**A Practical Approach & Practises for Building Quality Software with Automation.**

Vikas Kumar Choudhary

Pacific Academy of Higher Education and Research

Ph.D. Computer Science

**Abstract:**

In order to establish consistent standards and procedures for the operations, development, modification, and acceptance of products and services, any organization need to follow some best practises and approach.

This document provide some guidelines to practice Quality Standards and checkpoints during product and service development cycle to build both functional and operational quality from the start. Details discussed herewith are independent of the type of Software Development Lifecycle (SDLC): Waterfall or Agile or any other software Life Cycle Methodologies.

**1. Introduction:**

A software Development Organization has several test phases and environments in which monthly and/or quarterly release testing is executed. In the lower level of development and test environments, the focus is on **unit**, **functional product owner acceptance,** and **performanc**e testing.

Once code has passed these test phases, it is bundled and promoted to an internal integration test environment for **integration, regression**, and **end-to-end performance** testing.

Code deployments are also a planned/coordinated event which follow a consolidated implementation plan.

Once the internal integration test environment testing has been completed, the code is promoted to another Layer of Quality check to support **external customer** testing with real segregated/locked-down environment. After the partner testing cycle completes in, the **code is then promoted to production** for general use.



At each level/phase of development and testing there should be **Quality gates** in place *prior* to code promotion to the next environment. Defects should be formally tracked, and metrics should be routinely produced to ensure consistent quality.

**2. Operating Model:**

Quality Engineering principles, tools, and practices should be designed to help plan and build quality from the start hence stakeholders must focus on the following key principles:

🡪Execute automation in all environments for speed and quality

🡪Drive customer feedback to development by proactively monitoring production and incidents

🡪Enable performance and scale from the start

🡪Promote common tools and standard development-test practices with a focus on audit compliance --🡪Evangelize quality culture by fostering engineering communities

Quality improvements are only possible when there’s role clarity, a clear focus, and capacity allocation to build enablers.

Professionals should come together to contribute to common tools and practices by embracing open source culture to build enterprise-wide assets and enable re-use across organization.

**3. Maturity Assessment Model**

The quest for holistic quality starts with understanding and benchmarking current practices, use of enterprise standard tools, and maturity.

🡪Code Quality: assessment related to code, security checks, and Continuous Integration

🡪Automation: assessment related to automation layers, reporting, environment, and metrics

🡪Performance: performance strategy, tools, types of testing, reporting, and metrics

🡪Quality Checks: gating checks throughout Software Development Life Cycle and environments

**What Does Quality Gating do**?

Quality gates provide key technology checkpoints that must be leveraged as exit criteria to advance an initiative and/or a feature from Planning to Development, Integration, Customer Acceptance, and Production. These gates, in addition to the gates set by Business Operations, will provide a holistic framework to ensure operational and functional quality.

Business operations gates will compliment these quality gates by clearly articulating other must have artifacts.

**4. Planning:**

All stakeholders, must engage early in the planning cycle to understand requirements, prioritization, acceptance criteria, estimations, and integration points to deliver end-to-end test and performance strategies.

**🡪Performance Targets**

Stakeholders should have product owner document, then build the performance strategy covering testing and tuning strategies including load, breakpoint, stress, endurance, spike, front end benchmarking, and other forms of performance tests.

**🡪Test Strategy and Environments**

Stakeholders, must build an end-to-end test strategy detailing requirements such as test data, testing types (even covering disaster recovery, field testing, migration testing, etc.), environments, connectivity, tools, simulators, lab setups, test suites, defect lifecycle, acceptance criteria, and such.

**🡪Testing Approaches**

It should ensure that proper estimation of stories and definition of Story READY and DONE. Definition of Done must include in-sprint automation (unit, functional, customer journey, and performance), monitoring, logging, and other aspects required for Development exit gating.

**🡪Quality Gates**

All Quality criteria for planning exit gating are met and should be signed off prior to entering development phase.

**5. Development Phase**

**🡪Code Check-in Process:**

Each code check-in initiates an incremental build of the code that was changed (i.e. not a clean build of the entire codebase). Build results (indicating success of the build, including unit test results, of the changed code) should be automatically emailed to each member of the team after each check-in.

