Theoretical Analysis — AI in Software Engineering

Course: AI for Software Engineering — Week 4
Student: Lynn Bitok
Date: 2025-10-29

Instructions

Answer each question concisely. Suggested length:

- Q1, Q2, Q3: **120–200 words each**
- Case Study: **150–250 words**

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

Answer:
AI-driven code generation tools accelerate development by providing contextual autocompletions, boilerplate code, and API usage examples directly in the editor. They reduce repetitive typing (e.g., CRUD boilerplate), suggest idiomatic patterns, and speed up onboarding by showing example calls for unfamiliar libraries. This often shortens the edit-compile-debug cycle and helps developers prototype faster.

Limitations include: (1) **Hallucinations** — the model can suggest incorrect or insecure code; (2) **Context limits** — suggestions depend on the visible context and may miss global constraints; (3) **Licensing and provenance** — generated code may unintentionally resemble licensed code; (4) **Security risks** — it can suggest insecure patterns or reveal private snippets if trained poorly. Mitigations: human review, unit tests, static analysis, secure code scanning, and adherence to licensing policies. Overall, these tools are productivity multipliers when used with careful review and test coverage.

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

Answer:
Supervised learning for bug detection requires labeled examples (e.g., commits labeled "bug" or "not bug"). Models (logistic regression, random forests, or neural nets) learn patterns associated with past bugs and can classify new instances. Pros: high precision when labels are accurate; clear evaluation metrics. Cons: requires labeled data (costly to produce) and may overfit historical bias.

Unsupervised learning (anomaly detection, clustering) does not rely on labels; instead it identifies unusual behavior in metrics, logs, or traces that deviate from learned normal patterns. Pros: useful in new projects with little labeled data and for discovering unknown failure modes. Cons: higher false positive rates and more work to map anomalies to real bugs.

In practice, hybrid approaches (semi-supervised learning, weak supervision) combine the strengths: use unsupervised methods to flag candidates, then label and train supervised models for higher precision.

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

Answer:
Bias mitigation is essential because personalization systems influence what users see and how they behave. If training data reflects historical inequities or unbalanced representation, personalization may reinforce stereotypes, marginalize minority users, or systematically favor certain groups (e.g., showing career opportunities to one gender more than another). Consequences include unfair treatment, loss of trust, legal risk, and reduced product inclusivity.

Mitigation practices: collect diverse and representative datasets; apply fairness-aware training (re-weighting, adversarial debiasing); use fairness metrics (demographic parity, equal opportunity); run audits and A/B tests across cohorts; and keep humans in the loop for sensitive decisions. Transparent explanations and options to opt-out or override personalization also reduce harm and increase user trust.

Case Study: "AI in DevOps — Automating Deployment Pipelines" — How AIOps improves deployment efficiency (two examples)

Answer:
AIOps applies machine learning to operational data (logs, metrics, traces) and improves deployments in several ways:

1. **Predictive Scaling / Resource Optimization:** By analyzing historical traffic and performance metrics, AIOps predicts load spikes and triggers auto-scaling policies before latency increases. This leads to fewer failed deployments due to resource exhaustion and smoother performance during rollouts.
2. **Automated Rollbacks & Root Cause Detection:** AIOps systems monitor key metrics and detect anomalies after a release; if regression patterns are recognized (higher error rates, latency), the system can automatically rollback the deployment or trigger targeted canary rollbacks. Additionally, ML-based log analysis helps pinpoint offending services or code paths faster than manual triage.

Other gains include smarter alerting to reduce noise, automated remediation for known incidents, and faster incident resolution via clustered log/error summaries. These reduce mean time to recovery (MTTR) and increase deployment velocity with lower operational risk.