University of Zurich^UZH

Department of Informatics IfI
University of Zurich
Binzmühlestrasse 14
CH—8050 Zürich
Switzerland
URL: https://www.ifi.uzh.ch

Instructors:
Prof. Dr. Alberto Bacchelli
Prof. Dr. Burkhard Stiller
Prof. Dr. Ivan De Oliveira Nunes
Dr. Marco D'Ambros
Assistants:
Konstantinos Kitsios

Fundamentals of Software Systems (FSS) 03SM22MI0002

Software Evolution - Part I Assignment

Submission Guidelines

To correctly complete this assignment you **must**:

- Carry out the assignment in a team of three students.
- Carry out the assignment with your team only. You are allowed to discuss solutions with other teams, but each team should develop its own personal solution. A strict plagiarism policy will be applied to all artifacts submitted for evaluation.
- Submit a solution to OLAT as a single ZIP file by November 24, 2025 @ 23:55. Each solution must consist of:
  - An `fss_se_assignment.py` or an `fss_se_assignment.ipynb` containing the source code of your solution.
  - A `requirements.txt` containing all the necessary packages.
  - A `README.md` containing the documentation of your solution. At the very least, this file should contain the plots requested below, the answers to the questions below, as well as design decisions you took in this assignment.

Use of Generative AI. While you may use tools like Generative AI for assistance, you must ensure that the final solution is your own original work. Generative AI should only be used as a tutor or assistant (e.g., to help troubleshooting or learning a new API), not for directly producing the solutions. If you make use of AI, **you have to document all the prompts you have used and an overall explanation of how you used Generative AI in the `README.md` file.**

Background

Hugging Face Transformers is a popular open-source library for natural language processing (NLP) that provides state-of-the-art machine learning models for tasks like text classification, translation, summarization, and question answering. It has a large and active codebase, making it an interesting candidate for analysis in the context of software systems and quality.

Prerequisites

To set up the assignment's environment, follow the steps below:

- Clone the Transformers repository.
- Checkout the latest release tag, which is *v4.57.0*.
- Create a Python virtual environment using Python 3.12. Any package you may need should be installed in this fresh virtual environment, and exported to the `requirements.txt` deliverable.

Task 1: Defect Analysis

Defective hotspots are files where defects are frequently found. We will implement a simple yet effective approach to identify these hotspots by analyzing commit messages for keywords that indicate defect resolutions, such as "fix", "bug", "error", or "issue". To complete this task, follow the steps described below.

- Use Git to extract all the commit messages after 2023-01-01. The cutoff date is chosen for computational reasons and will be used for the rest of the tasks as well.
- Analyze these messages to detect the presence of specific keywords of your choice related to defect fixes.
- Calculate and plot the total number of defects per month. Why do you think the number of defects dropped sharply in October 2025?
- Calculate and plot the number of defects per month for the two files with the highest number of defects.
  - In which month were the most defects introduced? How would you explain it? Manually examine the repository for that month (e.g., change logs, releases, commit messages) and come up with a hypothesis.
- What are the limitations of this method for finding defective hotspots?

Metric	Description	How To Measure
Cyclomatic Complexity (CC)	Quantifies the number of linearly independent paths through a program's source code by measuring the control flow within the program.	Using python libraries such as lizard, pydriller, radon, or McCabe.
Lines of code (LoC)	Measures the number of lines of code in a file.	Using python libraries such as lizard, cloc, or radon. For a simple but approximate solution, Python's `splitlines()` can also be used.
Number of code changes (NCC)	Measures the number of times each file has been changed in a certain timeframe or between two releases.	As in Task 1.
Indentation-based complexity (IC)	Analyses the visual shape of the code, such as the level of indentation. This approach is language-neutral and can be applied to different programming languages.	Counting whitespace "blocks" at the beginning of each line. A block is defined by a set number of spaces, i.e., 4 or a tab character for indentation.

Table 1: Complexity Metrics.

Task 2: Complexity Analysis

Measuring complexity is useful in many cases, like pinpointing refactoring opportunities, and a number of different metrics exist for this task, which are shown in Table 1. A complexity hotspot is a file that might be problematic due to its high complexity. With that in mind, complete the following analysis:

- Select two complexity metrics of your choice.
- Calculate the complexity of all `.py` files in the repository using the selected metrics.[1]
- Visualize the complexity hotspots. The visualization should effectively convey which parts of the code are more complex or change more frequently. Feel free to use any visualization of your choice and explain the rationale behind your decision.
- What can you say about the correlation between the two complexity measures in this repository? For example, if you selected CC and LoC, what can you say for the statement "Files with more lines of code tend to have higher cyclomatic complexity"?

[1]For NCC, consider the commits since 2023-01-01.

- A colleague of yours claims that "Files with higher complexity tend to be more defective". What evidence can you present to support or reject this claim for the selected complexity measures in this repository?

Task 3: Coupling Analysis

Logical coupling occurs when two seemingly distinct files are functionally related. It can be detected by mining software repositories to see which files tend to be committed together frequently over time. With that in mind, complete the following analysis:

- Calculate the logical coupling for each file pair in the repository. Visualize the 10 most coupled file pairs using a visualization of your choice that effectively conveys the coupling relationships. Select one of these 10 most coupled file pairs and comment on their relationship.

- Repeat the steps of the bullet point above, but consider only file pairs where the one file is a Python test file, i.e., starts with "`test_`", and the other is a Python non-test file. How would you explain this type of coupling? Is it a code smell that requires attention and signals potential refactoring opportunities or is it something different?

- Writing tests is a time-consuming task and developers often omit it, thus, automated test generation tools have been implemented and are widely used. One of the most popular test generation tools for Python is Pynguin, that takes as input a `.py` file and generates passing tests for that file. Pynguin writes the generated tests to a new file in a separate folder, isolated from the project's test suite. Suppose that you are tasked with implementing an option for Pynguin to place the tests directly in the project's test suite, specifically in the test file that is most closely "related" with the input `.py` file. Discuss at least three (3) implementations for selecting the most "related" test file given a (non-test) `.py` file. You do not have to implement these options at this stage.

- Select two of the three test placement implementations you proposed above. Where would they place automatically-generated tests for the `src/transformers/generation/utils.py` file?