**🡪Automated Nightly Build Process**

1. Each night CI (Continuous Integration system) performs a clean build of entire codebase.
2. Runs all unit, functional and performance tests (except long running end to end tests).
3. Build results (indicating success of the build, including test results) are automatically emailed to each member of the team after each nightly build.

🡪Unit Testing

1. Unit tests are created by the developers during the iteration.
2. Unit tests should run automatically as part of the **code check in process** and as part of the **automated nightly build process.**
3. Unit performance testing to be performed as part of the CI build process.

**🡪Peer Reviews**

A story should be peer reviewed within the team to be considered complete

Code must be checked into the main branch only after peer review, pull requests are used for peer code reviews. Code cannot be checked-in to a remote branch without an approved pull request.

**🡪Functional Testing**

Functional test automation programs should be developed by the Stakeholders where feature development velocity exceeds test development velocity) for each API and UI component**.** In some cases these tests will require a virtualized API (mock) of external systems or environments with production-like data. The respective Stakeholder should work with the developer to implement the mock and write automated scripts for data seeding.

Once Story is marked Completed, it is reviewed by the Product Owner/delegate for acceptance.

**5.1 Development Acceptance**

Acceptance of developed stories should be done in a dedicated Staging environment. Types of testing in this environment includes:

**Deployment Validation Testing**

Do preliminary automated testing to reveal simple failures severe enough to reject a prospective software release. Then cover the most important functionality of a component or system to ascertain if crucial functions of the software work correctly.

**Critical Regression Testing**

Subset of regression test set that can be run on a daily basis to ensure critical functionality is not broken. This is also used for quick regression testing while promoting production hot fixes.

**User Acceptance Testing should be** performed by the product owner.

**Quality Gates:**

All Quality criteria for Development exit gating are met and signed off prior to promotion to next environment for end-to-end Integration testing and performance.

**6. Integration - Performance Engineering, Testing, and Tuning**

1. A dedicated, but scaled environment (in terms of capacity), with production hardware and configuration should be prepared.
2. Unit Performance tests should be there for each API and UI component. A subset of the performance tests (including some load tests) should be run as part of the automated nightly build process.
3. Do automated data setup, execution and analysis of the end-to-end performance tests with the goal of automating this process incrementally using the Continuous Integration (CI) tool stack.
4. Longer running end-to-end performance tests should be executed in a dedicated end-to-end performance environment on a regular cadence.
5. Performance Quality Gatesto be met before promoting the code to customer test environment. This includes no blocker performance issues and sign off on performance.

**6.1 Types of Performance Testing**

Each type of performance tests (API, UI and mobile) capabilities must capture end-to-end response times, including last mile, under real user and real device conditions

1. *Performance Smoke Test*: Test early in development using self-service tools
2. *Stress Testing*: Test beyond normal or peak load
3. *Load Testing*: Test for normal or peak loads for SLAs
4. *Capacity Testing*: Test for scaling systems future needs
5. *Soak Testing*: Test for normal or peak load for an extended duration
6. *Spike Testing*: Test for sudden spike loads and impact on systems
7. *Break Point Testing*: Test to determine system break points

**7) Integration – Functional Testing**

Integration testing of applications and services are done in a dedicated and controlled environment such as ITF and/or dedicated Stage environment. This is the first end-to-end integrated environment containing all systems to perform various tests and exercise implementation plans for production like deployment rhythm.

Preliminary automated testing to reveal simple failures severe enough to reject a prospective software release.

**Exploratory Testing**

An informal design technique where the stakeholders actively controls the design of the tests and uses information gained to design new and better scenarios. As part of exploratory testing, consumer touchpoints can be used to validate customer experience from different geographies and devices using Crowd sourced testing.

**Full Regression Testing**

Full validation of an application, following modification, to ensure defects have not been introduced in unchanged areas of the software.

Performed when the software or the environment has changed.

Must cover key customer journeys for a given product(s) or service(s)

**Customer Issue Testing**

A set of test cases that have been generated based on prior customer reported or impacted defects, problems, and incidents.

**New Functionality Testing**

Test cases that are based on new features or functionality.

