Random data in Unit Tests?

Asked 16 years, 3 months ago Modified 2 months ago

Viewed 75k times

188

I have a coworker who writes unit tests for objects which fill their fields with random data. His reason is that it gives a wider range of testing, since it will test a lot of different values, whereas a normal test only uses a single static value.

I've given him a number of different reasons against this, the main ones being:

- random values means the test isn't truly repeatable (which also means that if the test can randomly fail, it can do so on the build server and break the build)
- if it's a random value and the test fails, we need to a) fix the object and b) force ourselves to test for that value every time, so we know it works, but since it's random we don't know what the value was

Another coworker added:

- If I am testing an exception, random values will not ensure that the test ends up in the expected state
- random data is used for flushing out a system and load testing, not for unit tests

Can anyone else add additional reasons I can give him to get him to stop doing this?

(Or alternately, is this an acceptable method of writing unit tests, and I and my other coworker are wrong?)

unit-testing tdd mocking

Share

Improve this question

Follow

edited Aug 29, 2008 at 18:15

travis
36.4k ● 21 ● 72 ● 97

asked Aug 28, 2008 at 14:45

Adam V
6,336 ● 3 ● 41 ● 52

48 "random values means the test isn't truly repeatable" not true, since the tets will be using pseudo-random numbers. Provide the same initial seed, get the same sequence of "random" tests. – Raedwald Sep 8, 2011 at 16:06

28 Anecdote: I once wrote a CSV export class, and random testing revealed a bug when control characters were placed at the end of a cell. Without random testing, I would not have ever thought to add that as a test case. Did it always fail? No. Is it a perfect test? No. Did it help me catch and fix a bug? Yes. – Tyzoid May 9, 2017 at 15:30

5 Tests can also serve as documentation, to explain when the code expects as input and what is expected as output. Having a test with clear arbitrary data can be simpler and more explanatory than code that generates random data. – splintor Oct 8, 2018 at 15:42

3 If your unit test fails because of a randomly generated value, and this value is not part of an assert, good luck with debugging your unit test. – eriksmith200 Nov 27, 2018 at 9:57

Small cost to random tests is that it can make it harder to refactor. When a random test fails in refactoring, you're not certain the bug is due to the refactoring. It also helps to debug execution of the same test payload on the master and refactoring branch on *different machines*. So random testing really needs a framework to easily export and import test payloads – Jake Stevens-Haas Jun 3, 2022 at 19:19

23 Answers

Sorted by: Highest score (default)

94

There's a compromise. Your coworker is actually onto something, but I think he's doing it wrong. I'm not sure that totally random testing is very useful, but it's certainly not invalid.

A program (or unit) specification is a hypothesis that there exists some program that meets it. The program itself is then evidence of that hypothesis. What unit testing ought to be is an attempt to provide counter-evidence to refute that the program works according to the spec.

Now, you can write the unit tests by hand, but it really is a mechanical task. It can be automated. All you have to do is write the spec, and a machine can generate lots and lots of unit tests that try to break your code.

I don't know what language you're using, but see here:

Java http://functionaljava.org/

Scala (or Java) http://github.com/rickynils/scalacheck

Haskell http://www.cs.chalmers.se/~rjmh/QuickCheck/

.NET:
http://blogs.msdn.com/dsyme/archive/2008/08/09/fscheck-0-2.aspx

These tools will take your well-formed spec as input and automatically generate as many unit tests as you want, with automatically generated data. They use "shrinking" strategies (which you can tweak) to find the simplest possible test case to break your code and to make sure it covers the edge cases well.

Happy testing!

