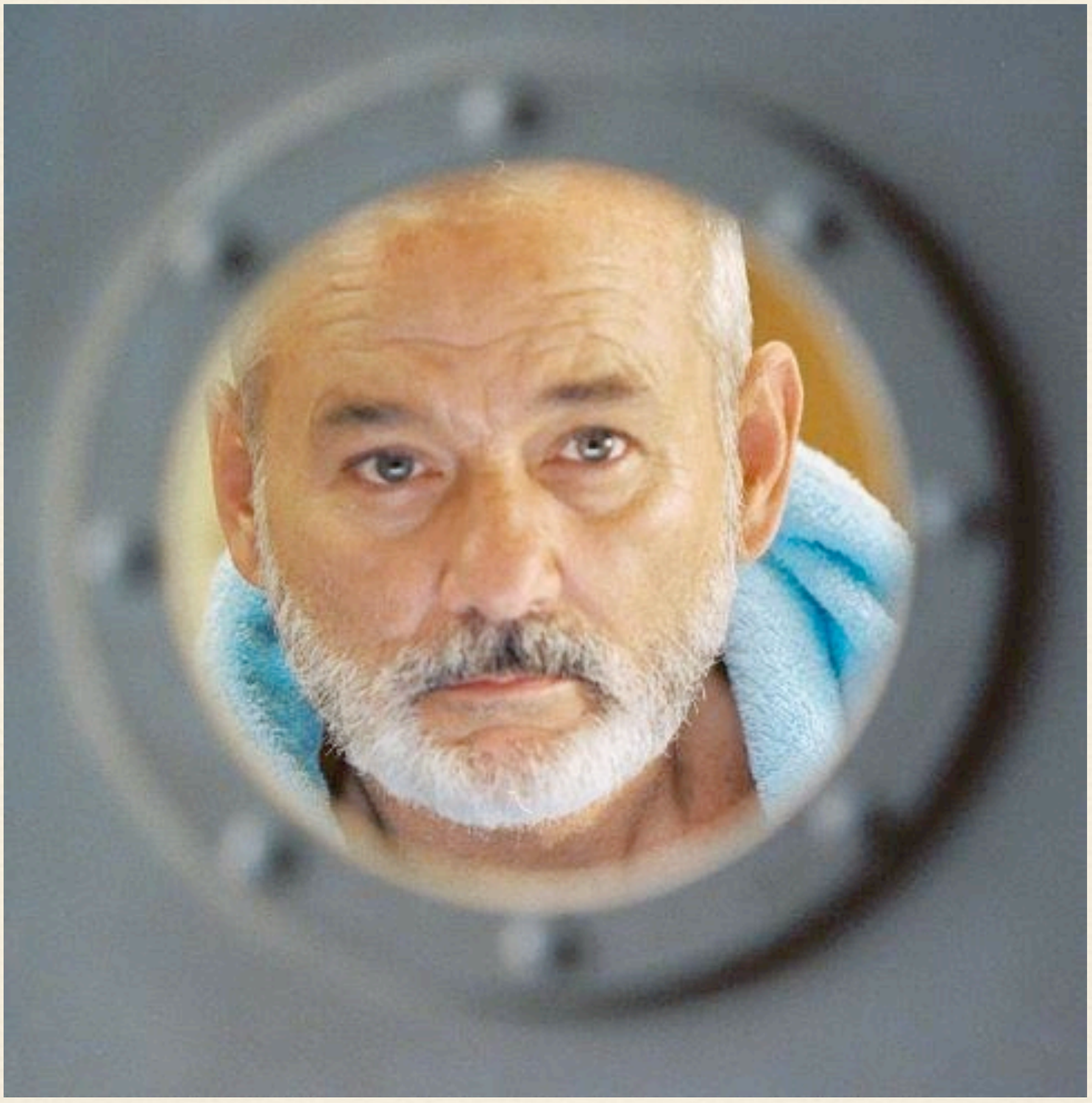