**Crowd Sourced Testing (Consumer Solutions)**

Allows you to identify specific market/regional testing scenarios in specific geographical locations using real devices on real networks in real user conditions by:

🡪Targets specific demographics/scenarios by selecting testers from a pool of 200,000+ testers in 200+ countries.

🡪Desired mix of device, OS, carrier and browser coverage for both mobile and web.

**Quality Gates**

All Quality criteria for Integration exit gating are met and signed off prior to promotion to next environment for end-to-end Customer Acceptance.

**8. Customer Acceptance**

Customer testing of applications and services are done in a dedicated and controlled environment (i.e. sandbox). Customer testing environment must have data set ups and network topology that closely mirror production set ups and network connectivity to customer test environments and labs.

**Deployment Validation Testing**

Preliminary automated testing to reveal simple failures severe enough to reject a prospective software release.

**Functionality**: Customer performs testing for compatibility and key functionality by driving tests from customer test environment/labs.

**Critical Regression and Customer Journeys validation (Including Customer Journeys and Persona)** This testing to be conducted by Organization, if required in partnership with customers, to validate end customer journeys and personas using production like customer data and accounts set ups. Automating these customer tests and reporting enables speed to market and instils confidence in our customers by enabling them focus on new feature testing rather than regression.

**Usability and exploratory testing*:*** Crowd sourced testing to validate et flows (regional functionality, regression, localization) before launching to wider customer base.

**Quality Gates**

All Quality criteria for Customer Acceptance/Pre-production gating should met and signed off prior to promotion to next environment for end-to-end Customer Acceptance.

**9. Production Launch**

During this phase, software is deployed into Production and will follow with below steps.

**Deployment Validation Testing should be done**

Preliminary automated testing to reveal simple failures severe enough to reject a prospective software release.

**Field Validation**

Some applications go through Field validation by crowd source testers during launches by whitelisting accounts. This helps to capture customer experience, validate localization (e.g., terms and conditions), accessibility, etc. before the broader release.

**User Experience/Synthetic monitoring:**

Leverage critical regression scripts covering key customer functionality to execute synthetic tests in production at a regular frequency (say every 5 to 10 minutes). This will help to confirm our systems are not only reported available within our internal monitors but are always working for our customers.

Quality Gates: All applicable Post-Production gating activities are carried continuously to service customers with highest availability and performance.

**10. Pre-Development**

Stories must be articulated to detail monitoring/alerting, capacity planning, testing, security compliance risks, and downstream impacts.

**Monitoring/Alerting:**

The product has operational diagnostic tools installed and tested.

Logging standards are documented for developers to aid in system and business event discovery.

The product has a business transactions and system health dashboard validated by Business and Platform operations.

Synthetic monitoring from different geo-locations is set up to capture end user experience.

**Capacity planning:**

Business pipeline and system performance metrics are combined to arrive at capacity planning (compute, network, and storage) for short-term, medium term, and long term.

**Security compliance:**

Compliance strategy and validation plans to meet as per standards. Other data privacy needs must be explicitly captured (e.g. on-soil needs for a specific country, etc.)

**11. Pre-Production**

As part of pre-production checks, Operations and Business Operations need documentation to effectively monitor and support the service.

**Documentation:**

Technical Architecture Design (TAD) should be in place with necessary updates to articulate system architecture including connectivity with other internal and/or 3rd party systems with detailed server/network information.

Release notes covering new functionality and defect fixes, customer communications, and product training document for customer facing staff.

**Validations:**

Validations must adhere to Quality Engineering Integration gating requirements and sign-offs.

**12. Post-Deployment**

Business operations should monitor program health, periodic validations and monitoring, availability metrics, customer impacting incidents, and problem management in ITSM Systems

Customer support teams to capture business and availability metrics for external reporting and driving customer feedback.

**13. Acknowledgement**: I will like to thank Dr. Sanjay Chaudhary for constant support and guidance.

**References**:

1. Clean code:  **by Robert C. Martin**

# Code Complete: A Handbook of Agile Software Craftsmanship (Robert C. Martin)

1. Software Engineering: A Practitioner's Approach Paperback – 1 Jul 2017,Publisher**:** McGraw Hill Education
2. The Pragmatic Programmer by Andrew Hunt, Pearson Education,