What's the difference between unit, functional, acceptance, and integration tests? [closed]

Asked 13 years, 10 months ago Modified 6 years, 11 months ago

Viewed 249k times

846

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 9 years ago.

The community reviewed whether to reopen this question 10 months ago and left it closed:

> Original close reason(s) were not resolved

Improve this question

What is the difference between unit, functional, acceptance, and integration testing (and any other types of tests that I failed to mention)?

testing terminology definition

Share

Improve this question

Follow

edited Dec 31, 2017 at 14:46

Mogsdad

45.7k ● 21 ● 160 ● 284

asked Feb 4, 2011 at 23:59

Andrew

238k ● 195 ● 529 ● 717

2 See also sqa.stackexchange.com/a/23396/8992 – Michael Durrant Dec 4, 2016 at 13:01

3 I think you forgot to include load testing! – Talk is Cheap Show me Code May 9, 2018 at 14:48

Based on the target on which you are testing (Test Setup), also you can categorize tests into HIL(Hardware), MIL(Machine), SIL(Softwre) (IN Loop). – np2807 Dec 1, 2020 at 7:31

8 Answers

Sorted by: Highest score (default)

1404

Depending on where you look, you'll get slightly different answers. I've read about the subject a lot, and here's my distillation; again, these are slightly wooly and others may disagree.

Unit Tests

Tests the smallest unit of functionality, typically a method/function (e.g. given a class with a particular state, calling x method on the class should cause y to happen). Unit tests should be focussed on one particular feature

(e.g., calling the pop method when the stack is empty should throw an `InvalidOperationException`). Everything it touches should be done in memory; this means that the test code **and** the code under test shouldn't:

- Call out into (non-trivial) collaborators
- Access the network
- Hit a database
- Use the file system
- Spin up a thread
- etc.

Any kind of dependency that is slow / hard to understand / initialise / manipulate should be stubbed/mocked/whatevered using the appropriate techniques so you can focus on what the unit of code is doing, not what its dependencies do.

In short, unit tests are as simple as possible, easy to debug, reliable (due to reduced external factors), fast to execute and help to prove that the smallest building blocks of your program function as intended before they're put together. The caveat is that, although you can prove they work perfectly in isolation, the units of code may blow up when combined which brings us to ...

Integration Tests

Integration tests build on unit tests by combining the units of code and testing that the resulting combination

functions correctly. This can be either the innards of one system, or combining multiple systems together to do something useful. Also, another thing that differentiates integration tests from unit tests is the environment. Integration tests can and will use threads, access the database or do whatever is required to ensure that all of the code **and** the different environment changes will work correctly.

If you've built some serialization code and unit tested its innards without touching the disk, how do you know that it'll work when you are loading and saving to disk? Maybe you forgot to flush and dispose filestreams. Maybe your file permissions are incorrect and you've tested the innards using in memory streams. The only way to find out for sure is to test it 'for real' using an environment that is closest to production.

The main advantage is that they will find bugs that unit tests can't such as wiring bugs (e.g. an instance of class A unexpectedly receives a null instance of B) and environment bugs (it runs fine on my single-CPU machine, but my colleague's 4 core machine can't pass the tests). The main disadvantage is that integration tests touch more code, are less reliable, failures are harder to diagnose and the tests are harder to maintain.

Also, integration tests don't necessarily prove that a complete feature works. The user may not care about the internal details of my programs, but I do!

Functional Tests

Functional tests check a particular feature for correctness by comparing the results for a given input against the specification. Functional tests don't concern themselves with intermediate results or side-effects, just the result (they don't care that after doing x, object y has state z). They are written to test part of the specification such as, "calling function Square(x) with the argument of 2 returns 4".

Acceptance Tests

Acceptance testing seems to be split into two types:

Standard acceptance testing involves performing tests on the full system (e.g. using your web page via a web browser) to see whether the application's functionality satisfies the specification. E.g. "clicking a zoom icon should enlarge the document view by 25%." There is no real continuum of results, just a pass or fail outcome.

The advantage is that the tests are described in plain English and ensures the software, as a whole, is feature complete. The disadvantage is that you've moved another level up the testing pyramid. Acceptance tests touch mountains of code, so tracking down a failure can be tricky.

