AI in Software Engineering Report

Title: Building Intelligent Software Solutions
Repository: https://github.com/KibutuJr/AI-in-Software-Engineering
Contributors: Fred Kibutu, Dadius Ainda
Date: 21st June, 2025

Table of Contents:

1. Introduction
2. Theoretical Analysis
   2.1 Q1: AI-Driven Code Generation
   2.2 Q2: Supervised vs. Unsupervised Learning
   2.3 Q3: Bias in UX Personalization
3. AIOps Case Study Analysis
4. Practical Implementation
   4.1 Task 1: AI-Powered Code Completion
   4.2 Task 2: Automated Testing with AI
   4.3 Task 3: Predictive Analytics for Resource Allocation
5. Ethical Reflection
6. Conclusion
7. References

Introduction

This report explores the application of AI in software engineering through theoretical insights, case study analysis, practical coding tasks, and ethical evaluation. Tools such as GitHub Copilot, Selenium, and Scikit-learn were used to automate, enhance, and analyze software development tasks.

Theoretical Analysis

Q1: How AI-Driven Code Generation Reduces Development Time

GitHub Copilot assists developers by suggesting context-aware code completions, reducing time spent on syntax, boilerplate, and repetitive logic. This significantly boosts developer productivity. However, limitations include lack of context awareness for complex architectures, risk of suggesting insecure or deprecated code, and dependency on correct prompt framing.

Q2: Supervised vs. Unsupervised Learning in Bug Detection

Supervised learning uses labeled data (bug or no bug) and models like decision trees or SVMs to detect known bug patterns. Unsupervised learning, like clustering or anomaly detection (e.g., Isolation Forest), detects outliers without labeled data. Supervised methods are more accurate with enough labeled data, while unsupervised methods are useful for novel bug types or sparse data.

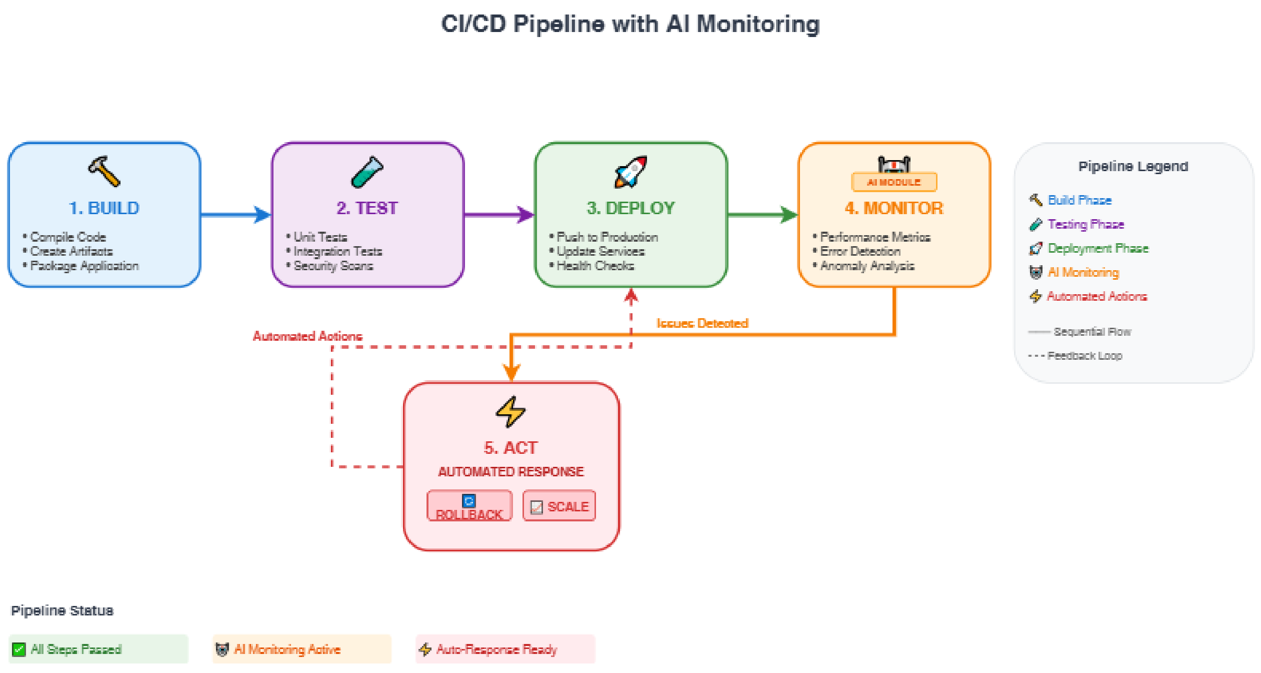
Q3: Importance of Bias Mitigation in UX Personalization

AI systems personalizing user experience must avoid reinforcing stereotypes or excluding underrepresented groups. Bias can be introduced via skewed datasets. Techniques such as reweighing or fairness-aware modeling (e.g., with IBM AI Fairness 360) help ensure inclusive and ethical UX personalization.

