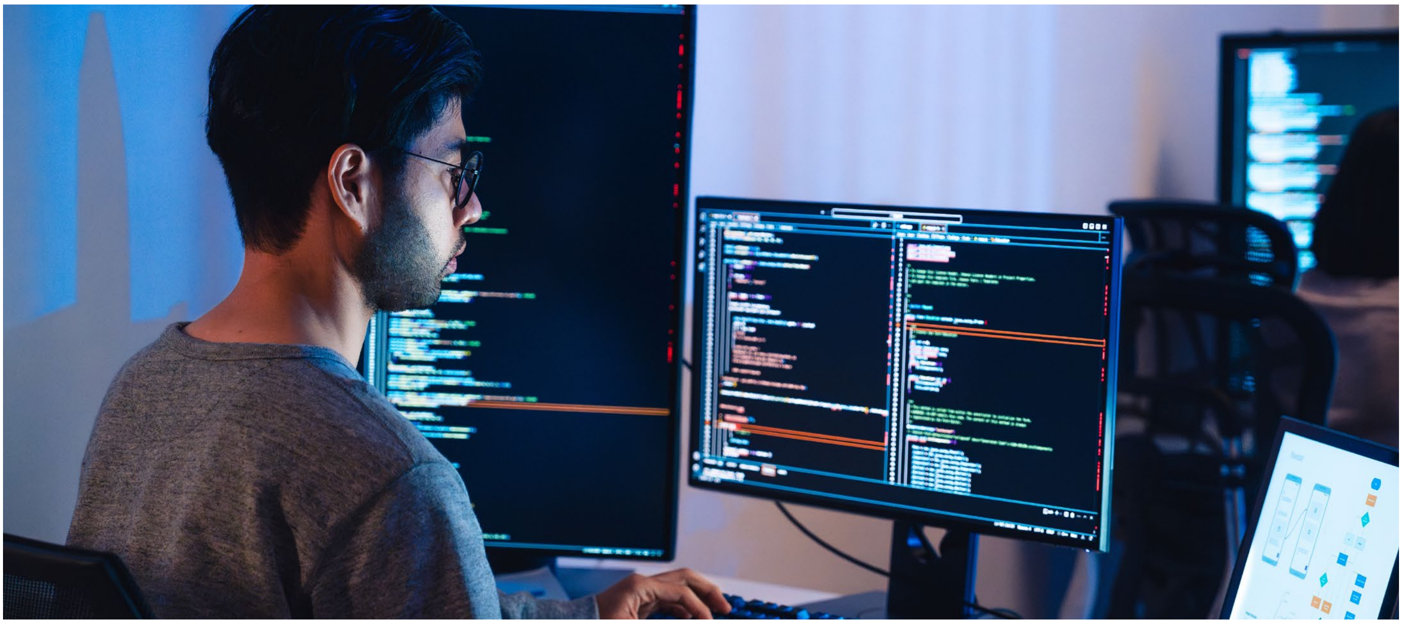
# OpenText DevOps Aviator and GitHub Copilot Extensions

Unlock trusted AI for faster, safer software delivery in enterprise DevOps



## Benefits

- Unify and streamline automated UI testing
- Fortify CI/CD pipelines with built-in security
- Boost workflow productivity and control
- Resolve defects seamlessly with contextual insights
- Clarify requirements for testable outcomes

Organizations in highly regulated and complex industries—such as automotive, finance, and healthcare—face significant challenges in software delivery. Manual documentation, fragmented test generation, and siloed defect tracking slow down development and complicate compliance within iterative, fast-paced release cycles.

OpenText™ DevOps Aviator™, now integrated with GitHub Copilot Extensions, transforms how teams manage these challenges. As part of the OpenText™ Core Software Delivery Platform, this combined solution embeds AI for DevOps, empowering developers to accelerate delivery without compromising on quality, traceability, or compliance.

Using natural language prompts, developers can automatically generate functional and compliance-ready tests, create accurate documentation on the fly, and decompose complex features into clear, manageable tasks. All of this happens within the developer's native workflow—streamlining processes, surfacing risks early through proactive testing, and enhancing overall delivery efficiency.

