LogiGear

Test Design for Automation: Anti-Patterns

These are 8 Anti-Patterns to watch for in Automation and How to Fix Them:

In the "Action Based Testing" method, tests can be organized in easy to manage"test modules" that look like spreadsheets, which we edit and manage in our tool, TestArchitect. The test modules contain test cases that are written as sequences of "actions," which are spreadsheet lines with an action keyword and zero or more action arguments.

What is an Anti-Pattern?

An Anti-Pattern is a situation to look for in a test that potentially can be harmful for maintainability and scalability.

An Anti-Pattern has 4 key factors:

01
It is a practice considered harmful, "the opposite of a pattern".

02
It is a pneumonic. These catchy names are helpful for memory recall and are helpful for spotting or describing the risk in each Anti-Pattern.

03
The Anti-Patterns often occur in combination and may have some overlap as well.

04
In the case of automated tests, Anti-Patterns may lead to risks in areas like: effectiveness, efficiency, readability, manageability, maintainability.

Anti-Pattern	How to Fix
Enter Enter Click Click: Many tests have been designed as long sequences detailed UI steps. This makes it difficult to manage and maintain those tests. For example, it will be hard to factor the impact of changes in the application under test into the tests.	Rather than trying to optimize bottom-up, create a high-level test design first. Then for tests with a business scope make sure the steps and matching actions are also at business level.
Interaction Heavy: I've seen in many projects that testers focus only on the interaction, and not much, if any on business tests. This makes the tests shallow which results in missing potential business level problems, thereby making the test interaction heavy.	A main distinction that testers should make recommend that testers make is between "business tests" that focus on business objects, rules and processes on one hand, and "interaction tests" that focus on the interaction with the application.
No-life/Lifeless: Missing life cycle steps of business objects. Most applications work on "business objects," like orders, invoices, products, customers, etc. These objects have their life-cycles in the application, like creation, update, retrieval and closure. However, the tests for such life-cycles are often scattered and as a result, are both hard to find and incomplete.	Life cycle tests are usually not hard to design. Identify the business objects, and for each of them identify the operations on them, including variations (like canceling an order). Design life cycle tests as business tests, not interaction tests.
Lame: No depth or variety, no testing techniques used. Time pressure and other factors often result in shallow test cases that don't challenge the application much. The tests will also be boring to read.	Try to think as a true "tester", somebody who wants to break things. Applying testing techniques and interactions with various stakeholders can help you in this process. Testing should be fun.

Anti-Pattern	How to Fix
Clueless: No clear scope for the tests. A very common situation is lack of scope for tests. The tests then are hard to find and updated if there are application changes, and may do work that is also done in other tests.	Design with clear scope for tests (esp. test modules). Understand the scope, and try not to deviate from it.
Over-Checking: Checks not relevant for the scope. Since test designers often follow an approach of steps with an expected result for each step, tests do many checks that do not fit the scope of such tests. These checks are unnecessary and probably over-lap similar checks elsewhere which end up cluttering result statistics.	Start with clear test design to avoid this Anti-Pattern. And then, when developing the tests, resist the temptation to check after every step. Only checks that fit the scope are welcome.
Cocktail: Interaction tests mixed with business tests. Even if tests are testing business functionalities, like business object life cycles and business rules, calculations and processes, they are often mixed with tests on interaction details, resulting in a convoluted and hard to maintain mix. A common example is to describe a log-in process in detail in all tests that start with a login.	This Anti-Pattern is usually a symptom of other Anti-Patterns like "clueless" so pay attention. A good modular test design can provide the direction you need to avoid cocktails.
Sneaky Checking: Checks hidden in actions. Even though it is good to have business level actions that hide unneeded details for many of the tests, try to avoid hiding too much. In particular, checks should be explicit and visible in the main test (the test modules), at the appropriate level of detail.	Try to avoid hiding too much. Design checks so they are explicit and visible in main test at the appropriate level of detail. An outsider should be able to understand what is being tested by just looking at the test module, without a need to review the implementations of the individual actions.

How to Avoid Anti-Patterns:

Being attentive in recognizing test design choices that can potentially complicate automation (and possibly testing quality) can help to make tests more concise, effective and easy to maintain. The Anti-Patterns described in this infographic may help spot issues earlier. A good way to avoid them altogether is to have a well thought out design and organization of the tests from the beginning.

Steps to fix your tests if you find an Anti-Pattern:

1. A good first step is to look at the overall test design.
2. Make a list of your tests and see if you can determine test modules with clear, unambiguous and differentiated scopes.
3. Then within each of these test modules, make sure to stay within the differentiated scope you established in step

Final note: *Anti-Patterns are a tool to discuss practices. If you communicate about Anti-Patterns, be careful that the other party understands the concept. A sentence like "this test is 'lame'" may not always come across well...*