Share Improve this answer Follow

edited Jan 28, 2013 at 16:37
RobertB
4,602 ● 1 ● 32 ● 29

answered Aug 28, 2008 at 15:46
Apocalisp
35k ● 8 ● 108 ● 158

3 +1 to this. ScalaCheck does a phenomenal job of generating minimized, random test data in a repeatable way.
– Daniel Spiewak Sep 18, 2008 at 1:20

25 It's not random. It's arbitrary. Big difference :) – Apocalisp Sep 18, 2008 at 5:10

reductiotest.org now longer seems to exist, and Google did not point me anywhere else. Any idea where it is now? – Raedwald Sep 8, 2011 at 16:18

It's now part of the Functional Java library. Link edited. But I would just use Scalacheck to test Java code. – Apocalisp Sep 9, 2011 at 8:21

1 @Apocalisp This is how one does testing of digital hardware while simulating it. The semiconductor industry term is 'constrained random verification'. – Tudor Timi Sep 13, 2017 at 17:20

59

This kind of testing is called a Monkey test. When done right, it can smoke out bugs from the really dark corners.

To address your concerns about reproducibility: the right way to approach this, is to record the failed test entries, generate a unit test, which probes for the *entire family* of the specific bug; and include in the unit test the one specific input (from the random data) which caused the initial failure.

Share Improve this answer Follow

answered Sep 18, 2008 at 1:21

Silver Dragon
5,550 ● 7 ● 44 ● 73

it's also possible to print the seed that was used for the random data generator each time the test fail. It will be therefore possible to reproduce them using the same seed in your home env. – ganqqwerty May 13, 2022 at 14:35

31

There is a half-way house here which has some use, which is to seed your PRNG with a constant. That allows you to generate 'random' data which is repeatable.

Personally I do think there are places where (constant) random data is useful in testing - after you think you've done all your carefully-thought-out corners, using stimuli from a PRNG can sometimes find other things.

Share Improve this answer Follow

answered Aug 28, 2008 at 15:09
Will Dean
39.5k ● 11 ● 92 ● 118

4 I have seen this work well on a system that had lots of locking and threads. The 'random' seed was writen to a file on each run, then if a run failed, we could work out the path the code took and write a hand written unit test for that case we had missed. – Ian Ringrose Mar 13, 2009 at 12:48

What does PRNG stand for? – Geoffrey Dec 6, 2017 at 14:15

Pseudo Random Number Generator – Will Dean Dec 6, 2017 at 23:10

28

I am in favor of random tests, and I write them. However, whether they are appropriate in a particular build environment and which test suites they should be included in is a more nuanced question.

Run locally (e.g., overnight on your dev box) randomized tests have found bugs both obvious and obscure. The obscure ones are arcane enough that I think random testing was really the only realistic one to flush them out. As a test, I took one tough-to-find bug discovered via randomized testing and had a half dozen crack developers review the function (about a dozen lines of code) where it occurred. None were able to detect it.

Many of your arguments against randomized data are flavors of "the test isn't reproducible". However, a well written randomized test will capture the seed used to start the randomized seed and output it on failure. In addition to allowing you to repeat the test by hand, this allows you to trivially create new test which test the specific issue by hardcoding the seed for that test. Of course, it's probably nicer to hand-code an explicit test covering that case, but laziness has its virtues, and this even allows you to essentially auto-generate new test cases from a failing seed.

The one point you make that I can't debate, however, is that it breaks the build systems. Most build and continuous integration tests expect the tests to do the same thing, every time. So a test that randomly fails will create chaos, randomly failing and pointing the fingers at changes that were harmless.

A solution then, is to still run your randomized tests as part of the build and CI tests, but *run it with a fixed seed, for a fixed number of iterations*. Hence the test always does the same thing, but still explores a bunch of the input space (if you run it for multiple iterations).

Locally, e.g., when changing the concerned class, you are free to run it for more iterations or with other seeds. If randomized testing ever becomes more popular, you could even imagine a specific suite of tests which are known to be random, which could be run with different seeds (hence with increasing coverage over time), and where failures wouldn't mean the same thing as deterministic CI systems (i.e., runs aren't associated 1:1 with code changes and so you don't point a finger at a particular change when things fail).

There is a lot to be said for randomized tests, especially well written ones, so don't be too quick to dismiss them!

