How much a tester should know about internal details of code?

Asked 16 years, 2 months ago Modified 12 years, 10 months ago
Viewed 858 times

2

How useful, if at all, is for the testers on a product team to know about the internal code details of a product. This does not mean they need to know every line of code but a good idea of how the code is structured, what is the object model, how the various modules are inter-linked, what are the inter-dependencies between various features etc.? This can argubaly help them in finding related issues or defects once they hit one. On the other side, this can potentially 'bias' their "user-centric" approach towards evaluating and certifying the product and can effect the testing results in the end.

I have not heard of any specific model for such interaction. (Lets assume a product that users, potentially non-technical consume, and not a framework or API that the testers are testing - in the latter case the testers may need to understand the code to test that because the user is another programmer).

testing

Share

Improve this question

Follow

edited Feb 11, 2012 at 15:26

skaffman

403k ● 96 ● 824 ● 774

asked Oct 24, 2008 at 12:45

Ather

1,600 ● 11 ● 17

You can make the argument that even people testing an API should not know the code behind it; after all, if it's not open-source, your "customers" won't know it either, so you don't want your testers working on aspects of the API based on their knowledge of how it's written. – Dave DuPlantis Oct 24, 2008 at 12:58

There's good definitions of black-, white- and grey-box testing on wikipedia: en.wikipedia.org/wiki/Grey_box_testing#Testing_methods – Stewart Johnson Oct 24, 2008 at 14:11

14 Answers

Sorted by: Highest score (default)

4

That entirely depends upon the type of testing being done.

For functional system testing, the testers can and probably should be oblivious to the details of the implementation -- if they know the details they may

inadvertently account for that in their test strategy and not properly test the product.

For performance and scalability testing it's often helpful for the testers to have some high-level knowledge of the structure of the codebase, as it's beneficial in identifying potential performance hotspots, and therefore writing targetted test cases. The reason this is important is that generally performance testing is a broad open-ended process, so anything that can be done to focus the testing to get results is beneficial to everybody.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:49

Stewart Johnson

14.4k ● 7 ● 63 ● 70

3

This sounds similiar to this previous question: [Should QA test from a strictly black-box perspective?]

Share Improve this answer

Follow

edited May 23, 2017 at 12:24

Community Bot

1 ● 1

answered Oct 24, 2008 at 13:25

matt b

140k ● 66 ● 284 ● 350

3

I've never seen a circumstance where a tester who knew a lot about the internals of system was disadvantaged.

I would assert that there are self justifying myths that an informed tester is as adequate or even better than a deeply technical one because:

1. It allows project managers to use 'random or low quality resources' for testing. The 'as uninformed as the user myth'. If you want this type of testing - get some 'real' users to test your stuff.
2. Testers are still often seen as cheaper and less valuable than developers. The 'anybody can do blackbox testing myth'.
3. Development can defer proper testing to the test team. Two myths in one 'we don't need to train testers' and 'only the test team does testing' myths.

Share Improve this answer

Follow

answered Oct 24, 2008 at 13:52

tonylo

3,352 ● 3 ● 29 ● 27

3

What you are looking at here is the difference between black box (no knowledge of the internals), white box (all knowledge) and grey box (some select knowledge).

The answer really depends on the purpose of the code. For integration heavy projects then where and how they communicate, even if it is entirely behind the scenes, allows testers to produce appropriate non-functional test cases.

These test cases are determining whether or not a component will gracefully handle the lack of availability of a dependency. It can also be used to identify performance related issues.

For example: As a tester if I know that the Web UI component defers a request to a orchestration service that does the real work then I can construct a scenario where the orchestration takes a long time (high load). If the user then performs another request (simulating user impatience) and the web service will receive a second request while the first is still going. If we continually repeat this the web service will eventually die from stress. Without knowing the underlying model it would not be easy to find the problem

In most cases for functionality testing then black box is preferred, as soon as you move towards non-functional or system integration then understanding the interactions can assist in ensuring appropriate test coverage.

Not all testers are skilled or comfortable working/understanding the component interactions or internals so it is on a per tester/per system basis on whether it is appropriate.