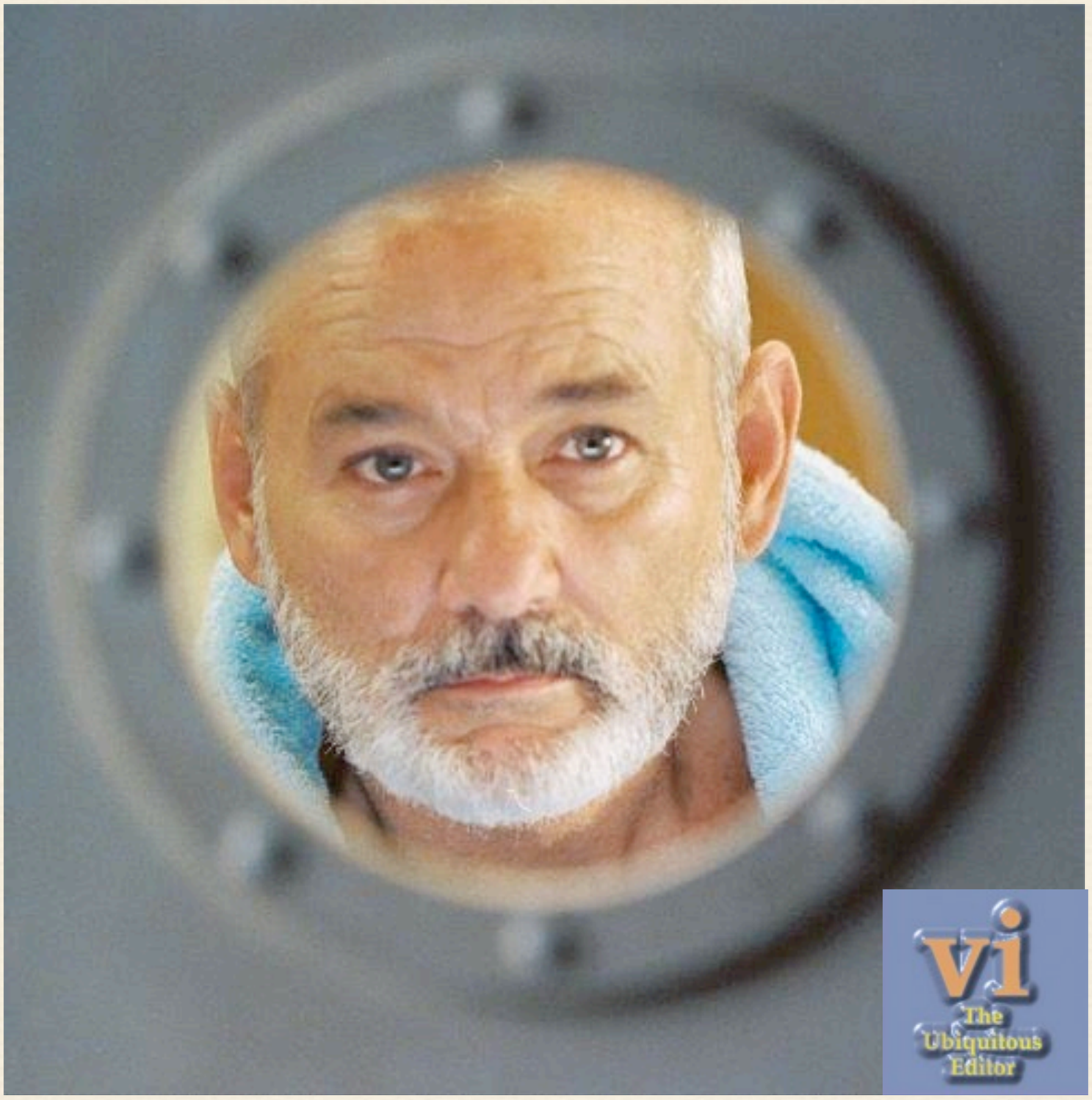
