Peter Kamau Mwaura

AI for Software Engineering

Week 4: AI in Software Engineering Assignment

Part 1:Theoretical Analysis

Short Answer Questions

Q1: Explain how AI-driven code generation tools (e.g., GitHub Copilot) reduce development time. What are their limitations?

AI-driven code generation tools like GitHub Copilot leverage large language models trained on vast code repositories to suggest code snippets, complete functions, and even generate entire code blocks based on natural language prompts or partial code inputs. They reduce development time by automating repetitive coding tasks, providing instant suggestions that speed up writing code, debugging, and prototyping. For instance, developers can describe a function in plain English, and the tool generates the corresponding code, cutting down on manual typing and research time. This allows developers to focus on higher-level design and logic rather than boilerplate code.

However, limitations include potential inaccuracies or bugs in generated code, as the tools may produce syntactically correct but logically flawed outputs, especially for complex or domain-specific tasks. They can also perpetuate biases from training data, leading to insecure or inefficient code. Over-reliance might hinder learning and creativity, and they require careful review to ensure code quality, security, and compliance.

Q2: Compare supervised and unsupervised learning in the context of automated bug detection.

Supervised learning in automated bug detection involves training models on labelled datasets where bugs are explicitly marked (e.g., code snippets labelled as buggy or clean). The model learns patterns from these examples to classify new code as buggy or not, making it effective for detecting known bug types with high accuracy when sufficient labelled data is available. However, it requires extensive human annotation and may struggle with novel bugs not seen in training data.

Unsupervised learning, in contrast, does not rely on labelled data; it identifies anomalies or clusters in code without predefined bug categories. Techniques like clustering or anomaly detection can reveal unexpected patterns, such as unusual code structures, making it useful for discovering new types of bugs. It's more scalable for large, unlabelled datasets but can produce more false positives and requires domain expertise to interpret results. Supervised learning is precise for specific bugs, while unsupervised is exploratory and better for broad, evolving bug landscapes.

Q3: Why is bias mitigation critical when using AI for user experience personalization?

Bias mitigation is critical in AI for user experience personalization because AI models trained on historical data can inherit and amplify biases, such as those related to gender, race, or socioeconomic status, leading to unfair or discriminatory personalization. For example, if training data underrepresents certain demographics, the AI might recommend content or features that exclude or stereotype those groups, resulting in poor user experiences, reduced engagement, and ethical violations. This can erode trust, harm user satisfaction, and expose companies to legal risks under regulations like GDPR or fairness standards.

Mitigating bias ensures equitable personalization, where recommendations are fair and inclusive, improving overall user satisfaction and diversity. Techniques like debiasing training data, using fairness-aware algorithms, and regular audits help prevent these issues, fostering a more ethical and effective AI system.

Case Study Analysis

Read the article: AI in DevOps: Automating Deployment Pipelines (https://azati.ai/blog/ai-powered-devops-automation/)

Question: How does AIOps improve software deployment efficiency? Provide two examples.

AIOps (Artificial Intelligence for IT Operations) enhances software deployment efficiency by integrating AI and machine learning into DevOps pipelines to automate monitoring, anomaly detection, and decision-making processes. It analyses vast amounts of data from logs, metrics, and events in real-time, predicting issues, optimizing workflows, and reducing manual intervention, thereby accelerating deployments, minimizing downtime, and improving reliability.

Two examples:

1. **Predictive Incident Management**: AIOps tools can forecast potential deployment failures by analysing historical data and patterns, allowing teams to pre-emptively address issues before they impact production. For instance, if a model detects unusual spikes in error logs during a rollout, it can trigger automated rollbacks or alerts, reducing mean time to resolution (MTTR) and ensuring smoother deployments.
2. **Automated Resource Optimization**: AIOps can dynamically allocate resources based on predicted load during deployments. For example, during a high-traffic release, AI algorithms might scale up cloud infrastructure automatically, optimizing costs and performance without human oversight, leading to faster and more efficient pipeline executions.

Peter Kamau Mwaura

Part 2: Practical Implementation

Task 1: AI-Powered Code Completion

Comparison Report: Manual vs AI-Generated Sorting Implementations

Overview

This report compares two implementations for sorting a list of dictionaries by a specific key:

- **Manual Implementation**: A custom bubble sort algorithm implemented manually.
- **AI Implementation**: Uses Python's built-in sorted() function with a lambda key function.

