

The Problem(s) With (standard) Unit Testing

Other ways of ensuring software correctness

Ricky Elrod
Red Hat, Inc.
August 7, 2014

What if I said to you:

**“You can create provably
bug-free programs.”**

Would you believe me?

**I am a person with an interest
in improving the state of our
industry.**

**I am a person with an interest
in improving the state of our
industry. Because it needs it. :-)**

But what are our common goals?

To create a working initial product as quickly as possible

To be able to maintain a working product as time goes on


My goal today

To show you some new (better) ways of obtaining these goals

To spark some interest in improving currently-used methodologies

ABRT reports on my laptop.

Yes - most of these are about the **kernel!**

My (39) System (2)

kernel Kerneloops	Monday 14
python Python	Last week 1
kernel Kerneloops	2 weeks ago 4
kernel Kerneloops	2 weeks ago 10
kernel Kerneloops	3 weeks ago 6
kernel Kerneloops	Last month 1
kernel Kerneloops	Last month 1
kernel Kerneloops	Last month 3
kernel Kerneloops	Last month 1
kernel Kerneloops	Last month 1
kernel Kerneloops	Last month

Automatic Bug Reporting Tool

Delete

kernel crashed

WARNING: CPU: 2 PID: 31288 at drivers/gpu/drm/i915/intel,

Name kernel

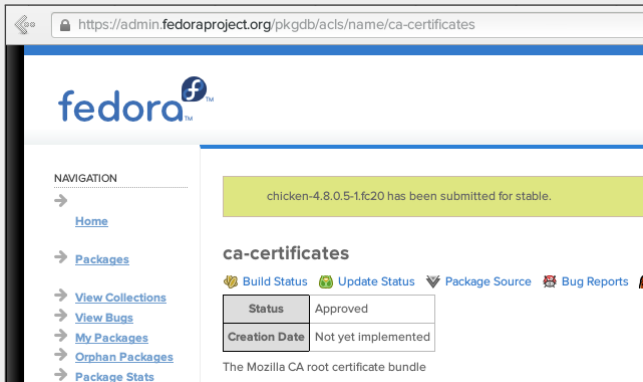
Version N/A

Detected 04/29/2014

Reported cannot be reported

A kernel problem occurred, but your kernel has been tainted (flags:GW). Kernel maintainers are unable to diagnose tainted reports.

Pkgdb1 shows a Bodhi flash message.



The screenshot shows the Fedora Package Database (pkgdb) admin interface. The browser address bar displays the URL: `https://admin.fedoraproject.org/pkgdb/acls/name/ca-certificates`. The page features the Fedora logo and a navigation sidebar on the left with the following links: Home, Packages, View Collections, View Bugs, My Packages, Orphan Packages, and Package Stats.

A yellow flash message at the top of the main content area states: "chicken-4.8.0.5-1.fc20 has been submitted for stable."

The main content area is titled "ca-certificates" and includes several utility links: Build Status, Update Status, Package Source, and Bug Reports. Below these links is a table with the following data:

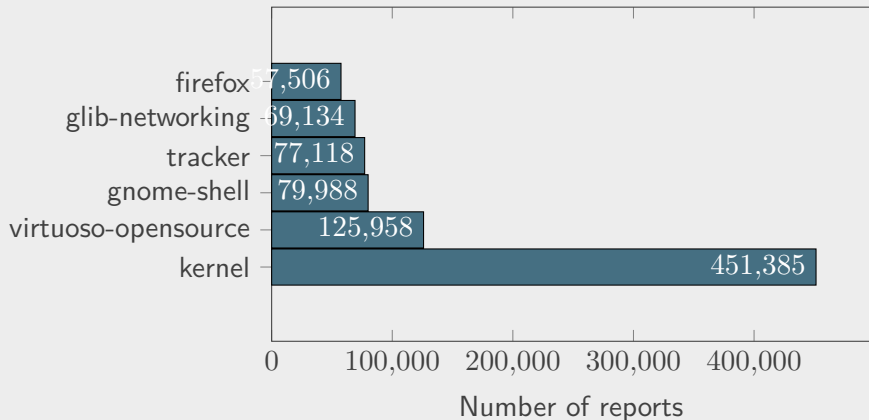
Status	Approved
Creation Date	Not yet implemented

Below the table, the text reads: "The Mozilla CA root certificate bundle"

FAS shows a 500 error after my session timed out.



