# Approach for Defect Prediction

## Objective

The primary objective is to predict defects in software development by leveraging machine learning models based on historical data.

## Data Understanding and Preparation

Dataset Description: Loaded dataset containing defect-related information.
Data Inspection: Examined dataset structure, checked for missing values, and performed exploratory data analysis (EDA).
Summary Statistics: Calculated summary statistics for numerical columns.
Categorical Columns Exploration: Investigated categorical columns and their value counts.
Visualizations: Utilized histograms and other visualizations to understand data distributions.

## Feature Selection and Preprocessing

Feature Selection: Selected relevant features based on their potential impact on defect prediction.
Data Preprocessing: Handled missing values and encoded categorical variables for model compatibility.

## Model Building and Evaluation

Initial Model: Utilized a Decision Tree Classifier as a baseline model for defect prediction.
Model Evaluation: Assessed model performance using classification metrics such as accuracy, precision, recall, and F1-score.
Model Refinement: Employed a RandomForestClassifier with hyperparameter tuning using GridSearchCV.
Cross-Validation: Performed cross-validation to ensure model robustness and generalizability.
Advanced Evaluation: Calculated ROC-AUC scores and visualized ROC curves and confusion matrices.

## Challenges and Considerations

Data Quality: Addressed missing values and outliers during preprocessing.
Model Selection: Chose appropriate models based on the nature of the problem.
Hyperparameter Tuning: Tuned models for optimal performance.
Evaluation Metrics: Considered various metrics for a comprehensive evaluation.

# Approach for Smart Test Selection

### Objective

The goal is to implement an intelligent test selection mechanism based on code changes to optimize testing efficiency.

### Data Retrieval and Preprocessing

Data Sources: Simulated retrieval of code changes, test execution logs, and code coverage reports.
Data Integration: Merged and preprocessed data to create a unified dataset for analysis.

### Test Selection Strategy

Criteria Definition: Outlined criteria for selecting tests based on coverage and execution time.
Implementation: Developed logic to select tests based on high coverage and shorter execution times.

### Challenges and Considerations

Mapping Accuracy: Ensured accurate mapping between code changes and relevant tests.
Coverage vs. Efficiency: Balanced test coverage without sacrificing efficiency.
Test Relevance: Maintained alignment of test suites with evolving codebases.
Data Availability: Required sufficient historical data for ML-based approaches.
Integration Complexity: Ensured seamless integration within CI/CD pipelines.