EXPLORATORY

TESTING

eGuide

TECHWELL™

TECHWELL™

As attention, effort, and spending shift from manual to automated software testing, many in the software industry are questioning the role of the human software tester in this new landscape. While we often see increases in speed, efficiency, and cost savings from automated testing, there will always be testing functions that can only be done effectively by a skilled exploratory tester. This eGuide explains why exploratory testing still plays a critical role in the development process and how it fits in today's agile and DevOps-focused world.

In This Exploratory Testing eGuide

6 Skills Needed for Exceptional Exploratory Testing

While anyone can claim to be an exploratory tester, only those with a set of honed skills will discover hard-to-find bugs that could impact your mobile app or website. Exploratory testers must possess these six skills if they are to find the edge cases that could derail a successful software release.

Finding a Middle Ground between Exploratory Testing and Total Automation

The automator wants to get rid of human exploration—they want a robot to cut down a forest and stack the wood. The explorer, on the other hand, sees tools more like a chainsaw—they allow humans to go ten times faster, but a human is still driving the process. Finding a middle ground is the best test strategy.

Continuous Exploratory Testing: Expanding Critical Testing across the Delivery Cycle

Continuous testing entails executing automated tests to obtain rapid feedback on business risks. Where does that leave exploratory testing? Obviously, it doesn't make sense to repeat the same exploratory tests across and beyond a sprint, but exploratory testing can be a continuous part of each software delivery cycle.

Integrating Exploratory Testing into Product Design

Exploratory testing, or ET, is a good fit for agile processes, can be done by any member of the dev/test team, and helps develop applications that map to customers' needs. Kevin Dunne writes how with increased use of ET, testing becomes an intellectual pursuit driving product quality and agility.

Using Tours to Structure Your Exploratory Testing

In testing, a tour is an exploration of a product that is organized around a theme. Tours bring structure and direction to exploration sessions, so they can be used as a fundamental tool for exploratory testing. They're excellent for surfacing a collection of ideas that you can then further explore in depth one at a time, and they help you become more familiar with a product—leading to better testing.

Insight from around the Industry

Find out what experienced agile practitioners have to say about implementing agile.

Additional Exploratory Testing Resources

6 Skills Needed for Exceptional Exploratory Testing

By Nicholas Roberts

While anyone can claim to be an exploratory tester, only those with a set of honed skills will discover hard-to-find bugs that could impact your mobile app or website. It requires skills that go above and beyond.

Here are six skills needed to be an exceptional exploratory tester.

1. Lateral thinking

Lateral thinking is when you solve a problem by an indirect approach, and it usually involves seeing the issue in a new way that no one else has previously. Essentially, it's examining a problem with a creative mindset. After all, it's hard to find bugs without thinking outside the box.

With lateral thinking skills, you see the various ways users will interact with your app or website, and you can find bugs that would have otherwise been undiscovered.

2. Critical thinking

Critical thinking is the ability to use reasoning in a rational manner. This lets you discover hidden relationships between variables, which increases the possibility of finding high-risk bugs that conventional thinking would not reveal. Critical thinking gets rid of the biases associated with personal beliefs, leading you to see valid reasons with an objective perspective.

Those who possess critical thinking skills make actionable suggestions because of their ability to weigh the consequences and risks associated with the search for bugs.

3. Investigation skills

Investigation skills provide a methodology that exploratory testers can use to discover and reproduce bugs. More importantly, being comfortable with using these investigative capabilities allows you to stray from the set methodology when necessary.

With investigation skills, you can set up a plan of attack when searching for bugs and will be familiar with the signs that your chosen method needs adjustments.

4. Storytelling skills

Exploratory testing requires the ability to tell a story—in this case, the story of the app or website you test. As you examine it, you will create accounts of various uses for the software and how people will utilize it. When you find a bug, this is the climax of the story, leading to the resolution, or the developers making adjustments to eliminate the problem.

5. Communication skills

The ability to communicate with other members of the team is essential for any exploratory tester. This allows you to coordinate with additional inspectors, determining who will work on a particular aspect of the testing, so you do not overlook any part.

Once you find a bug, communication skills are once again essential in order to succinctly and accurately describe the issue as well as the steps leading up to it. Developers will need to ask you questions and receive clear explanations in response.

6. Technical skills

Some types of testing do not require vast technical skills, only a familiarity with the basics. But exploratory testing may look at the full stack, from the user interface to the layers of software underneath, so you must have the skills to understand the programs and general coding.

