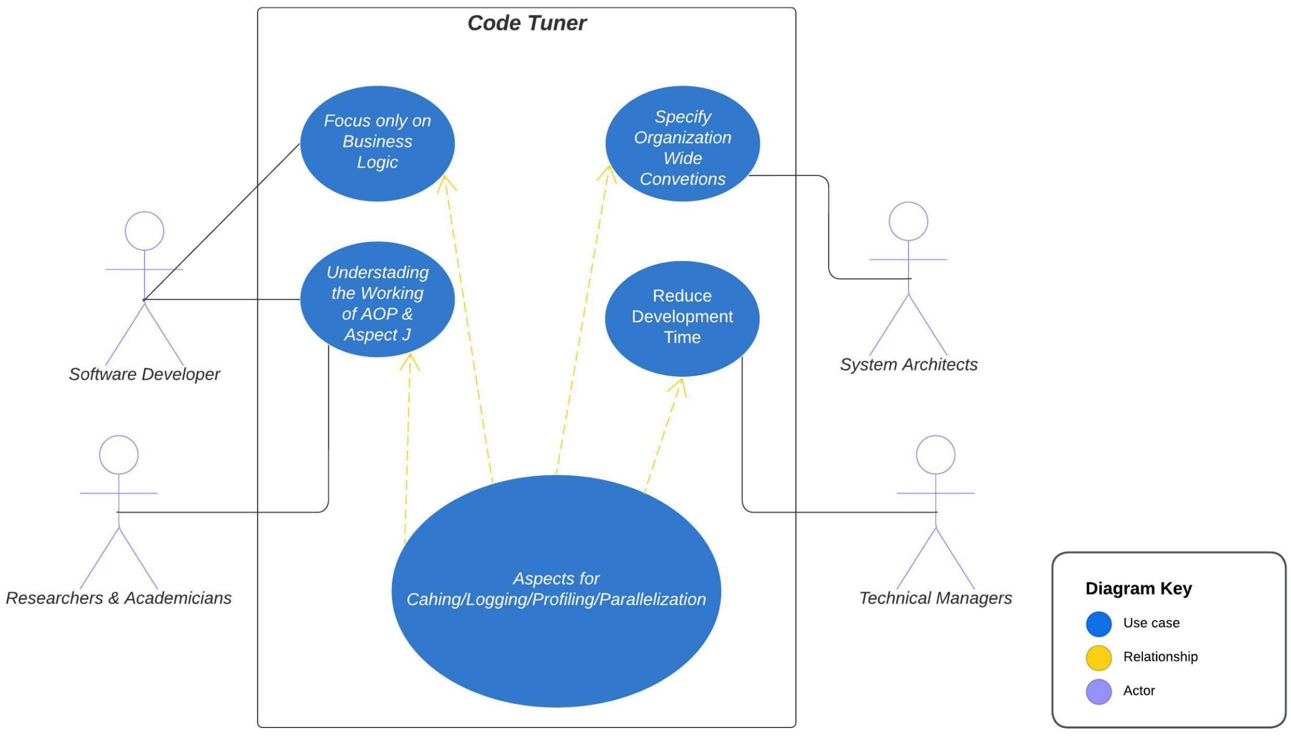
The intended audience for this project includes:

1. **Software Developers:** The tool is designed for software developers who work with Java applications and leverage the AspectJ framework for implementing cross-cutting concerns. They can use the tool to simplify the development of features like Logging, Profiling, Caching, and Parallelization in their applications, allowing them to focus on the core business logic.

2. **System Architects:** System architects who are responsible for defining the overall structure and design of software applications can benefit from the AspectJ tool. They can use the tool to enforce cross-cutting concerns across the system architecture and ensure consistent implementation of functionalities like Logging, Profiling, Caching, and Parallelization in the application.

3. **Technical Managers:** Technical managers who oversee software development projects can benefit from the AspectJ tool as it can help in reducing development time and effort by providing ready-to-use functionalities for cross-cutting concerns.

4. **Researchers and Academics:** Researchers and academics in the field of software engineering, aspect-oriented programming (AOP), and cross-cutting concerns can find value in the AspectJ tool. They can use the codebase as a reference or a tool for their research, experimentation, or educational purposes, and gain insights into the practical implementation of cross-cutting concerns in Java applications.


