Unit Test

A kind of AutomatedTest, though some would say a better name is DeveloperTest

"Unit" casually refers to low-level test cases written in the same language as the production code, which directly access its objects and members.

Under the strict definition, for QA purposes, the failure of a UnitTest implicates only one unit. You know exactly where to search to find the bug.

TestDrivenDevelopment produces DeveloperTests. The failure of a test case implicates only the developer's most recent edit. This implies that developers don't need to use MockObjects to split all their code up into testable units. And it implies a developer may always avoid debugging by reverting that last edit.

(Note that although most of the text on this Wiki uses the term UnitTest, the canonical ExtremeProgramming name has been changed to ProgrammerTest.)

UnitTests are programs written to run in batches and test classes. Each typically sends a class a fixed message and verifies it returns the predicted answer.

UnitTests are a key component of software engineering, and the ExtremeProgramming discipline leverages them to permit easy code changes. Developers write tests for every class they produce. The tests are intended to test every aspect of the class that could conceivably not work. (Do test adding and removing, don't test accessors.)

Key here is to CodeUnitTestFirst.

UnitTests are all combined into a huge suite of tests, using e.g. KentBeck's TestingFramework. When developers go to release new code, they run all the unit tests, not just theirs, on the integration machine. The tests **must** run at 100%. If any test fails, they figure out why and fix the problem. The problem certainly resides in something they did ... since they know the tests ran at 100% the last time anything was released.

Of course, sometimes the tests let something slip through. When that happens, developers unconditionally enhance the unit tests so that that problem, and any similar one that comes to mind, won't happen again.

-- RonJeffries

C3 [ChryslerComprehensiveCompensation] has over 1300 unit tests, performing over 13,000 individual checks. They run in about 10 minutes in VisualWorks. -- RonJeffries

---

Also, did you introduce tests late in development that identified unforeseen problems? If so, did your fixes to those problems break other things? In other words, how did you fight the "if it ain't broke don't fix it" attitude? -- KieranBarry

It is the UnitTests themselves that allow you to get away from the "if it ain't broke don't fix it attitude." You must be able to modify anything in the entire system at anytime. Ah the freedom of it! The only way you can do that is to require the UnitTests be run before any code is released at a SingleReleasePoint. -- DonWells

This isn't 100% true though. It is quite possible to introduce a modification, run the tests and have 100% test coverage with all tests passing, and still have broken the system. It's just less likely the more tests there are (but more likely the more complex the system is). In these cases the passing tests have induced a false sense of security. The probability of a change causing a bug to slip through a set of tests can only approach zero for an arbitrary system, although some specific systems may have such a zero probability. This is because an arbitrary system can describe the HaltingProblem, which is provably undecidable, therefore an algorithm that describes it is untestable, which means that arbitrary systems are not 100% testable, and of course if there aren't tests for a piece of code, a bug can slip though. Contrived, sure, but true in lots of other more practical examples too. -- MikeAmy

I don't think Don's talking about whether or not unit test can guarantee that a function is bug-free. I believe he's talking about how they can "cure" the "If it ain't broke, don't fix it" attitude. BTW, the inability of unit test's to guarentee correct behavior is not tied to the halting problem. We can construct languages where the halting problem (for that language) is solvable, but still can't be 100% unit tested.

What about things that can't be tested without human interaction? For example, suppose you have some code that handles printing. (And suppose the code is sufficiently deep in the system that it's not directly tested by a customer-written acceptance test. And suppose that it is necessary to test it at each integration, because breaking it can have subtle effects that will not easily be identified with this code without unit tests.) For most projects, the only practical way to test this printing code is to have a human inspect the paper that comes out of the printer and verify that it appears as it should. How does the XP testing process deal with this?

You can always install a pseudo printer (virtual printer like PDFCreator for Windows) and have the application print to it. It would generate post script or PDF file which you can easily compare with a valid .ps or .pdf file. Perhaps you picked the wrong example in printing.

If you CodeUnitTestFirst your design will be far different from what you might expect. Most likely you will isolate the code which actually prints something to a physical printer from the code which formats something to be printed. Once the physical printer driver is tested it is unlikely to be changed anytime soon. That puts it into the not-likely-to-break category and is then exempt from unit tests. If it ever does change it is tested again by inspection of the physical output. The code that formats output destined for printing can be unit tested as usual.

Also remember that AcceptanceTests (XP style) cover anything that is of value to the customer. If it is not valuable enough to figure out how to test it then it should be removed immediately if not sooner. Where there is a will there is a way to test. -- *DonWells*

How does one test ArtificialIntelligence algorithms? I'm inclined to think they require human interaction, and they also tend to change often. -- YuriKhan

Creating ArtificialIntelligence algorithms like ArtificialNeuralNetwork or GeneticAlgorithm usually starts with creating a testset. This testset is often used in the process of creating the algorithm, for example for training a neural network or for the selection in genetic programming.

