CASE STUDY

Exploratory Testing in Software Development

Rakibul 12 Mar 2025 0 317 0 Share

Exploratory Testing is a software testing approach where testers actively explore an application without predefined test cases. Unlike traditional scripted testing, this method relies on the tester's experience, intuition, and creativity to identify defects. It is particularly effective in detecting unexpected bugs, usability issues, and edge cases that structured testing might miss.

Since Exploratory Testing is unscripted, testers can go beyond the predefined scenarios of automated or manual scripted tests, uncovering defects that might otherwise be missed.

🔹 **Example**:
Imagine testing an e-commerce checkout page. While automated tests verify standard inputs (valid credit card, valid address), an exploratory tester might try:

- Entering a long string as a name (e.g., 500 characters).
- Using special characters in the address field.
- Switching between payment methods quickly.
- These actions could reveal crashes, UI glitches, or security vulnerabilities that weren't covered in scripted test cases.

2. Enhances Test Creativity

Exploratory Testing allows testers to think outside the box and use their domain expertise to simulate real-world use cases.

🔹 **Example**:
In a healthcare application where doctors upload patient reports, a tester may:

- Upload an extremely large medical report file.
- Attempt to upload multiple reports at the same time.
- Check how the system reacts if the internet disconnects mid-upload.
- Such testing can help uncover issues like performance bottlenecks, UI freezes, or incorrect error messages.

3. Saves Time

Because there's no need to write and maintain extensive test cases, exploratory testing can be conducted quickly, making it ideal for tight deadlines and Agile environments.

🔹 **Example**:
A startup developing a mobile app for food delivery has a two-day deadline to test a new feature. Instead of writing detailed test scripts, testers explore:

- How the app responds when a user places multiple orders at once.
- If the app functions well on different screen sizes.
- If discount coupons apply correctly across different restaurants.
- This approach allows for rapid feedback and issue identification before release.

4. Complements Automated Testing

While automated testing is great for regression testing and verifying expected behaviors, it often struggles with edge cases, usability issues, and new feature validation. Exploratory Testing fills these gaps.

🔹 **Example**:

- Explore how the app behaves if the session expires in the middle of a transaction.
- Check whether the app correctly handles currency conversion for international transfers.
- These tests help identify issues that automated scripts may not consider.

5. Mimics Real-User Behavior

Users do not interact with applications in a perfectly structured manner; they explore, make mistakes, and try different workflows. Exploratory Testing replicates this behavior, ensuring a more user-friendly experience.

🔹 **Example**:
For a travel booking app, a tester could:

- Attempt to book a flight while switching between Wi-Fi and mobile data.
- Enter incorrect details (e.g., an invalid passport number) and check the error messages.
- Navigate back and forth rapidly between pages to see if data persists correctly.
- Such scenarios reflect real-world conditions, leading to a more resilient and user-friendly application.

When to Use Exploratory Testing?

Exploratory Testing is a flexible approach suitable for various situations. Here's when it is most useful:

1. During Early-Stage Software Development for Quick Feedback

At the beginning of development, the application may not be stable enough for automated tests, but exploratory testing helps in identifying major issues early.

🔹 **Example:**
A team is developing a new **health-tracking mobile app**. In the first few iterations, basic features like step counting and heart rate tracking are added. Instead of waiting for full test cases, testers explore:

- Whether the step counter updates correctly.
- If the heart rate monitor responds to different levels of activity.
- How the app behaves when switching between different screens.
- This provides immediate feedback, ensuring core functionalities work before more structured testing begins.

2. When Performing Ad-Hoc Testing to Validate Functionality

Ad-hoc testing is unstructured testing done without detailed planning. Exploratory Testing is ideal for quickly checking if a feature works as expected.

🔹 **Example:**

- Try canceling scheduled rides at various points to see if the refund process is handled correctly.
- Change the pickup location last minute to see if the UI updates properly.
- This quick validation helps identify immediate issues before formal testing.

3. For Applications with Frequent Changes, Such as Agile and DevOps Environments

In fast-paced Agile and DevOps settings, new updates are released frequently, and maintaining detailed test scripts for every change can be challenging.