Share Improve this answer Follow

answered Jun 16, 2016 at 0:51
BeeOnRope
64.6k ● 18 ● 235 ● 432

Hey, I'm wondering if you can share an example where randomized tests found a bug? I've read this argument frequently but somehow cannot imagine :) Usually, I'm trying to write easy and obvious code and cannot come up with an example where I could benefit from a randomized test. Thanks! – kon Feb 9, 2022 at 17:51

23

If you are doing TDD then I would argue that random data is an excellent approach. If your test is written with constants, then you can only guarantee your code works for the specific value. If your test is randomly failing the build server there is likely a problem with how the test was written.

Random data will help ensure any future refactoring will not rely on a magic constant. After all, if your tests are your documentation, then doesn't the presence of constants imply it only needs to work for those constants?

I am exaggerating however I prefer to inject random data into my test as a sign that "the value of this variable should not affect the outcome of this test".

I will say though that if you use a random variable then fork your test based on that variable, then that is a smell.

Share Improve this answer Follow

answered Jun 14, 2013 at 14:39

Jimmy Bosse
1,404 ● 1 ● 13 ● 24

+1 for mentioning not relying on magic constants. Happens too easily when developers don't understand the code and just rely on tests passing meaning it works really.
– Eljas Hyyrynen Jul 14, 2023 at 12:42

20

In the book Beautiful Code, there is a chapter called "Beautiful Tests", where he goes through a testing strategy for the Binary Search algorithm. One paragraph is called "Random Acts of Testing", in which he creates random arrays to thoroughly test the algorithm. You can read some of this online at Google Books, page 95, but it's a great book worth having.

So basically this just shows that generating random data for testing is a viable option.

Share Improve this answer

Follow

answered Aug 28, 2008 at 15:12

mreggen
3,389 ● 2 ● 28 ● 25

13

Your co-worker is doing fuzz-testing, although he doesn't know about it. They are especially valuable in server systems.

Share Improve this answer

Follow

answered Mar 13, 2009 at 5:46

Robert Gould
69.7k ● 61 ● 191 ● 275

2 but isn't this a fundamentally different thing from unit tests? and done at a different time? – endolith Jun 11, 2014 at 1:22

6 @endolith there is no law of physics forcing you to run particular tests at particular times – Criticize SE actions means ban Jan 6, 2015 at 6:06

2 @immibis But there are good reasons to do particular tests at particular times. You don't run a battery of unit tests every time a user clicks an "Ok" button. – endolith Jan 6, 2015 at 15:21

2 @user253751 I think we should print some t-shirts with your quote! :) – Kirby Sep 8, 2020 at 15:24

13

One advantage for someone looking at the tests is that arbitrary data is clearly not important. I've seen too many tests that involved dozens of pieces of data and it can be difficult to tell what needs to be that way and what just happens to be that way. E.g. If an address validation method is tested with a specific zip code and all other data is random then you can be pretty sure the zip code is the only important part.

Share Improve this answer Follow

answered Dec 20, 2010 at 17:10
Trystan Spangler
1,769 ● 12 ● 20

10

> - if it's a random value and the test fails, we need to a) fix the object and b) force ourselves to test for that value every time, so we know it works, but since it's random we don't know what the value was

If your test case does not accurately record what it is testing, perhaps you need to recode the test case. I always want to have logs that I can refer back to for test cases so that I know exactly what caused it to fail whether using static or random data.

Share Improve this answer Follow

edited Sep 18, 2008 at 1:10
John Millikin
201k ● 41 ● 215 ● 227

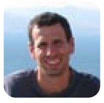
answered Aug 28, 2008 at 14:57
EBGreen
37.6k ● 12 ● 68 ● 86

5

You should ask yourselves what is the goal of your test.
Unit tests are about verifying logic, code flow and object interactions. Using random values tries to achieve a different goal, thus reduces test focus and simplicity. It is acceptable for readability reasons (generating UUID, ids, keys,etc.).
Specifically for unit tests, I cannot recall even once this method was successful finding problems. But I have seen many determinism problems (in the tests) trying to be clever with random values and mainly with random dates.
Fuzz testing is a valid approach for **integration tests** and **end-to-end tests**.