In order to deliver high-quality products, you need to ensure that your testing finds the bugs that impact their usage. Exploratory testers must possess these six skills if they are to find the edge cases that could derail a successful software release.

Finding a Middle Ground between Exploratory Testing and Total Automation

By Matthew Heusser

Testers seem to be having the same argument over and over again.

The automator wants to get rid of human exploration—that is, they want to press a button, get a green bar, and ship to production. In some cases, they might want to commit to version control, have something else automatically press the button, and automatically ship to production. This is akin to having robots cut down a forest and stack the wood: no humans involved.

The explorer, on the other hand, wants a human intervention step. They see tools more like a chainsaw. The chainsaw allows the human to go ten times as fast, but a human is still in charge, driving the process. The explorer doesn't want robots to do everything automatically; they want to be a cyborg, a six-million-dollar man, to balance the human and the machine.

When the explorer says of course tools are important, the automator gets angry because that is not what the automator means. The result could be a no-hire, a lost chance to collaborate, or even the end of a friendship.

I believe the two have something to learn from each other.

The implicit assumption of the automator is that the automated tests are all there is. Of course, that is not the case; most web applications automated with Selenium still need to test printing, tab order, font size, and plenty of other features that are hard (if not impossible) to automate.

The tools influence the thinking, tempting testers to ignore risks the tool does not support. The best testers still make a list of these other risks and invest some time into exploration.

Also, writing automation takes a long time. There are plenty of risks that are expensive to put into code and unlikely to break if they work now, so testers can get away with one-time tests. The automator's worldview ignores these problems.

Meanwhile, the explorer is dismissing the value of tools that run all the time, unattended. At best, the explorer might have Selenium running on Jenkins on every commit, or create virtual servers on demand—these are tools that significantly reduce risk and can be done for better or worse. Many explorers dismiss these tools because they don't understand them, because they think they're someone else's job, or because they're maintained by the programmers.

The explorer can benefit by expanding his idea of risk management, and the same goes for the automator.

I suggest we start the conversation from what we agree on, explain to the other person where we differ, then figure out if there is a way to blend the ideas, like peanut butter and chocolate. The ideal would be to have the code deployed to production on every commit while continuously exploring production and staging for emergent risks, using the logs, customer feedback, new features, version control, and developer interviews to help inform us of those risks.

It's a tall order, I know. Still, I think a collaboration is better than either of those ideas on their own.

Continuous Exploratory Testing: Expanding Critical Testing across the Delivery Cycle

By Ingo Philipp

Continuous testing is the process of executing automated tests to obtain rapid feedback on the business risks associated with a software release. Where does that leave exploratory testing? It's not automated, but it's certainly critical for determining whether a release candidate has an acceptable level of risk.

Test automation is perfect for repeatedly checking whether incremental application changes break your existing functionality. However, where test automation falls short is at helping you determine if new functionality truly meets expectations. Does it address the business needs behind the user story? Does it do so in a way that's easy to use, resource-efficient, reliable, and consistent with the rest of your application?

Exploratory testing promotes the creative, critical testing required to answer these questions. Obviously, it doesn't make sense to repeat the same exploratory tests continuously across and beyond a sprint, but exploratory testing can be a continuous part of each delivery cycle.

Here are a few ways teams embed exploratory testing throughout their process.

Perform Ad Hoc Exploratory Testing as Each User Story is Implemented

This is the exploratory testing equivalent of peer code review. When a developer completes a user story, they sit down with a tester. First, the tester starts testing while providing a running commentary on what they are doing and why. Next, the developer takes control, explaining how they would test the software given their knowledge of the implementation details and challenges. The developer gains a user- and business-focused perspective of the functionality, and the tester learns about the inherent technical risks.

Another tactic is to have the developer and a tester separately test the same feature simultaneously, then discuss their findings at the end of the session. Often, this turns testing into a competition, where each participant tries to uncover the most or "best" issues in the allotted time.

Align Exploratory Testing Sessions with Full Regression Testing

It's simply not possible to perform exploratory testing or full regression testing on every code commit. That's what smoke testing is for. Instead, many teams run full regression testing and session-based exploratory testing in parallel a few times per week, whenever they've implemented new functionality that an end-user could feasibly exercise.

For optimal results, these sessions should be lightly planned, tightly timeboxed, include diverse perspectives, and really take the six thinking hats theory seriously.