🔹 **Example:**
A **SaaS-based project management tool** frequently updates its task management dashboard. Testers use exploratory testing to:

- Verify if the drag-and-drop functionality still works after an update.
- Check if task filters return correct results.
- Explore different user roles (Admin, Member, and Guest) to ensure permissions are properly enforced.
- Since exploratory testing is flexible, it helps in quickly assessing new changes.

4. When There Are No Clear Test Requirements or Specifications

Some projects may lack detailed documentation, making scripted testing difficult. Exploratory Testing helps testers adapt to uncertain or changing requirements.

🔹 **Example:**
A startup is developing an **AI-powered chatbot** for customer support, but the specifications keep evolving. Testers explore:

- If the bot correctly understands different question phrasings.
- How the bot responds to unexpected inputs like emojis or gibberish.
- If the chatbot's responses change when users modify previous queries.
- By exploring the system, testers uncover usability issues and gaps in functionality.

5. To Validate Critical Areas Before Production Releases

Before launching a product or update, exploratory testing helps uncover last-minute issues in critical functionalities.

🔹 **Example:**
A **financial trading platform** is about to go live with a new real-time stock trading feature. Testers focus on:

- Ensuring transactions are processed correctly under heavy load.
- Checking if order execution timing is accurate.
- Testing scenarios like sudden market crashes or multiple rapid trades.
- This ensures that no major issues exist before deployment.

Testers define a specific objective, explore the application for a fixed time, document observations, and analyze findings.

🔹 **Example:**
A tester sets a goal to evaluate the **checkout process** in an **e-commerce app**. They explore:

- Entering different invalid payment details (expired card, incorrect CVV, insufficient balance).
- Applying and removing discount coupons to check if the price updates correctly.
- Simulating a network disconnection while making a payment.
- After the session, findings are discussed to improve future tests.

2. Freestyle Testing

An entirely unscripted approach where testers freely navigate the application without predefined goals.

🔹 **Example:**
A tester explores a **mobile banking app**, randomly attempting actions such as:

- Opening and closing different sections rapidly to check for crashes.
- Transferring funds while switching between dark and light mode.
- Attempting security settings changes to find inconsistencies.
- Freestyle testing can uncover unexpected usability issues.

3. Error Guessing

Testers use their intuition and experience to predict problem areas in an application.

🔹 **Example:**
In a **social media platform**, the login feature had issues in previous releases. The tester focuses on:

- Entering incorrect credentials multiple times to check account lock behavior.
- Log in from two devices simultaneously to detect session handling issues.
- Testing forgotten passwords and two-factor authentication flows.
- By focusing on historically weak areas, testers improve defect detection.

4. Boundary Value Testing
Tests input limits and edge cases to check if the system behaves correctly.

🔹 **Example:**
A **user registration form** allows usernames between 1 and 50 characters. The tester tries:

- **0 characters** (empty input) to check error handling.
- **1 character** to verify the lower limit.
- **50 characters** to verify the upper limit.
- **51 characters** to check if it gets rejected properly.
- This approach helps identify input validation issues.

🔹 **Example:**
A UI tester and a backend tester explore a **hotel booking app** together. They:
- Check if the UI booking details match the backend database.
- Analyze discrepancies in price calculations.
- Validate that room availability updates correctly across different devices.
- Pair testing combines different perspectives, improving test coverage.

Steps to Conduct Exploratory Testing

1. Define Scope

Before starting, identify which part of the application you want to test. This helps keep the session focused and ensures that relevant areas are covered.

🔹 **Example:**
A tester is exploring a **food delivery app**. The scope might include:
- The restaurant search functionality.
- The checkout and payment process.
- Delivery tracking updates.

By defining the scope, the tester ensures they don't get sidetracked and focus on the most critical areas.

2. Set Goals

Establish clear objectives to guide the testing session. Goals should align with user needs and business requirements.

🔹 **Example:**
For an **online banking app**, the goals could be:
- Verify if users can transfer money between accounts.
- Check if login via biometric authentication (fingerprint/face ID) works correctly.
- Ensure that users can download their transaction statements without errors.

Setting goals helps testers measure success and focus their testing efforts effectively.