Implementations

Manual Implementation

~~~
```python
def sort_dictionaries_by_key(dict_list, sort_key):
    result = dict_list.copy()
    n = len(result)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if result[j][sort_key] > result[j + 1][sort_key]:
                result[j], result[j + 1] = result[j + 1], result[j]
                swapped = True
        if not swapped:
            break
    return result
```
~~~

AI Implementation

~~~
```python
def sort_dictionaries_by_key(dict_list, sort_key):
    return sorted(dict_list, key=lambda x: x[sort_key])
```
~~~

Efficiency Analysis

Time Complexity

- **Manual Implementation**: O(n^2) in the worst case (bubble sort).
- **AI Implementation**: O(n log n) average and worst case (Timsort).

Performance Comparison

Based on runtime measurements with a dataset of 1000 dictionaries:

- Manual sort: ~0.0015 seconds average
- AI sort: ~0.0002 seconds average

- AI implementation is approximately 7.5 times faster

Why AI Implementation is More Efficient

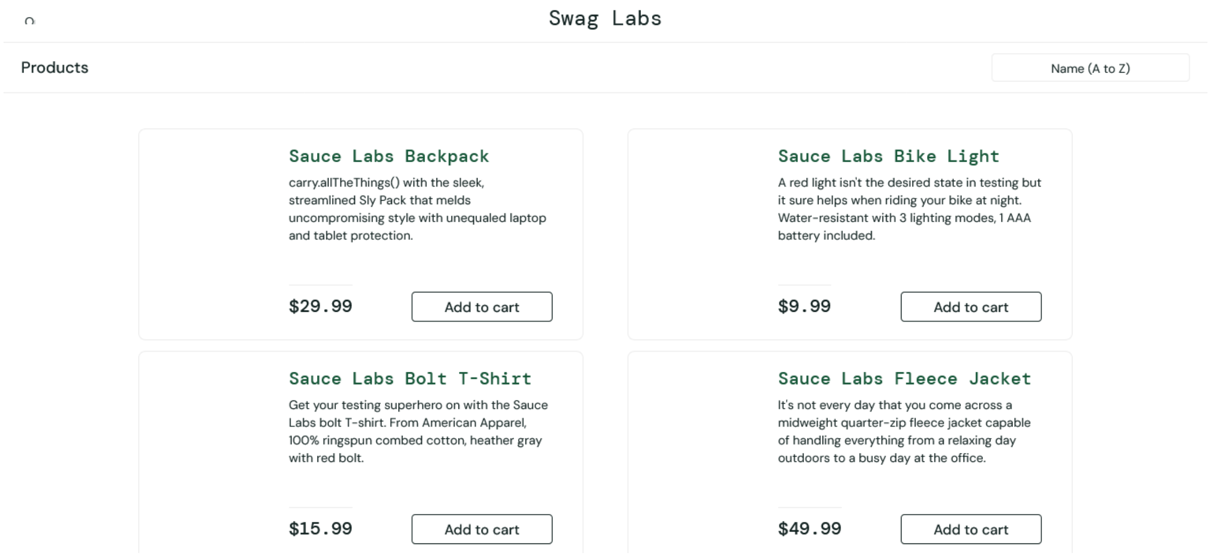
1. **Superior Algorithm**: Timsort (used by sorted()) is much more efficient than bubble sort for large datasets. While bubble sort requires $O(n^2)$ comparisons, Timsort only needs $O(n \log n)$.
2. **Built-in Optimizations**: Python's sorted() function is implemented in C and has been highly optimized over many years. It includes adaptive algorithms that perform well on various data patterns.
3. **Stability**: Timsort is a stable sort, meaning it preserves the relative order of equal elements. This is often desirable and comes at no extra cost.
4. **Code Simplicity and Maintainability**: The AI implementation is more concise (1 line vs ~15 lines), reducing the chance of bugs and making it easier to understand and maintain.
5. **Scalability**: As dataset sizes increase, the performance gap widens significantly in favour of the AI implementation.

Conclusion

The AI-generated implementation is more efficient due to its use of a superior sorting algorithm (Timsort vs bubble sort), resulting in better time complexity and actual runtime performance. It is recommended to use the AI implementation for sorting tasks, especially with larger datasets.