Code Tuner
Focus only on Business Logic
Specify Organization Wide Convetions
Understading the Working of AOP & Aspect J
Reduce Development Time
Aspects for Caching/Logging/Profiling/Parallelization
Software Developer
System Architects
Researchers & Academicians
Technical Managers
Diagram Key
Use case
Relationship
Actor

Use Case Diagram

2. Overall Description of the Project

This project aims to create a solution that offers advanced functionalities to address cross-cutting concerns in software programs, such as Logging, Profiling, Caching, and Parallelization. The solution will enable software developers to streamline their application's business logic and delegate other concerns to our code, reducing the development time and effort required.

2.1 Novelty & Value Addition

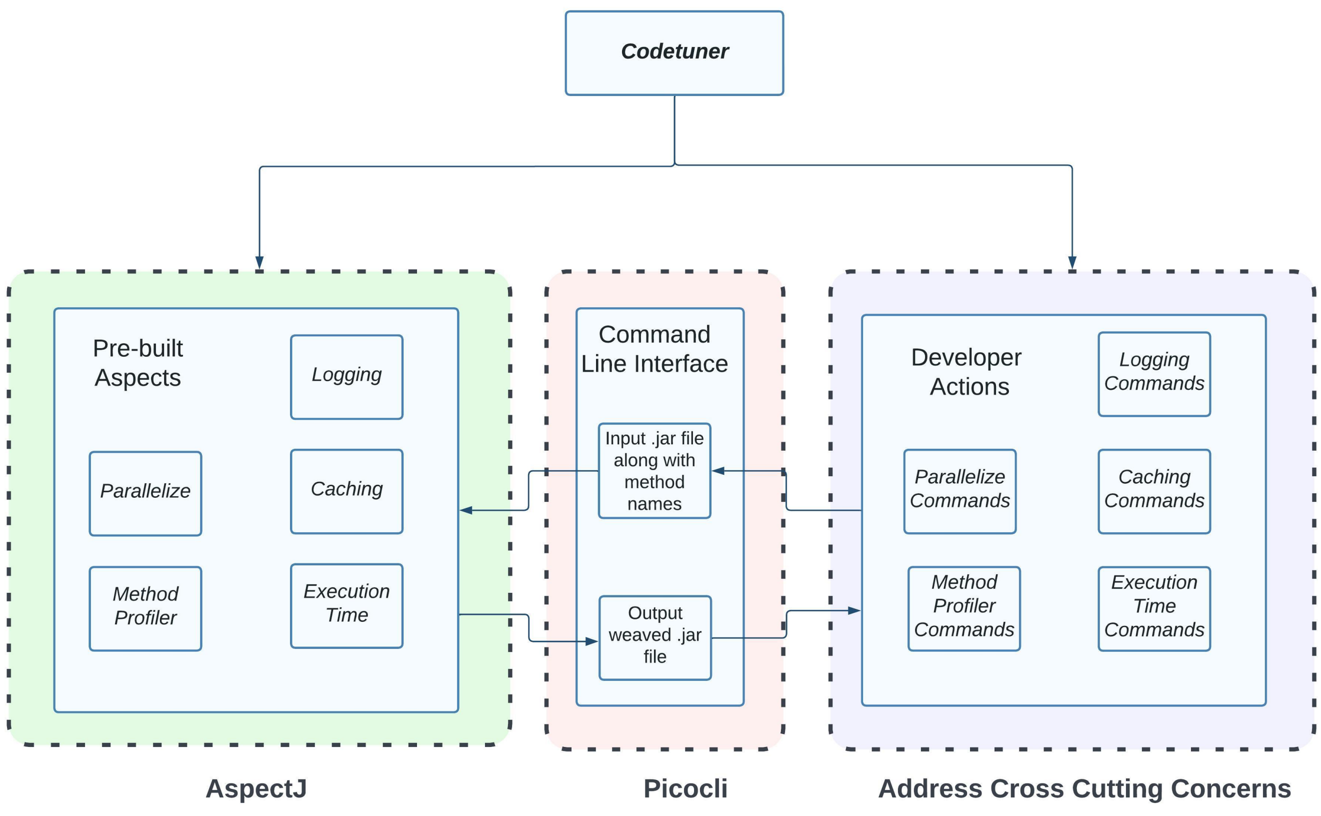
The current projects available in the domain offer limited solutions and do not provide a comprehensive one-stop solution for addressing a variety of cross-cutting concerns, such as logging, caching, profiling, and more. The project aims to fill this gap and provide a comprehensive AspectJ tool that can cater to various cross-cutting concerns. It will benefit all the stakeholders mentioned above, including software developers, system architects, technical managers, and researchers/academics in their respective domains.