Host Blitz Exploratory Sessions for Critical Functionality

The best way to uncover user experience issues before end-users is to get a broad array of feedback prior to release. One way is to host "blitz" exploratory testing sessions. When you're wrapping up work on critical new functionality, invite people from a variety of backgrounds and teams to participate in a short timeboxed session. Incentives can help drive participation, maximize results, and make testing fun.

Using test automation to continuously check the integrity of existing functionality is certainly critical. However, if you're not also making exploratory testing a continuous part of your process, how will you know if the new functionality meets expectations?

The goal of continuous testing is to understand whether a release candidate has an acceptable level of risk. Exploratory testing is perfectly suited for helping you answer that critical question.

Integrating Exploratory Testing into Product Design

by Kevin Dunne

Exploratory testing, or ET, is becoming increasingly important on today's testing teams, especially for companies embracing agile. Simply put, ET is a testing method that relies on investigative approaches and is implemented through parallel learning, test design, and test execution.

By supplementing manual scripted and automated testing methodologies with ET, software teams can employ a free-form exploration of an application that should dramatically improve its overall quality, performance, security, and usability. ET is a valuable tool for finding critical defects in software that wouldn't have been discovered otherwise, uncovering potential improvements in code that are hidden in an untested area, and double-checking that a critical customer feature will work under all circumstances.

ET fits well with agile processes because it doesn't depend on the heavy documentation of scripted testing, so it speeds time to market. And any member of the dev/test team can do exploratory testing, regardless of their specific skill sets. This gives product teams needed flexibility to pull resources into testing when schedules are tight.

The goal, of course, is developing higher-quality applications that map more closely to customer needs. Below are a few ideas for integrating ET into your team and work processes.

Pair with developers: A highly collaborative way to get started (and that has been proven to improve software quality) is through paired testing and development. It's the fastest way to validate new functionality, as there's a much smaller lapse between code release and testing to write an automated test or manual script.

Team-based ET: Some software teams will engage a group of developers and testers into a phase of exploratory testing in between sprints or at other ad hoc times. By breaking out areas of the application into assignments for individual exploration, you could expose a vital new area for product enhancement and usability. Another bonus: You are simultaneously driving better information and idea-sharing between team members and greater understanding of the product's overall functionality.

ET for UAT: Convincing product owners or subject matter experts to take a turn in exploratory testing, especially on critical features, is an effective way to drive ownership around the QA process and results. Because product owners typically have a closer relationship to the customer and business, they may bring a new perspective to the usefulness of an application or its features.

Beta testing: Beta testers from the community who aren't typically skilled in development or testing supply the critical customer viewpoint of a product. Instead of a free-for-all approach, give beta testers a specific charter or mission based on whatever aspects of the application deserve attention.

Replace traditional testing: Many organizations can replace some of the manual scripted tests being performed with automated ones—or, if manual tests are duplicative or ineffective, they can be gotten rid of altogether. With this newly found bandwidth, you can put those testers to use in exploratory testing sessions. In testing, as in all areas of product design and development, efficiency is as important as speed.

Any tester or developer on your team already has the skills to help with ET efforts. By integrating exploratory testing, you can add value quickly and easily to your overall product development goals.

Using Tours to Structure Your Exploratory Testing

By Nishi Grover Garg

I had just started working with a new product, a web-based platform that was a fairly complex system with a large number of components, each with numerous features. Going into each component and inside every feature would take too much time; I needed a quick, broad overview and some feedback points I could share as queries or defects with my team.

I realized my exploration of the application would need some structure around it. Using test sessions and predefined charters, I could explore set areas and come back with relevant observations—I had discovered tours.

Cem Kaner describes tours as an exploration of a product that is organized around a theme. Tours help bring structure and a definite direction to exploration sessions, so they can be used as a fundamental tool for exploratory testing.

Tours are excellent for surfacing a collection of ideas that you can then further explore in depth one at a time. Tours testing provides a structure to the tester on the way they go about exploring the system, so they can have a particular focus on each part and not overlook a component. The structure is combined with a theme of the tour, which provides a base for the kind of questions to ask and the type of observations that need to be made.

In the course of conducting a tour, testers can find bugs, raise questions, uncover interesting aspects and features of the software, and create models, all done on the basis of the theme of the tour being performed.

Tours testing provides a structure to the tester on the way they go about exploring the system, so they can have a particular focus on each part and not overlook a component.