Share Improve this answer Follow

answered Oct 5, 2014 at 22:11



Ohad Bruker
524 ● 6 ● 10

I would add that using random input for fuzzing is a poor substitute for coverage-guided fuzzing when it is possible.
– gobenji Apr 23, 2020 at 3:10

4

Can you generate some random data once (I mean exactly once, not once per test run), then use it in all tests thereafter?

I can definitely see the value in creating random data to test those cases that you haven't thought of, but you're right, having unit tests that can randomly pass or fail is a bad thing.

Share Improve this answer
Follow

answered Aug 28, 2008 at 14:55

Tim Frey
9,931 ● 9 ● 45 ● 61

3

If you're using random input for your tests you need to log the inputs so you can see what the values are. This way if there is some edge case you come across, you *can* write the test to reproduce it. I've heard the same reasons from people for not using random input, but once you have insight into the actual values used for a particular test run then it isn't as much of an issue.

The notion of "arbitrary" data is also very useful as a way of signifying something that is *not* important. We have some acceptance tests that come to mind where there is a lot of noise data that is no relevance to the test at hand.

Share Improve this answer
Follow

answered Oct 9, 2008 at 3:26

craigb
16.9k ● 7 ● 53 ● 63

3

I think the problem here is that the purpose of unit tests is not catching bugs. The purpose is being able to change the code without breaking it, so how are you going to know that you break your code when your random unit tests are green in your pipeline, just because they doesn't touch the right path? Doing this is insane for me. A different situation could be running them as integration tests or e2e not as a part of the build, and just for some specific things because in some situations you will need a mirror of your code in your asserts to test that way. And having a test suite as complex as your real code is like not having tests at all because who is going to test your suite then? :p

Share Improve this answer Follow

edited Feb 26, 2021 at 0:59

answered Feb 26, 2021 at 0:54

L Leonardo Mangano

141 ● 1 ● 3

3

A unit test is there to ensure the correct behaviour in response to particular inputs, in particular all code paths/logic should be covered. There is no need to use random data to achieve this. If you don't have 100% code coverage with your unit tests, then fuzz testing by the back door is not going to achieve this, and it may even mean you occasionally don't achieve your desired code coverage. It may (pardon the pun) give you a 'fuzzy' feeling that you're getting to more code paths, but there may not be much science behind this. People often check code coverage when they run their unit tests for the first time and then forget about it (unless enforced by CI), so do you really want to be checking coverage against every run as a result of using random input data? It's just another thing to potentially neglect.

Also, programmers tend to take the easy path, and they make mistakes. They make just as many mistakes in unit tests as they do in the code under test. It's way too easy for someone to introduce random data, and then tailor the asserts to the output order in a single run. Admit it, we've all done this. When the data changes the order can change and the asserts fail, so a portion of the executions fail. This portion needn't be 1/2 I've seen exactly this result in failures 10% of the time. It takes a long time to track down problems like this, and if your CI doesn't record enough data about enough of the runs, then it can be even worse.

Whilst there's an argument for saying "just don't make these mistakes", in a typical commercial programming setup there'll be a mix of abilities, sometimes relatively junior people reviewing code for other junior people. You can write literally dozens more tests in the time it takes to debug one non-deterministic test and fix it, so make sure you don't have any. Don't use random data.

Share Improve this answer Follow

answered Jan 20, 2022 at 13:15

Keeely
1,005 ● 11 ● 22

2

In my experience unit tests and randomized tests should be separated. Unit tests serve to give a certainty of the correctness of some cases, not only to catch obscure bugs. All that said, randomized testing is useful and should be done, separately from unit tests, but it should test a series of randomized values. I can't help to think that testing 1 random value with every run is just not enough, neither to be a sufficient randomized test, neither to be a truly useful unit test.

