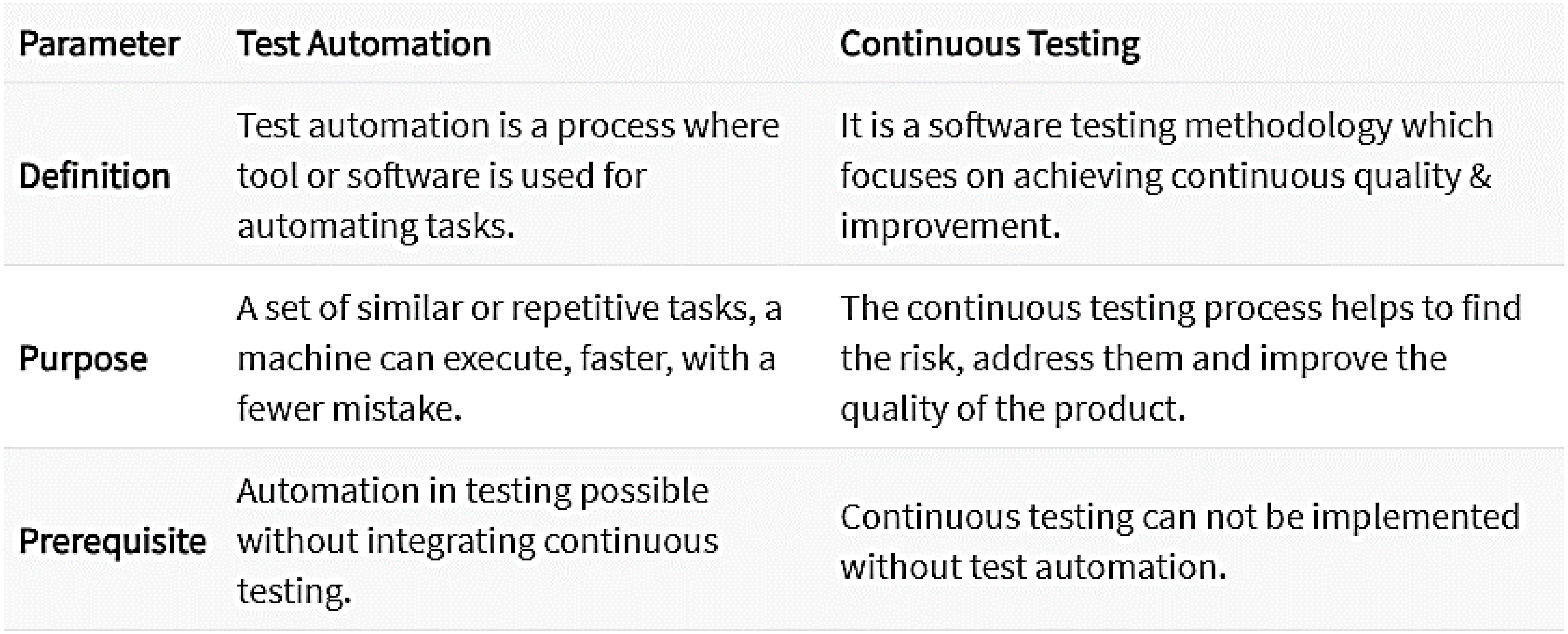
innovate achieve lead

BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Module 9 ,10

How Is Continuous Testing Different from Test Automation?



Parameter	Test Automation	Continuous Testing
Definition	Test automation is a process where tool or software is used for automating tasks.	It is a software testing methodology which focuses on achieving continuous quality & improvement.
Purpose	A set of similar or repetitive tasks, a machine can execute, faster, with a fewer mistake.	The continuous testing process helps to find the risk, address them and improve the quality of the product.
Prerequisite	Automation in testing possible without integrating continuous testing.	Continuous testing can not be implemented without test automation.

Time	Software release can take a month to years.	Software release may be released weekly to hourly.
Feedback	Regular feedback after testing each release.	Feedback at each stage needs to be instant.
History	Automated testing has been done for decades to make the testing process faster.	Continuous testing is a relatively newer concept.

