Project Insight

In this project, we developed a dynamic risk scoring system built on top of starter code provided by the instructors, which included repository mining tools using PyDriller and a harness for evaluating predictions. Our primary goal was to design a method that analyzes historical commit activity to assign a risk score to each file, indicating its likelihood of containing defects. Early on, we focused on exploring different metrics such as code churn, modification frequency, and bug-fix history, to understand their effectiveness in predicting future defects.

As we iterated, we observed that overly complex models often introduced noise, particularly with features like import frequency and developer centrality. These were eventually removed in favor of cleaner, more interpretable inputs. We also expanded our detection of bug-fix commits by enriching the keyword list used to classify commit messages, helping improve the precision of our bug-fix metric across diverse projects.

Methodology

Our methodology focused on extracting meaningful metrics for each file, including modification frequency, bug-fix commit count, code churn (lines added and removed), recency of changes, and developer inexperience based on first contribution timing. Initially, we combined these features using a variance-based weighting system, where features with greater variability across the dataset were given more influence. This allowed the model to dynamically adapt to the characteristics of each repository. However, we found that this approach introduced complexity without a meaningful accuracy gain. To improve interpretability and maintain control over feature influence, we transitioned to a static weighting scheme, assigning the highest weights to recent bug activity, bug-fix frequency, and modification frequency, with smaller weights given to churn and developer experience.

In the final model, we prioritized recent bug activity, bug-fix frequency, and modification frequency as the most predictive features, assigning them the highest weights. Code churn and developer inexperience were included with lower weights to account for edge cases. Additionally, we implemented file rename resolution to ensure that historical commit activity was not lost during file renames; an important enhancement that improved prediction accuracy by preserving complete risk histories.

Final Approach

Our final risk scoring system uses a simple approach based on statically weighted metrics: recent bug activity, bug-fix frequency, modification frequency, code churn, and developer inexperience. We expanded bug detection with a broader keyword set and ensured full compatibility with the provided harness. The result is a lightweight, practical tool for identifying high-risk files in real-world software projects.