Another aspect is validating the test results. If you have random inputs, you have to calculate the expected output for it inside the test. This will at some level duplicate the tested logic, making the test only a mirror of the tested code itself. This will not sufficiently test the code, since the test might contain the same errors the original code does.

Share Improve this answer Follow

answered Jan 25, 2022 at 9:29

maybesomename
121 ● 1 ● 11

2

This is an old question, but I wanted to mention a library I created that generates objects filled with random data. It supports reproducing the same data if a test fails by supplying a seed. It also supports JUnit 5 via an extension.

Example usage:

```
Person person = Instancio.create(Person.class);
```

Or a builder API for customising generation parameters:

```
Person person = Instancio.of(Person.class)
    .generate(field("age"), gen -> gen.ints.min(18).ma
    .create();
```

Github link has more examples:
https://github.com/instancio/instancio

You can find the library on maven central:

```
<dependency>
    <groupId>org.instancio</groupId>
    <artifactId>instancio-junit</artifactId>
    <version>LATEST</version>
</dependency>
```

Share Improve this answer Follow

answered Apr 6, 2022 at 6:45

armandino
18.4k ● 17 ● 73 ● 85

1

You didn't mention what tech stack you use. It's called fuzzing, sometimes you can also find it under old name 'property testing', and there's lots of good stuff for the task:

- Java / Kotlin/JVM: Instancio, Jazzer
- Kotlin/Multiplatform: kotest-property
- Python: Hypothesis, Atheris, or combination of them
- Rust: libfuzzer-sys, arbitrary, or combination of them
- C/C++/LLVM/WASM: libfuzzer

All those tools can be configured to generate only one piece of data, or to run in a loop. There are also domainspecific libraries (e.g. for random dates, strings, SSNs, etc), but they are rather too limited for most apps.

Share Improve this answer Follow

answered Sep 5 at 11:05

iirekm
9,368 ● 5 ● 39 ● 47

0

Depending on your object/app, random data would have a place in load testing. I think more important would be to use data that explicitly tests the boundary conditions of the data.

Share Improve this answer Follow

answered Aug 28, 2008 at 14:51

EBGreen
37.6k ● 12 ● 68 ● 86

0

We just ran into this today. I wanted **pseudo-random** (so it would look like compressed audio data in terms of size). I TODO'd that I also wanted **deterministic**. rand() was different on OSX than on Linux. And unless I re-seeded, it could change at any time. So we changed it to be deterministic but still psuedo-random: the test is repeatable, as much as using canned data (but more conveniently written).

This was **NOT** testing by some random brute force through code paths. That's the difference: still deterministic, still repeatable, still using data that looks like real input to run a set of interesting checks on edge cases in complex logic. Still unit tests.

Does that still qualify is random? Let's talk over beer. :-)

Share Improve this answer Follow

answered Sep 18, 2008 at 1:15

Tim James

1,603 ● 1 ● 15 ● 15

0

I can envisage three solutions to the test data problem:

- Test with fixed data
- Test with random data
- Generate random data *once*, then use it as your fixed data

I would recommend doing *all of the above*. That is, write repeatable unit tests with both some edge cases worked out using your brain, and some randomised data which you generate only once. Then write a set of randomised tests that you run *as well*.

The randomised tests should never be expected to catch something your repeatable tests miss. You should aim to cover everything with repeatable tests, and consider the randomised tests a bonus. If they find something, it should be something that you couldn't have reasonably predicted; a real oddball.

Share Improve this answer

answered Oct 26, 2011 at 16:32

Follow

Tom W
5,393 ● 4 ● 31 ● 58

0

I routinely use random values in *specific types* of tests to indicate that the method should support any arbitrary value of the expected type. Take this simple `CreateCustomer` method as example:

```
public Customer CreateCustomer(string name)
{
    var customer =
_customerRepository.InsertCustomer(
        name: name);
    _messageBus.Publish(new CustomerCreated(
        customerId: customer.CustomerId,
        name: customer.Name));
    return customer;
}
```

No functionality differs based on the value of the `name` parameter, but it is important to verify that the same value gets passed to the repository, the message bus, and is set on the returned customer object. Similarly, it's important to verify that the customer ID returned from the repository is passed to the message bus and is set on the returned customer object.