Also, in agile software development, user acceptance testing involves creating tests to mirror the user stories created by/for the software's customer during development. If the tests pass, it means the software should meet the customer's requirements and the stories

can be considered complete. An acceptance test suite is basically an executable specification written in a domain specific language that describes the tests in the language used by the users of the system.

Conclusion

They're all complementary. Sometimes it's advantageous to focus on one type or to eschew them entirely. The main difference for me is that some of the tests look at things from a programmer's perspective, whereas others use a customer/end user focus.

Share Improve this answer Follow

edited Mar 9, 2014 at 15:18

answered Feb 5, 2011 at 1:32

Mark Simpson
23.4k ● 2 ● 46 ● 44

24 +1. @Mark Simpson Could functional and acceptance testing to be summed up as "system testing"? Where do end-to-end tests fit in? (too much different vocabulary for my taste) – Torsten Engelbrecht Mar 5, 2013 at 16:45

3 @Franz I was speaking about the ability and ease with which you can *reduce* risk via isolating units of code and testing them. You're right though, the language I used was a bit loose, as tests cannot prove that code is bug-free. – Mark Simpson Dec 11, 2013 at 1:23

21 Despite the up-votes, this is completely wrong. Unit-tests do not test even the "trivial" collaborators; any injected dependency must be mocked. Functional tests do not test

"behavior"; they test only "function", i.e. "f(A) returns B". If side-effects matter, it is "behavioral". If these include system-calls, they are also "system" tests, as in "behavioral system tests". (See testerab@ below.) "Acceptance" tests are a subset of "behavioral system tests" which cover the full-stack. "Integration" tests upward, simulating actual usage; it tests that all dependencies can be integrated in practice.
– cdunn2001 Mar 8, 2014 at 17:59

10 @cdunn2001: Don't worry, constructive criticism is always good :) Your comment taught me a few things I didn't know and cleaned up my terminology somewhat. I'm always keen to learn new things from developers who are keen on testing. I remember the first time I discovered Miško Hevery's blog -- it was like a treasure trove :)
– Mark Simpson Mar 10, 2014 at 12:03

13 @MarkSimpson although your answer is very good i would like a little more detail regarding functional tests. I mean in your answer, for me, is hard to distinguish between functional tests and unit tests. I hope you have time for this, keep up the great work! – Andrew Starlike Jun 18, 2014 at 9:14

99

The important thing is that you know what those terms mean to your colleagues. Different groups will have slightly varying definitions of what they mean when they say "full end-to-end" tests, for instance.

I came across Google's naming system for their tests recently, and I rather like it - they bypass the arguments by just using Small, Medium, and Large. For deciding which category a test fits into, they look at a few factors - how long does it take to run, does it access the network, database, filesystem, external systems and so on.

http://googletesting.blogspot.com/2010/12/test-sizes.html

I'd imagine the difference between Small, Medium, and Large for your current workplace might vary from Google's.

However, it's not just about scope, but about purpose. Mark's point about differing perspectives for tests, e.g. programmer vs customer/end user, is really important.

Share Improve this answer

Follow

answered Feb 5, 2011 at 16:39

testerab

1,475 ● 13 ● 22

7 +1 for the google test naming thing as it helps give a bit of perspective on why various organisations / people have different definitions for tests. – Mark Simpson Feb 5, 2011 at 22:49

This is also a very nice article going into why you'd use different levels of test and what you get out of them: kentcdodds.com/blog/unit-vs-integration-vs-e2e-tests – testerab May 8, 2020 at 11:24

75

http://martinfowler.com/articles/microservice-testing/

Martin Fowler's blog post speaks about strategies to test code (Especially in a micro-services architecture) but most of it applies to any application.