Let's discuss some common types of tours that are useful for testers and look at some examples.

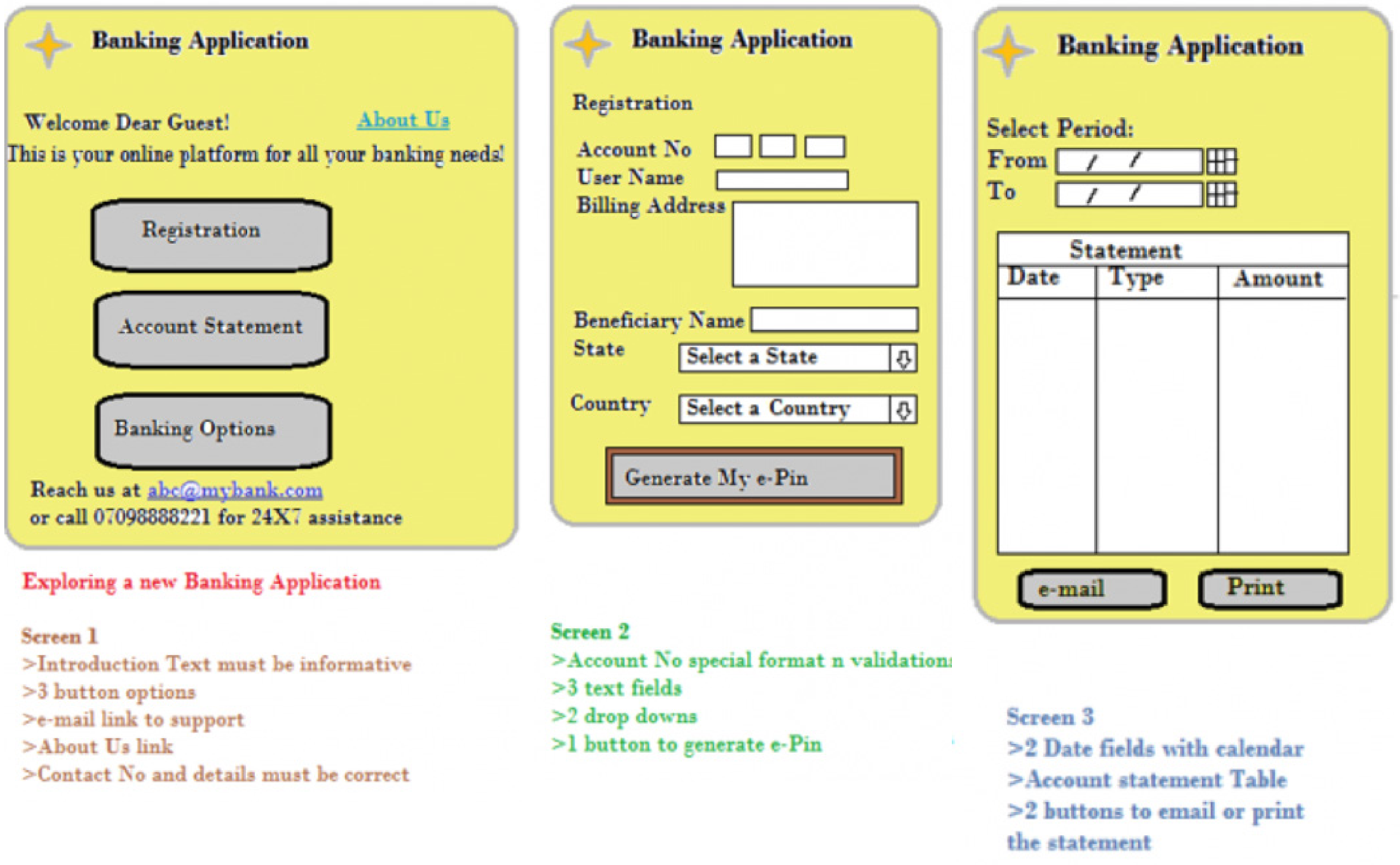
Application Tours

An application tour traverses the application screen by screen, with the aim of exploration as well as learning the application.

This is the first kind of tour I used, as an exploration method for my new project, by investigating all the screens one by one, noting the options and features present, and getting familiar with the application.

Let's look at a sample banking mobile application with three screens. The first screen is the welcome screen with options, the second is the registration screen, and the third is where we can fetch our account statement based on the time period range we select.