3. Explore the Application

Interact with the application freely, following real-world scenarios and user behaviors. During this phase, testers:
- Navigate different sections.
- Perform various user actions (e.g., sign-ups, transactions, searches).
- Check if everything works as expected.

🔹 **Example:**
In a **ride-sharing app**, a tester might:
- Try booking a ride from an airport to a nearby city.

This hands-on approach helps uncover usability issues and unexpected bugs.

4. Note Observations

While exploring, document unusual behaviors, UI glitches, or performance issues. This information helps in debugging later.

🔹 **Example:**
A tester is checking the **cart functionality** in an e-commerce app and notices:

- Items disappear from the cart after refreshing the page.
- Discount codes don't apply correctly when multiple coupons are used.
- Product images sometimes fail to load on slower internet connections.

These observations help developers understand problem areas and fix defects quickly.

5. Analyze and Report Bugs

Once issues are identified, testers should:

- Clearly describe the bug.
- Provide steps to reproduce it.
- Attach screenshots or videos to support the report.

🔹 **Example:**
Bug report for a **finance tracking app**:

- **Issue:** Transactions don't update in real time.
- **Steps to Reproduce:**
    a. Open the app and check the latest transactions.
    b. Make a new transaction using the linked bank account.
    c. Refresh the transaction history page.
- **Expected Result:** The new transaction should appear immediately.
- **Actual Result:** The transaction doesn't appear until the app is restarted.
- **Attachments:** Screenshot showing the missing transaction.

Clear bug reports improve communication between testers and developers.

6. Review Findings

Discuss insights and findings with the development team to improve overall software quality. This is also a chance to prioritize which issues need immediate attention.

🔹 **Example:**
During a **review meeting for a travel booking website**, testers and developers:

- Discuss why users face payment failures in certain countries.
- Prioritize fixing a security loophole found in the login process.
- Identify UI issues that need refinement before the next release.

Regular reviews ensure that testing efforts lead to meaningful improvements.

Although exploratory testing is mostly **manual**, certain tools can improve efficiency and documentation.

1. Session Recorders

These tools track user actions during a testing session, making it easier to analyze findings later.

- **TestRail** – Helps manage test sessions and record exploratory testing notes.
- **qTest** – Allows tracking test execution history.
- **Testpad** – Supports exploratory and checklist-based testing.

🔹 **Example:**
A tester using **TestRail** records a session while testing a **new search feature in an online marketplace**. Later, they review session logs to recall all steps taken when they encountered a bug.

2. Bug Tracking Tools

Used for logging, tracking, and managing defects.

- **Jira** – Widely used for bug tracking in Agile teams.
- **Bugzilla** – An open-source tool for managing software defects.
- **Trello** – This can be used for lightweight bug tracking with visual task boards.

🔹 **Example:**
A QA team logs a critical issue in **Jira** when a **banking app crashes during fund transfers**. Developers use the bug-tracking tool to assign and track the issue until it is resolved.

3. Screen Capture & Logging Tools

Used for capturing screenshots, recording videos, and monitoring network logs.

- **Snagit** – A Simple tool for capturing screenshots and annotating them.
- **Loom** – Records video walkthroughs of issues.
- **Charles Proxy** – Captures and analyzes network traffic for debugging API-related issues.

🔹 **Example:**
A tester finds a **checkout page error in an e-commerce app**. They:

- Take a **screenshot using Snagit** to highlight the issue.
- Record a **video using Loom** to show how the issue occurs.
- Use **Charles Proxy** to check if there's a failed API request causing the issue.

By using these tools, testers can provide **detailed** bug reports, making it easier for developers to fix issues.

Challenges of Exploratory Testing & How to Overcome Them

1. Lack of Documentation

Since exploratory testing is unscripted, it can be difficult to track what has been tested and what issues were found.

- **Record sessions**: Use tools like **Loom** or **TestRail** to capture video recordings.
- **Use exploratory test charters**: Define what areas need testing to maintain focus.

🔹 Example:
A tester explores an **online banking app** and finds a bug where **fund transfers fail intermittently**. Without documentation, they might struggle to reproduce the issue later. By using **Loom** to record their session, they can replay their steps and provide developers with detailed insights.

