A Deep Dive Into Exploratory Testing

By Anna Sharpe

asharpe213@gmail.com

Abstract

In software development, ensuring quality is crucial for delivering reliable, efficient, and user-friendly applications. There are many methods available to quality engineers today and exploratory testing can be a strong tool in an engineer's toolbox. Exploratory testing is an approach which differs from traditional scripted testing in its adaptability, creativity, and focus on learning while testing. This paper explores the concept of exploratory testing, its benefits, challenges, and best practices, as well as a case study in how to apply exploratory testing to your team.

Biography

I graduated with a BS in computer science from Whitworth University in 2014 and started my career as an intern test engineer in 2012. I am now a full time software test engineer with a part-time passion for writing websites for non-profits. This is my second paper for PNSQC and along with attending the conference six times over the years, I have also presented a poster for PNSQC. I am thoroughly invested in continuing to add new tools to my testing toolbox and sharing my experiences with my teammates. I am currently employed remotely out of Sacramento, Ca working with healthcare software.

1. What is exploratory testing?

Exploratory testing is a software testing technique where testers actively explore the application without predefined test scripts. It is often referred to as "testing as learning" because testers investigate the software in real time using their intuition, experience, and insights to identify defects. Unlike scripted testing, where each step is predetermined and executed systematically, exploratory testing allows testers to simultaneously design and execute tests, fostering a deeper understanding of the system and its functionality as a whole.

The tester's primary goal during exploratory testing is not only to identify defects but also to uncover edge cases which might not be obvious through standard test cases. Testers leverage their domain knowledge, creativity, and critical thinking skills to interact with the software in unpredictable ways, enabling the discovery of bugs which otherwise would have been missed.

2. Why do we care about exploratory testing?

An expert quality assurance engineer needs a toolbox full of tools. While testing software is inherently tedious and redundant, there needs to be a tool to test outside the box. Exploratory testing can be that tool when applied with a clear motive and knowledge of the software under test. Additionally, running the same static tests over and over again can only find so many bugs. Whereas exploratory testing allows the engineer to flex their creative muscles and find the bugs no one else can.

3. Characteristics of exploratory testing

Several key characteristics define exploratory testing:

1. **Creativity and flexibility**: Testers apply creativity to explore unexpected paths or combinations of inputs which may lead to potential failures.

2. **Focus on learning**: Exploratory testing aims to learn about the software's functionality, which helps give insight into where to focus a tester’s energy.

3. **Bug bashes**: Exploratory testing is also useful in bug bash sessions where there might be guidelines about a new feature, but no exact test scripts. These sessions can help uncover questions not asked during development and bugs around the new feature.

4. **Time-bound sessions**: Exploratory testing typically involves time-boxed sessions, where testers work within a fixed duration (e.g., 1 hour) to explore the system and document findings.

4. Benefits of exploratory testing

Exploratory testing offers several benefits which make it a powerful addition to a software testing toolbox. By testing in an unscripted and unpredictable manner, exploratory testing is more likely to uncover edge cases, usability issues, and critical bugs traditional tests may miss. Exploratory testing can adapt to changes in the software, including new features or unexpected behavior. Since

there are no predefined scripts, exploratory testing eliminates the need for test case creation, making it ideal for projects with tight deadlines or rapidly changing requirements. Exploratory testing also encourages testers to actively engage with the software, utilizing their domain knowledge and creativity, which can lead to higher motivation and improving critical thinking with each session. Having tests which do not set a limit on what needs to be tested forces the tester to interact with the software in an unstructured manner, which can lead to previously untested scenarios in the software. Thus the tester is exploring the software with different paths in each test session with the goal of finding new bugs.

5. Challenges of exploratory testing

While exploratory testing has numerous benefits, it also presents certain challenges:

1. **Lack of documentation**: Since there are no predefined test scripts, exploratory testing can suffer when there is insufficient documentation. Insufficient documentation causes the tester to have unanswered questions during testing which would lead to a bug being ignored because the observed behavior was assumed to be correct due to a lack of documentation.

2. **Inconsistent results**: Because the tests are not standardized, exploratory testing may not always result in bugs found nor confidence in the product's quality.

3. **Requires experienced testers**: Exploratory testing demands a high level of experience, intuition, and domain knowledge from testers. Inexperienced testers may struggle to identify critical defects or focus their exploration effectively.