NumericalOptimization code is tricky to UnitTest exactly, but it is possible to UnitTest it heuristically. Has the accuracy of the learner increased over several training epochs? If not, maybe the LearningRate is too high. (This test may be too high-level to be considered a UnitTest, PerSe.) Finer-grained tests include asserting that analytic computations of gradients (used for BackProp in NeuralNetworks) is equal to numerical computations thereof, PlusOrMinus some epsilon. -- JosephTurian

---

I just came across this paper, which oozes XP-ness: GuerillaHci on usability testing. It's by JakobNielsen, author of UsabilityEngineering, and though he doesn't mention the word UnitTest (he talks of 'scenarios'), it is a kind of unit

testing for usability (of course, it's not automated; but his tests are not meant to be full acceptance tests). Interesting reading. -- BrianEwins

---

Has anyone tried a local Wiki to store unit tests in? -- ThaddeusOlczyk

---

I find it hard to write UnitTests where the unit being tested is heavily network-related, especially where it relies on components across the network to function properly. Does anyone have experience with UnitTestingNetworkFunctionality? -- GabrielWachob

I would suggest looking at MockObject to help here. Your code should be designed to separate the model from network interaction (perhaps through appropriate APIs). The model can be tested properly with a set of MockObjects implementing your networking API. Testing the networking API itself can be done without involving your model too. But then again, your particular problem might just be the exception to the rule. -- ChanningWalton

---

What if you have most of your code outside the classes? Like in PhpLanguage, with PhpUnit. -- DavidEscala

Maybe you are not programming object oriented? -- XaviAlbaladejo

Yes. In php there is almost always some code outside the classes. I know I ought to make everything ObjectOriented, but sometimes it is difficult to do so. -- DavidEscala

The key point of testing, of course, is that you get good enough test coverage, cheaply enough. This is why unit tests are good: you can test the same things with functional tests, but you can generally do it more easily and cheaply with unit tests. That is why the real trick is not to work harder on writing the tests, but to write code that is more easily testable. When you take this approach, I think you'll find your logic naturally migrating out of pages and into classes.

Should unit testing test the public interface of a class or the private implementation?

(Apologies if this is an old chestnut)

I'm coding in Java and I frequently find that I want to test private methods of my classes in isolation. I usually resort to giving the methods default access instead and testing from a test case in the same package. However, this smells and feels like a violation of encapsulation.

I've seen a number of commentators state that unit tests should all be done against the PublicInterface. However, frequently when I resort to testing the public interface, my tests become harder to construct and take on the feel of acceptance tests. This is perhaps because my objects are too big. However, if I were to break them down into smaller public classes to facilitate testing, aren't these additional public classes just exposing the implementation anyway?

Perhaps I'm really missing the point given Michael Hills comment above in **Are we testing too much?** about testing privates? I believe unit testing is about testing method implementations not public interfaces, but there doesn't seem an easy way to do this in languages like Java. Now I'm really missing that much slighted *friend declaration* from my C++ days!

-- MartinBayly

See ExtremeProgrammingTestingPrivateMethods, and MethodsShouldBePublic. You specifically do **not** want to test method implementations unless you are really desperate to put in some tests (perhaps when UnitTestingLegacyCode). If the implementation changes, the tests break; not fun. You don't want the tests to force you into a particular implementation. You want the tests to work as scaffolding, not a cage.

I've found that when I really get into testing, I end up refactoring the code into LotsOfShortMethods, most public or package, and most with their own tests. There are several ways to deal with this "interface pollution":

1. Ignore it
2. ReFactor a big class into many smaller classes and use delegation instead of having the class do everything on its own
3. Use package visibility rather than public
4. Expose your class through an interface that doesn't use all the methods

Actually, the first one isn't so bad during development because it gives you more opportunities to do refactorings such as MoveMethod. The second one is probably an all-around good thing to do anyway since it will help your code conform to the LawOfDemeter. There are probably other ways I haven't considered...

Recently I have been involved in lots of discussions about WritingTestableClasses. The essence of this is that often you morph your ClassUnderTest to facilitate good testing.

-- *MikePorter*

This is where ReFactoring ties in.

I think unit tests should test the interface: if the interface didn't change, how could it break the application?... invariants (ClassInvariant) are good for testing internals. -- PanuKalliokoski

---

I'm not sure if it's good practice, but it seems logical to do white-box testing by exercising package-private methods in the parallel test hierarchy. I have found myself adding package-private methods just for testing, it seems like an ideal solution to the problem you describe.

-- *Orbfish*

UnitTests for XML/XSLT

I am really stuck on testing XSLT (and CSS as well). Things in my world seem to be headed toward XML for data and XSLT for presentation and transformation (with possible use of CSS, XFORM, etc.).