2. Tester Dependency

Exploratory testing relies heavily on the tester's experience, intuition, and knowledge, making results inconsistent across different testers.

🔹 How to Overcome:
- **Rotate testers**: Different testers will bring unique perspectives, improving test coverage.
- **Pair testing**: Involve two testers working together to share knowledge.
- **Use domain experts**: Have business analysts or product managers participate in testing.

🔹 Example:
For a **healthcare application**, a QA engineer might focus on **functional bugs**, while a **medical domain expert** might identify issues related to **data accuracy in patient records**. By involving multiple testers, the overall quality of testing improves.

3. Difficult-to-Measure Coverage

Since exploratory testing doesn't follow predefined test cases, it can be hard to determine if all critical areas have been tested.

🔹 How to Overcome:
- **Use test charters**: Define objectives for each testing session.
- **Leverage test tours**: Break down testing into categories like UI testing, security testing, or performance testing.
- **Use mind maps**: Visualize test coverage by mapping different test scenarios.

🔹 Example:
A tester working on a **travel booking app** might use a **Test Tour** approach, where they:
- Focus on **payment gateway functionality** in one session.
- Explore **booking modifications and cancellations** in another session.
- Test **multi-language support** separately.
- This structured approach ensures that key areas are tested without missing anything.

Best Practices for Exploratory Testing

1. Stay Curious
Adopt a user mindset and question how the application behaves in unexpected situations.

- Try to **schedule a delivery for a past date.**
- Explore what happens if a user **removes all saved payment methods** before checking out.
- This mindset helps uncover real-world issues that users might encounter.

2. Use Mind Maps
Mind maps help visualize different test scenarios, ensuring broader test coverage.
🔹 Example:
For an **e-learning platform**, a tester creates a mind map covering:

- **Login methods** (email, Google, Facebook, mobile number).
- **Course interactions** (video playback, quizzes, assignments).
- **Payment options** (credit card, PayPal, coupons).
- This helps testers explore different areas systematically.

3. Combine with Automated Testing
Use exploratory testing to identify **new test scenarios**, then automate repetitive tasks for efficiency.
🔹 Example:
In an **e-commerce app**:

- Exploratory testing uncovers an issue where users can **apply multiple discount codes incorrectly**.
- The bug is fixed, and then an **automated test case** is written to ensure it doesn't reappear in future updates.

By combining exploratory and automated testing, teams improve efficiency while maintaining quality.

4. Regularly Share Findings
Discussing test findings helps refine testing strategies and catch overlooked defects.
🔹 Example:
During a **weekly QA meeting**, testers report that:

- A **flight booking app crashes when selecting specific airlines.**
- A **subscription-based video platform fails to renew memberships for some users.**
- By sharing these insights early, developers can prioritize fixes.

5. Keep Learning
Stay updated with new testing methodologies, tools, and industry trends.
🔹 Example:
A QA engineer following industry trends learns about **AI-powered testing tools** and decides to:

- Use **Testim.io** for smarter automated tests.
- Explore **chaos engineering** to improve resilience testing.
- Continuous learning enhances testing effectiveness.

Conclusion on Exploratory Testing

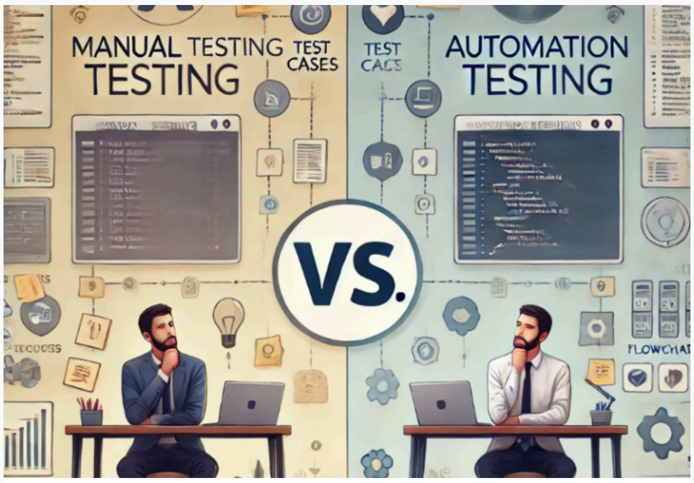
Exploratory Testing is a **dynamic, flexible, and efficient** approach to software testing that focuses on **real-world user experiences** rather than predefined scripts. It allows testers to **uncover hidden defects, usability issues, and unexpected edge cases** that traditional testing might miss. This method is particularly valuable in **Agile and DevOps environments**, where rapid changes and