Packages which have produced the most reports in F20 ...over the past year.



Fedora 20 users have reported almost 1.6 million crashes over the past year.

Let's pretend for a minute that unit tests are the solution.
...it might help to at least **RUN** them.

```
$ grep -c '%check' gnome-shell.spec  
0
```

Let's pretend for a minute that unit tests are the solution.
...it might help to at least **RUN** them.

```
$ grep -A 3 '%check' tracker.spec
%check
desktop-file-validate %{buildroot}%{_datadir}/
    applications/tracker-*.desktop

%post -p /sbin/ldconfig
```

Let's pretend for a minute that unit tests are the solution.
...it might help to at least **RUN** them.

```
$ grep -c '%check' glib-networking.spec  
0
```

Let's pretend for a minute that unit tests are the solution.
...it might help to at least **RUN** them.

```
$ grep -c '%check' firefox.spec  
0
```



Before we continue...

I am going to be using mostly **Java-style** syntax

If you don't understand something **please** ask

Other languages/syntax may be used for emphasis/clarity

Nothing in this talk is specific to any particular language

Motivation

You're browsing through some code and open a file and see nothing but:

```
int add10(int n);
```

You think to yourself, “What might this function do?”

You go off to write some unit tests to try to confirm your suspicion.

Motivation

```
int add10(int n);
```

```
test(  
    "The add10 function applied to 5 should return 15",  
    add10(5) == 15);
```

```
test(  
    "The add10 function applied to 20 should return 30",  
    add10(20) == 30);
```

```
test(  
    "The add10 function applied to 3000 should return  
    3010",  
    add10(3000) == 3010);
```

Motivation

```
[info] Total for specification Add10Spec
[info] Finished in 28 ms
[info] 3 examples, 0 failure, 0 error
[info] Passed: : Total 3, Failed 0, Errors 0, Passed 3,
    Skipped 0
[success] Total time: 2 s, completed May 19, 2014
    3:41:37 AM
All tests passed.
```

Motivation

You dig a bit deeper into the code and find a file that implements the function:

```
int add10(int n) {  
    if (n < 10000000)  
        return (n + 10);  
    else  
        return (n / 2);  
}
```

But... You should have added more tests!

What are the most common reasons for unit testing?

To *find problems early*

To *prevent regressions*

Ask yourself: How well do they actually do that?

Traditional unit testing only works when you remember to test things.

It is very easy to forget test cases - especially edge cases.

Adding tests is laborious and boring.

Adding tests can account for up to 50 percent of the cost of software development! [1]

[1] [Claessen and Hughes, 2000] - (<http://dl.acm.org/citation.cfm?id=351266>)

Traditional unit testing has other problems, too.

Unit tests tend to re-enforce biases rather than expose bugs.

We are bad at trying to pick random values to test with.

We often overlook and forget about obvious edge cases.

Despite these downfalls, we rely on them instead of reasoning about our code.

Tests re-inforce confidence in our belief that we are being responsible programmers.

But, by relying on them, we are not.

But what other options do we have?

A brief agenda for the remainder of the talk

Parametricity

Ongoing research: Dependent types

Property-based testing

Code review

Types!

“But by far the most popular and best established lightweight formal methods are type systems [...]”

-Benjamin Pierce (TaPL)

“A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.”

-Benjamin Pierce (TaPL)

Types!

“But by far the most popular and best established lightweight **formal methods** are type systems [...]” -Benjamin Pierce (TaPL)

“A type system is a tractable syntactic method for **proving** the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.” -Benjamin Pierce (TaPL)



Programming Is Not Math

When I learned to program, back when dinosaurs walked the earth and the internet had no cats, there was an idea: much, no one steered me towards programming. I came to it accidentally, in college, when I took an elective progra

So my first programming language was Fortran, an abbreviation of "Formula Translation." As you might expect from the diagram below.

Since I'm not a Fortran expert, I'll just show you a simple example of a Fortran program that calculates the area of a circle.

