SCENARIO TESTING

Scenario testing

- Tag line
  - Tell a persuasive story
- Fundamental question or goal
  - Challenging cases that reflect real use.
- Paradigmatic case(s)
  - Appraise product against business rules, customer data, competitors' output
  - Life history testing (Hans Buwalda's "soap opera testing.")
  - Use cases are a simpler form, often derived from product capabilities and user model rather than from naturalistic observation of systems of this kind.

Scenario testing

- The ideal scenario has several characteristics:
  - The test is ***based on a story*** about how the program is used, including information about the motivations of the people involved.
  - The story is ***credible***. It not only *could* happen in the real world; stakeholders would believe that something like it probably *will* happen.
  - The story involves a ***complex use*** of the program ***or a complex environment or a complex set of data***.
  - The test results are ***easy to evaluate***. This is valuable for all tests but is especially important for scenarios because they are complex.

What is a Test Scenario?

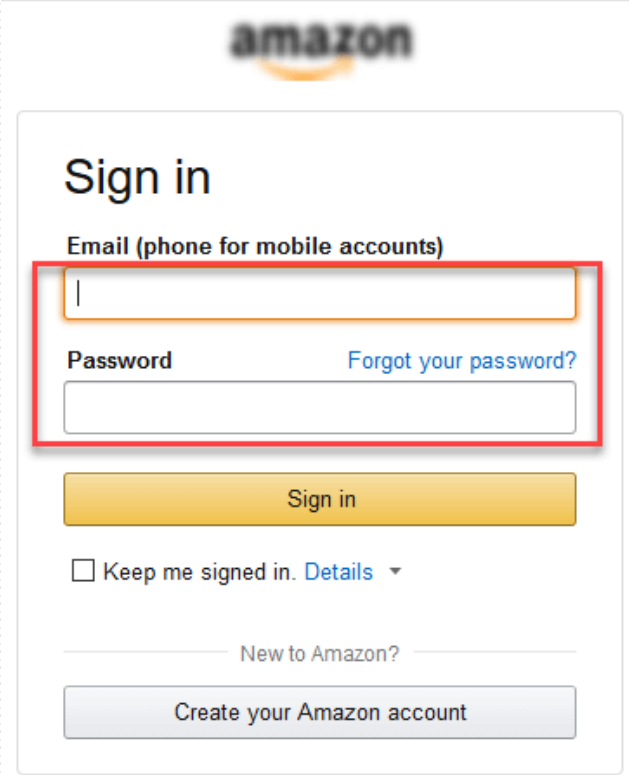
- A Test Scenario is defined as any functionality that can be tested.
- It is also called Test Condition or Test Possibility.
- As a tester, you should put yourself in the end user’s shoes and figure out the real-world scenarios and use cases of the Application Under Test.

Ex: Test Scenarios for a Banking Site

- Test Scenario 1: Check the Login and Authentication Functionality
- Test Scenario 2: Check Money Transfer can be done
- Test Scenario 3: Check Account Statement can be viewed
- Test Scenario 4: Check Fixed Deposit/Recurring Deposit can be created
- …….

Test Scenario for eCommerce Application

- Test Scenario 1: Check the Login Functionality



Specific test cases for this Test Scenario would be:

1. Check system behavior when valid email id and password is entered.
2. Check system behavior when invalid email id and valid password is entered.
3. Check system behavior when valid email id and invalid password is entered.
4. Check system behavior when invalid email id and invalid password is entered.
5. Check system behavior when email id and password are left blank and Sign in entered.
6. Check Forgot your password is working as expected
7. Check system behavior when valid/invalid phone number and password is entered.
8. Check system behavior when “Keep me signed” is checked

Why use scenario tests?

- Learn the product
- Connect testing to documented requirements
- Expose failures to deliver desired benefits
- Explore expert use of the program
- Make a bug report more motivating
- Bring requirements-related issues to the surface, which might involve reopening old requirements discussions (with new data) or surfacing not-yet-identified requirements.

Scenarios

- Designing scenario tests is much like doing a requirements analysis but is not requirements analysis. They rely on similar information but use it differently.
  - The requirements analyst tries to foster agreement about the system to be built. The tester exploits disagreements to predict problems with the system.
  - The tester doesn't have to reach conclusions or make recommendations about how the product should work. Her task is to expose credible concerns to the stakeholders.
  - The tester doesn't have to make the product design tradeoffs. She exposes the consequences of those tradeoffs, especially unanticipated or more serious consequences than expected.
  - The tester doesn't have to respect prior agreements. (Caution: testers who belabor the wrong issues lose credibility.)
  - The scenario tester's work need not be exhaustive, just useful.

Sixteen ways to create good scenarios

1. Write life histories for objects in the system. How was the object created, what happens to it, how is it used or modified, what does it interact with, when is it destroyed or discarded?

   For a library management system, track the lifecycle of a book - from acquisition, cataloguing, lending, returning, and eventually removal from the system due to wear and tear.