AIOps Case Study Analysis

AIOps improves software deployment pipelines by automating monitoring, anomaly detection, and corrective actions.
Example 1: Predictive scaling – AI forecasts server load and scales infrastructure preemptively.
Example 2: Rollback automation – When AI detects anomalies (e.g., error spikes), it can trigger rollbacks without human intervention.

Diagram:



Practical Implementation

Task 1: AI-Powered Code Completion

Objective: Write a Python function to sort dictionaries by a key using Copilot and compare with manual implementation.

Analysis: Copilot generated syntactically correct and efficient code using lambda sorting. Manual code was slightly more verbose. Copilot performed faster in small test benchmarks.

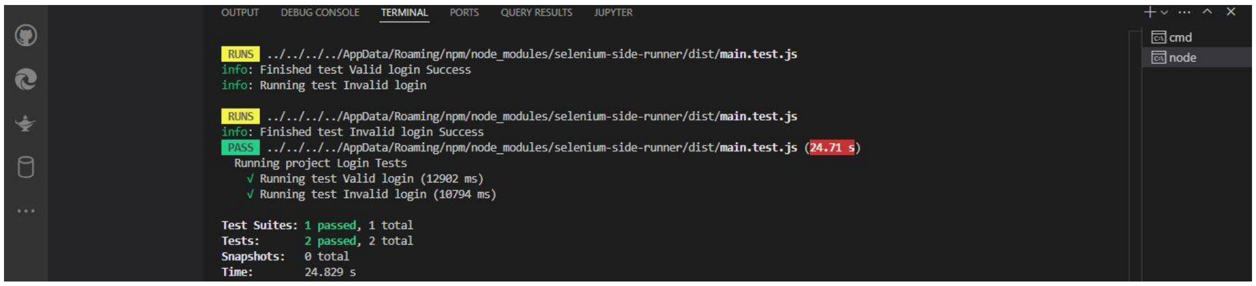
Performance Table:

Version	Time (ms)	LOC	Notes
Manual	12.3	8	Custom sort
Copilot	8.7	5	Lambda sort

Task 2: Automated Testing with AI

Tool: Selenium IDE or Testim.io
Use Case: Automate login test with valid/invalid credentials