Continuous Testing Tools

1) QuerySurge

QuerySurge is the smart data testing solution that is the first-of-its-kind full DevOps solution for continuous data testing. Key features include Robust API with 60+ calls, detailed data intelligence & data analytics, seamless integration into the DevOps pipeline for continuous testing, and verifies large amounts of data quickly.

2) Jenkins

Jenkins is a Continuous Integration tool which is written using Java language. This tool can be configured via GUI interface or console commands.

3) Travis

Travis is continuous testing tool hosted on the GitHub. It offers hosted and on-premises variants. It provides a variety of different languages and a good documentation.

4) Selenium

Selenium is open-source software testing tool. It supports all the leading browsers like Firefox, Chrome, IE, and Safari. Selenium WebDriver is used to automate web application testing.

Best Practices for SQA Implementation

Core Principles for Effective SQA

Define Clear Quality Objectives:

- Set measurable goals like defect density thresholds, code coverage percentages, and customer satisfaction scores.
- Align objectives with project requirements and expectations.

Adopt a Comprehensive SQA Plan:

- Document scope, schedule, risk assessment, and metrics.
- Continuously update the plan to reflect lifecycle changes

Early QA Involvement and Standardization

Involve QA Early in Development:

- Apply the shift-left approach to integrate QA in requirements and design phases.
- Conduct requirements validation and design reviews.

Use Standardized Processes and Guidelines:

- Implement coding standards, design principles, and testing protocols.
- Follow standards like ISO 9001, CMMI, or IEEE

Automation and Continuous Testing

Automate Testing Where Possible:

- Leverage tools like Selenium, JUnit, or TestNG for unit, regression, and performance testing.
- Integrate testing automation into CI/CD pipelines.

Embrace Continuous Testing:

- Include testing in CI/CD workflows using tools like Jenkins or GitLab CI.
- Execute automated tests after every code commit to detect issues early.