It seems that future applications may consist of little more than XML, XSLT and maybe a little script glue.

Anyone care to make any suggestions on how to apply XP unit testing on this stuff? Object-oriented it ain't!

-- DanVokt

See XmlUnit, XsltUnit

For data transformers (filters) such as XSLT, a good unit testing method is often to drive the transformation once, check that it is what was intended, store the result, and run **diff -u** against the stored result and the current result of the transformation. -- PanuKalliokoski

There is another approach. It is possible to verify transformation logic of one template only: whether it is invoked via call-template or apply-templates. Some time ago I've created a small Java library allowing to call or apply transformations of one template and then to verify result of the transformation. Verification can be achieved via XPath. I did not have any time to publish this tool, or to continue its developing, so if anyone interested in it one could contact me. -- PavelSher

---

Are we confusing tradition with necessity?

Perhaps the distinction between an AcceptanceTest (nee "functional test") and a UnitTest is not what they test or how they're done, but in who they satisfy. The acceptance test makes the customer satisfied that the software provides the business value that makes them willing to pay for it. The unit test makes the programmer satisfied that the software does what the programmer thinks it does.

It could be that some people confuse traditions with the actual definition of the tests. One might say, for example, "A unit test is testing an individual class in isolation," or "An acceptance test tests the entire program." That would be wrong. They are not the definitions, just traditions resulting from the forces that act upon you when you are doing testing. There is absolutely nothing wrong with the programmer writing a so-called unit test that tests the whole program, or with the customer defining an acceptance test that stubs out part of the system. It's up to the people doing it to weigh the costs and benefits, not to be arbitrarily constrained to tradition.

However, some authors such as MartinFowler and MatthewFoemmel believe that "Unit tests are written by developers and typically test an individual class or small group of classes." [http://www.martinfowler.com/articles/continuousIntegration.html#N222] It may be that the authors are providing definitions or it may be that they are acknowledging a traditional understanding of the terms. In any case, MartinFowler would probably be the first to say do what works and not be constrained to traditions when they do not serve well.

[DoBothUnitAndAcceptanceTests.]

From the XP mailing list:

> I'm trying to introduce some XP practices where I work. People are resisting unit tests for a couple of reasons and I'm looking for help. One of the problems is that the tests are fragile in the face of XP's dynamic and evolving design. Someone goes to the effort of writing a bunch of unit tests, and then they realize that their design was wrong. To change the design they have to modify some or all of the tests. Typically they just comment out the tests. The feeling is that it would be easier to write the tests once the design has solidified a bit more. -- Chris

Chris, it sounds as if there are a few issues in your testing approach that could be fixed.

- **Is the testing framework awkward?** Most of the time, it should be easier to add a new test than to add the code that passes it.
- **Are we writing mini-functionals?** A common newbie mis-step is to write UnitTests which focus on an object's behavior in complex interactions with other objects. Focus on isolated behavior. Sometimes tests can be mini-functionals, but we can move with more confidence if we know that the collaborators are themselves thoroughly tested.
- **But we can't test this object without pulling in all those other objects.** This is often a smell. Objects should be tiny and stupid. That way we can write tests that are tiny and stupid. Bang at the object until you *can* test it in isolation. Or abandon the design that created that object. Investigate MockObjects and other strategies that help to put one object at a time under the microscope.
- **Are the tests looking too deep into the guts?** If our tests poke too deep into mechanisms, then every time we change the mechanism we have to change all the tests. The less you expose to the world, the fewer the unit tests you will write.
- **Are we testing too much?** The purpose of unit testing is to increase developer mobility. It is *not* to verify the correctness of your program. Are you testing privates? Are you testing simple accessors? Are you trying to make unit tests perform 100% code coverage? These are all practices that may have some value, but also may serve to defeat the real purpose of unit tests.

It's a wonderful humbling experience, UnitTests. I don't write code any other way anymore. My code has less problems, I have more confidence and management has more confidence. -- sg

---

Writing test often needs more time than implementing the features, but I think that's good because testing is the most important activity to create good software.

AnswerMe. Does anyone have experience unit testing Informatica mappings? -- EricHodges

Every day. I work there. - AnonymousDonor

I found that unit testing add a very nice immediate access to the code I'm currently working on. If I don't UnitTest then before testing a chunk of code I must run the whole app (wait 5s), load some data (1s to 1 minute) and walk across the application until a place I can exerce [?] my new code. When I UnitTest I just run the test.

I never see anyone notice this here but it happens to be a very nice improvement in my working process. Am I just unlucky working on application with heavy setup or does other people see this as an basic GoodSense improvement ?

