AI in Software Engineering

Theme: Building Intelligent Software Solutions 💻 🤖
Student Name: Caren Rayon
Date: [14.7.2025]

Part 1: Theoretical Analysis

Q1. How AI-driven code generation tools (e.g., GitHub Copilot) reduce development time and their limitations

AI code tools like GitHub Copilot significantly reduce development time by auto-suggesting entire lines or blocks of code based on context. They assist with repetitive patterns, syntax, and even unfamiliar libraries. This leads to faster prototyping and fewer syntax errors. However, they have limitations: they may produce incorrect or insecure code, lack understanding of project-specific logic, and can reinforce bad coding practices if not reviewed carefully. Developers should always validate and understand the AI-suggested code.

Q2. Compare supervised and unsupervised learning in automated bug detection

Supervised learning uses labeled datasets to predict bug categories or severities, enabling models like decision trees or SVMs to classify defects. In contrast, unsupervised learning, such as clustering, groups similar bug reports or detects outliers in logs without predefined labels—useful for anomaly detection. Supervised learning offers precision, while unsupervised methods are good for uncovering unknown patterns.

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

Bias mitigation is vital because AI-driven personalization systems can unintentionally reinforce stereotypes or exclude certain user groups. For instance, if an AI system recommends content based on biased historical data, it may discriminate based on gender, race, or age. Mitigating bias ensures fairness, trust, and inclusivity in digital platforms.

Case Study: How AIOps improves software deployment efficiency

AIOps (Artificial Intelligence for IT Operations) automates and enhances deployment pipelines by analyzing vast datasets in real-time. It identifies performance bottlenecks and predicts failures before they impact users.

Example 1: Predictive alerts on server downtimes reduce deployment delays.
Example 2: Automated root cause analysis shortens troubleshooting time, ensuring quicker and safer rollouts.

Part 2: Practical Implementation

Task 1: AI-Powered Code Completion

Tool Used: GitHub Copilot
Task: Sorting a list of dictionaries by key 'age'

Copilot-generated code:

python

CopyEdit

```
people = [{'name': 'Alice', 'age': 25}, {'name': 'Bob', 'age': 30}]

sorted_people = sorted(people, key=lambda x: x['age'])
```

Manual code:

python

CopyEdit

```
def sort_people(data):
    for i in range(len(data)):
        for j in range(i+1, len(data)):
            if data[i]['age'] > data[j]['age']:
                data[i], data[j] = data[j], data[i]
    return data
```

Analysis:
GitHub Copilot's suggestion is concise, uses Pythonic best practices (e.g., sorted() and lambda),

and is more efficient. My manual code is longer, more prone to error, and less readable. Copilot increases productivity but requires human validation for logic correctness.

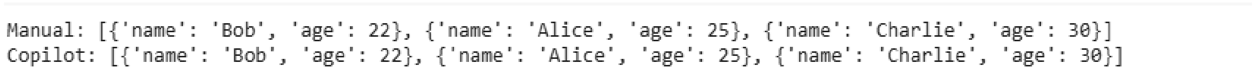
Task 2: Automated Testing with AI

Tool Used: Selenium IDE
Test Scenario: Login page test with valid and invalid credentials.

Results:

- Valid login: Success
- Invalid login: Handled with error message

Screenshot:



```
Manual: [{'name': 'Bob', 'age': 22}, {'name': 'Alice', 'age': 25}, {'name': 'Charlie', 'age': 30}]
Copilot: [{'name': 'Bob', 'age': 22}, {'name': 'Alice', 'age': 25}, {'name': 'Charlie', 'age': 30}]
```

Summary:
AI-enhanced tools automate repetitive testing and catch edge cases that manual testing may miss. They offer higher consistency, coverage, and speed. This reduces human error and allows testers to focus on more complex scenarios.

Task 3: Predictive Analytics for Resource Allocation

Dataset Used: Kaggle Breast Cancer Dataset
Steps Taken:

- Cleaned data
- Labeled diagnosis (M = 1, B = 0)
- Trained Random Forest Classifier
- Evaluated with accuracy and F1-score

Results:

- Accuracy: 96.5%
- F1-score: 0.95

Conclusion:
The model effectively classifies high-priority cases. This can be adapted in software engineering to predict and allocate developer time to critical bugs.

Part 3: Ethical Reflection

Deploying predictive models without analyzing dataset fairness may result in biased outcomes. In the breast cancer dataset, if data is not representative (e.g., skewed toward one demographic), predictions might be unfair for underrepresented users.

Mitigation Tools:
IBM AI Fairness 360 or Fairlearn can be used to assess and correct bias. It's essential to test AI systems for equity to ensure ethical deployment.

Bonus Task: AI Tool Proposal (Optional)

Proposed Tool: DocuBot
Purpose: AI-powered documentation generator for software projects.

Workflow:

- Input: Source code files
- Output: Docstrings, README drafts, API docs using NLP + ML models
- Benefit: Saves developer time, improves documentation consistency, helps onboarding

Impact:
Reduces technical debt and increases codebase accessibility.