Quality Assurance in Different Development Methodologies

Waterfall Model

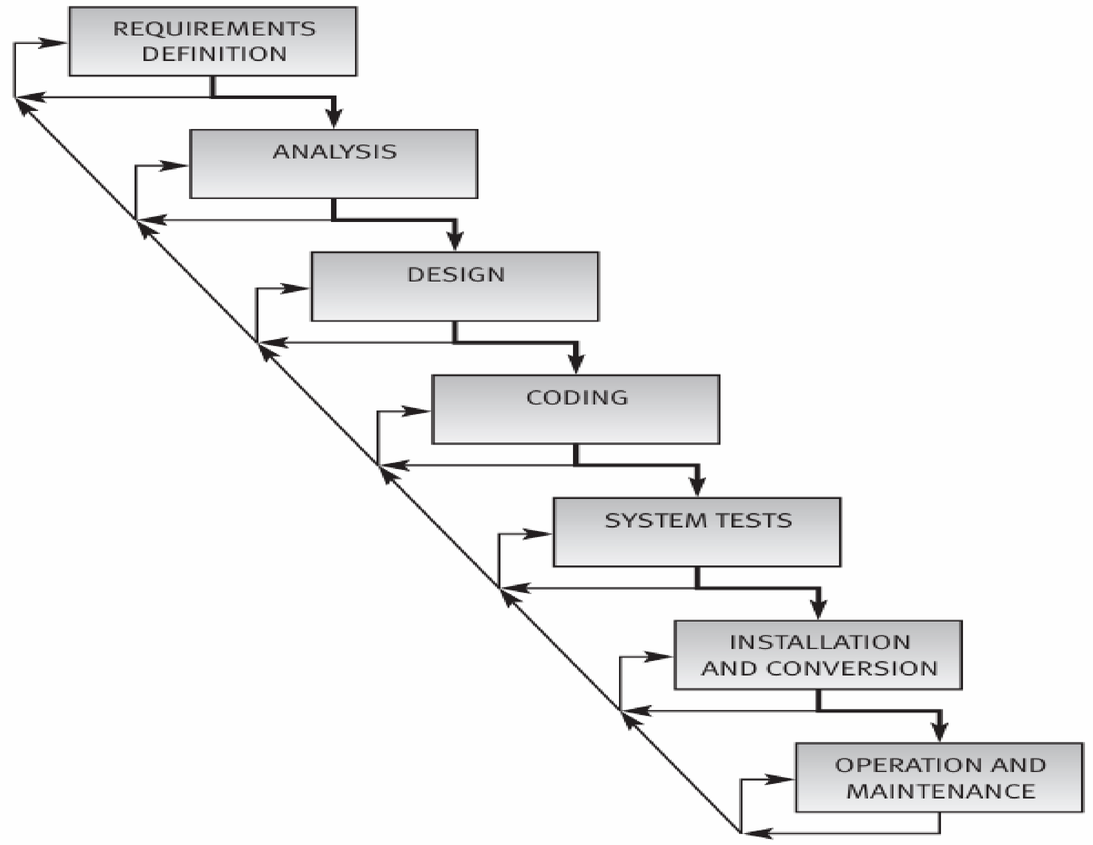


Figure 7.1: The waterfall model
Source: After Boehm (1981) and Royce (1970) (© 1970 IEEE)

1. **Design**: Define outputs, inputs, processing procedures, data structures, databases, and software architecture in detail.
2. **Coding**: Translate the design into code with quality assurance activities like inspections, unit tests, and integration tests.
3. **System Tests**: Conduct comprehensive tests to identify and correct errors. Developers perform these tests, followed by customer acceptance tests or joint system tests to ensure commitments are met.
4. **Installation and Conversion**: Install the approved system as part of a larger setup or replace an existing system, ensuring uninterrupted operations during conversion.
5. **Regular Operation and Maintenance**: Begin system operation, providing:
    - **Corrective Maintenance**: Fix user-reported faults.
    - **Adaptive Maintenance**: Adjust to new requirements.
    - **Perfective Maintenance**: Enhance system performance with minor features.

Prototyping Model

Iterative development using prototypes for user feedback.

Evolved prototypes lead to a refined system or complete software.

Advantages: High user involvement and reduced failure risks.

Limitations: Reduced developer flexibility.

Spiral Model

Combines iterative development, risk analysis, and user feedback.

Activities in each iteration:

1. Planning and defining objectives.
2. Risk analysis and resolution.
3. Engineering activities (design, coding, testing).
4. Customer evaluation and feedback.

Object-Oriented Model

The **object-oriented model** focuses on **reusing software components** (objects or components) through a **software component library** to streamline development (Figure 7.5).

Development Process:

1. Conduct object-oriented **analysis and design**.
2. Acquire **reusable components** from the library, if available; otherwise, proceed with regular development.
3. Stock newly developed components in the library for future reuse.

Benefits:

- **Economy**: Reusing components is more cost-effective than developing new ones.
- **Improved Quality**: Reused components are more reliable due to prior defect detection by other users.

Importance of a Quality Culture

📌 **Why Quality Culture Matters:**

- Ensures **high-quality software** and **customer satisfaction.**
- Encourages **proactive quality management** rather than reactive fixes.
- Fosters a **collaborative environment** where quality is a shared responsibility.

📉 **Consequences of Poor Quality Culture:**

- Increased **software defects, security risks, and customer dissatisfaction.**
- Higher **failure costs** due to post-release issues and rework.
- Lack of accountability leads to **inefficiency and inconsistent product quality.**

✅ **Solution:** Establish a strong **quality-first mindset** across the organization.