2.2 Operating Environment

The application is designed and tested to run on the following environment

- `AJC Compiler Version` : 1.9.X
- `Java` : 18.X
- `Operating System` : Ubuntu LTS

3. Block Diagram



4. Functional Requirements

4.1 Address Cross-Cutting Concerns

No.	Concern	Description
1.	Cahing	This requirement aims to provide a caching solution for Java functions. Caching is a common issue in software development that can have a significant impact on application performance. The proposed caching functionality in the Codetuner will enable users to reduce the time required for repetitive computations and improve the overall performance of the application. It will also facilitate the maintenance of a consistent coding style and cache properties across the codebase.
2.	Resource Utilization (Profiling)	This requirement aims to optimize the utilization of system resources, including CPU, memory, and disk space, while executing AspectJ code. The resource utilization should be monitored and optimized through techniques such as profiling, caching, and parallelization aspects to ensure efficient execution.
3.	Execution Time	This requirement aims to measure the execution time of Java Functions. The measurement of execution time will enable developers to take corrective actions if required or reduce the execution time by using other functionalities like caching, parallelization
4.	Parallelization	This requirement aims to provide support for parallelization in Java code. Parallelization is a technique that can significantly improve application performance by distributing workload across multiple threads. The AspectJ tool should be designed

No.	Concern	Description
		to support parallel execution of Java code while maintaining thread safety and minimizing resource contention.
5.	Logging	This requirement aims to provide a flexible and configurable logging mechanism for Java code. Logging is an essential functionality in software development that can help diagnose issues and improve application quality.

4.2 Integration

The tool should easily integrate with existing software applications and frameworks to support easy development.

4.3 Testing

The tool should be rigorously tested to ensure its reliability, performance, and compatibility with various environments and use cases. Unit testing should be carried out to ensure the tool meets the required quality standards.

5. Nonfunctional Requirements

5.1 Performance

The Java Code Tuner should be efficient and performant, with minimal impact on the overall performance of the software application. The tuner should be designed to avoid delays or errors.

5.2 Scalability

The Java Code Tuner should be scalable and able to accommodate the growth of the software application. It should be designed to handle an increasing number of code files, invocations & complexity.

5.3 Reliability

The Java Code Tuner should be reliable and available at all times. It should be designed to handle errors and exceptions gracefully and recover from any failures quickly.

5.4 Security

The Java Code Tuner should be designed with security in mind. It should adhere to best practices for security and protect sensitive client data and make sure propietary code files never leave the user system.

5.5 Usability

The Java Code Tuner should be easy to use and integrate into existing software applications. The tool should have clear documentation, and examples to help developers understand and implement its features.