Sarah Hill



BULL-only

Programming Is No Mat

When I learned that the first man to walk the earth and the internet had no cats, there was an idea. Much, no one would have thought of programming. I came to it accidentally, in college, when I took an elective program

So my first programming language was `CTT225`, an abbreviation of "Formula Translation." As you might expect from the diagram below,

Programming is **EXACTLY** math

A quick look at Curry-Howard

The *Curry-Howard isomorphism* says that **programs** directly correspond to **proofs**.

Types are *propositions*, programs are *proofs* of them.

(This extends even further into math (c.f. Curry-Howard-Lambek) via category theory.)

Programming is EXACTLY math

A quick look at Curry-Howard

Logic / Math	Programming
universal quantification	generalized function space
existential quantification	generalised cartesian product
implication	function type
conjunction	product type
disjunction	sum type
true formula	unit type
false formula	bottom type

But I digress...

Kind of.

Parametricity:

Types as documentation

What is Parametricity?

Theorems for free!

Phillip Wadler [1] says:

Write down the definition of a polymorphic function on a piece of paper. Tell me its type, but be careful not to let me see the function's definition. I will tell you a theorem that the function satisfies.

The purpose of this paper is to explain the trick.

[1] [Wadler, 1989] - <http://ttic.uchicago.edu/~dreyer/course/papers/wadler.pdf>

What is Parametricity?

An (non)example

```
int doSomething1(int n);
```

We can state a lot about what this function does *not* do.

e.g., it will never return a string or a floating point number.

But there are also an enormous amount of possibilities of things that it *can* do.

What is Parametricity?

Another (non)example

```
ArrayList<int> doSomething(ArrayList<int> xs);
```

Does it double all the numbers in the list?

Does it add them up and order the list depending on if the sum is even?

Who knows?!

What is Parametricity?

A real example

```
<A> ArrayList<A> doSomething3(ArrayList<A> xs);
```

Now, the function is polymorphic.

I can tell you a lot about how this function works.

I can do this with a lot of confidence.

I can also tell you a lot about what it does not do.

Readability and comprehensibility of the code has increased.

Theorem

```
<A> List<A> doSomething3(ArrayList<A> xs);
```

Every element in the output list was also in the input list.

Contraposed Theorem

```
<A> List<A> doSomething3(ArrayList<A> xs);
```

If an element does not appear in the input list, it also does not appear in the output list.

*It would not have compiled otherwise!**

* Well, kind of.

Well...

Except for...*

Type-casing (e.g. Python `is` keyword)

Type-casting

Type-value (e.g. Python `type` keyword)

Unsafe universal functions (e.g. Python `str`, `int`, etc)

Side effects

null (Python `None`)

Exceptions

Infinite recursion

* list taken from Tony Morris's DjangoCon slide deck, via CC-BY.

Type system holes

Dealing with logical inconsistencies

We have a name for these logical inconsistencies.

They are called “bottom types” and denoted \perp in literature.

They make life annoying - but we can deal with it!

Some non-total languages don't even have most of these!

~~Type-casing (e.g. Python `is` keyword)~~

~~Type-casting~~

~~Type-value (e.g. Python `type` keyword)~~

~~Unsafe universal functions (e.g. Python `str`, `int`, etc)~~

~~Side-effects~~

~~null (Python `None`)~~

~~Exceptions~~

~~Infinite recursion~~

Fast and Loose Reasoning [1]

It's morally correct!

Functional programmers often reason about programs as if they were written in a total language, expecting the results to carry over to non-total (partial) languages. We justify such reasoning.

[1] “Fast And Loose Reasoning is Morally Correct”, 2006, Danielsson, Hughes, Jansson, Gibbons.

Theorem

```
<A> List<A> doSomething3(ArrayList<A> xs);
```

Every element in the output list was also in the input list.
With Fast and Loose Reasoning in mind, can you disprove this?

So what?

So we have free documentation, and free theorems. So what?

How does this help us know what the function *does*?

We write unit tests.

But not the kind you're thinking of.