Building a quality culture in organizations

- **1. Leadership Commitment**
  - Leaders must **prioritize quality** and **set measurable goals**.
  - Provide resources and create an **accountability-driven environment**.
- **2. Shared Responsibility & Continuous Improvement**
  - Quality is **not just QA's job**—developers, testers, and stakeholders must contribute.
  - Regular **evaluations, retrospectives, and root cause analysis** ensure improvement.
- **3. Employee Empowerment & Communication**
  - Train and equip teams with **tools and autonomy** for better decision-making.
  - Foster **open communication channels** (e.g., Slack, MS Teams) for collaboration.
- **4. Customer-Centric Approach & Standardization**
  - Align quality efforts with **customer expectations and feedback**.
  - Follow **best practices and industry standards** (ISO, CMMI) to ensure consistency.

Case Studies of Successful SQA Implementations

Introduction

- SQA ensures software meets defined quality standards and user expectations.
- Tailored strategies improve efficiency, reduce risks, and address project-specific challenges.
- Explore six case studies demonstrating successful SQA implementations across various industries.

NASA – Rigorous SQA for Space Missions

Context: Precision required for space missions to avoid catastrophic outcomes.

SQA Strategies:

- Comprehensive Testing: From requirements to deployment. Automated
- Tools: Simulators to mimic space conditions . Peer Reviews: Regular code inspections.
- Continuous Monitoring: Post-deployment performance tracking.

Outcome: High-quality, reliable software for missions like Mars Rover; benchmark for high-stakes industries.

Google: Automated Testing for Scalable Systems

Context: Reliability and scalability for millions of global users.

SQA Strategies:

- Test-Driven Development (TDD): Tests written before coding.
- Continuous Testing: Automated unit, integration, and regression tests.
- Simulations: Stress tests for high user loads.
- Defect Tracking: Centralized tools for quick resolution.

Outcome: Faster delivery cycles and scalable, reliable applications like Search and YouTube.

Lessons Learned from Successful Software Quality Assurance (SQA) Projects

Implementing SQA effectively leads to improved software reliability, reduced risks, and higher customer satisfaction.

Here are key lessons derived from successful SQA projects across industries:

Emerging Technologies and Their Impact on SQA

An Overview of Transformations in Software Quality Assurance

Introduction

Emerging technologies have significantly transformed Software Quality Assurance (SQA), introducing new methodologies, tools, and challenges.

Artificial Intelligence (AI) and Machine Learning (ML)

Impact:

- Test Automation: AI generates test cases, identifies defects, and automates repetitive tasks.
- Predictive Analytics: ML predicts defect-prone areas to prioritize testing.
- Dynamic Testing: AI adapts test cases in real-time.

Challenges:

- High initial setup costs and expertise requirements.
- Difficulty in testing AI systems due to non-deterministic behavior.

Cloud Computing

impact:

- Scalable Testing Environments: Reduces infrastructure costs.
- Cross-Platform Testing: Covers diverse platforms and configurations.
- Disaster Recovery: Cloud-based backups ensure robust data retention.

Challenges:

- Security and compliance concerns in cloud environments.

Internet of Things (IoT)

Impact:

- Complex Scenarios: QA validates diverse device interactions and real-time data.
- Performance Testing: Ensures reliability under high device loads.
- Security Testing: Identifies vulnerabilities in devices and data transmission.

Challenges:

- Managing vast combinations of devices, protocols, and platforms.

Blockchain Technology

Impact:

- Smart Contract Testing: Ensures correctness and security of blockchain systems.
- Decentralized Testing: Requires unique test strategies.

Challenges:

- Debugging is complex due to immutability.
- Limited expertise in blockchain testing.

Big Data and Analytics

Impact:

- Data Validation: Ensures accuracy and integrity of large datasets.
- Performance Testing: Evaluates handling of high-volume, high-velocity data.
- Testing Insights: Analytics identify patterns and root causes of defects.

Challenges:

- Managing and processing large-scale data for testing.

Automation and Robotics

Impact:

- Robotic Process Automation (RPA): Automates repetitive tasks.
- Hardware-in-the-Loop Testing: Integrates hardware and software testing.

Challenges:

- Testing robotic systems demands specialized environments and expertise.