In almost all cases we start with black box and head towards white as the need sees.

Share Improve this answer

Follow

edited Oct 30, 2008 at 1:09

Adam Bellaire

110k ● 19 ● 152 ● 165

answered Oct 24, 2008 at 12:59

Ryan Boucher

359 ● 2 ● 8

As I knew it, Black Box meant "no access to code", while White Box meant "full access to code" and Gray Box meant "read-only access to code". So, both white and gray boxes can get to know all of it, but only white boxes get to change it. – schonarth Oct 24, 2008 at 13:50

There's good definitions of black-, white- and grey-box testing on wikipedia: en.wikipedia.org/wiki/Grey_box_testing#Testing_methods – Stewart Johnson Oct 24, 2008 at 14:06

1

A tester does not need to know internal details.

The application should be tested without any knowledge of interal structure, development problems, externals depenednecies.

If you encumber the tester with those additional info you push him into a certain testing scheme and the tester should never be pushed in a direction he should just test from a non coder view.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:48

Peter

5,778 ● 22 ● 23

1

There are multiple testing methodologies that require code reviewing, and also those that don't.

The advantages to white-box testing (i.e. reading the code) is that you can tailor your testing to only test areas that you know (from reading the code) will fail.

Disadvantages include time wasted from actual testing to understand the code.

Black-box testing (i.e. not reading the code) can be just as good (or better?) at finding bugs than white-box.

Normally both types of testing can happen on one project, developers white-box unit testing, and testers black-box integration testing.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:50

GavinCattell

3,953 ● 22 ● 22

1

I prefer Black Box testing for final test regimes In an ideal world...

- Testers should know nothing about the internals of the code

- They should know everything the customer will - i.e. have the documents/help required to use the system/application.(this definetly includes the API description/documents if it's some sort of code deliverable)

If the testers can't manage to find the defects with these limitations, you haven't documented your API/application enough.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:50

NotJarvis

1,247 ● 11 ● 18

1 Hmm. I agree with the general point for acceptance testing, but I'm not sure I want to rely on a testing regime which only finds the bugs if the documentation is adequate. Adequate docs are difficult to achieve (even formal specs get amended), and there are no tests you can run to check they're OK. – Steve Jessop Oct 24, 2008 at 13:04

Agreed - I was referring more to Acceptance testing before it goes out the door. In the ideal world there would be a number of testing regimes for a release. – NotJarvis Oct 29, 2008 at 13:34

Black box *testing* says nothing about what black box *testers* should know. I've never worked in an environment that accepted good testing was possible without some "under the hood" knowledge. – testerab Dec 13, 2010 at 0:59

If they are dedicated testers (Only thing they do) then I think they should know as little about the code as

1

possible that they are attempting to test.

Too often they try to determine why its failing, that is the responsibility of the developer not the tester.

That said I think developers make great testers, because we tend to know the edge cases for certain types of functionality.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:52

Brian Schmitt

6,068 ● 1 ● 26 ● 36

1

Here's an example of a bug which you can't find if you don't know the code internals, because you simply can't test all inputs:

```
long long int increment(long long int l) {
    if (l == 475636294934LL) return 3;
    return l + 1;
}
```

However, in this case it would be found if the tester had 100% code coverage as a target, and looked at only enough of the internals to write tests to achieve that.

Here's an example of a bug which you quite likely won't find if you *do* know the code internals, because false confidence is contagious. In particular, it is usually not possible for the author of the code to write a test which catches this bug:

```
int MyConnect(socket *sock) {
    /* socket must have been bound already, but
that's OK */
    return RealConnect(sock);
}
```

If the documentation of MyConnect fails to mention that the socket must be bound, then something unexpected will happen some day (someone will call it unbound, and presumably the socket implementation will select an arbitrary local address). But a tester who can see the code often doesn't have the mindset of "testing" the documentation. Unless they're really on form, they won't notice that there's an assumption in the code not mentioned in the docs, and will just accept the assumption. In contrast, a tester writing from the docs could easily spot the bug, because they'll think "what possible states can a socket be in? I'll do a test for each". Since no constraints are mentioned, there's no reason they won't try the case that fails.