- **AI-generated UI testing:** Leverage Java and Selenium to automate UI testing, significantly reducing manual effort and potential errors in your testing processes.
- **AI-generated unit testing:** Implement JUnit tests to rigorously validate every fix, ensuring functionality and preventing regressions.
- **BDD requirements generation:** Transform ambiguous instructions into structured, executable BDD scenarios using Gherkin syntax, ensuring precise test coverage and rapid validation.
- **Defect analysis:** Convert bug reports and discussions into clear, actionable steps, enabling quick and accurate issue resolution.
- **Task generation:** Break down complex features into smaller, well-defined tasks, simplifying project management and execution.
- **Vulnerability detection:** Integrate STRIDE-based threat modeling directly into your CI/CD pipeline to identify and mitigate security risks early in the development cycle.

**OpenText DevOps Aviator deployment options:**

Run anywhere and scale globally in the OpenText public cloud

- DevOps Aviator runs on the OpenText Core Software Delivery Platform in the OpenText Public Cloud with a user subscription

## Unify and streamline automated UI testing

Automated testing becomes the backbone of your delivery process with OpenText DevOps Aviator and GitHub Copilot. This combined solution enforces rigorous validation through AI-generated unit tests (e.g., JUnit) to catch regressions early and maintain confidence in every release.

Beyond unit testing, teams can incorporate Java and Selenium for UI test automation, reducing manual effort while improving test reliability. By unifying functional and UI testing into one AI-enhanced pipeline, you ensure consistent quality, performance, and user experience across every deployment.

## Fortify CI/CD pipelines with built-in security

Security is integrated, not bolted on. DevOps Aviator embeds STRIDE-based threat modeling into your CI/CD workflows via the GitHub Copilot Extension, enabling real-time vulnerability detection and remediation—before code is deployed.

This shift from reactive patching to proactive threat management helps teams build security into the fabric of development. With continuous insights into your application’s risk posture, you significantly reduce the likelihood of breaches, compliance violations, and costly rework.

## Boost workflow productivity and control

Large features often introduce scope creep and delivery delays. AI for DevOps helps. DevOps Aviator helps teams break down complex features and user stories into well-defined, manageable tasks, improving estimation accuracy and reducing project risk.

This structured decomposition enables tighter control over resources, better alignment to sprint goals, and greater predictability—allowing teams to deliver high-quality software on time and within scope.

## Resolve defects seamlessly with contextual insights

Sifting through lengthy bug reports and fragmented comments wastes valuable time. DevOps Aviator uses AI for DevOps to distill defect reports and related conversations into clear, actionable insights.

Developers receive concise summaries highlighting root causes, related discussions, and resolution steps—streamlining debugging efforts and accelerating mean time to resolution. The result: reduced downtime and higher product reliability.

## Clarify requirements for testable outcomes

Ambiguous requirements can derail entire test cycles. DevOps Aviator leverages Gherkin syntax to formalize requirements into clear, shared, and executable BDD scenarios.

By transforming vague instructions into structured, testable behaviors, your teams can eliminate misinterpretation, foster collaboration, and ensure test coverage aligns precisely with business expectations, enabling faster validation and higher confidence in delivery.

## Resources

LinkedIn ›

X ›

Webpage ›

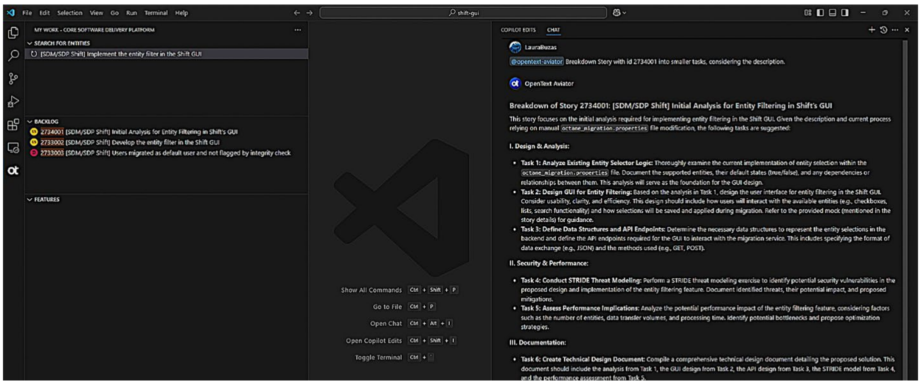
Solution overview ›

Interactive Tour ›

DevOps Cloud blog ›

Unlike generic copilots, OpenText DevOps Aviator and GitHub Copilot Extensions bring enterprise-grade precision to your development process—with built-in test optimization, traceability, and compliance-aware automation. It's purpose-built for industries where software quality and governance are non-negotiable.