5.6 Maintainability

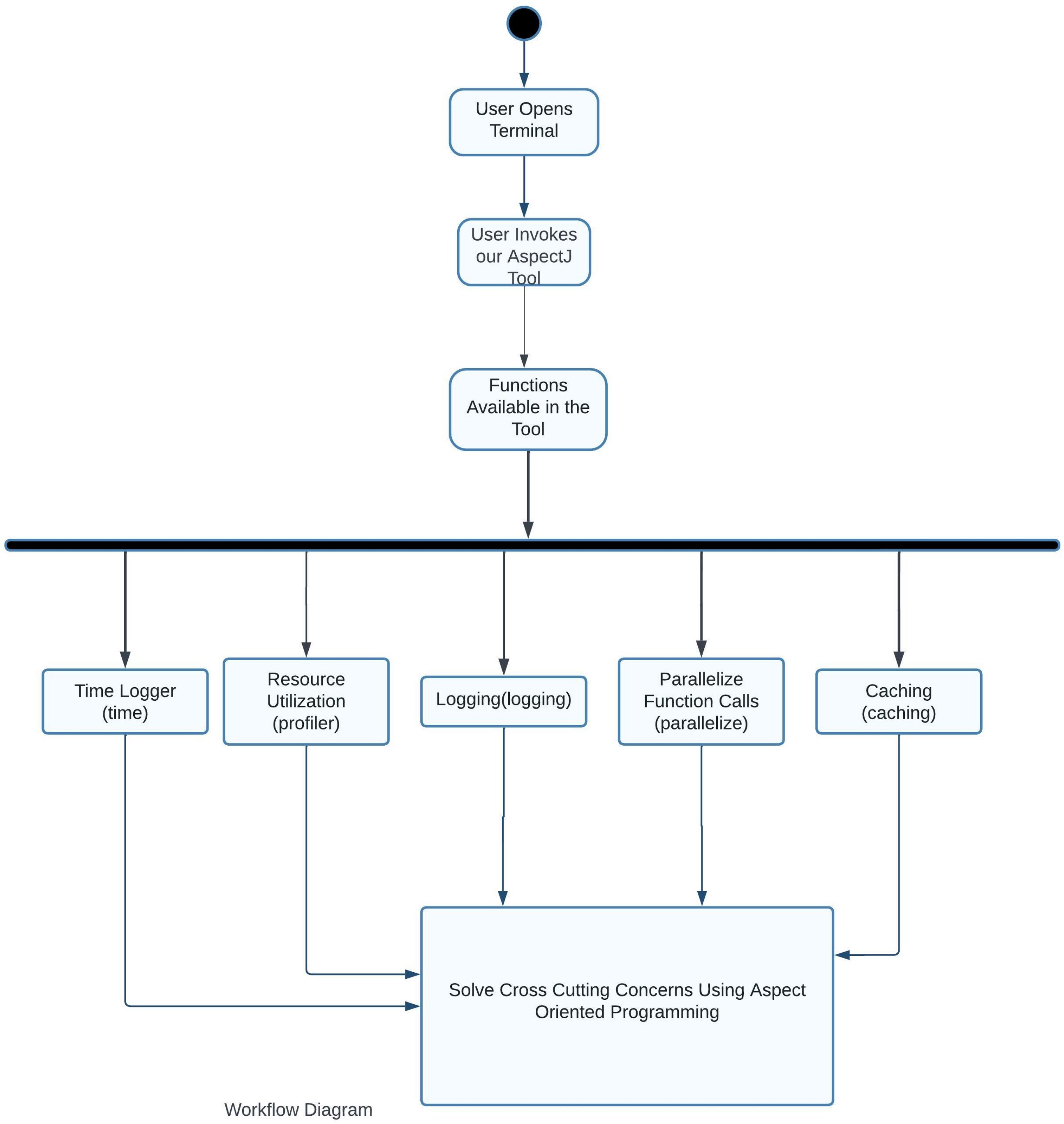
The Java Code Tuner should be easy to maintain and update. It should have clear code structure, consistent naming conventions, and documentation to help developers understand and modify the library.

6. Interface Requirements

6.1 Command Line Interface

The tool is powered by a picocli interface, allowing users to use it easily. Picocli is a popular one file framework that allows creating rich command line applications for Java applications.