I'll quote from his summary slide:

- Unit tests - exercise the smallest pieces of testable software in the application to determine whether they behave as expected.
- Integration tests - verify the communication paths and interactions between components to detect interface defects.
- Component tests - limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.
- Contract tests - verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.
- End-To-End tests - verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

answered Feb 21, 2015 at 22:08

Maxim

7,348 ● 1 ● 33 ● 45

That's a great article by the way. However I don't completely understand what contract test are for. Aren't they redundant

in light of component and integration tests? – wheleph Apr 16, 2015 at 16:03

In some languages (that Mr Fowler uses) you can implement an interface that is not exposed when using the standard definition of a class e.g. void IMyInterface.MyMethod(). Which in turn would logically have its own tests. Although at that point you are heading back towards BDD.. Which ironically Mr Fowler has had a land grab at as well.
– Skarsnik May 5, 2015 at 5:30

2 it's not Fowler article btw, just posted there. Contract tests are test that are made after clients starts to use your service, you then write tests that checks if you don't broke something for that particular clients, i.e. change service api.
– Rafał Łużyński Dec 29, 2015 at 0:46

@wheleph unit, integration and component tests speak mostly for the software internals that are heavily controllable by the developer. An issue in the first three means changing your source to fix the issue. -- The contract tests touch what is promised to you in functionality but you may not be able to change directly in the face of defect. This requires adding support code to work around those possible issues instead of just fixing the defect. -- So you would work around a web service giving you back malformed json even if the contract specification told you it be of a certain structure. – Yemi Bedu Nov 2, 2016 at 20:18

1 To evolve more, the acceptance test. as highlighted by Dave Farley, can also be an integration test, or and e2e test. Acceptance Testing is a fit in a process, rather than a specific style of test. – Mario Galea Jan 20, 2022 at 14:51

Unit Testing - As the name suggests, this method tests at the object level. Individual software components are tested for any errors. Knowledge of the program is

35

needed for this test and the test codes are created to check if the software behaves as it is intended to.

Functional Testing - Is carried out without any knowledge of the internal working of the system. The tester will try to use the system by just following requirements, by providing different inputs and testing the generated outputs. This test is also known as closed-box testing or black-box.

Acceptance Testing - This is the last test that is conducted before the software is handed over to the client. It is carried out to ensure that the developed software meets all the customer requirements. There are two types of acceptance testing - one that is carried out by the members of the development team, known as internal acceptance testing (Alpha testing), and the other that is carried out by the customer or end user known as (Beta testing)

Integration Testing - Individual modules that are already subjected to unit testing are integrated with one another. Generally the two approachs are followed :

1) Top-Down
2) Bottom-Up

edited Nov 11, 2015 at 13:59

pedromendessk

3,618 ● 2 ● 27 ● 36

answered Jun 25, 2015 at 22:12

Shah
553 • 5 • 13

What do you mean by top-down and bottom-up? Is integration testing the same as end to end testing?
– tamj0rd2 Sep 7, 2019 at 9:27

20

This is very simple.

1. Unit testing: This is the testing actually done by developers that have coding knowledge. This testing is done at the coding phase and it is a part of white box testing. When a software comes for development, it is developed into the piece of code or slices of code known as a unit. And individual testing of these units called unit testing done by developers to find out some kind of human mistakes like missing of statement coverage etc..
2. Functional testing: This testing is done at testing (QA) phase and it is a part of black box testing. The actual execution of the previously written test cases. This testing is actually done by testers, they find the actual result of any functionality in the site and compare this result to the expected result. If they found any disparity then this is a bug.
3. Acceptance testing: know as UAT. And this actually done by the tester as well as developers, management team, author, writers, and all who are involved in this project. To ensure the project is finally ready to be delivered with bugs free.

4. Integration testing: The units of code (explained in point 1) are integrated with each other to complete the project. These units of codes may be written in different coding technology or may these are of different version so this testing is done by developers to ensure that all units of the code are compatible with other and there is no any issue of integration.

Share Improve this answer

Follow

edited Jul 2, 2017 at 0:59

Community Bot

1 ● 1

answered May 7, 2015 at 13:24

Rakesh Kumar

317 ● 2 ● 3

2 @OlegTsyba the answer came 4 years after the question was answered. – bentesha Sep 3, 2017 at 12:38

18 We should never start an answer with "This is very simple", especially if it is a complex topic such as this one. – milosmns Mar 29, 2019 at 10:06

6

Some (relatively) recent ideas against excessive mocking and pure unit-testing:

- https://www.simple-talk.com/dotnet/.net-framework/are-unit-tests-overused/
- http://googletesting.blogspot.com/2013/05/testing-on-toilet-dont-overuse-mocks.html