By harnessing the power of AI across testing, security, planning, and debugging, your teams can accelerate value delivery, boost engineering efficiency, and reduce operational risk—giving you a strategic edge in today's competitive software landscape.



[SDM/SDP Shift] Implement the entity filter in the Shift GUI

2734001 [SDM/SDP Shift] Initial Analysis for Entity Filtering in Shift's GUI
2733002 [SDM/SDP Shift] Develop the entity filter in the Shift GUI
2733003 [SDM/SDP Shift] Users migrated as default user and not flagged by integrity check

LauraBuzas

@opentext-aviator Breakdown Story with id 2734001 into smaller tasks, considering the description.

OpenText Aviator

**Breakdown of Story 2734001: [SDM/SDP Shift] Initial Analysis for Entity Filtering in Shift's GUI**

This story focuses on the initial analysis required for implementing entity filtering in the Shift GUI. Given the description and current process relying on manual `octane_migration.properties` file modification, the following tasks are suggested:

**I. Design & Analysis:**

- **Task 1: Analyze Existing Entity Selector Logic:** Thoroughly examine the current implementation of entity selection within the `octane_migration.properties` file. Document the supported entities, their default states (true/false), and any dependencies or relationships between them. This analysis will serve as the foundation for the GUI design.
- **Task 2: Design GUI for Entity Filtering:** Based on the analysis in Task 1, design the user interface for entity filtering in the Shift GUI. Consider usability, clarity, and efficiency. This design should include how users will interact with the available entities (e.g., checkboxes, lists, search functionality) and how selections will be saved and applied during migration. Refer to the provided mock (mentioned in the story details) for guidance.
- **Task 3: Define Data Structures and API Endpoints:** Determine the necessary data structures to represent the entity selections in the backend and define the API endpoints required for the GUI to interact with the migration service. This includes specifying the format of data exchange (e.g., JSON) and the methods used (e.g., GET, POST).

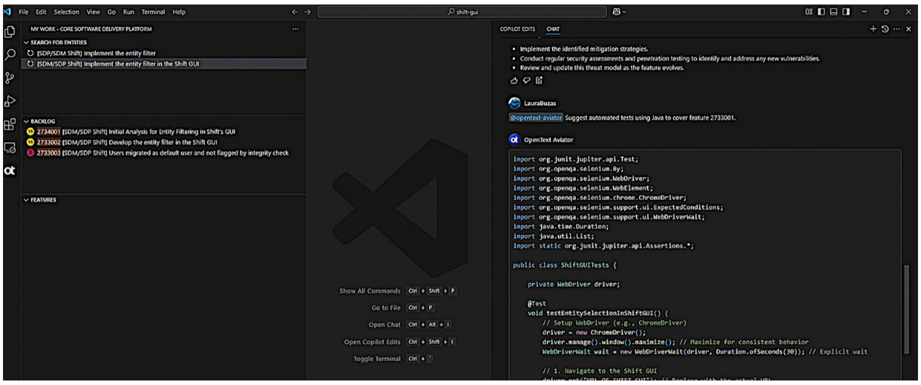
**II. Security & Performance:**

- **Task 4: Conduct STRIDE Threat Modeling:** Perform a STRIDE threat modeling exercise to identify potential security vulnerabilities in the proposed design and implementation of the entity filtering feature. Document identified threats, their potential impact, and proposed mitigations.
- **Task 5: Assess Performance Implications:** Analyze the potential performance impact of the entity filtering feature, considering factors such as the number of entities, data transfer volumes, and processing time. Identify potential bottlenecks and propose optimization strategies.

**III. Documentation:**

- **Task 6: Create Technical Design Document:** Compile a comprehensive technical design document detailing the proposed solution. This document should include the analysis from Task 1, the GUI design from Task 2, the API design from Task 3, the STRIDE model from Task 4, and the performance assessment from Task 5.

Break user stories down into tasks.