Here are the details of our application tour, which gives us a general overview of the app.



Banking Application

Welcome Dear Guest! About Us

This is your online platform for all your banking needs!

Registration

Account Statement

Banking Options

Reach us at abc@mybank.com

or call 07098888221 for 24X7 assistance

Banking Application

Registration

Account No

User Name

Billing Address

Beneficiary Name

State: Select a State

Country: Select a Country

Generate My e-Pin

Banking Application

Select Period:

From / /

To / /

Statement		
Date	Type	Amount

e-mail Print

Exploring a new Banking Application

Screen 1

>Introduction Text must be informative

>3 button options

>e-mail link to support

>About Us link

>Contact No and details must be correct

Screen 2

>Account No special format n validations

>3 text fields

>2 drop downs

>1 button to generate e-Pin

Screen 3

>2 Date fields with calendar

>Account statement Table

>2 buttons to email or print the statement

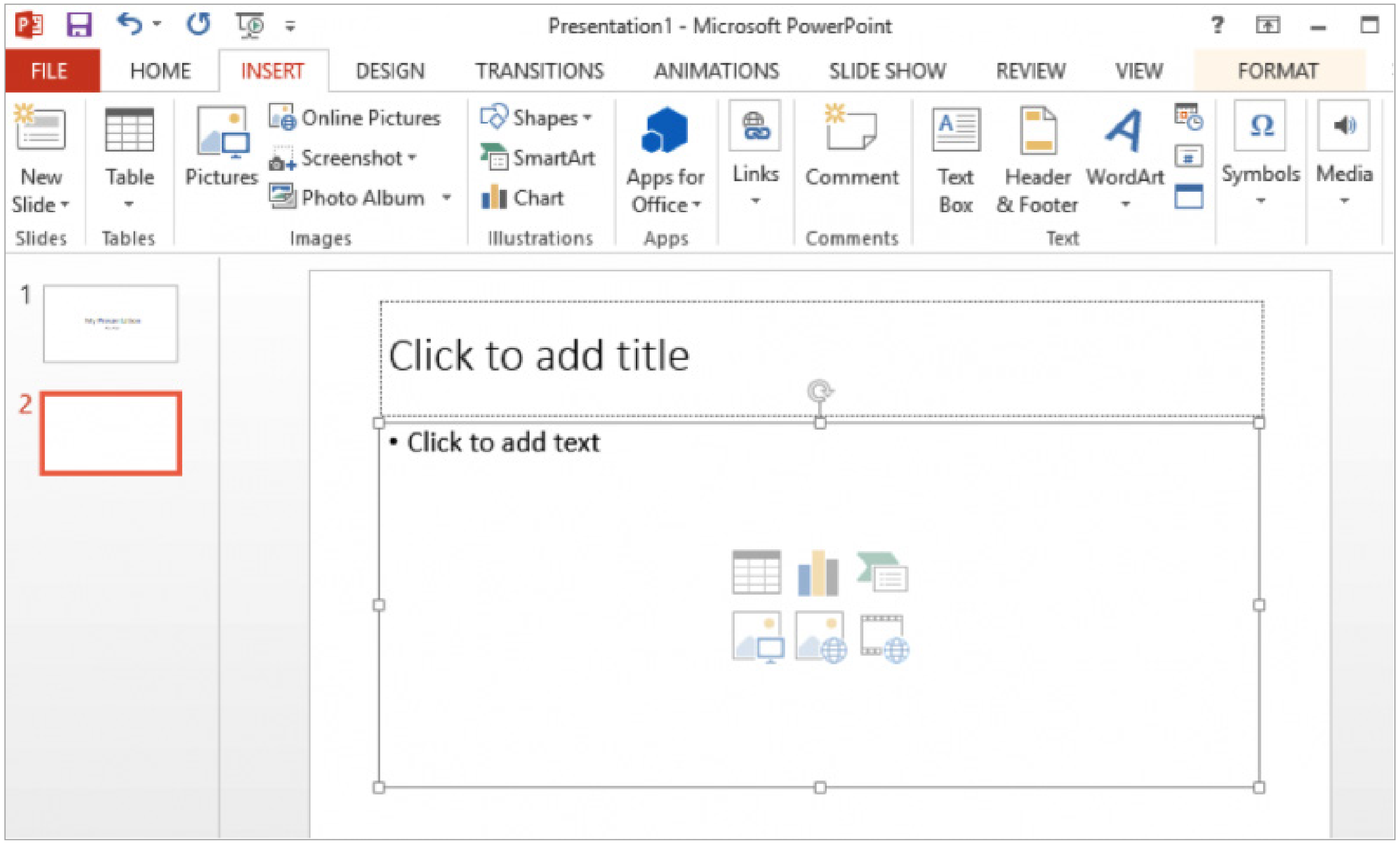
Feature Tours

After becoming familiar with the application through the application tour and noting my interest areas and observations, I proceeded to perform a feature tour.