I HaveThisPattern. The current behaviour I'm testing would take tedious setup and 10 to 15 minutes of running to get to without UnitTests. Not only would this be a boring pain in the ass, but it would be a constant flow breaker. Also, it's intricate enough that even with a written down test procedure, I'd probably never end up refactoring the code once it works; I'd not want to go through and manually test the code again, when it's not even fresh in my mind. Unfortunately, I'm still arranging the app to be easier to set up state to test other such situations. -- JoeWeaver

---

WayneMiller's question moved to ProgrammerTest

All runround..wondering.. What To Do Best and How ??

See

Specific tips on doing UnitTests:

DeveloperTestTechniques, XpTestFaq, UnitTestExamples, CodeUnitTestFirst, TestEverythingThatCouldPossiblyBreak, OptimizingUnitTests, UnitTestExamplesAndGuidelines, MockObject, UnitTestingErrors, UnitTestingCorba, FragileTests

Domain-specific advice:

UnitTestsAndDatabases, GuiUnitTesting, UnitTestingMarsOrbiters, UnitTestingLegacyCode, UnitTestsForSynchronizationLogic, TestPrintedOutput, UnitTestsForLibraries

Stop-gap measures for when you can't figure out how to test it properly:

DontChangeTheCodeTest, NoTestsYetTest, UntestableUnits

How does unit-testing change the way you think about programming?

UnitTestingIsDesign, ArguingThroughUnitTests, ResilienceVsAnticipation, ProgrammerTest

Semantics & renaming:

RenamingUnitTests, UnitTestsDefined

RefactoringUnitTests, UnitTestsDefined, XpVsStandardDefinitionOfUnitTest, ProgrammerTest

Others:

UnitTestingCostsBenefits, UnitTestTrial, SourceTest, QaIsNotQc, IsUnitTestingExtreme, AcceptanceTests, SeparateCodingAndTesting, PoorMansTestingFramework

New: LateOnsetTesting - writing tests for code that's already written

---

Here's a StupidQuestion: could someone define what a 'Unit' is in the lexicon of XP? I tend to think "compiler unit", or "integration unit", but I'm thinking that these units are more like at the level of individual methods. Is there any official definition?

The Nomenclature of Unit Tests http://smallwiki.unibe.ch/nomenclatureofunittests/ says a Unit is ... um ... nevermind.

---

http://www.testingfaqs.org/t-unit.html has a list of JavaUnitClones

---

I am starting a new web application where we will be using SSL certificates for authentication. I would like to use a web testing framework like HttpUnit or HtmlUnit to test that user Alice with certificate A gets the right privileges, user Bob with certificate B gets his different set of privileges, and user Sam (the Saboteur) with certificate S gets no privileges. The members of the HttpUnit mailing list told me that HttpUnit couldn't do that, and that I should roll my own using the Jakarta Commons HttpClient library, but there seems to be no way to explicitly say what binary certificates the client should offer the server.

Is there a tool I've missed? -- EricJablow

---

http://ojesunit.blogspot.com/ is a new way of writing unit test in php

---

Unit testing for our teams has become a proven tool and provides the developers a comfortable way in which they can reach the levels of desired quality. Especially on projects where we use test driven development for system as wel as integration tests, we see a high increase in commitment towards testing from the developers when applying unit testing to realize and fulfill requirements for different test stages. We currently succesfully apply this in agile as well as traditional project approaches. -- RonaldvanAken

---

What happens when a one or more functions are dependent on a function whose behavior is likely to change with time? i.e., a function for selecting tax rate based on income will probably be changed from year to year. Therefore, if another function computes the tax owed using this changing function, its unit tests will fail with every change. Worse yet, a function to compute the gross income, which uses both of these functions, will also have its unit tests break. -- mb

Consider that a CodeSmell. If your code isn't working for UnitTests, your code needs fixed. In this case, either give the tax-rate function an extra parameter (year), so you can simply add new UnitTests to handle it. Or consider use of a tax-rate ContextObject with a appropriate FunctorObjects configured for region and year.

Any ideas on how to test asynchronous code. For example I want to test the Lucene indexing & querying mechanism. And both in concerto. Should I change the production code to have hooks where I can put my testing code into? ~~~

This is the FAQ "How do I test { complexity | performance | concurrency | security | look-n-feel | awesomeness | etc }?"

The answer is you don't, because ordinary tests are for SoftwareEngineering, and exhaustively examining any of those topics is ComputerScience. For each of those things, you pick from the ComputerScience literature a good-enough algorithm. Then you use DeveloperTests to check that each detail of the algorithm is correctly implemented. (See http://broadcast.oreilly.com/2009/02/merb-mind-maps.html for an example implementing MinimumCostSpanningTree.)

For concurrency, a test case should configure a mock object to perform one way, then another, and you would assert that your production code dealt with the results correctly.

CategoryTesting

Last edit March 1, 2013, See github about remodeling.