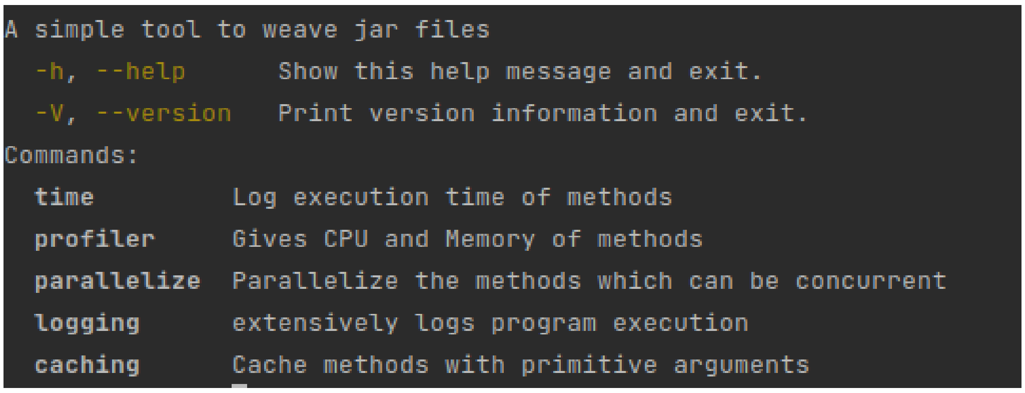
The following is the workflow diagram for using the command line interface.


User Opens Terminal
User Invokes our AspectJ Tool
Functions Available in the Tool
Time Logger (time)
Resource Utilization (profiler)
Logging(logging)
Parallelize Function Calls (parallelize)
Caching (caching)
Solve Cross Cutting Concerns Using Aspect Oriented Programming