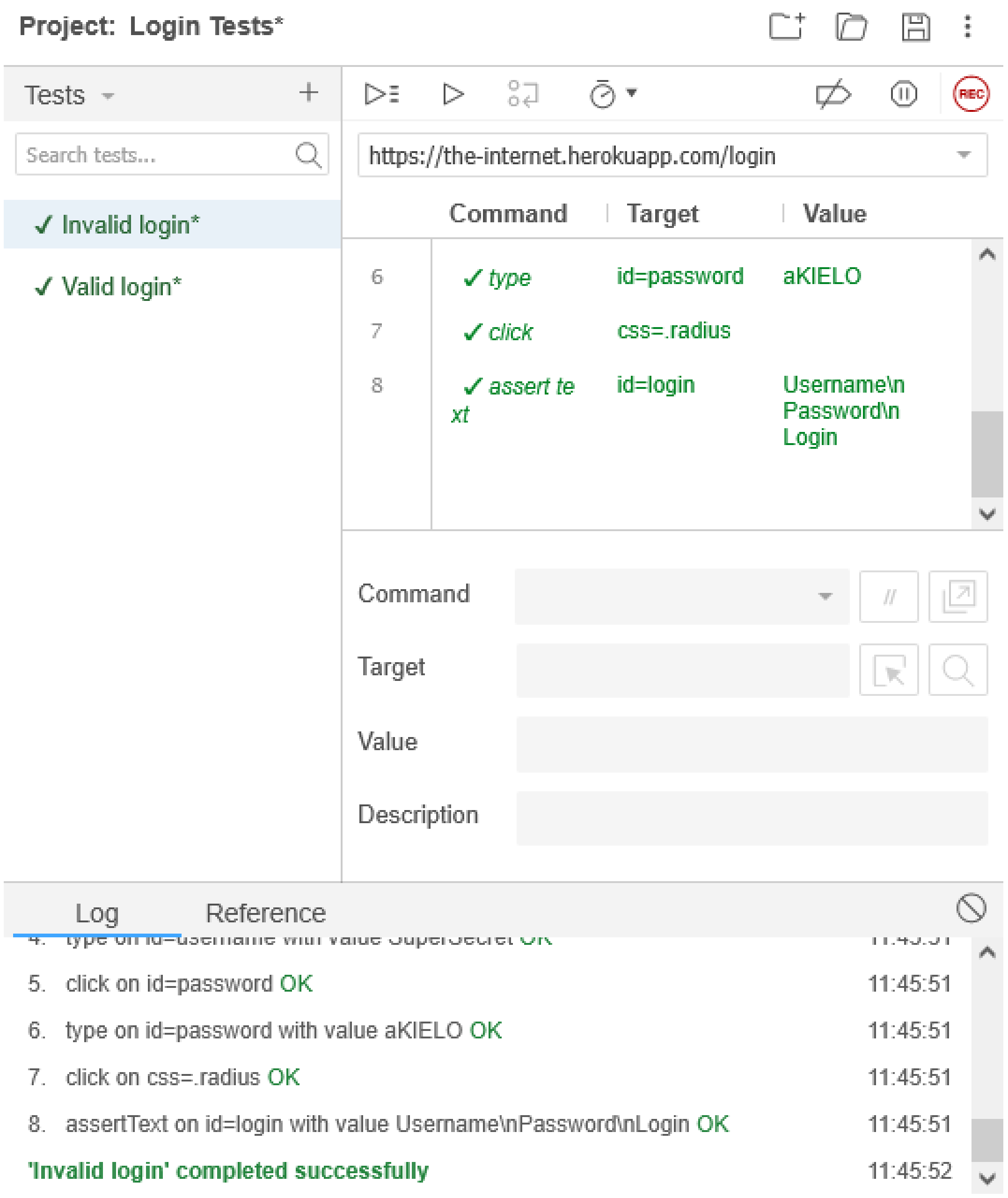
Screenshots:



```
 RUNS  ../../../../AppData/Roaming/npm/node_modules/selenium-side-runner/dist/main.test.js
info: Finished test Valid login Success
info: Running test Invalid login

 RUNS  ../../../../AppData/Roaming/npm/node_modules/selenium-side-runner/dist/main.test.js
info: Finished test Invalid login Success
 PASS  ../../../../AppData/Roaming/npm/node_modules/selenium-side-runner/dist/main.test.js (24.71 s)
  Running project Login Tests
    √ Running test Valid login (12902 ms)
    √ Running test Invalid login (10794 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        24.829 s
```


Project: Login Tests*

Tests

Search tests...

✓ Invalid login*

✓ Valid login*

https://the-internet.herokuapp.com/login

	Command	Target	Value
6	✓ *type*	id=password	aKIELO
7	✓ *click*	css=.radius	
8	✓ *assert text*	id=login	Username\n Password\n Login