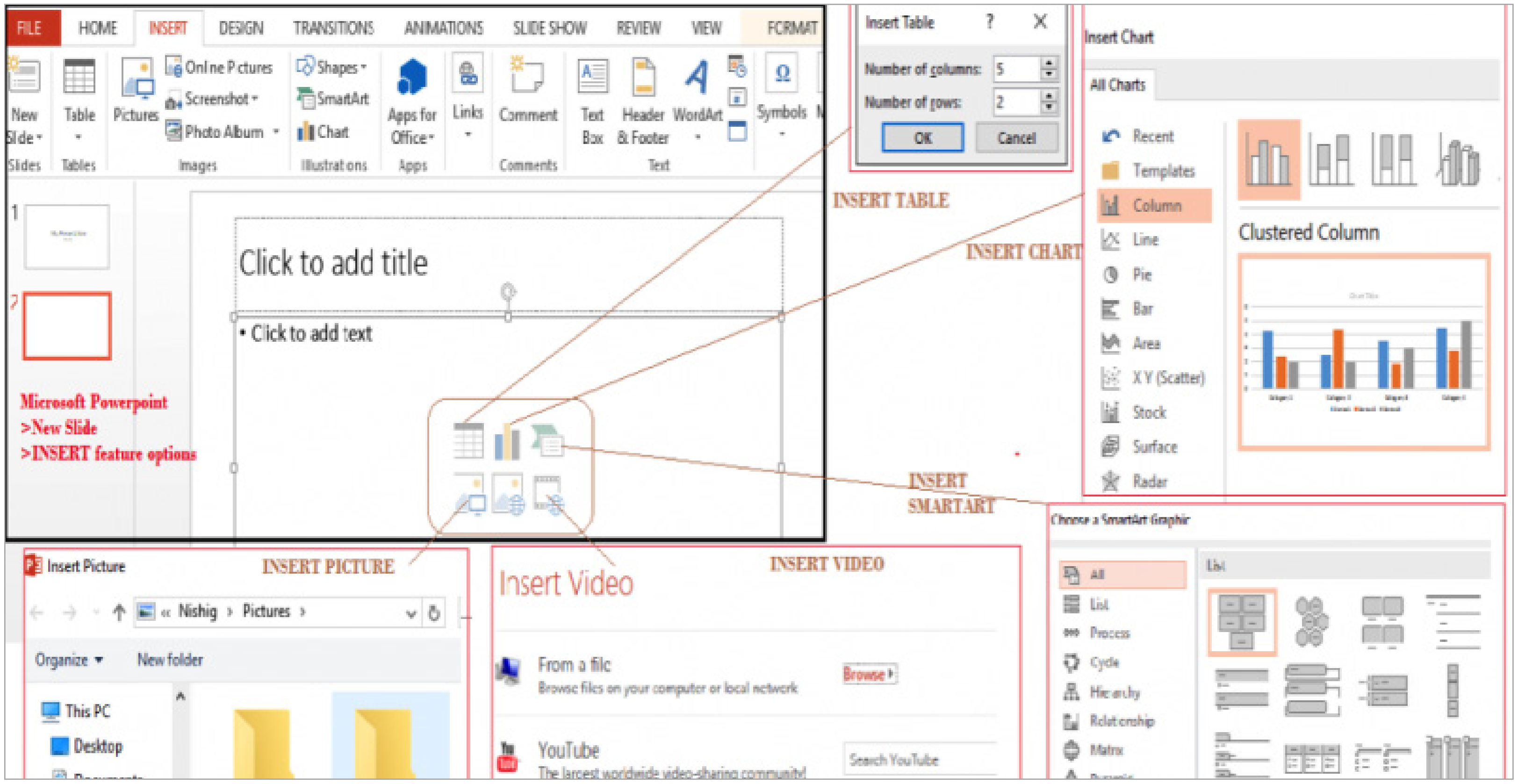
For this I selected a concerned test area and tried out all the features of that area. I customized this tour a bit by focusing on the most used and most valuable features, which I needed to learn for my upcoming user stories.