Property based testing:

Writing tests that actually find bugs

Property-based testing

Writing tests first forces you to think about the problem you're solving. Writing property-based tests forces you to think way harder. – Jessica Kerr (@jessitron) April 25, 2013

Property-based testing

What is it?

Make statements about the result of your code based on input.

Statements are in the form of “for all input X: Y is true.”

Verify these statements for many different (randomized) inputs.

Property-based testing

An example

```
prop_rev_rev :: Eq a => [a] -> Bool
prop_rev_rev xs = reverse (reverse xs) == xs
```

```
> quickCheck prop_rev_rev
+++ OK, passed 100 tests.
```

Property-based testing

Another example

```
prop_intMult_assoc :: Int -> Int -> Int -> Bool
prop_intMult_assoc x y z = (x * y) * z == x * (y * z)
```

```
> quickCheck prop_intMult_assoc
+++ OK, passed 100 tests.
```

Property-based testing

Yet another example

```
prop_floatMult_assoc :: Float -> Float -> Float -> Bool
prop_floatMult_assoc x y z = (x * y) * z == x * (y * z)
```

```
> quickCheck prop_floatMult_assoc
*** Failed! Falsifiable (after 9 tests and 173 shrinks):
1.0e-45
1.00756055e-2
49.624866
```


Property-based testing in other languages

QuickCheck(-alikes) are everywhere!

Python: qc, paycheck, pytest-quickcheck, pyqcy

Ruby: rantly

Java: <https://bitbucket.org/blob79/quickcheck>

Scala: ScalaCheck

etc.

Property-based testing in Python

```
from pyqcy import qc, int_, float_, list_, main

@qc
def prop_rev_rev(xs=list_(of=int_(min=0, max=1024),
                          min_length=16,
                          max_length=2048)):
    assert xs[::-1][::-1] == xs

if __name__ == '__main__':
    main()
```

```
$ python tests.py
prop_rev_rev: passed 100 tests.
```

Property-based testing in Python

```
from pyqcy import qc, int_, float_, list_, main

@qc
def prop_intMult_assoc(a=int_(), b=int_(), c=int_()):
    assert (a * b) * c == a * (b * c)

if __name__ == '__main__':
    main()
```

```
$ python tests.py
prop_intMult_assoc: passed 100 tests.
```

Property-based testing in Python

```
from pyqcy import qc, int_, float_, list_, main

@qc
def prop_floatMult_assoc(a=float_(), b=float_(), c=
    float_()):
    assert (a * b) * c == a * (b * c)

if __name__ == '__main__':
    main()
```

```
$ python tests.py
prop_floatMult_assoc: failed (only 64 out of 100 tests
    passed).
```

```
Failure encountered for data:
```

```
    a = -5.961979798923219e+18
```

```
    c = 6.869588745645083e+18
```

```
    b = -7.772808141146458e+18
```

```
Exception:
```

```
...
```

An active research topic:

The crazy (awesome) world of
dependent types

Dependent Types

A very quick overview of an active research topic

A **dependent type** is a type that **depends** on a **value**.

Dependently typed languages e.g. **Agda**, **Coq**, and **Idris**.

(These three are packaged in Fedora!)

Dependent Types

A boring, canonical example — Vector length

```
total
append : Vect n a -> Vect m a -> Vect (n + m) a
append Nil      ys = ys
append (x :: xs) ys = x :: append xs ys
```

The function terminates (i.e., is total).

The output **MUST** have a length of the sum of the two inputs.

It wouldn't have compiled otherwise.

Dependent Types

Type-safe printf:

<https://www.youtube.com/watch?v=fVBck2Zngjo>

```
*Printf> :type printf
Printf.printf
  : (s : String) -> interpFormat (formatString s)
```


Dependent Types

Type-safe printf:

<https://www.youtube.com/watch?v=fVBck2Zngjo>

```
*Printf> printf "%s %d" "a string" "whoops"  
(input):1:8:When elaborating an application of  
  constructor __infer:  
    Can't convert  
      String  
  with  
    Int
```

Dependent Types

Type-safe printf:

<https://www.youtube.com/watch?v=fVBck2Zngjo>

```
*Printf> printf "%s %d" "a string" 123 "too many!"
(input):1:8:When elaborating an application of
  constructor __infer:
    printf "%s %d"
      "a string"
      (fromInteger 123) does not have a
        function type (String)
```