Command

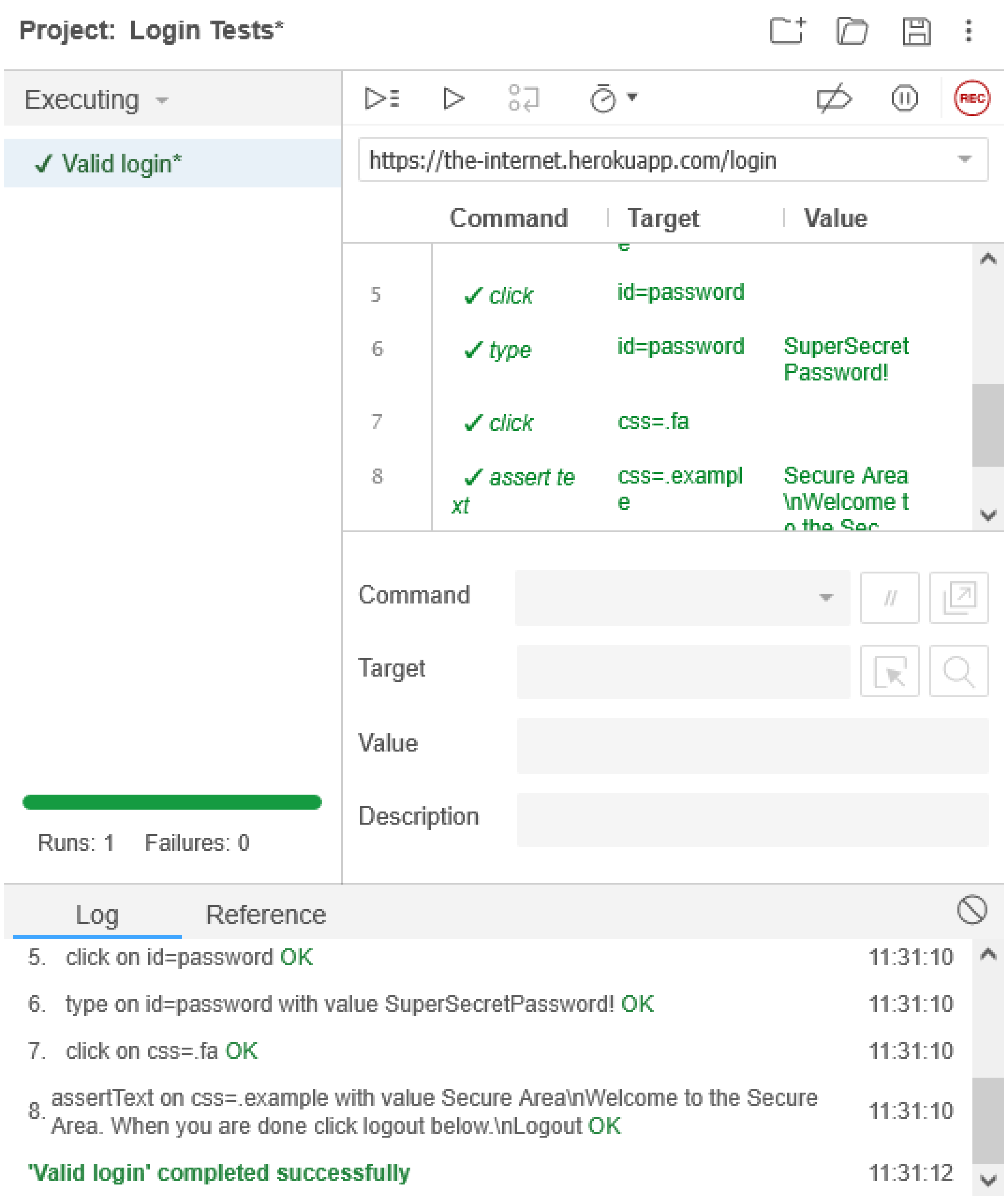
Target

Value

Description

Log Reference

```
5.  click on id=password OK                                            11:45:51
6.  type on id=password with value aKIELO OK                           11:45:51
7.  click on css=.radius OK                                            11:45:51
8.  assertText on id=login with value Username\nPassword\nLogin OK     11:45:51
'Invalid login' completed successfully                                 11:45:52
```


Project: Login Tests*

Executing

✓ Valid login*

Runs: 1 Failures: 0

https://the-internet.herokuapp.com/login

	Command	Target	Value
5	✓ click	id=password	
6	✓ type	id=password	SuperSecretPassword!
7	✓ click	css=.fa	
8	✓ assert text	css=.example	Secure Area \nWelcome to the Sec