4. **Difficult to measure coverage**: Unlike scripted testing, where it's easier to measure how much of the application has been tested, exploratory testing does not lend itself easily to measuring test coverage.

5. **Time constraints**: Although exploratory testing can offer quick feedback, it can also be time-consuming, particularly when testers are not familiar with the application, leading to longer test sessions and potentially missing key defects.

6. Best practices for exploratory testing

To overcome the challenges associated with exploratory testing and maximize its effectiveness, testers should have a guideline for best practices. Time-boxing allows testers to focus their efforts and make the most of their exploration within a fixed period. For example having a one hour session after the completion of an epic with other testers is a great way to utilize the collective team's testing power. Testers should create a testing goal for each session. The goal guides the tester's exploration and ensures the session is focused on specific areas of the application. It is also very helpful to have a leader for the meeting who can give a demo of the work to be tested to define the testing goal for the group. Exploratory testing may uncover unanswered questions or holes in existing documentation. It is always helpful to document findings or update existing documentation to avoid confusion in the future. Having a collective document the group can use to write down questions and bugs is a great way of keeping everything in one place. Bug bashes can

be a more structured approach to exploratory testing where a feature is demoed and then testers are asked to find bugs. This can be extremely helpful because fresh eyes are going to see the feature in a new way. Also, the additional testers are not going to be as familiar with the feature, thus are going to ask difficult questions which may uncover new bugs. Collaboration is key to understanding the software's functionality and identifying areas which require deeper exploration. Testers should work closely with developers, product owners, and other stakeholders to ensure they are targeting the most critical aspects of the application. As a group, discuss new bugs, how they were found and the priority of the bug.

7. How to educate new testers

New testers can benefit greatly from exploratory testing if they are given the right guidance. Sharing results of previously found bugs for an epic they are involved in can give them some ideas of where potential issues might be. Every tester has critical thinking skills, however, it is not always obvious to new testers how to use those skills. Showing new testers how to recognize patterns can help them come to their own conclusions later on. Also, newer testers might not be confident enough to question the status quo, so it is important to communicate to them questions are good. Making assumptions is not a good practice and new testers should be informed as such. Involve the new testers in bug triage meetings to show them where customers are having issues so they know the important scenarios in the application.

8. Exploratory testing in Agile and DevOps

Exploratory testing is particularly well-suited for Agile and DevOps environments, where fast-paced iterations and continuous delivery are essential. In these environments, software requirements may evolve rapidly, making it difficult to maintain extensive test scripts. Exploratory testing offers a flexible approach to ensure software quality remains high, even as the software evolves.

In Agile teams, exploratory testing fits well within the iterative development process. Testers can explore new features during each sprint, providing valuable feedback to the development team. Furthermore, the collaborative nature of exploratory testing aligns with Agile principles, allowing testers to communicate with developers and product owners to address issues promptly.

In DevOps environments, where continuous integration and continuous delivery (CI/CD) pipelines require fast-paced testing, exploratory testing can complement automated tests. Testers can focus on areas which are difficult to automate, such as user experience, usability, and edge cases, while relying on automated tests for repetitive, predictable tasks.

9. How to integrate exploratory testing into the SDLC

Developing software always has deadlines which makes it difficult to budget time for exploratory testing. It is the responsibility of the lead tester for the project to schedule time for testing scenarios outside the direct scope of the work done by the team. It can be an easy trap to fall into where testers are only testing the acceptance criteria outlined in the project and not testing adjacent domains which might also be unknowingly impacted. Time can be set aside each week of the project to ensure testing is happening throughout the development process and avoid a mad

scramble of testing at the end. The team can even set up a reoccurring meeting each week, every Friday at 10am for an hour for example, to ensure exploratory testing becomes routine. These hour sessions could be done solo or additional people could be brought in to bounce ideas off of.

10. Case study of exploratory testing

Description of work completed

This project's goal was to add a login functionality to the existing checkout process to help customers expedite checkout of future purchases. Additional work done was to collect the user's data for future analysis.

If you find a bug:

Open a bug for team Cobalt

Add a label Wave_1_Bug_Bash

Scenarios to test:

Test steps	Expected	Screenshot	Tester
Customer logs in with valid credentials	Customer should see their previous purchases		Billy
Customer is denied login with invalid credentials	Customer should see a link to reset their password		Anna
Customer has zero previous purchases	Customer's previous purchases should not give an error, just be blank		Emily
Customer has a large amount of purchase history	The page should not fail and all purchases should show correctly		David
View data on the back end	Customer's data is accurate		Frank

Accounts to login with:

Username	Password	Tester
anna.anna@gmail.com	AnnasPassword!	
sharpe.sharpe@gmail.com	SharpesPassword!	