In a feature tour , the tester is encouraged to move through the application and get familiar with all features they come across. It can be used for the purpose of learning as well as checking the functionality. Along with the most common features, the tester may also try out the less popular features of the application, as time and exploration permit.

For example, let’s conduct a mock feature tour of Microsoft PowerPoint. First we’ll start with adding a new slide. Our tour will focus on the “Insert” features in the “Add Slide” contents.



On the new slide, the default options that appear are "Add Header" and "Add Body." Inside the "Add Body" option, the tour should include all the feature options available, one by one—table, chart, picture, etc.—looking at the dialog boxes that pop up and touching on each available option.



Of course we do not need to go into exhaustive detail for each feature. The only thing we are looking for is a structure and direction. This tour lets us pick where to focus more time and effort, while other features we may choose to just touch upon.

Menu Tours

Menu tours navigate through all menus and submenus in the app, visiting each one for understanding and checking.

I used a menu tour as a structured charter for a one-hour test session, traversing all menus, submenus, and items in the application. This brought me final clarity of what I needed to know about the application, the use of various options, and what lay within each one. I was free to go and explore the ones I needed in detail later.

Here's how to perform a menu tour:

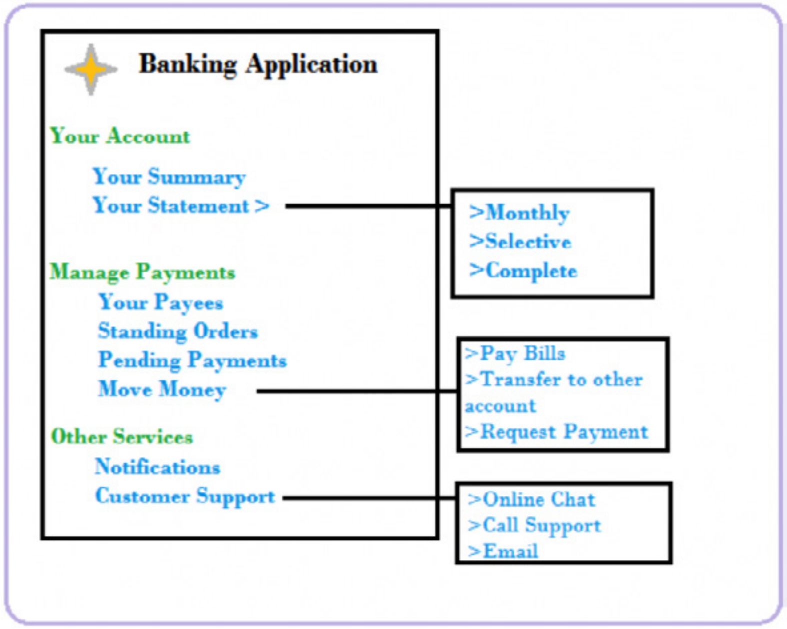
- Go through all menus and submenus available in the application.
- Check for context-sensitive menus that become active or inactive under certain situations.
- Check for context-sensitive menus that come up with a right-click.
- Check for shortcut keys to activate menus and features.
- Look for option settings that reveal new features.
- Look for user-dependent features or menus.

If required or if you find it useful, you may record all the options in these menus and submenus as lists, diagrams, screenshots, or mind maps. Let's go back to the banking app example to demonstrate. A menu tour traverses all the menu and submenu items with the above pointers in mind, and you could diagram it as above, right.

All these tours helped me speed up my learning and perform my exploration of applications in a directed manner. You can adopt these tours as a part of your functional tests, exploratory cycles, or system tests.

Other Types of Tours

Depending on your testing needs and context, there are even more tours you could perform:



- **Documentation tour:** In this tour, you explore the user manual and help guide of the product and check the instructions for correctness of information provided, updates made for the new release features, user-friendliness, and the ease of understandability of the documentation.
- **Interoperability tour:** This tour aims to investigate the interaction of the system with third-party apps and systems, focusing only on the interface points of the applications.
- **Continuous use tour:** Here, you tour the system for long, ongoing usage with multiple screens and files open for a prolonged duration in order to observe memory usage.

I hope this introduction helps you start performing tours as part of your exploratory test sessions so that you can be more deliberate with your observations, craft more useful tests, and better understand your products.