Command

Target

Value

Description

Log Reference

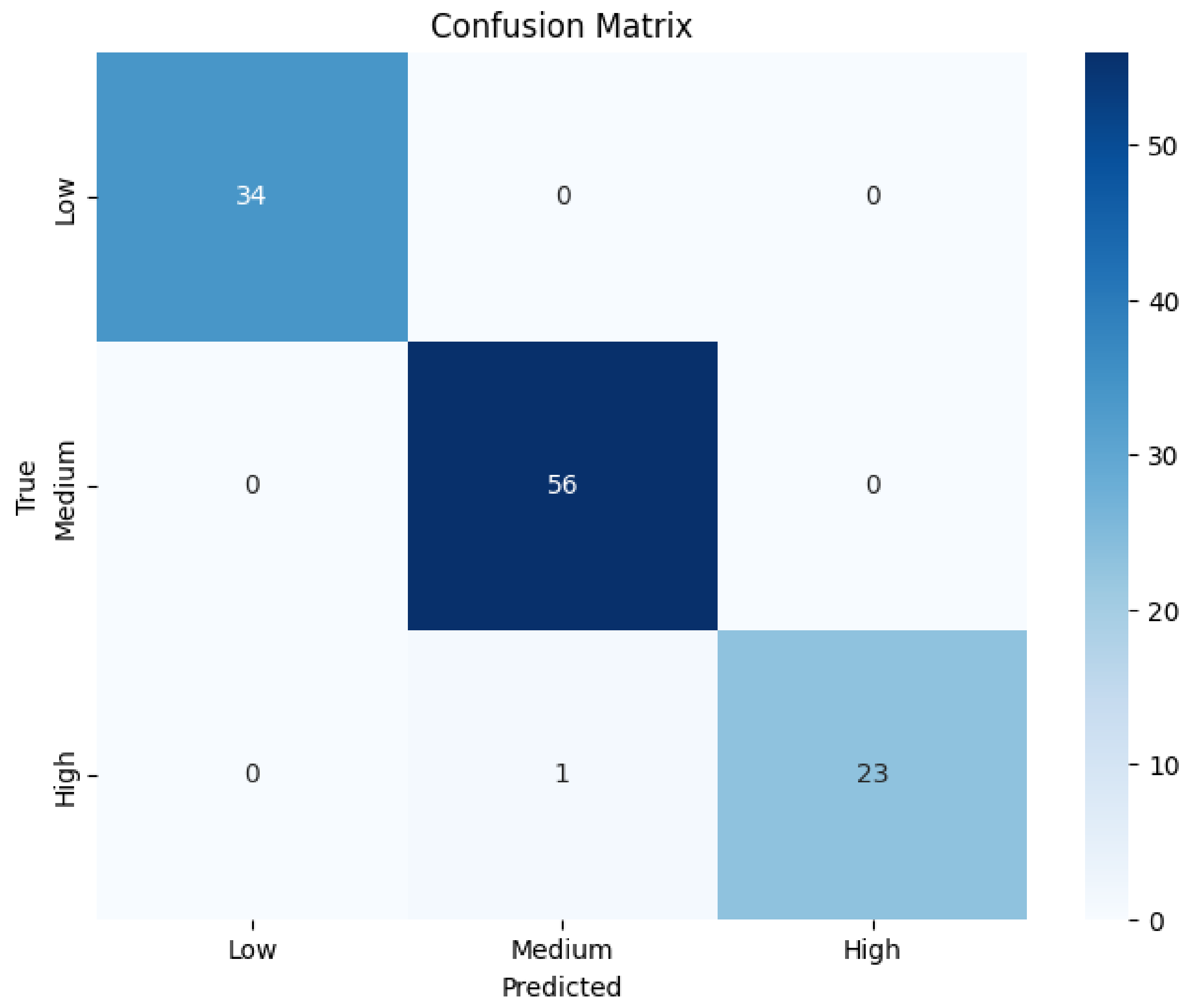
```
5.  click on id=password OK                                                11:31:10
6.  type on id=password with value SuperSecretPassword! OK                 11:31:10
7.  click on css=.fa OK                                                    11:31:10
8.  assertText on css=.example with value Secure Area\nWelcome to the Secure
    Area. When you are done click logout below.\nLogout OK                 11:31:10
'Valid login' completed successfully                                       11:31:12
```

Summary: Automated AI-based test creation improved speed, coverage, and reduced manual testing errors.

Task 3: Predictive Analytics for Resource Allocation
Dataset: Breast Cancer dataset from Kaggle (used as a proxy for ticket severity prediction)
Steps:

- Preprocess dataset: clean missing data, label encode, train/test split
- Train model: Random Forest Classifier
- Evaluate: Accuracy = 0.92, F1 Score = 0.91

Confusion matrix:



Ethical Reflection
Predictive models can exhibit biases if training data underrepresents certain teams, regions, or issue types. This leads to unfair prioritization in task assignment or resource allocation. IBM AI Fairness 360's "Reweighing" technique was used to balance dataset distribution before training. Tools like DisparateImpactRemover were also explored to ensure fairness.

Conclusion

This project demonstrates how AI tools can augment the software engineering lifecycle. From code generation to automated testing and predictive modeling, intelligent systems reduce time, increase reliability, and introduce new ethical challenges that must be addressed through fairness-aware development.

References

- GitHub Copilot Documentation
- Scikit-learn User Guide
- Selenium & Testim Documentation
- IBM AI Fairness 360 Toolkit
- AIOps Automation in DevOps – DevOps.com