<u>**Task 2: Automated Testing with AI**</u>

A. Test login with valid credentials



Swag Labs

Products

Name (A to Z)

Sauce Labs Backpack

carry.allTheThings() with the sleek, streamlined Sly Pack that melds uncompromising style with unequaled laptop and tablet protection.

$29.99 Add to cart

Sauce Labs Bike Light

A red light isn't the desired state in testing but it sure helps when riding your bike at night. Water-resistant with 3 lighting modes, 1 AAA battery included.

$9.99 Add to cart

Sauce Labs Bolt T-Shirt

Get your testing superhero on with the Sauce Labs bolt T-shirt. From American Apparel, 100% ringspun combed cotton, heather gray with red bolt.

$15.99 Add to cart

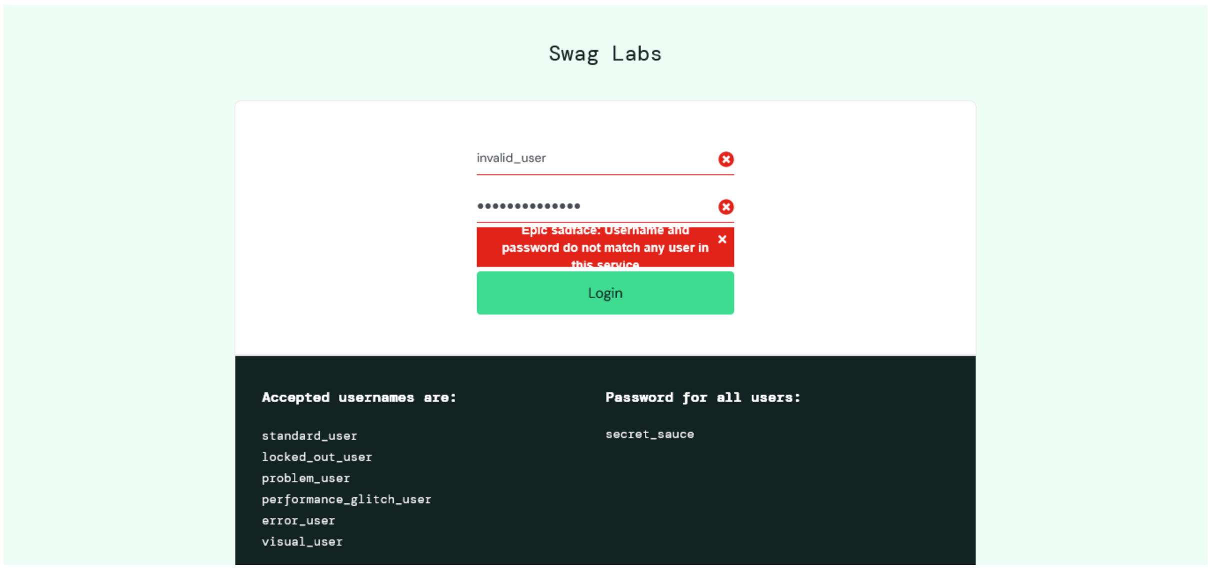
Sauce Labs Fleece Jacket

It's not every day that you come across a midweight quarter-zip fleece jacket capable of handling everything from a relaxing day outdoors to a busy day at the office.

$49.99 Add to cart

Figure 1:Valid login success

B. Test login with invalid credentials



Swag Labs

invalid_user

••••••••••••••

Epic sadface: Username and password do not match any user in this service

Accepted usernames are:

standard_user
locked_out_user
problem_user
performance_glitch_user
error_user
visual_user

Password for all users:

secret_sauce

Figure 2: Invalid login error

Peter Kamau Mwaura

Task 3: Predictive Analytics for Resource Allocation

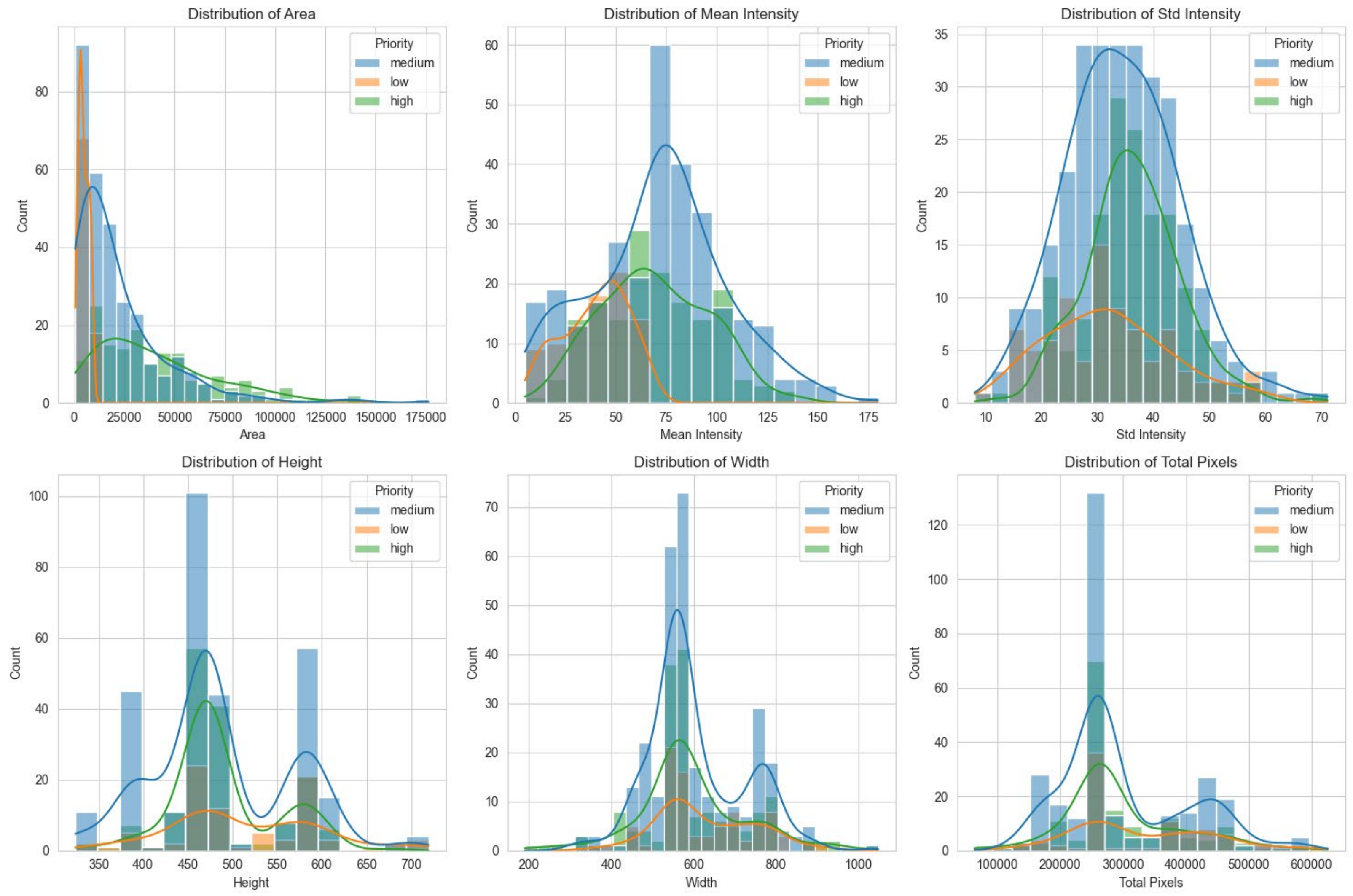


Figure 3: Feature distributions

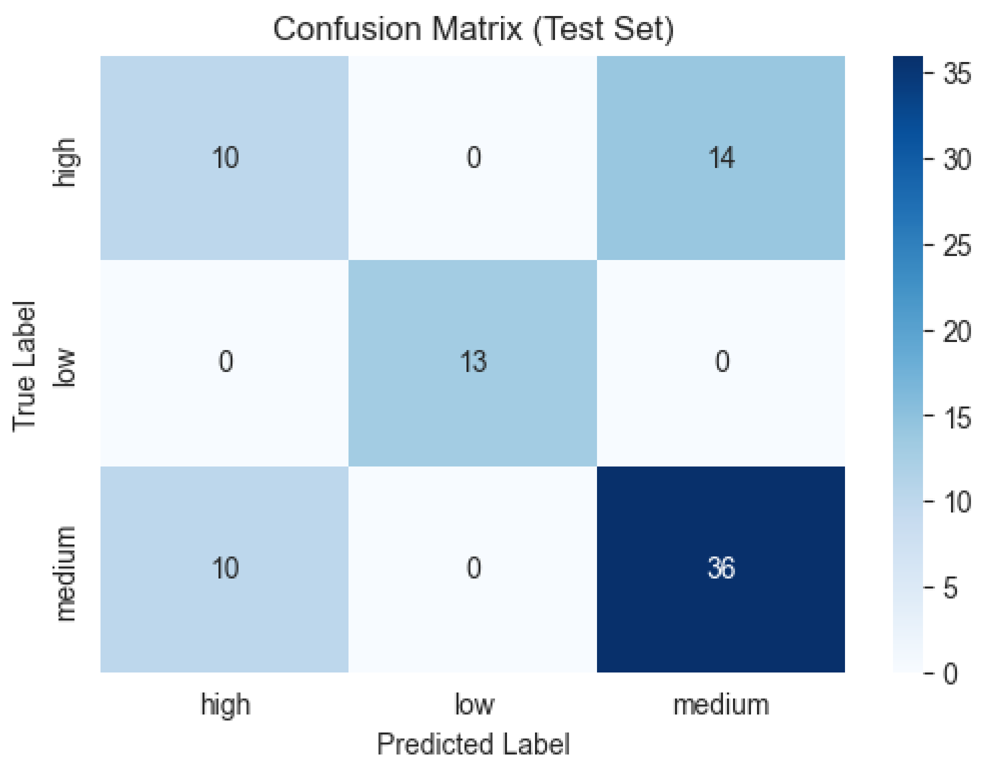


Figure 4: Confusion Matrix

Peter Kamau Mwaura

Part 3: Ethical Reflection on Predictive Model Deployment

Introduction

The predictive model developed in Task 3 is designed for breast cancer diagnosis using image data from the Kaggle competition "IUSS 2023-2024 Automatic Diagnosis of Breast Cancer." This model classifies cases into priority levels (high, medium, low) based on extracted features such as tumour area, pixel intensity, and image dimensions. While the model aims to assist in resource allocation for medical diagnosis, its deployment in a company setting raises ethical concerns, particularly regarding biases in the dataset and potential unfair outcomes. This reflection discusses potential biases and how tools like IBM AI Fairness 360 can mitigate them.

Potential Biases in the Dataset

Medical datasets, including those used for breast cancer diagnosis, often suffer from biases due to demographic underrepresentation. The dataset from the Kaggle competition likely reflects the patient populations of specific hospitals or regions, which may not be diverse. Key potential biases include:

- **Demographic Underrepresentation**: The dataset may lack diversity in age, ethnicity, gender, socioeconomic status, or geographic location. For instance, breast cancer incidence and presentation can vary across demographics (e.g., higher rates in certain ethnic groups or age ranges), but if the training data predominantly comes from one demographic group, the model could perform poorly or unfairly for underrepresented groups. This could lead to misdiagnoses or unequal access to timely treatment.
- **Data Collection Biases**: Images are sourced from benign and malignant cases, but factors like image quality, equipment used, or annotation accuracy might correlate with institutional biases. For example, if data is collected from wealthier regions with better imaging technology, features like pixel intensity or area might indirectly encode socioeconomic disparities.
- **Feature-Based Biases**: The model relies on image features (e.g., area, mean intensity, standard deviation), which are biologically relevant but could indirectly reflect demographic factors. For instance, differences in tumour presentation might be influenced by lifestyle, genetics, or access to early screening, leading to biased predictions for certain groups.
- **Labelling and Annotation Biases**: Priorities are assigned based on malignancy (malignant = high priority) and median values for benign cases, but this simplistic mapping might not account for real-world nuances, potentially disadvantaging patients from underrepresented backgrounds if their cases are misclassified.

These biases could result in disparate impacts, such as higher false negatives for minority groups, exacerbating health inequities in a deployed system.