- http://codebetter.com/iancooper/2011/10/06/avoid-testing-implementation-details-test-behaviours/
- http://cdunn2001.blogspot.com/2014/04/the-evil-unit-test.html
- http://www.jacopretorius.net/2012/01/test-behavior-not-implementation.html
- Why Most Unit Testing is Waste

Share Improve this answer Follow

edited Apr 19, 2014 at 17:39

answered Mar 10, 2014 at 2:18

cdunn2001
18.2k ● 8 ● 57 ● 46

I am new to testing code. Unit tests seem mostly like a waste of time. I thought I was doing unit testing but I was doing integration testing and then I read about unit testing and it seems silly, maybe for people with very little experience? There's a chance I'm missing some sort of point. – Z2VvZ3Vp Mar 6, 2015 at 4:51

If *Unit* is defined broadly, then you are properly unit-testing. I oppose testing implementation details. A private class should not be "unit-tested". However, if you have several public classes, you might be tempted to mock one while testing another. That's the real debate. Is the *Unit* (a) your entire library? (b) each public class within the library? Or (c), each public method within each class? I prefer to test a given library as an integrated component, but to mock or fake external dependencies (unless they are fast and reliable). So I think I'm with you. – cdunn2001 Mar 6, 2015 at 17:57

1 @PixMach: actually it's the other way around. Not having (good) unit tests in place, wastes a lot of your time, if you (or somebody else) have to change that code in the future. If you have experience maintaining code with and without unit tests, you'll know the difference. The idea is, that if a unit test breaks, you should know exactly which part of the code has to be fixed. Failing big scale acceptance / integration tests often only tell you: it does not work. And then you have to start old school debugging... – Goodsquirrel Aug 2, 2016 at 12:26

1 @Goodsquirrel, it depends what you call a "unit". That's the problem. Bad tests will be deleted during refactoring. Good tests will still be helpful. Bad tests add no value and get in the way. Good tests are self-documenting and greatly appreciated. Let's get specific. I have a private method to return a value if another value is True, otherwise a default value. (Legacy code.) Should that method be tested? I say no. Another private method returns the nth Fibonacci number. Should that be tested? I say yes. – cdunn2001 Aug 3, 2016 at 15:55

2 The smallest *exposed* code. Big difference. – cdunn2001 Mar 3, 2018 at 18:04

5

I will explain you this with a practical example and no theory stuff:

A developer writes the code. No GUI is implemented yet. The testing at this level verifies that the functions work correctly and the data types are correct. This phase of testing is called Unit testing.

When a GUI is developed, and application is assigned to a tester, he verifies business requirements with a client

and executes the different scenarios. This is called functional testing. Here we are mapping the client requirements with application flows.

Integration testing: let's say our application has two modules: HR and Finance. HR module was delivered and tested previously. Now Finance is developed and is available to test. The interdependent features are also available now, so in this phase, you will test communication points between the two and will verify they are working as requested in requirements.

Regression testing is another important phase, which is done after any new development or bug fixes. Its aim is to verify previously working functions.

Share Improve this answer

Follow

edited Jan 6, 2017 at 18:38

revelt

2,400 ● 1 ● 26 ● 38

answered Feb 12, 2015 at 10:53

fahad shaikh

94 ● 1 ● 3

1 "A developer writes the code. No GUI is implemented yet. The testing at this level verifies that the functions work correctly and the data types are correct. This phase of testing is called Unit testing" This is not true. GUI is actually just a "plugin". You can already write E2E tests to your API output. (or any response object you generate) – ssibal Mar 12, 2018 at 10:23

4

unit test: testing of individual module or independent component in an application is known to be unit testing , the unit testing will be done by developer.

integration test: combining all the modules and testing the application to verify the communication and the data flow between the modules are working properly or not , this testing also performed by developers.

funcional test checking the individual functionality of an application is mean to be functional testing

acceptance testing this testing is done by end user or customer whether the build application is according to the customer requirement , and customer specification this is known to be acceptance testing

Share Improve this answer

Follow

answered Apr 26, 2014 at 12:33

malini

187 ● 1 ● 2

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.