[SDP/SDM Shift] Implement the entity filter
[SDM/SDP Shift] Implement the entity filter in the Shift GUI

2734001 [SDM/SDP Shift] Initial Analysis for Entity Filtering in Shift's GUI
2733002 [SDM/SDP Shift] Develop the entity filter in the Shift GUI
2733003 [SDM/SDP Shift] Users migrated as default user and not flagged by integrity check

- Implement the identified mitigation strategies.
- Conduct regular security assessments and penetration testing to identify and address any new vulnerabilities.
- Review and update this threat model as the feature evolves.

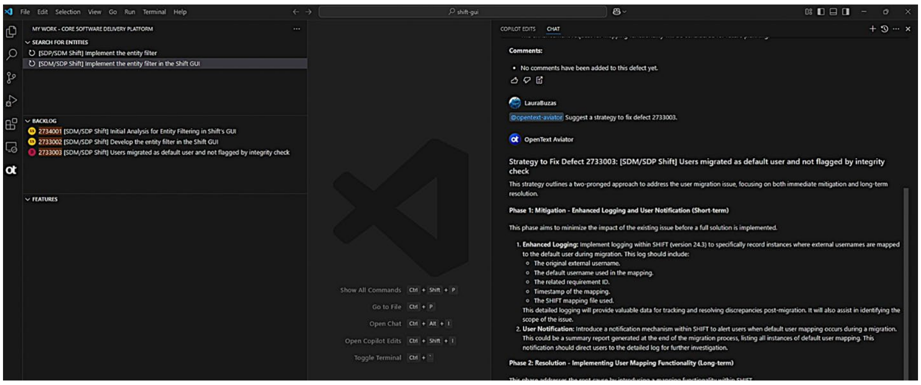
LauraBuzas

@opentext-aviator Suggest automated tests using Java to cover feature 2733001.

OpenText Aviator

```java
import org.junit.jupiter.api.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import java.time.Duration;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;

public class ShiftGUITests {

    private WebDriver driver;

    @Test
    void testEntitySelectionInShiftGUI() {
        // Setup WebDriver (e.g., ChromeDriver)
        driver = new ChromeDriver();
        driver.manage().window().maximize(); // Maximize for consistent behavior
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(30)); // Explicit wait

        // 1. Navigate to the Shift GUI
        driver.get("URL_OF_SHIFT_GUI"); // Replace with the actual URL
```

Generate automated test suggestions using Java.



[SDP/SDM Shift] Implement the entity filter
[SDM/SDP Shift] Implement the entity filter in the Shift GUI

2734001 [SDM/SDP Shift] Initial Analysis for Entity Filtering in Shift's GUI
2733002 [SDM/SDP Shift] Develop the entity filter in the Shift GUI
2733003 [SDM/SDP Shift] Users migrated as default user and not flagged by integrity check

**Comments:**

- No comments have been added to this defect yet.

LauraBuzas

@opentext-aviator Suggest a strategy to fix defect 2733003.

OpenText Aviator

**Strategy to Fix Defect 2733003: [SDM/SDP Shift] Users migrated as default user and not flagged by integrity check**

This strategy outlines a two-pronged approach to address the user migration issue, focusing on both immediate mitigation and long-term resolution.

**Phase 1: Mitigation - Enhanced Logging and User Notification (Short-term)**

This phase aims to minimize the impact of the existing issue before a full solution is implemented.

1. **Enhanced Logging:** Implement logging within SHIFT (version 24.3) to specifically record instances where external usernames are mapped to the default user during migration. This log should include:
   - The original external username.
   - The default username used in the mapping.
   - The related requirement ID.
   - Timestamp of the mapping.
   - The SHIFT mapping file used.

   This detailed logging will provide valuable data for tracking and resolving discrepancies post-migration. It will also assist in identifying the scope of the issue.
2. **User Notification:** Introduce a notification mechanism within SHIFT to alert users when default user mapping occurs during a migration. This could be a summary report generated at the end of the migration process, listing all instances of default user mapping. This notification should direct users to the detailed log for further investigation.

**Phase 2: Resolution - Implementing User Mapping Functionality (Long-term)**

This phase addresses the root cause by introducing a mapping functionality within SHIFT.

Generate suggestions to fix defects.