Conclusion

- Emerging technologies have revolutionized SQA by enabling automation and addressing challenges.
- Adoption requires overcoming skill gaps, high investments, and adapting to rapid advancements.
- SQA professionals must stay updated to ensure software quality in an evolving landscape.

innovate achieve lead

BITS Pilani
Pilani Campus

Artificial Intelligence and Machine Learning in Quality Assurance

Introduction

- AI and ML are revolutionizing Quality Assurance (QA).
- Enhances testing efficiency, early defect detection, and predictive analytics.
- Shifts QA from manual testing to intelligent, automated approaches.

Key Concepts

Artificial Intelligence (AI)

- Simulates human intelligence to make decisions and solve problems.
- Used in QA for test automation, defect prediction, and anomaly detection.

Machine Learning (ML)

- Subset of AI where algorithms learn from data to make predictions or decisions.
- Helps in identifying patterns, predicting failures, and improving test coverag

Automated Testing

How It Works:

- AI automates repetitive tasks like test case generation, execution, and reporting.
- ML optimizes test suites by identifying redundant tests.

Benefits:

- Faster test execution.
- Improved accuracy and reduced human errors.

Tools: Selenium, Testim, Applitools.

Defect Prediction and Prevention

How It Works:

- ML analyzes historical data to predict defect-prone areas.
- AI identifies potential failure points before they occur.

Benefits:

- Reduces debugging time.
- Enhances software reliability.

Anomaly Detection

How It Works:

- AI analyzes logs, system behavior, and performance metrics.
- ML flags unusual patterns indicating defects.

Benefits:

- Early detection of performance bottlenecks.
- Ensures system stability.

Test Case Prioritization

How It Works:

- ML ranks test cases based on defect probability, impact, or execution history.
- Focuses on critical areas for testing.

Benefits:

- Saves resources by testing high-risk areas first.
- Improves test efficiency.

Predictive Analytics

How It Works:

- ML analyzes past trends to predict defect patterns and testing timelines.

Benefits:

- Helps in resource allocation.
- Reduces risks in project planning.

Visual Testing

How It Works:

- AI compares screenshots of applications to detect UI inconsistencies.

Benefits:

- Ensures visual consistency across platforms.
- Speeds up regression testing

Natural Language Processing (NLP)

How It Works:

- NLP analyzes requirements and generates test cases automatically.
- Chatbots assist in real-time test result analysis.

Benefits:

- Simplifies test case creation and management.
- Enhances team communication

Self-Healing Test Scripts

Title: Self-Healing Test Scripts

How It Works:

- AI detects changes in application code or UI and updates test scripts automatically.

Benefits:

- Reduces test maintenance efforts.
- Ensures continuous testing in dynamic environments

Challenges in AI/ML Adoption for QA

- **Data Dependency:** Requires large datasets for training ML models.
- **Complexity:** AI/ML implementation can be technically challenging.
- **Cost:** High initial investment in tools and infrastructure.
- **Lack of Expertise:** Requires specialized skills and training.

Best Practices for Using AI/ML in QA

- **Start Small**: Implement AI/ML in specific areas before scaling.
- **Leverage Open-Source Tools**: Use TensorFlow, PyTorch, Selenium AI plugins.
- **Collaborate Across Teams**: Encourage communication between QA, development, and data science teams.
- **Monitor and Improve Models**: Regularly update ML models with new data.
- **Focus on High-Risk Areas**: Automate repetitive and critical testing tasks.

Benefits of AI/ML in QA

- **Efficiency**: Faster testing cycles with higher accuracy.
- **Scalability**: Can handle complex, large-scale projects.
- **Cost Reduction**: Minimizes manual effort and resource usage.
- **Improved Quality**: Early defect detection improves reliability.
- **Continuous Learning**: ML models improve over time.

Conclusion

- AI and ML are transforming QA with automation, efficiency, and predictive insights.
- While challenges exist, strategic adoption improves software quality and delivery timelines.
- Organizations leveraging AI/ML effectively will stay competitive in modern software development.