Insight from Around the Industry

"Exploratory testing is a time-based, minimum-planning, maximum-execution hands-on approach to testing. It focuses on finding known-unknown and unknown-unknown defects, which normally cannot be found with automated or any type of scripted tests which cover known-known and unknown-known issues. Exploratory testing is where true testers really get to exercise their skills, curiosity, and passion for testing, apply mnemonic devices, and think outside of the box to find most hard-to-find bugs."
» Tanya Kravtsov

"Sometimes when we do exploratory testing we find that the developer is trying to get that burn-down chart to behave, and so they're pushing out code as fast as they can. Sure, the automated test they wrote passed, but they didn't necessarily do their due diligence to really examine the code and play with it."
» Matt Attaway

"Testers have to take on new skills. No doubt. There still is manual testing, but it's going to be more exploratory testing, ad hoc testing. The role of the manual tester, having dedicated manual testers, is going away. It's an automate-first culture, movement, if you will, and that requires that testers be engineers and have some level of programming expertise."
» Adam Auerbach

"We really need exploratory testing for these things. We need human judgment and really what I see the role of automation doing is freeing up the tester from the things that would basically keep them tied to the assembly line. We don't need testers to press buttons. We don't need testers to execute the same recipe over and over. We want humans who actually can think and design and explore."
» Stephen Vance

"Testing is something that is shifting from more of a black box perspective into more developers and we're blurring that role. Then over the years, how that has kind of caused us to be confused about whether or not automation is really automation and that testing is merely an exploratory thing. So everyone seems to have a lot of confusion around testing."
» Tariq King

We want humans who actually can think and design and explore.

"The way that I look at it is, you know, the creative part of testing, which is the exploratory manual testing, is not going to go away. However, I would say that we would need a right mix of both—test automation and manual testing. If somebody tells me that they're doing everything with automation, I'd be very cautious about it and vice versa."
» Kalyan Konda

"What tends to happen is developers and testers tend to collaborate much closer, and they tend to share a lot of these things. So the testers still own the exploratory testing, and they're still doing that, but except what they're doing it is they're doing it in close proximity of the developers as tiny pieces are being completed, they're doing it. So it's more iterative."
» Jeff "Cheezy" Morgan

"The whole team owns quality, the automation is just a task on the sprint, and we need to have a few people on the team who can perform that task. With that said, I do encourage and train those testers that want to learn on how to code. I believe in the T-shaped Scrum team—the more multi-skilled people on the team, the less "scrum-merfall" there is."
» Mary Thorn

TECHWELL™

Additional Resources

MORE INFORMATION FOR SOFTWARE PROFESSIONALS

STICKYMINDS™ A TECHWELL COMMUNITY

StickyMinds is home to thousands of software testing resources, including articles, *Better Software* magazine articles, conference presentations, and interviews with industry notables.

CLICK HERE

NARROW YOUR SEARCH TO A SPECIFIC TYPE OF RESOURCE:

StickyMinds Articles

StickyMinds articles cover a wide range of software testing topics including exploratory testing, test automation, test management, test design techniques, agile testing, test process improvement, test tools, and much more. **Click here** to read exploratory testing articles on StickyMinds.

Better Software Magazine Articles

Better Software magazine is a digital quarterly filled with expert analysis, how-to articles, and real-world case studies covering all aspects of software development. **Click here** to join StickyMinds and access *Better Software* magazine articles about exploratory testing.

TechWell Conference Presentations

Couldn't make it to a TechWell conference to sharpen your exploratory testing skills and knowledge? TechWell conference presentations are available to StickyMinds members soon after a conference ends. **Click here** to join StickyMinds and access conference presentations related to exploratory testing.

Interviews

Each year, TechWell interviews dozens of software professionals including well-known thought leaders, seasoned practitioners, and respected conference speakers. **Click here** to read, listen to, and watch interviews with exploratory testing experts.

STAR Conferences

The STAR conferences are full of keynotes, tutorials, and classes covering everything from agile testing to UX testing. Learn from experts in the field and network with your peers to get the most immersive agile conference experiences possible. **Learn More.**

SQE TRAINING A TECHWELL COMPANY

SQE Training offers both foundational and specialized agile, DevOps, and software testing courses to help your team deliver better software. Learn more about their exploratory testing, test automation, and test planning courses at sqetraining.com/testing.