Answer: do both. One way to do this is to **write a test suite before you see/write the code**, and then add more tests to cover any special cases you introduce in your implementation. This applies whether or not the tester is the same person as the programmer, although obviously if the programmer writes the second kind of test, then only one person in the organisation has to understand the code. It's arguable whether it's a good long-term strategy to have code only one person has ever understood, but it's widespread, because it certainly saves time getting something out the door.

[Edit: I decline to say how these bugs came about. Maybe the programmer of the first one was clinically insane, and for the second one there are some restrictions on the port used, in order to workaround some weird network setup known to occur, and the socket is supposed to have been created via some de-weirdifying API whose existence is mentioned in the general sockets docs, but they neglect to require its use. Clearly in both these cases the programmer has been very careless. But that doesn't affect the point: the examples don't need to be realistic, since if you don't catch bugs that only a very careless programmer would make, then you won't catch all the actual bugs in your code unless you never have a bad day, make a crazy typo, etc.]

Share Improve this answer Follow

edited Oct 24, 2008 at 13:43

answered Oct 24, 2008 at 13:18

Steve Jessop

279k ● 40 ● 469 ● 709

1

I guess it depends how good of testing you want. If you just want to sanity check the common scenarios, then by all means, just give the testers / pizza-eaters the application and tell them to go crazy.

However, if you'd like to have a chance at finding edge cases, performance or load issues, or a whole lot of other issues that hide in the depths of your code, you'd

probably be better off hiring testers who know how and when to use white box techniques.

Your call.

Share Improve this answer

Follow

answered Oct 26, 2008 at 5:12

Alan

1,045 ● 7 ● 14

1

IMHO, I think the industry view of testers is completely wrong.

Think about it ... you have two plumbers, one is extremely experienced, knows all the rules, the building codes, and can quickly look at something and know if the work is done right or not. The other plumber is good, and get the job done reliably.

Which one would you want to do the final inspection to make sure you don't come home to a flooded house? In fact, in what other industry do they allow someone who knows hardly anything about the system they are inspecting to actually do the inspection?

I have seen the bar for QA go up over the years, and that makes me happy. In time, QA may become something that devs aspire to be.

In short, not only should they be familiar with the code being tested, but they should have an understanding that rivals the architects of the product, as well as be able to

effectively interface with the product owner(s) / customers to ensure that what is being created is actually what they want. But now I am going into a whole seperate conversation ...

Will it happen? Probably sooner than you think. I have been able to reduce the number of people needed to do QA, increase the overall effectiveness of the team, and increase the quality of the product simply by hiring very skilled people with dev / architect backgrounds with a strong aptitude for QA. I have lower operating costs, and since the software going out is higher quality, I end up with lower support costs. FWIW ... I have found that while I can backfill the QA guys effectively into a dev role when needed, the opposite is almost always not true.

Share Improve this answer

Follow

answered Oct 26, 2008 at 5:43

user25306

421 ● 2 ● 6 ● 7

1

If there is time, a tester should definitely go through a developers code. This way, you can improve your tests to get better coverage.

So, maybe if you write your black box tests looking at the spec and think you have the time to execute all of those and will still be left with time, going through code cannot be a bad idea.

Basically it all depends on how much time you have.. Another thing you can do to improve coverage is look at

the developers design documents. Those should give you a good idea of what the code is going to look like...

Testers have the advantage of being familiar with both the dev code and the test code!

Share Improve this answer

Follow

answered Oct 29, 2008 at 8:18

user32285

21 • 1

0

I would say they don't need to know the internal code details at all. However they do need to know the required functionality and system rules in full detail - like an analyst. Otherwise they won't test all the functionality, or won't realise when the system misbehaves.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:51

Tony Andrews

132k • 23 • 227 • 264

0

For user acceptance testing the tester does not need to know the internal code details of the app. They only need to know the expected functionality, the business rules. When a bug is reported

Whoever is fixing the bug should know the inter-dependencies between various features.

Share Improve this answer

Follow

answered Oct 24, 2008 at 12:54

Aaron Palmer

8,982 9 50 77