2. List possible users, analyze their interests and objectives

   In a fitness app, identify users like fitness enthusiasts, beginners, and professional athletes, and tailor the app's features to their specific fitness goals and routines.

3. Consider disfavored users: how do they want to abuse your system?

   For an online store, think about how a fraudulent user might try to exploit payment systems and implement safeguards against such abuses

Sixteen ways to create good scenarios

4. List system events. How does the system handle them?
   - In a weather forecasting application, handle events like data updates from meteorological stations, user location changes, and emergency weather alerts.
5. List special events. What accommodations does the system make for these?
   - For an event management software, accommodate special occasions like leap years, daylight saving time changes, or local holidays.
6. List benefits and create end-to-end tasks to check them.
   - In a project management tool, verify benefits like improved team collaboration by simulating the complete process of a team working on a project.

Sixteen ways to create good scenarios

7. Look at the specific transactions that people try to complete, such as opening a bank account or sending a message. What are all the steps, data items, outputs, displays, etc.?

   For a banking app, walk through the entire process of opening a bank account, including steps like identity verification, initial deposit, and receiving account details.

8. What forms do the users work with? Work with them (read, write, modify, etc.)

   In a healthcare system, ensure forms for patient information are easy to fill, read, and modify by healthcare providers.

Sixteen ways to create good scenarios

9. Interview users about famous challenges and failures of the old system.

   Talk to users of a legacy CRM system to understand pain points and areas for improvement.

10. Work alongside users to see how they work and what they do.

    Spend a day with a retail store manager to see how they use an inventory management system in real-life scenarios.

11. Read about what systems like this are supposed to do. Play with competing systems.

    Analyse how other e-commerce platforms handle user engagement and compare it to your system's approach.

12. Study complaints about the predecessor to this system or its competitors.

    Review feedback on older versions of a travel booking app to identify and address common issues in the new version.

Sixteen ways to create good scenarios

13. Create a mock business. Treat it as real and process its data.

    Simulate a restaurant business in a point-of-sale system to test its efficiency in processing orders and payments.

14. Try converting real-life data from a competing or predecessor application.

    Import data from an older accounting software into a new one to ensure compatibility and data integrity.

15. Look at the output that competing applications can create. How would you create these reports / objects / whatever in your application?

    Compare how different medical record systems generate patient reports and strive to improve clarity and detail in your system.

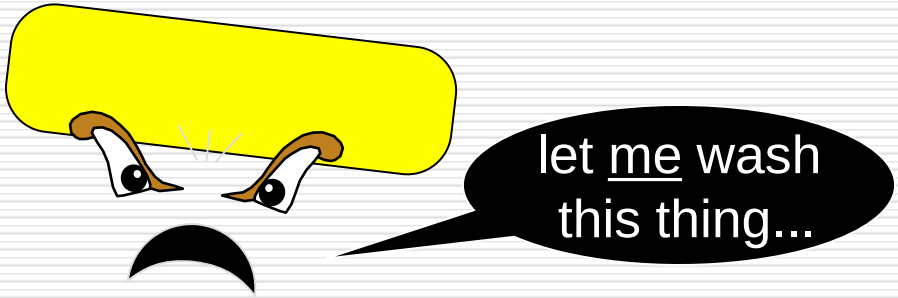
16. Look for sequences: People (or the system) typically do task X in an order. What are the most common orders (sequences) of subtasks in achieving X?

    In a customer service software, analyze common patterns in how service tickets are handled, from initial receipt to resolution, and optimize the workflow accordingly.

Soap operas

- Build a scenario based on real-life experience. This means client / customer experience.
- Exaggerate each aspect of it:
  - example, if a scenario can include a repeating element, repeat it lots of times
  - make the environment less hospitable to the case (increase or decrease memory, printer resolution, video resolution, etc.)
- Create a real-life story that combines all of the elements into a test case narrative.

“Killer Soaps”



- More specifically aimed at finding hidden problems
- Run when everything else has passed
- One option: put a killer soap at the end of a normal cluster
- Ask the “specialists” for input

From a talk by Hans Buwalda

Risks of scenario testing

- Other approaches are better for testing early, unstable code.
  - A scenario is complex, involving many features. If the first feature is broken, the rest of the test can't be run. Once that feature is fixed, the next broken feature blocks the test.
  - Test each feature in isolation before testing scenarios, to efficiently expose problems as soon as they appear.
- Scenario tests are not designed for coverage of the program.
  - It takes exceptional care to cover all features or requirements in a set of scenario tests. Statement coverage simply isn't achieved this way.
- Reusing scenarios may lack power and be inefficient
  - Documenting and reusing scenarios seems efficient because it takes work to create a good scenario.
  - Scenarios often expose design errors but we soon learn what a test teaches about the design.
  - Scenarios expose coding errors because they combine many features and much data. To cover more combinations, we need new tests.
  - Do regression testing with single-feature tests or unit tests, not scenarios.

Exercise

- eKYC
  - How to cheat eKYC process
- OrangeHRM/MentorUS
- Moodle Assignment
- Apply for mentorUS