Dependent Types

Type-safe printf:

<https://www.youtube.com/watch?v=fVBck2Zngjo>

```
*Printf> printf "%s %d" "not enough"  
\i => prim__concat "not enough " (prim__toStrInt i)  
      : Int -> String
```

Dependent Types

Type-safe printf:

<https://www.youtube.com/watch?v=fVBck2Zngjo>

```
*Printf> printf "%s %d" "a string" 123  
"a string 123" : String
```

Code Review

Code Review

Useful for fleshing out ideas.

Useful for fleshing out implementation details.

Useful for pointing out ways of abstraction.




There are tools made for doing code review

```
894 -- XXX DUPLICATION
895 pkgQual dflags = alwaysQualify { queryQualifyPackage = qual_pkg dflags }
896 qual_pkg dflags pkg_key
897     | pkg_key == mainPackageKey
898     -- Skip the lookup if it's main, since it won't be in the package
899     -- database!
900     = False
901     | [_] <- [() | pkg' <- eltsUFM (pkgIdMap (pkgState dflags))
```


simonmar

You can use `not . Util.lengthExceeds`


Or just use GitHub Issues

  **larsrh** commented on an outdated diff on Jun 29  Hide outdated diff

```
tests/src/test/scala/scalaz/concurrent/TaskTest.scala
((16 lines not shown))
226 226
227 227     val r = Nondeterminism[Task].nmap6(t1, t2, t3, t4, t5, t6)(List(
228 -   r.run must_== List("a","b","c","d","e","f")
229 -   sb.toString must_==(“ecdfba”)
228 +   val chars = List('a','b','c','d','e','f')
229 +   r.run must_== chars
230 +   sb.toString.toList.sorted must_==(chars)
```

 **larsrh** added a note on Jun 29 Owner

I think the intention was to actually prove that the execution runs in parallel, hence the different waiting times. I don't know if this behaviour can be examined in some other way. Maybe by looking at the thread IDs?

 **Blaisorblade** added a note on Jun 29

Ah, makes sense.

The thread IDs are a neat idea. I prefer the thread names, since IDs might be reused. I'm dropping the waiting times since they aren't guaranteed to help. I coded this up, and after using a non-default thread pool (the default one is bounded to the number of processors), I could see six threads.

Summary

Tests of any kind are more useful if actually run.

Types are awesome, given an expressive type system.

When possible, make bugs prevent compilation.

Property-based testing finds way more bugs than unit testing.

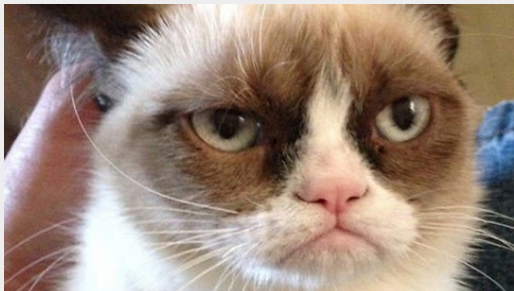
Tests are necessary to the extent that you don't have types.

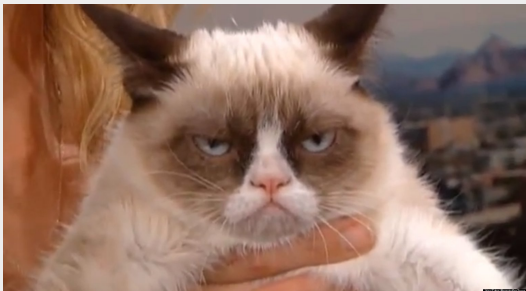
Summary

Don't rely on tests to reason about code, rely on types.

Try to use languages which make this possible.

Let's stop insisting on writing broken, buggy code.





Summary

The tools mentioned in this talk are just one step toward doing this.

But it takes effort. Creating working code is hard — but worth it.

We can do better — we already have

...the industry just needs to catch up!

Contact

ricky@elrod.me, @relrod6 on Twitter, “relrod” on Freenode IRC.
Let's talk!