I could choose some constant arbitrary values to use in my test, i.e.: `name` => "Customer Name", `CustomerId` => 13. But, that test could be satisfied with a generally useless, hardcoded implementation, i.e.:

```
public Customer CreateCustomer(string name)
{
    var customer =
_customerRepository.InsertCustomer(
        name: "Customer Name");
    _messageBus.Publish(new CustomerCreated(
        customerId: 13,
        name: "Customer Name"));
    return new Customer(
        customerId: 13,
        name: "Customer Name"));
}
```

So instead I would use randomly generated name and customer ID values in my test, with a dynamic seed for the random to ensure it's *not* just using the same values in every test run. Something like this:

```
[TestMethod]
public void CreateCustomer()
{
    var random = new
Random(Guid.NewGuid().GetHashCode());
    var name = random.NextDouble().ToString();

    var customerId = random.Next();
    var repositoryCustomerName =
random.NextDouble().ToString();
    var customerRepository = new
Mock<ICustomerRepository>(MockBehavior.Strict);
    customerRepository
        .Setup(r => r.InsertCustomer(name))
        .Returns(new Customer(
            customerId: customerId,
            name: repositoryCustomerName))
        .Verifiable();

    var messageBus = new Mock<IMessageBus>
(MockBehavior.Strict);
    messageBus
```

```
        .Setup(b =>
b.Publish(It.Is<CustomerCreated>(c =>
            c.CustomerId == customerId
            && c.Name == repositoryCustomerName)))
        .Verifiable();

    var sut = new CustomerService(
        customerRepository:
customerRepository.Object,
        messageBus: messageBus.Object);

    var customer = sut.CreateCustomer(name: name);

    customerRepository.Verify();
    messageBus.Verify();
```

Share Improve this answer Follow

answered Oct 5, 2023 at 22:10

Jack Olsen
1 ● 2

0

Unit testing VS fuzzing

Random data can (and should) be used to test the resistance or the resilience of the software, and fuzzytests are the perfect place for randomly generated input data.

Unit tests, on the other hand, are definitely **not** the place to put randomness in.

From wikipedia

Unit tests

> unit testing [...] is a form of software testing by which isolated source code is tested to validate expected behavior

Fuzzing

> *fuzzing* or *fuzz testing* is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program

Let me make an example: you cannot test non-regression with randomly generated data. E.G.: imagine that someone find a bug in your software, then you write a patch and a non-regression test to check that your patch is not reverted.

This can hardly be done with randomic data.

Instead you can use fuzzing *in addition* to unit testing in your pipeline to production

Share Improve this answer

Follow

answered Sep 26 at 16:34

Marco Carlo Moriggi

384 ● 5 ● 13

▲

-1

How can your guy run the test again when it has failed to see if he has fixed it? I.e. he loses repeatability of tests.

While I think there is probably some value in flinging a load of random data at tests, as mentioned in other replies it falls more under the heading of load testing than anything else. It is pretty much a "testing-by-hope" practice. I think that, in reality, your guy is simply not thinkng about what he is trying to test, and making up for that lack of thought by hoping randomness will eventually trap some mysterious error.

So the argument I would use with him is that he is being lazy. Or, to put it another way, if he doesn't take the time to understand what he is trying to test it probably shows he doesn't really understand the code he is writing.

Share Improve this answer Follow

edited Aug 28, 2008 at 22:55

answered Aug 28, 2008 at 22:02

Greg Whitfield

5,719 ● 2 ● 31 ● 32

4 It's possible to log the random data or random seed so that the test can be reproduced. – cbp May 29, 2009 at 6:06

in addition to logging the random numbers, generally a test that fails with random inputs, even if the random inputs are not logged, that test can be re-run in a `while(true)` loop and will most likely fail in the time that it takes to check your snapchat feed. – Kirby Sep 8, 2020 at 15:16