Workflow Diagram

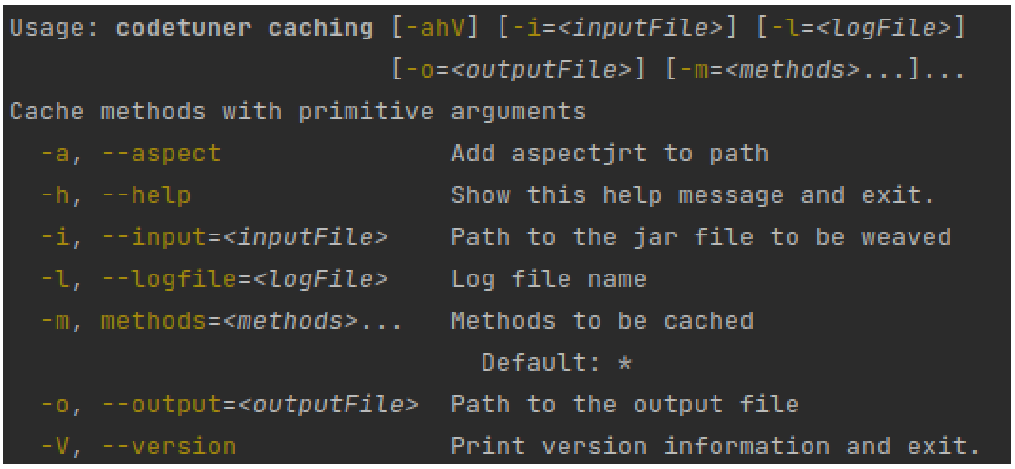
7. Interface Design

7.1 Home Screen



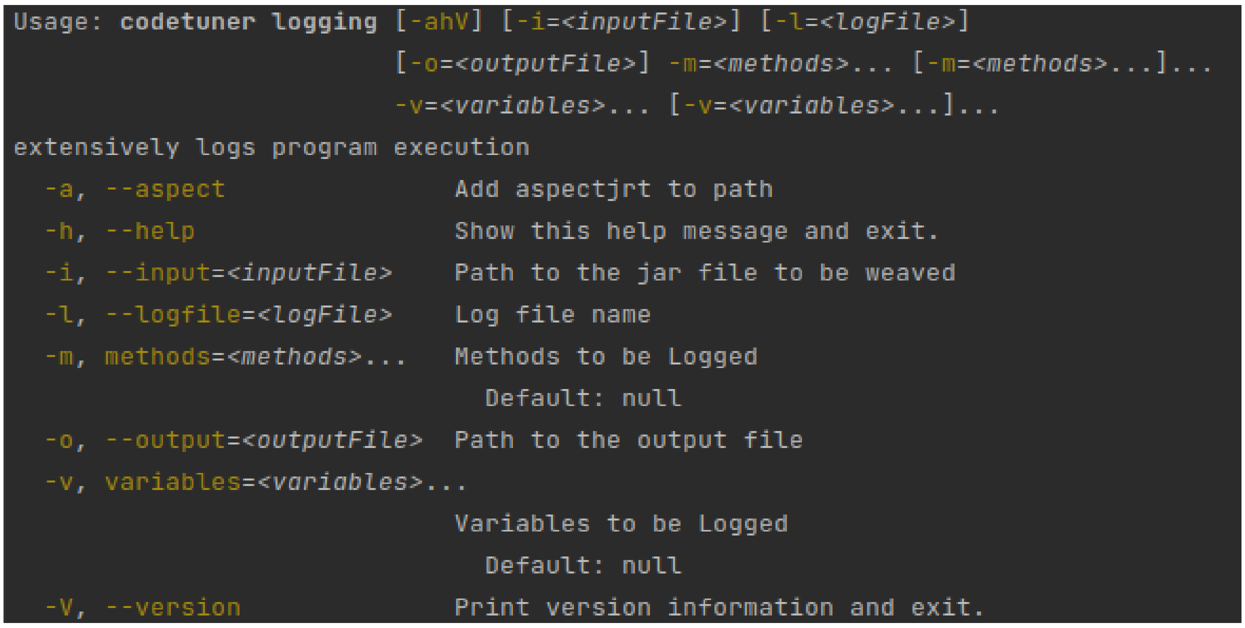
```
A simple tool to weave jar files
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  time          Log execution time of methods
  profiler      Gives CPU and Memory of methods
  parallelize   Parallelize the methods which can be concurrent
  logging       extensively logs program execution
  caching       Cache methods with primitive arguments
```

7.2 Caching Menu



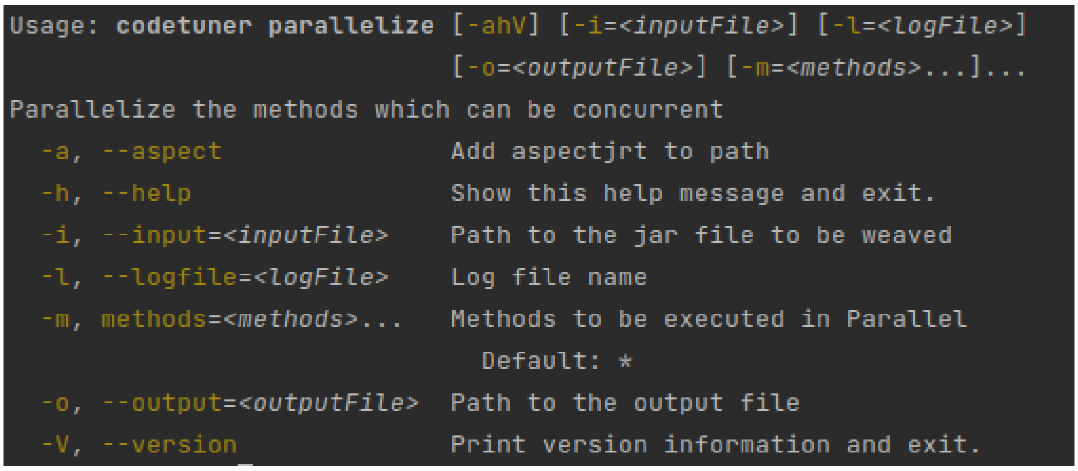
```
Usage: codetuner caching [-ahV] [-i=<inputFile>] [-l=<logFile>]
                         [-o=<outputFile>] [-m=<methods>...]...
Cache methods with primitive arguments
  -a, --aspect               Add aspectjrt to path
  -h, --help                 Show this help message and exit.
  -i, --input=<inputFile>    Path to the jar file to be weaved
  -l, --logfile=<logFile>    Log file name
  -m, methods=<methods>...   Methods to be cached
                               Default: *
  -o, --output=<outputFile>  Path to the output file
  -V, --version              Print version information and exit.
```

