A Recipe for Successful Exploratory Testing

Written by Daniel Kraus with help from Christine Paras

Exploratory testing can be messy without the proper guidance. This infographic breaks down the 3 basic ingredients which make a successful recipe for exploratory testing.

In "Exploratory Testing Explained" James Bach defines exploratory testing as "simultaneous learning, test design and test execution" and describes it as follows: "[…] any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests."

Doesn't this sound good? Indeed, but without some guidance, exploratory testing can be a mess. Let's show you three basic ingredients which make a successful recipe for testing.

OVERVIEW

Prep time: 5 min

(To set up your machine)

Cooking time: 30 - 60 min

(To "explore" your system under test)

Result: Well-tested Software

(And happy users)

INGREDIENTS

1. PROTOCOL

+

2. TESTING TOUR

+

3. TIMEBOX

TOUR EXAMPLES

Antisocial
(Always do the opposite)

Landmark
(Most important features in different orders)

Garbage Collector
(Go street by street, house by house)

Supermodel
(GUI) only)

For successful exploratory testing, it is important to start with the *timebox* first. The *timebox* defines the amount of time available for exploratory testing (usually 30 – 60 min). This allows you to be extremely focused and makes planning easy as you can decide per sprint, week, or day, on how much time you want to spend on testing.

Next comes the *testing tour*. According to Michael Bolton's blog post "Of Testing Tours and Dashboards," the history of "touring" goes back to the mid-90s. The idea is largely based on when tourists find themselves in a new city, they will explore it on the basis of various topics that personally interest them. This concept can be applied to exploratory testing. Testing tours help to structure and organize exploratory testing so that people don't test the same feature over and over again, while other parts of the system under test (SUT) remain untested.

Distinct and catchy names like the ones above will spice up the team's vocabulary and ensure that they will get used to the SUT.

The 3rd and final ingredient are *protocols*. Although they are quite unpopular, they can be very helpful when it comes to exploratory testing. Their ability to record what has been tested allows us to go back in time and see why a particular bug hasn't been found. It also gives the team the opportunity to see what has worked in the past and what hasn't. Furthermore, if a test case has proven itself to be effective, the corresponding protocols can be used as a blueprint for automating it.

This protocol (right) uses Markdown for formatting, which can be utilized to extract other formats such as HTML or PDF (e.g. with Pandoc). The protocol itself should be lightweight and informal. This encourages people to write and read it. Plus, a protocol like this can also help new team members get on board easier.

```
# 2018-11-02T09:30

Scope: my software
Tour: landmark
Timebox: 30 minutes
Legend: (C)orrect, (B)ug, (?) unknown

# Test Cases

## 1

1) open menu "x" via "y" - C
2) select "z" and type in "foo" - C
3) fill out form bottom to top - B

...
```

Note that these are just loose guidelines. Some teams like to use Git-versioned Markdown files for their protocols, whereas others may prefer Excel sheets in a private cloud. Feel free to customize the entire process to your

team's/department's/organization's individual needs, but without disregarding the basic concepts.

In that sense, bon appétit!

This recipe was based on the original article by Daniel Kraus via medium.com: https://medium.com/@beatngu13/a-recipe-for-succesful-exploratory-testing-231dc8c44eea