

Aim, Fire

Kent Beck

“Never write a line of functional code without a broken test case.”

—Kent Beck

“Test-first coding is not a testing technique.”

—Ward Cunningham

Craig Larman talked about koans—the sound of one hand clapping, mountains aren’t mountains, and so on. Now I offer this: *Test-first coding isn’t testing*. Okay, Ward Cunningham, what is it? Never mind what is it—what’s it going to do to my code?

We have to start with how it works. Let’s say we want to write a function that adds two numbers. Instead of jumping to the function, we first ask, “How can we test this?” How about this code fragment:

```
assertEquals(4, sum);  
// cf JUnit.org for more  
// explanation
```

That’s a good start, but how do we compute `sum`? How about `int sum=? ...hmmm....`

What if we represent our computation as a static method on a `Calculator` class?

```
int sum=  
    Calculator.sum(2, 2);  
assertEquals(4, sum);
```

The fragment doesn’t compile? Pity. Now we go and write our `Calculator` class.

There are a bunch of other test cases that also have to pass? Excellent! Make a list and we’ll get to them all, one at a time.

History sidetrack

Test-first coding isn’t new. It’s nearly as old as programming. I learned it as a kid while reading a book on programming. It said that you program by taking the input tape (you know, like a long, skinny floppy disk) and typing in the output tape you expect. Then you program until you get the output tape you expect.

Until I started coding testing first, I had no idea how often I was coding without knowing what the right answer should be. Crazy, isn’t it? You wouldn’t do such a thing, but perhaps you have a co-worker who does.

Analysis

So, test-first is an analysis technique. We decide what we are programming and what we aren’t programming, and we decide what answers we expect. Picking what’s in scope and, more importantly, what’s out of scope is critical to software development. Following test-first religiously forces you to explicitly state what circumstances you were considering when you wrote the code. Don’t expect any circumstance not implied by one of the tests to work.

Design

But wait, there’s more. Test-first is also a design technique. You remember when we decided that `sum` would be a static method on the `Calculator` class? That’s logical, or inter-



face, design—just like my old college texts talked about. Of course, those texts never explained how to *do* logical design; they just made it clear that it must be done, it must be done before physical design and implementation, and heaven help anyone who mixes them up.

Test-first is a technique for logical design. You don't have the implementation yet, so you have to stretch to work on physical design. What you begin typing is an expression of the out-sides of the logic you are about to write.

Robert Martin makes the analogy to a virus. Viruses have special shapes that are designed to attach to molecules in the cell wall. The shapes are the inverse of the shape of the molecule. Test-first tests are like that. They are the inverse of the shape of the code we are about to write. And, when the shapes match exactly, we've written the code we expected.

Not testing

How much would you pay for tests like this? Don't answer that question. There's more. I'm a worrier. When I don't have my test blankie, I'm forever fussing about what did I miss, what did I forget, what did I just screw up? With the tests, bang, I have instant confidence.

Was that just a mistake? Push the button, run the tests, the bar is green, the code is clean. Ahhhh.

Next test.

All you testing gurus are probably sharpening your knives. "That's not testing. Testing is equivalence classes and edge cases and statistical quality control and...and...a whole bunch of other stuff you don't understand."

Hold on there—I never said that test-first was a testing technique. In fact, if I remember correctly, I explicitly stated that it *wasn't*.

However, reflecting on the difference between coding test-first and even automating tests immediately after coding, I think I have an order-of-magnitude fewer defects coding test-first. Do I have hard numbers? No, I don't. That's what PhDs are for, and I don't have one of those, either. (It

would make a great thesis, though, a la Laurie Williams' pair programming studies.)

I'm not submitting this as something everyone should do all the time. I have trouble writing GUIs test-first, for example (another thesis, eager student). It's certainly worth a try, though.

Better design

We saw how test-first drives design decisions. Surprise! Not only do the decisions come at a different time, the decisions themselves are different. Test-first code tends to be more cohesive and less coupled than code in which testing isn't part of the intimate coding cycle.

Some people worry that they can't even get their code done, and now they have to test, too. And we're asking them to design better, too? It can't be done.

Relax. You don't have to be a great designer—you just have to be creatively lazy. Here's an example.

Suppose we want to write a function that, given a person of a certain age, returns that person's mortality table (gruesome, I know, but that's life insurance for you). We might begin by writing

```
assertEquals(expectedTable,
    actualTable);
```

How are we going to compute the `actualTable`? Ask a `MortalityTable` class.

**Here's the beauty of it—
we didn't have to be
brilliantly prescient
designers to find a less
tightly coupled design.
We just had to be
intolerant of pointless
effort in writing the tests.**

```
MortalityTable actualTable=
    new MortalityTable(bob);
assertEquals(expectedTable,
    actualTable);
```

How do we create the person? There's a constructor that takes 11 parameters (or 11 setter methods, all of which have to be called, which amounts to the same thing).

```
Person bob= new Person("Bob",
    "Newhart", 48, Person.NON_
    SMOKER,
```

We're tired of typing about this point, and we don't feel like staring at the parameter list for the constructor any more. Is there some other solution?