7.3 Logging Menu



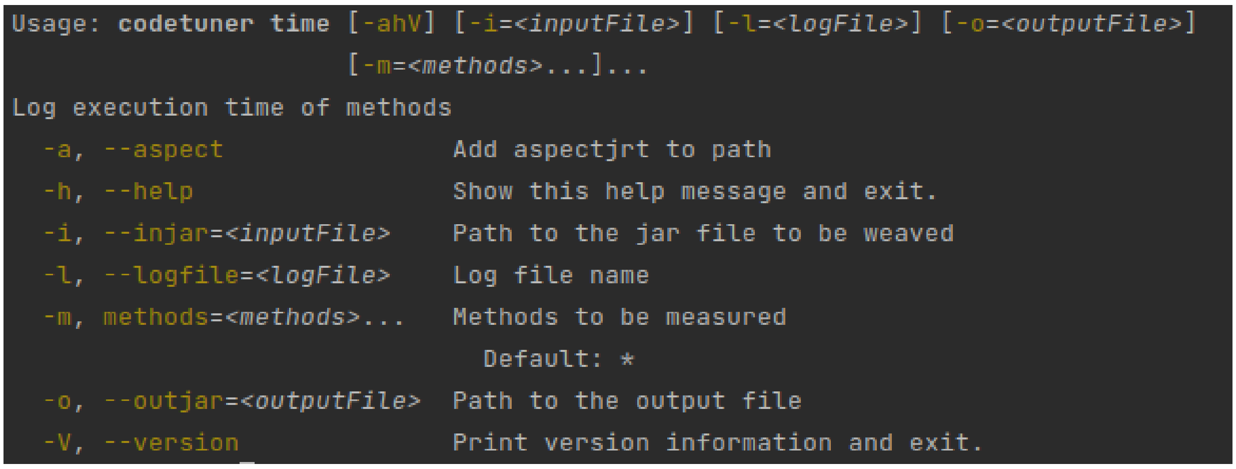
```
Usage: codetuner logging [-ahV] [-i=<inputFile>] [-l=<logFile>]
                         [-o=<outputFile>] -m=<methods>... [-m=<methods>...]...
                         -v=<variables>... [-v=<variables>...]...
extensively logs program execution
  -a, --aspect               Add aspectjrt to path
  -h, --help                 Show this help message and exit.
  -i, --input=<inputFile>    Path to the jar file to be weaved
  -l, --logfile=<logFile>    Log file name
  -m, methods=<methods>...   Methods to be Logged
                               Default: null
  -o, --output=<outputFile>  Path to the output file
  -v, variables=<variables>...
                             Variables to be Logged
                               Default: null
  -V, --version              Print version information and exit.
```

7.4 Parallelization Menu



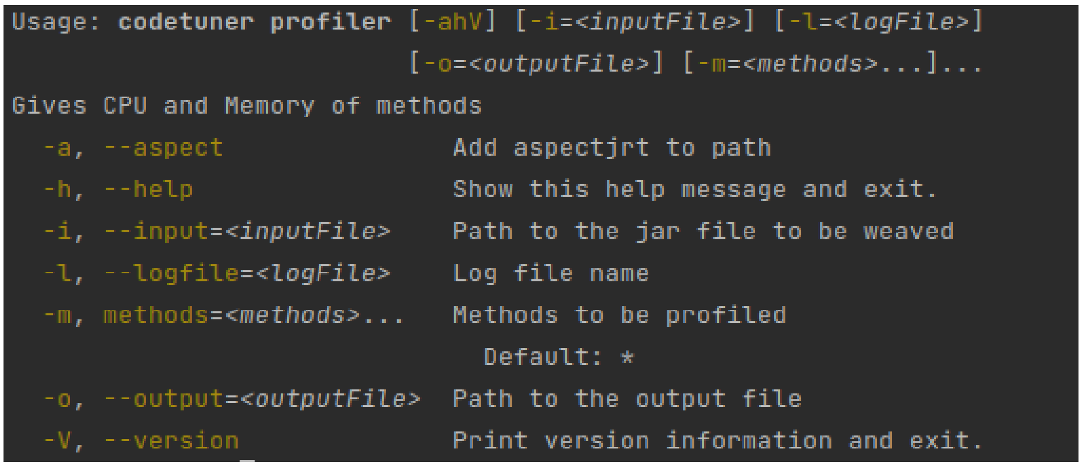
```
Usage: codetuner parallelize [-ahV] [-i=<inputFile>] [-l=<logFile>]
                             [-o=<outputFile>] [-m=<methods>...]...
Parallelize the methods which can be concurrent
  -a, --aspect               Add aspectjrt to path
  -h, --help                 Show this help message and exit.
  -i, --input=<inputFile>    Path to the jar file to be weaved
  -l, --logfile=<logFile>    Log file name
  -m, methods=<methods>...   Methods to be executed in Parallel
                               Default: *
  -o, --output=<outputFile>  Path to the output file
  -V, --version              Print version information and exit.
```