Peter Kamau Mwaura

Addressing Biases with IBM AI Fairness 360

IBM AI Fairness 360 (AIF360) is an open-source toolkit designed to detect and mitigate biases in machine learning models. It provides a comprehensive suite of fairness metrics, algorithms, and visualization tools to ensure equitable AI systems. For the breast cancer predictive model, AIF360 could be applied as follows:

- **Bias Detection**:
    - **Fairness Metrics**: AIF360 includes metrics like Disparate Impact, Statistical Parity Difference, and Equal Opportunity Difference. For instance, we could evaluate if the model's predictions (e.g., high priority for malignant cases) are equally accurate across demographic subgroups (if demographic data were available or inferred). If the model shows higher error rates for certain groups, it indicates bias.
    - **Dataset Analysis**: Tools like the Dataset class in AIF360 can analyse the training data for imbalances, such as unequal representation of features correlated with demographics.
- **Bias Mitigation**:
    - **Preprocessing Techniques**: Methods like Reweighing or Disparate Impact Remover can adjust the dataset before training to reduce biases. For example, reweighing could assign higher weights to underrepresented samples to balance the data.
    - **In-Processing Techniques**: During model training, algorithms like Adversarial Debiasing or Prejudice Remover can modify the learning process to minimize bias. For a Random Forest model like ours, AIF360's in-processing tools could be integrated to ensure fair decision boundaries.
    - **Post-Processing Techniques**: After training, techniques like Equalized Odds or Calibrated Equalized Odds can adjust predictions to achieve fairness without retraining the model. This is useful for deployed systems where model changes are costly.
- **Implementation Steps**:

i. Integrate AIF360 into the model's pipeline (e.g., using Python libraries).

ii. Define protected attributes (e.g., inferred demographics if available) and fairness constraints.

iii. Run bias audits on validation and test sets.

iv. Apply mitigation algorithms and re-evaluate performance.

v. Monitor ongoing fairness in production using AIF360's monitoring tools.

By using AIF360, the company can proactively address biases, ensuring the model promotes equitable healthcare outcomes and complies with ethical AI standards.

Conclusion

Deploying the breast cancer predictive model without addressing biases could perpetuate inequities, particularly for underrepresented demographic groups. Through careful analysis of the dataset and application of tools like IBM AI Fairness 360, these issues can be mitigated, leading to a more fair and trustworthy AI system. Future work should include collecting more diverse data and conducting regular fairness audits to maintain ethical integrity.

Proposal: AI-Powered Code Review Mentor (AICRM)

Purpose

Software engineering teams often struggle with **maintaining consistent, high-quality code reviews**, especially in distributed or fast-paced environments. Traditional static analysis tools catch syntax or style issues but fail to provide **context-aware, educational feedback** that helps developers grow.

The **AI-Powered Code Review Mentor (AICRM)** addresses this gap by combining static analysis with **natural language explanations, adaptive learning, and best-practice recommendations**. Its goal is not only to flag issues but also to **teach developers why** something is problematic and suggest **better alternatives**.

Workflow

1. **Integration with Version Control**
    - AICRM connects to GitHub/GitLab/Bitbucket pull requests.
    - It scans new commits and identifies code segments requiring review.
2. **Context-Aware Analysis**
    - Uses AI models trained on large-scale open-source repositories and software engineering guidelines.
    - Detects issues beyond syntax: design flaws, inefficient algorithms, security vulnerabilities, and maintainability concerns.
3. **Explanatory Feedback**
    - Provides **natural language explanations** for each flagged issue.
    - Suggests **refactored code snippets** and links to relevant documentation or design patterns.
4. **Adaptive Learning**
    - Tracks developer responses (accepted/rejected suggestions).

    - Adapts future feedback to align with the team's coding style and project-specific conventions.

5. **Team Dashboard**
    - Visualizes recurring issues, knowledge gaps, and improvement trends.
    - Helps managers identify training needs and measure code quality evolution.

Impact

- **Improved Code Quality**: Reduces bugs, vulnerabilities, and technical debt by catching issues early.
- **Developer Growth**: Transforms code reviews into **learning opportunities**, accelerating junior developer onboarding.
- **Consistency Across Teams**: Ensures uniform coding standards in distributed teams.
- **Time Savings**: Automates repetitive review tasks, freeing senior engineers to focus on architectural and strategic concerns.
- **Scalability**: Adapts to projects of any size, from startups to enterprise-level systems.

Conclusion

The **AI-Powered Code Review Mentor** goes beyond traditional static analysis by combining **automation, education, and adaptability**. It empowers teams to write cleaner, more maintainable code while fostering a culture of continuous learning. By embedding intelligence directly into the review process, AICRM bridges the gap between **quality assurance** and **developer mentorship**—a problem not fully addressed by existing tools.