There might be a simpler way to construct a `Person`, but there's an even easier solution: change the interface to `MortalityTable` to take the age (assuming that's the only parameter of interest, remember, this is an example), not the whole person. Now we don't have to create a `Person` at all:

```
MortalityTable actualTable=
    new MortalityTable(48);
assertEquals(expectedTable,
    actualTable);
```

Hang on a doggone minute

What just happened? We made a snap design decision that created more flexibility than we needed by passing the whole `Person` when creating a `MortalityTable`. Without immediate feedback, we would probably have just ridden that decision for years or decades without questioning it. Writing the test gave us feedback seconds later that caused us to question our assumption.

Here's the beauty of it—we didn't have to be brilliantly prescient designers to find a less tightly coupled design. We just had to be intolerant of pointless effort in writing the tests. We still had to demonstrate skill in finding and executing our alternative design. So, it's not that design skill goes away—the intensity of the feedback substitutes for the ability to guess right. We don't have to guess (well, not for long) about the right

design. We design something simple we think might work, and then seconds later we have confirmation. If the feedback shows we oversimplified, we make the design more complicated. If the feedback shows we overdesigned, we simplify.

So, test-first

- encourages you to be explicit about the scope of the implementation,
- helps separate logical design from physical design from implementation,
- grows your confidence in the correct functioning of the system as the system grows, and
- simplifies your designs.

Is that all? I'll leave you with two more points.

First, test-first helps you communicate clearly with your pair-programming partner. Martin and I had a fabulous example of this one time when we were programming in Munich. He wanted the expected value to be 0.1 and I wanted it to be 10. Can you spot the disagreement? (It took us far too long to realize we were talking about different representations of a percentage, but that's programming in public for you).

Second, test-first tests leave a Rosetta stone for the future, answering the all-important question, "What was this idiot thinking when he wrote this?"

When shouldn't you test first? Writing user interfaces test-first is hard. I'd love to find a UI testing strategy where test-first actually helped me design better interfaces.

Give it a spin. See how it goes. 

Kent Beck is director of Three Rivers Institute, exploring the interaction of technology, business, and humanity. He has pioneered Extreme Programming, CRC cards, the TimeTravel patterns, the xUnit family of testing frameworks including the award-winning JUnit, and the Hillside Group. He wrote *Planning Extreme Programming*, *Extreme Programming Explained: Embrace Change*, *Kent Beck's Guide to Better Smalltalk: A Sorted Collection*, and *Smalltalk Best Practice Patterns*. Contact him at PO Box 128, Merlin, OR 97532; kent@threeriversinstitute.org.

ADVERTISER / PRODUCT INDEX

SEPTEMBER / OCTOBER 2001

Advertising Personnel

James A. Vick

IEEE Staff Director, Advertising Businesses
Phone: +1 212 419 7767
Fax: +1 212 419 7589
Email: jv.ieeemedia@ieee.org

Marion Delaney

IEEE Media, Advertising Director
Phone: +1 212 419 7766
Fax: +1 212 419 7589
Email: md.ieeemedia@ieee.org

Marian Anderson

Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown

IEEE Computer Society, Business Development Manager
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Debbie Sims

Assistant Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: dsims@computer.org

Atlanta, GA

C. William Bentz III
Email: bb.ieeemedia@ieee.org
Gregory Maddock
Email: gm.ieeemedia@ieee.org
Sarah K. Huey
Email: sh.ieeemedia@ieee.org
Phone: +1 404 256 3800
Fax: +1 404 256 7942

San Francisco, CA

Matt Lane
Email: ml.ieeemedia@ieee.org
Telina Martinez-Barrientos
Email: Telina@husonusa.com
Phone: +1 408 879 6666
Fax: +1 408 879 6669

Chicago, IL (product)

David Kovacs
Email: dk.ieeemedia@ieee.org
Phone: +1 847 705 6867
Fax: +1 847 705 6878

Chicago, IL (recruitment)

Tom Wilcoxon
Email: tw.ieeemedia@ieee.org
Phone: +1 847 498 4520
Fax: +1 847 498 5911

New York, NY

Dawn Becker
Email: db.ieeemedia@ieee.org
Phone: +1 732 772 0160
Fax: +1 732 772 0161

Boston, MA

David Schissler
Email: ds.ieeemedia@ieee.org
Phone: +1 508 394 4026
Fax: +1 508 394 4926

Dallas, TX

Royce House
Email: rhouse@houseco.com
Phone: +1 713 668 1007
Fax: +1 713 668 1176

Japan

German Tajiri
Email: gt.ieeemedia@ieee.org
Phone: +81 42 501 9551
Fax: +81 42 501 9552

Europe

Glesni Evans
Email: ge.ieeemedia@ieee.org
Phone: +44 193 256 4999
Fax: +44 193 256 4998

Advertiser / Product

Page Number

Java Development Conference 2001	Cover 2
ParaSoft	Cover 4
Software Engineering Conference 2001	11
Sticky Minds	15

IEEE Software

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, California 90720-1314
Phone: +1 714 821 8380
Fax: +1 714 821 4010
<http://computer.org>
advertising@computer.org