# NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCE

# TestOps and Future Testing Report

# GROUP MEMBERS

# Name Roll No

# Muhammad Usman 22L-7991

# Mujtaba Ali 22F-3650

# Submitted to: Sir Jawad Khalid

# TestOps and Future Testing Report

## **Task 1: Create a report on needs of TestOps, what you have covered so far, and what is missing in your frameworkT**

### **1.1 Understanding TestOps**

**TestOps** (Testing Operations) refers to integrating testing seamlessly into software development and delivery pipelines. It focuses on streamlining test automation, enabling continuous testing, and fostering collaboration between development, testing, and operations teams.

### **1.2 Needs of TestOps**

1. **Centralized Test Management:** A unified platform to manage test cases, results, and environments.
2. **Continuous Integration and Continuous Delivery (CI/CD):** Automation of testing throughout the development lifecycle.
3. **Real-Time Analytics:** Insights into the testing process and its impact on software quality.
4. **Scalability and Flexibility:** Support for growing project demands and diverse test requirements.
5. **Team Collaboration:** Bridging communication gaps between teams.
6. **Test Environment Management:** Consistent test environments to minimize false positives or negatives.

### **1.3 Requirements of Adopting TestOps**

1. **Efficiency:** Accelerates feedback loops and bug identification.
2. **Quality Assurance:** Ensures testing becomes integral to development.
3. **Improved Collaboration:** Provides visibility and accountability across teams.
4. **Scalability:** Handles dynamic project demands and increases test coverage.

##### **1.4 Coverage and Missing Aspects in Our Framework**

**Framework Coverage**: The following are features already incorporated into our test automation framework:

* **Test Design Techniques**: Techniques such as Equivalence Class Partitioning (ECP) and Boundary Value Analysis (BVA) have been implemented.
* **Web UI Automation**: Leveraged tools like Selenium with Page Object Model (POM) and BDD (Behavior Driven Development) using Cucumber.
* **Data-Driven Testing**: Utilized parameterized tests that source data from external files, including Excel and Redis.
* **API and Performance Testing**: Automated API testing using RestAssured and performance testing using JMeter.
* **CI/CD Integration**: Configured GitHub workflows for automated testing and deployment.
* **Reporting**: Implemented Allure Reports for visual insights into test execution results.

**Missing Aspects**:

* **Centralized TestOps Management**: Our framework lacks a unified management solution for test cases and results. Tools like Katalon TestOps or Testkube could address this gap.
* **Real-Time Reporting**: Missing real-time test analytics and dashboards. Katalon Analytics or Testkube Metrics can provide these insights.
* **Chaos Engineering**: Fault tolerance and resilience testing are absent. Tools like Gremlin or Chaos Monkey can simulate failures and validate system robustness.
* **AI-driven Test Automation**: The framework does not leverage AI for intelligent test automation or anomaly detection. Tools like Applitools or Testim could add this capability.
* **Scalability and Containerization**: Our framework is not containerized or Kubernetes-ready, which limits scalability. Solutions like Testkube enable distributed testing in Kubernetes environments.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aspect** | |  | | --- | |  |  |  | | --- | | **Missing Feature** | | **Potential Tools** |
| Centralized TestOps Management | No unified test management solution. | Katalon TestOps, Testkube |
| Real-Time Reporting | Limited real-time analytics and test dashboards. | Katalon Analytics, Testkube Metrics |
| Chaos Engineering | Missing fault tolerance testing for systems. | Gremlin, Chaos Monkey |
| AI-driven Test Automation | No AI-based testing for anomaly detection or predictive tests. | Applitools, Testim |
| Scalability and Containerization | No containerized setup for distributed testing in Kubernetes environments. | Testkube (K8s native orchestration) |

### TestOps and Future Testing Report Explanation

#### **Task 1: Create a Report on Needs of TestOps**

##### **1.1 Understanding TestOps**

TestOps refers to the operationalization of testing as an integral part of the software development lifecycle (SDLC). It focuses on continuous testing within DevOps, ensuring testing activities are automated and seamlessly integrated into CI/CD pipelines. This paradigm enhances test visibility, promotes collaboration, and drives faster delivery of high-quality software.

##### **1.2 Needs of TestOps**

TestOps addresses critical needs in modern software testing:

1. **Centralized Test Management**: Consolidates all test cases, results, and environment configurations into a single platform, ensuring efficiency and traceability.
2. **Continuous Integration and Continuous Delivery (CI/CD)**: Ensures that testing occurs at every stage of the development lifecycle, minimizing risks and improving feedback loops.
3. **Real-Time Analytics**: Offers dashboards and actionable insights to track software quality, testing progress, and areas for improvement.
4. **Scalability and Flexibility**: Adapts to project growth, evolving requirements, and varying test environments.
5. **Team Collaboration**: Improves communication between development, QA, and operations teams, fostering transparency and accountability.
6. **Test Environment Management**: Provides consistent and predictable test environments to reduce flakiness and inconsistencies.