challenging, these can be addressed by **maintaining session notes, leveraging testing tools, and collaborating across teams.** Additionally, combining **exploratory testing with automated testing** ensures both **quick defect detection and long-term stability.**
Ultimately, Exploratory Testing **mimics real-user behavior, promotes creativity, and enhances overall software reliability.** By **staying curious, continuously learning, and sharing findings,** testers can contribute significantly to **building high-quality applications that deliver exceptional user experiences.**

testautomation qatesting manualtesting exploratorytesting softwarequalityassurance agiletesting bugdetection softwaretestingtechniques defectdiscovery userexperiencetesting testingstrategies

Share your thoughts Or Start discussion

Related Blogs

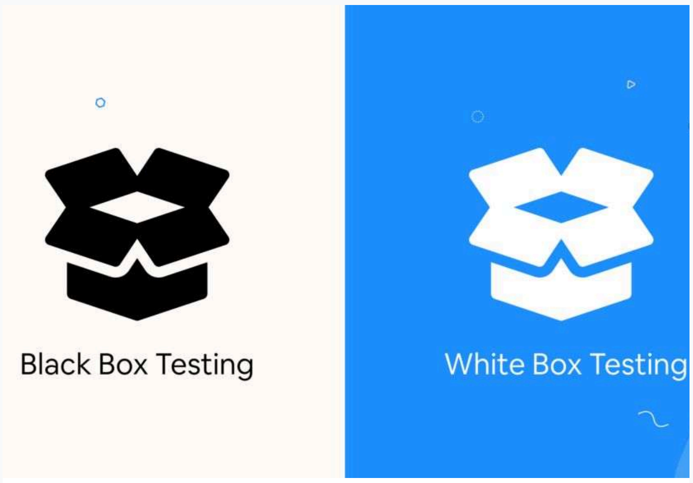


CASE STUDY 0 0 28

Can a QA build his career with only manual testing skills or ultimately does he need...

Can someone build a career in QA with only manual testing skills or automation i

Ali Hasan
24 Mar 2025



CASE STUDY 0 0 16

White Box vs Black Box Testing

🚀 White Box vs Black Box Testing: What Every QA Should Know!As a QA Engineer,

Anirudha
24 Mar 2025

sqa testing qa software testing qabrains testing tool

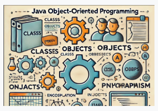
automationtesting softwaretesting mobiletesting selenium

View All

Popular Post

Can a Software Tester Become a Game Tester? Here’s What You Need t...

As the gaming industry continues to grow, fueled by innovations in virtual reali

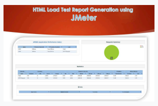


Understanding Java Object-Oriented Programming (OOP) Concepts

Java is a powerful and widely used programming language known for its versatilit

Essential Bugs to Check for in Game Testing: A Guide for Beginners

Game testing is crucial to ensure a smooth, engaging, and bug-free experience fo



JMeter: Short technique for Generating an HTML load test report using...

Pre-requisites:Install Java:Java Version: "1.8.0_291" or higher (minimum require

View All

Popular Discussion

01 Top Software Testing Interview Questions and Expert Tips from QA Leaders

02 AI tools for QA engineer

03 What is SQL?

04 Appium, WebDriver

05 What are the most effective strategies you've found for balancing speed and...

View All

QA Brains is the ultimate QA community to exchange knowledge, seek advice, and engage in discussions that enhance Quality Assurance testers' skills and expertise in software testing.

QA Topics

Web Testing

Interview Questions

Game Testing

See more ⟶

Quick Links

Discussion

About Us

Terms & Conditions

Privacy Policy

Follow Us

For Support

support@qabrains.com