Username	Password	Tester
bob.bob@hotmail.com	BobsPassword!	
andy.andy@icloud.com	AndysPassword!	
julia.julia@aol.com	JuliasPassword!	

Known issues:

- *Customers with purchases older than 2020 are not appearing correctly*
- *Customers with exactly five purchases have a distorted view of the list*

The above is an example of documentation to have ready for running a successful bug bash. The tester leading the bug bash should outline the work completed with as much detail as possible to give the testers a general idea of the work completed. They also should give a demo of this work to ensure everyone is clear about what is being tested. There should be steps on what to do if a tester finds a bug. Your company might have standards on how to title the bugs to indicate it was found as part of the bug bash or maybe adding a label indicating as such. Also, it is important to list any known issues to avoid duplicate bugs being opened.

The coworkers joining the bug bash, bashers, should be given some scenarios to follow to get the session started. The scenarios should follow a similar path to what was demoed at the start of the meeting and the bashers can spring board from there. It can be helpful to have the scenarios assigned to bashers to ensure everyone isn't picking the first scenario and there is some accountability for who tested what. Additionally, where necessary there should be logins created prior to the bug bash for use and again the accounts should be assigned.

Inviting a variety of coworkers to these bug bashes is the ideal scenario. If possible, invite the stakeholders of the feature so they can have a say in the quality of the work done. Most people want to see the company they work for succeed and having a bug bash with people from other departments and teams is a great way for everyone to be involved in the quality of the product. If your company has never had a bug bash, it might seem silly to invite 50 people to an hour long bug bash, but the meeting can be optional. Over time, the test team can show the value of these bug bashes to get more involvement.

Upon completion of the bug bash, the tester leading the meeting needs to meet with the team who originally completed the work to triage the bugs found. These bugs might spawn further questions or uncover missed requirements. It is important to get these bugs addressed as soon as possible since there might be an upcoming deadline for the project. Having stakeholders or product managers in these triage meetings can help speed up the process and keep them informed on the status of the project.

These bug bash meetings almost always lead to bugs being found and thus fixed before the customers can complain about them. While the meeting does have an up front cost, potentially 10 engineers for two hours in the bug bash, the reward can be invaluable. Let's imagine the bug bash above yields one high severity bug and three medium severity bugs, the bugs are then triaged out and fixed before the code goes to customers. Depending on the number of customers a company has, this one bug bash eliminated potentially hundreds of complaints from customers. It also eliminated the need for a patch and fewer patches means less stress on the engineering team. These statistics on value provided are easy to see, no one wants to patch software after customers complain and a planned out bug bash can eliminate the likelihood of high severity bugs being shipped. If your company has never organized a bug bash, it might be necessary to review past high severity bugs found by customers and discuss with stakeholders if a bug bash is a process improvement your team can make.

11. Conclusion

Exploratory testing is a vital software testing technique which promotes creativity, adaptability, and learning during the testing process. Its flexibility and ability to uncover hidden defects make it an essential tool for software quality assurance. While it presents challenges, such as documentation and coverage measurement, these can be mitigated through best practices like time-boxing, session-based test management, and collaboration with stakeholders. In Agile and DevOps environments, exploratory testing is particularly valuable, allowing teams to rapidly test and ensure the quality of software in dynamic, fast-paced development cycles. By combining exploratory testing with other testing strategies, organizations can build more robust, reliable, and user-friendly software.

12. References

Book:

James A. Whittaker. 2005. *Exploratory Software Testing: Tips, Tricks, Tours, and Technique to Guide Test Design.* Boston: Addison-Wesley Professional

Cem Kaner, and Rebecca L Fiedler. 2013. *Foundations of Software Testing.* Florida: Context-Driven Press

Mauricio Aniche. 2022. *Effective Software Testing: A Developer's Guide.* New York: Manning

Atul Gawande. 2011. *The Checklist Manifesto: How to Get Things Right.* New York: Metropolitan Books

Gene Kim, Kevin Behr, and George Spaffron. 2013. *The Phoenix Project: A Novel About IT, DevOps and Helping Your Business Win.* Oregon: IT Revolution Press