##### **1.3 Requirements of Adopting TestOps**

* **Efficiency**: Streamlined workflows reduce redundant efforts and increase productivity.
* **Quality Assurance**: Testing becomes a continuous and integrated process, ensuring robust software quality.
* **Improved Collaboration**: Enhances cross-functional team interactions, ensuring alignment on goals and deliverables.
* **Scalability**: Allows organizations to handle complex projects with increased test coverage.

##### **1.4 Coverage and Missing Aspects in Our Framework**

**Framework Coverage**: The following are features already incorporated into our test automation framework:

* **Test Design Techniques**: Techniques such as Equivalence Class Partitioning (ECP) and Boundary Value Analysis (BVA) have been implemented.
* **Web UI Automation**: Leveraged tools like Selenium with Page Object Model (POM) and BDD (Behavior Driven Development) using Cucumber.
* **Data-Driven Testing**: Utilized parameterized tests that source data from external files, including Excel and Redis.
* **API and Performance Testing**: Automated API testing using RestAssured and performance testing using JMeter.
* **CI/CD Integration**: Configured GitHub workflows for automated testing and deployment.
* **Reporting**: Implemented Allure Reports for visual insights into test execution results.

**Missing Aspects**:

* **Centralized TestOps Management**: Our framework lacks a unified management solution for test cases and results. Tools like Katalon TestOps or Testkube could address this gap.
* **Real-Time Reporting**: Missing real-time test analytics and dashboards. Katalon Analytics or Testkube Metrics can provide these insights.
* **Chaos Engineering**: Fault tolerance and resilience testing are absent. Tools like Gremlin or Chaos Monkey can simulate failures and validate system robustness.
* **AI-driven Test Automation**: The framework does not leverage AI for intelligent test automation or anomaly detection. Tools like Applitools or Testim could add this capability.
* **Scalability and Containerization**: Our framework is not containerized or Kubernetes-ready, which limits scalability. Solutions like Testkube enable distributed testing in Kubernetes environments.

#### **Task 2: Explore Current and Future Testing Needs**

##### **Selected Topics**

1. **Testing and Test Automation in AI and ML**
   * **Tools**: Deepchecks, Checklist.
   * **Focus**:
     + **Model Validation**: Ensuring models meet desired performance benchmarks.
     + **Fairness Testing**: Detecting and addressing biases in AI/ML models.
     + **Data Drift Detection**: Identifying changes in data distribution that could affect model accuracy.
     + **Robustness Testing**: Evaluating model performance against edge cases or adversarial inputs.
2. **System Reliability Testing with Chaos Engineering**
   * **Tools**: Gremlin, Chaos Monkey.
   * **Focus**:
     + Simulating failures in production-like environments.
     + Validating the system's ability to recover and maintain functionality.
     + Ensuring resilience under stress or adverse conditions.
3. **Containerization and CI/CD Pipelines for Test Automation**
   * **Tools**: Testkube, Docker, Kubernetes.
   * **Focus**:
     + Building containerized testing environments to improve scalability.
     + Automating orchestration of tests within Kubernetes.
     + Leveraging Kubernetes-native solutions like Testkube for efficient test management.

### **1. Testing in AI/ML**

Testing in AI/ML pipelines is critical due to the complexity and variability in data and models. It involves:

#### **a. Data Validation**

* Checking data quality, distribution, and consistency.
* Identifying missing, duplicate, or outlier data.

#### **b. Model Validation**

* Evaluating model performance using metrics (e.g., accuracy, precision, recall, F1 score).
* Validating robustness, fairness, and explainability.

#### **c. Deployment Testing**

* Ensuring the deployed model behaves as expected in real-world scenarios.
* Testing for model drift and data drift.

### **2. Test Automation in AI/ML**

Automation focuses on making testing repeatable, scalable, and efficient. Tools like **DeepChecks** and **CheckList** are invaluable.

#### **a. DeepChecks**

* **Purpose**: Automates data and model validation.
* **Key Features**:
  + Data validation: Check for missing values, distribution changes, and duplicates.
  + Model validation: Test for overfitting, underfitting, or incorrect assumptions.
  + Model drift: Detect data and prediction drift over time.

**Example Usage:**

**from deepchecks.tabular.suites import model\_evaluation**

**suite = model\_evaluation()**

**suite\_result = suite.run(train\_dataset=train\_dataset, test\_dataset=test\_dataset, model=model)**

**suite\_result.save\_as\_html() # replace this with suite\_result.show() or suite\_result.show\_in\_window() to see results inline or in window**

**# or suite\_result.results[0].value with the relevant check index to process the check result's values in python**

#### **. CheckList**

* **Purpose**: Automates NLP testing by evaluating models on specific behaviors and edge cases.
* **Key Features**:
  + Behavioral testing for NLP models.
  + Template-based creation of linguistic and logical tests.

Example Usage:

import checklist

from checklist.test\_suite import TestSuite

suite\_path = 'release\_data/sentiment/sentiment\_suite.pkl'

suite = TestSuite.from\_file(suite\_path)

### **Summary**

This report highlights the importance of TestOps in modern software testing and identifies gaps in the existing framework, providing a roadmap for future enhancements. Exploring advanced topics like AI/ML testing, chaos engineering, and containerized test automation ensures alignment with evolving industry trends, making the framework more robust and versatile.