Top 8 Signs of Bad Unit Testcases

howtodoinjava.com/best-practices/8-signs-of-bad-unit-test-cases

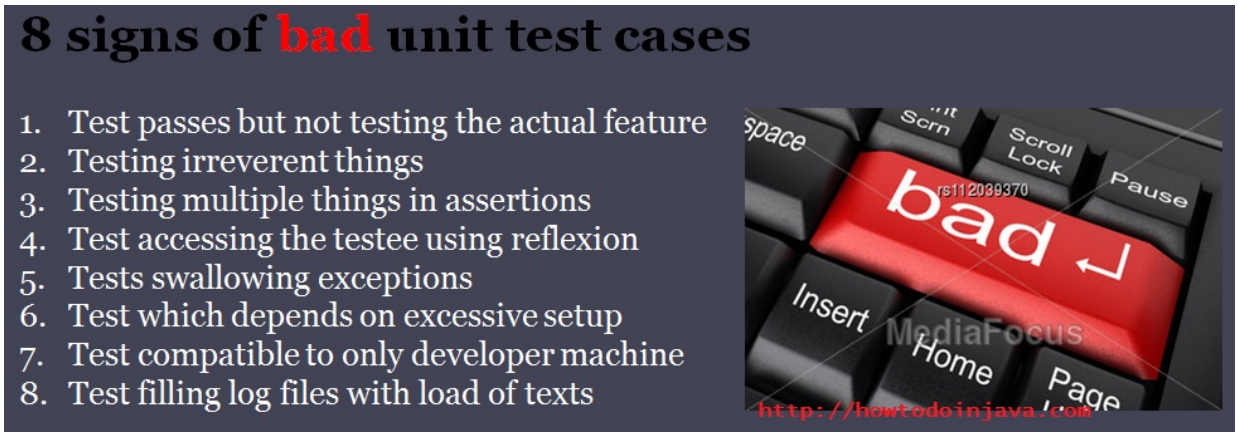
If you have been in software development for a long time, then you can easily relate with the importance of unit testing. Experts say that most of bugs can be captured in unit testing phase itself, which eventually get passed to quality teams, IF we follow these **best practices for writing junit unit tests**

"It's overwhelmingly easy to write bad unit tests that add very little value to a project while inflating the cost of code changes astronomically."

Though, bad unit tests are reality and everyone who does code review, face it occasionally (may be regularly). So what constitute a bad test case? How to identify bad test cases?

In this post, I am trying to identify 8 such signs, which will provide you subtle hints, that a particular test case it bad one and needs to be updated.

Table of Contents



8 signs of bad unit test cases

Lets see what these pointers mean, one by one:

1) Test passes but not testing the actual feature

Believe me, I have seen such test cases in my previous projects which seems to doing lots of stuff in code, but in actual they were doing nothing. They were sending requests to server and no matter what server respond, they were passing. Horror !!

Beware of such test cases making place in your code repository, by strict code reviews. Once they make their way in your code base, they will become liability on you to carry them, build them and running them every time but without adding any value.

2) Testing irrelevant things

This one is another sign of bad test case. I have seen developers checking multiple irrelevant things so that code passes with doing SOMETHING, well not necessarily the correct thing. Best approach is to follow single responsibility principle, which says, one unit test case should test only one thing and that's all.

3) Testing multiple things in assertions

This one seems to be similar to above sign, but it is not. Where in previous sign, test was testing irrelevant things, here in this sign, unit-test test the correct things but many of such things in one test case. This is again violation of single responsibility principle.

Well, please note that I am not encouraging the use of single assertion per test case. To test a single entity, you might need to use multiple assertions, do it as needed.

For example, one API which takes some parameters in post body and create Employee object and return it in response. This Employee object can have multiple fields like first name, last name, address etc. Writing a test case to validate only first-name, then another for last-name and another for address is duplicacy of code, without any value. Don't do it.

Instead, write a single positive test case for create employee and validate all fields in that unit test. Negative test cases should be written separately doing only one thing and one assertion in this case. e.g. One test case for blank first name, one test case for invalid first name and so on. All such negative test cases can be validate using single assertion which **expect exception in response**.

4) Test accessing the testee using reflexion

This one is real bad. Trying to change the testee to suite ones need. What happen when code refactoring on testee happen. test case will blow up. Do not use this or allow to be used either.

5) Tests swalloing exceptions

I got my fair share of such test cases. They silently swallow the exception in little catch block at the end of test case. And worse is that they do not raise a failed alarm also. Exceptions are signals that your applications throws to communicate that something bad has happened and you must investigate it. You should not allow the test cases to ignore these signals.

Whenever you see an unexpected exception, fail the test case without failure.

6) Test which depends on excessive setup

I do not like test cases which requires a number of things to happen before they start testing the actual thing. Such a scenario can be like: an application which facilitates online meetings. Now to test whether a user of particular type, can join the meeting, below are the steps test is performing:

- Create the User
- Set user permissions
- Create the meeting
- Set meeting properties
- Publish meeting joining information
- [Test] User join the meeting
- Pass/Fail

Now in above scenario, there are 5 complex steps which must be setup before actual logic is verified. For me, this is not a good test case.

A correct test system will have an existing user present in system for this activity, at least. It will reduce at least two steps in test case. If one can find appropriate, he can have some existing already created meetings also to make this test case really focused.

Another way to make it correct is using mock objects. They are there for this very purpose. Isn't it??

7) Test compatible to only developer machine

This is not widely seen but sometimes visible when written by freshers. They use system dependent file paths, environment variables or properties in place of using common properties/paths, or things like this. Watch out for them.

8) Test filling log files with load of texts

They do not seems creating problem in happy days. But when rainy days come, they make the life hell by putting unnecessary text without any information in log files and make the life hell for debugger who is trying to find something hidden in those log files.

Tests are not for debugging the application, so do not put debug logs kind statements. A single log statement for Pass/ Fail is enough. Believe me.

These are my thoughts based on my learning in last few years. I do not expect you to agree with me on all above points. But might have some other opinion which is perfectly cool. But I will surely like to discuss, what you feel on topic.

Happy Learning !!