littlepay

Quality Metrics

To ensure a high-quality release in an Agile environment, it is crucial to establish and track **quality metrics** that provide insight into the effectiveness and efficiency of the testing process. These metrics help measure both the product's quality and the testing process itself.

Here's a list of quality metrics that can be used to assess the test results for the Littlepay coding exercise and ensure a high-quality release.

These are the ideal metrics to be captured & maintained for any delivery process and teams can adopt that suit them best based on their strategy.

1. Test Coverage

Definition: Percentage of code or requirements covered by tests.

Why It's Important: High test coverage ensures that the majority of the application is tested and that no critical functionality is left untested.

How to Measure:

- **Code Coverage**: Use tools like JaCoCo, Cobertura, or SonarQube to measure the percentage of the codebase covered by unit tests.
- **Requirement Coverage**: Ensure that all business requirements (e.g., correct charge calculation, handling of tap events, etc.) are covered by tests, particularly edge cases like incomplete trips and cancelled trips.

Target: 80% or higher (depending on the project's complexity).

---

2. Defect Density

Definition: Number of defects found per unit of code (e.g., per 1,000 lines of code) or feature delivered.

Why It's Important: It indicates how many defects exist in the code relative to its size. A high defect density can indicate poor-quality code or inadequate testing.

How to Measure:

- Count the number of defects logged during testing (found by both automated and manual testing).

Target: Ideally, defect density should be low, and any critical defects should be resolved before release.

---

3. Defect Leakage

Definition: Percentage of defects found in production after the release compared to the total number of defects found in testing.

Why It's Important: Defect leakage indicates the effectiveness of your testing process. Low defect leakage means that most issues are found during testing, ensuring fewer bugs in production.

How to Measure:

- Track the defects found in the production environment after the release.

Target: Less than 5% (the goal is to catch as many issues as possible before production).

---

4. Test Pass Rate

Definition: Percentage of tests that pass successfully during the testing cycle.

Why It's Important: A high test pass rate indicates that the application behaves as expected according to the test cases.

How to Measure:

- Count the number of tests executed and the number that passed.
- Calculate the pass rate:
  $\text{Test Pass Rate} = \text{Number of Tests Passed} / \text{Total Number of Tests} \times 100$

Target: 95% or higher.

---

5. Defect Resolution Time

Definition: Average time taken to fix defects after they are identified.

Why It's Important: The faster defects are fixed, the sooner the testing team can revalidate the fixes and ensure the release is on time.

How to Measure:

- Track the time taken from defect identification to defect resolution.
- Calculate the average defect resolution time.

Target: Resolve defects within 1–2 days for high-priority issues, and within a sprint for lower-priority issues.

---

6. Test Execution Efficiency

Definition: Measure of how efficiently tests are executed, particularly in automated testing environments.

Why It's Important: Efficient test execution helps ensure that testing doesn't become a bottleneck in the development process.

How to Measure:

- Track the time it takes to run tests (both manual and automated).
- Measure the percentage of automated tests executed compared to manual tests.

Target: Automate at least 70% of tests, with 90% of automated tests executing within a reasonable time frame (e.g., under 30 minutes for a full suite).

---

7. Test Automation Rate

Definition: The percentage of automated tests compared to the total number of tests.

Why It's Important: High automation improves testing efficiency, especially for regression tests, and ensures tests are repeatable.

How to Measure:

- Count the number of automated tests and the total number of tests (manual + automated).
- Calculate the automation rate:
  Test Automation Rate=Number of Automated TestsTotal Number of Tests×100

Target: 70% or higher.

---

8. Test Stability

Definition: Measure of how often tests fail due to issues in the test itself (e.g., test flakiness) rather than the system under test.

Why It's Important: Unstable tests waste time and resources, as tests that fail for non-system reasons give false results.

How to Measure:

- Track the number of flaky or unstable tests (tests that fail intermittently).
- Calculate the percentage of flaky tests:
  Test Stability=Number of Stable TestsTotal Number of Tests×100

Target: 95% or higher stable tests.

---

9. User Satisfaction (Post-Release)

Definition: The satisfaction level of users or stakeholders after the release, measured through feedback or surveys.

Why It's Important: User satisfaction directly correlates with the perceived quality of the product. High satisfaction indicates the application is stable and meets user expectations.

How to Measure:

- Collect feedback from end-users or stakeholders after release.
- Use surveys or direct feedback to measure satisfaction on a scale (e.g., 1–5).

Target: 4 or higher (out of 5) on satisfaction surveys.

---

10. Continuous Delivery Metrics

Definition: Metrics that measure how quickly and safely changes are deployed to production.

Why It's Important: It indicates the speed of development and the reliability of the deployment pipeline.

How to Measure:

- Track deployment frequency (how often new versions are deployed to production).

- Track lead time (how long it takes from code being committed to being deployed to production).

Target:

- Deployment frequency: Multiple deployments per week.
- Lead time: Less than 1–2 days from commit to deployment.

---

11. Customer Impact

Definition: Measures how much the application impacts users (e.g., performance issues, downtime).

Why It's Important: Ensures that the application provides a smooth user experience and doesn't impact end users negatively.

How to Measure:

- Track incidents of downtime, performance degradation, or user complaints after release.
- Monitor key system performance indicators (e.g., load time, system crashes).

Target: No major incidents, and response time under acceptable thresholds (e.g., <2 seconds load time).

Conclusion

Tracking these quality metrics provides a comprehensive view of the testing process and the product's quality. Monitoring these metrics during the Agile cycle helps identify issues early, optimize the testing process, and ensure that the release is of high quality. Continuous improvement based on these metrics ensures that each release is better than the last and meets both functional and non-functional requirements.