Lab Assignment 9

Course: CS202 Software Tools and Techniques for CSE
Lab Topic: Module Dependency and Cohesion Analysis
Date: 20th March 2025

Objective

This lab focuses on analyzing software dependencies and cohesion using **pydeps** (for Python) and **LCOM** (Lack of Cohesion of Methods) (for Java).

Learning Outcomes

By the end of this lab, students will be able to:

- Use **pydeps** to generate and analyze dependency graphs for Python projects.
- Use **LCOM** to measure class cohesion in Java projects.
- Identify design flaws and code smells (anti-patterns).
- Implement modularization and refactoring strategies to optimize software structure.

Pre-Lab Requirements

- Operating System: Windows/Linux/MacOS
- Programming Languages: Python (3.7 or above) and Java (11 or later)
- Required Tools: **pydeps**, **LCOM**

  Read: https://github.com/thebjorn/pydeps

  Use: **LCOM.jar**

Lab Activities:

1. Select one real-world project with multiple modules (.py files organized into folders) and at least 500+ lines of source code (overall).
2. Navigate to the project directory. Run **pydeps** to generate a detailed dependency graph. **pydeps** also generates a dependency graph analysis (in JSON format) when the **--show-deps** command line option is passed. Store and utilize this JSON to calculate **fan-in** and **fan-out** of each module.
3. Analyze the generated dependency graph:
   - Identify highly coupled modules and their dependencies.
   - Detect cyclic dependencies and explain how they affect maintainability.
   - Check for unused and disconnected modules.
   - Assess the depth of dependencies.
4. Perform a dependency impact assessment:

- How would changes in the core module[1] affect the rest of the system?
- Which modules are at risk of breaking the system if modified?

5. Measuring Java Class Cohesi[2]on using **LCOM**:
   - Choose a Java project with at least 10 classes.
   - Navigate to the project directory and run LCOM analysis on the project. For example, assume **LCOM.jar** is in the current directory. Then, give the **source code path** to be analyzed as input and a folder where output should be stored in the below command:

```
java –jar LCOM.jar -i <input_path> -o <output_path>
```

6. Identify classes with high **LCOM** values and analyze:
   - What does a high **LCOM** value suggest about a class's design?
   - Is there a chance for performing functional decomposition?
7. Visualizing Cohesion in Classes:
   - Create a table of **LCOM** values for the selected classes.
   - Compare them **side-by-side** in the format below:

Java code	Output of **LCOM.jar**					
	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	YALCOM

`package lcom.testsubject;`
`public class AllDisconnected`
`{`
`  public int  f1, f2, f3;`
`  public void m1(){}`
`  public void m2(){}`
`  public void m3(){}`
`}`						

Resources

- Lecture 8 Slides
- https://pypi.org/project/modulegraph (Python Module dependency analysis)
- https://github.com/thebjorn/pydeps (Python Module dependency graphs)
- https://github.com/tushartushar/LCOM (LCOM{1..5} & YALCOM)

[1] Core Module refers to the module that serves as a critical dependency for multiple other modules. These modules often contain shared functionality of the system.