7.5 Execution Time Menu



```
Usage: codetuner time [-ahV] [-i=<inputFile>] [-l=<logFile>] [-o=<outputFile>]
                      [-m=<methods>...]...
Log execution time of methods
  -a, --aspect               Add aspectjrt to path
  -h, --help                 Show this help message and exit.
  -i, --injar=<inputFile>    Path to the jar file to be weaved
  -l, --logfile=<logFile>    Log file name
  -m, methods=<methods>...   Methods to be measured
                               Default: *
  -o, --outjar=<outputFile>  Path to the output file
  -V, --version              Print version information and exit.
```

7.6 Profiler Menu



```
Usage: codetuner profiler [-ahV] [-i=<inputFile>] [-l=<logFile>]
                          [-o=<outputFile>] [-m=<methods>...]...
Gives CPU and Memory of methods
  -a, --aspect               Add aspectjrt to path
  -h, --help                 Show this help message and exit.
  -i, --input=<inputFile>    Path to the jar file to be weaved
  -l, --logfile=<logFile>    Log file name
  -m, methods=<methods>...   Methods to be profiled
                               Default: *
  -o, --output=<outputFile>  Path to the output file
  -V, --version              Print version information and exit.
```

8. Unit Test Plan

8.1 Introduction

This section outlines the unit test plan for the Java Code Tuner. The purpose of this document is to provide an overview of the unit testing strategy, approach, and procedures to be followed during the project's testing phase. The tests will use JUnit Jupiter as the testing framework, Jacoco for code coverage, and Mockito for the creation of mock objects to test exception handling.

8.2 Scope

The scope of the unit testing will cover the individual components or units of the application. The main goal is to ensure that each component is functioning as expected and that any issues or defects are identified and fixed before moving on to integration testing.

8.3 Testing Approach

The testing approach will follow a combination of black box and white box testing. Black box testing will be used to verify that each component behaves as expected when provided with specific input and expected output. White box testing will verify the components' internal workings and ensure that all code paths have been exercised. Jacoco will be used to measure the code coverage of the unit tests.

8.4 Test Environment

The test environment will consist of a dedicated testing environment that is separate from the development and production environments. The testing environment will be set up to mimic the production environment as closely as possible to ensure that the results of the testing are as accurate as possible.

8.5 Pass/Fail Criteria

To successfully complete the test phase, the run rate must be 100% unless a valid reason is provided, and the pass rate must be at least 90%. We will continuously monitor and refine our testing approach to ensure we meet or exceed these criteria.

8.6 Components Tested

No.	Component	Description
1.	Main	Ensures that the CodeTuner tool boots up correctly when invoked.
2.	Weaver	Verifies that the selected aspect is properly woven with the user-provided JAR file to generate a modified JAR file.
3.	Caching	Validates that the caching aspect is properly applied to the user's code.
4.	Execution Time	Tests if the execution time aspect is applied correctly to the user's code and generates accurate metrics.
5.	Logging	Verifies that the logging aspect is applied correctly and captures relevant information.
6.	Method Profiling	Tests if the method profiling aspect is properly applied to the user's code and generates accurate performance metrics.
7.	Parallelization	Verifies that the parallelization aspect is applied correctly and improves the performance of the user's code.

9. Constraints

The tool built in this project has been tested only for Java Version 18.0.

10. Contributors

Name	Entry Number
Aditya Aggarwal	2020CSB1066
Akshat Toolaj Sinha	2020CSB1068

Name	Entry Number
Pranavkumar Mallela	2020CSB1112
Shahnawaz Khan	2020CSB1123
Vishnusai Janjanam	2020CSB1142