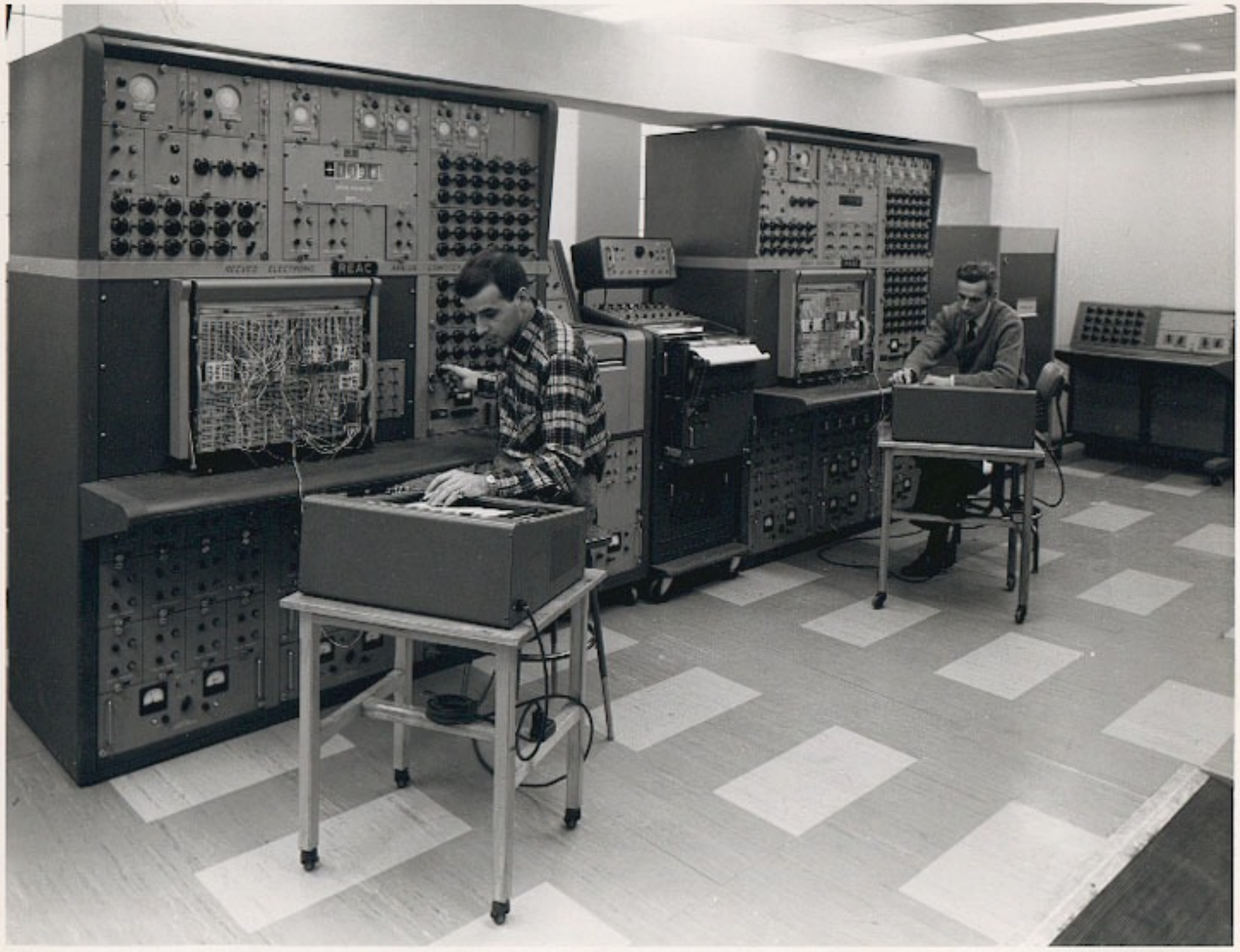
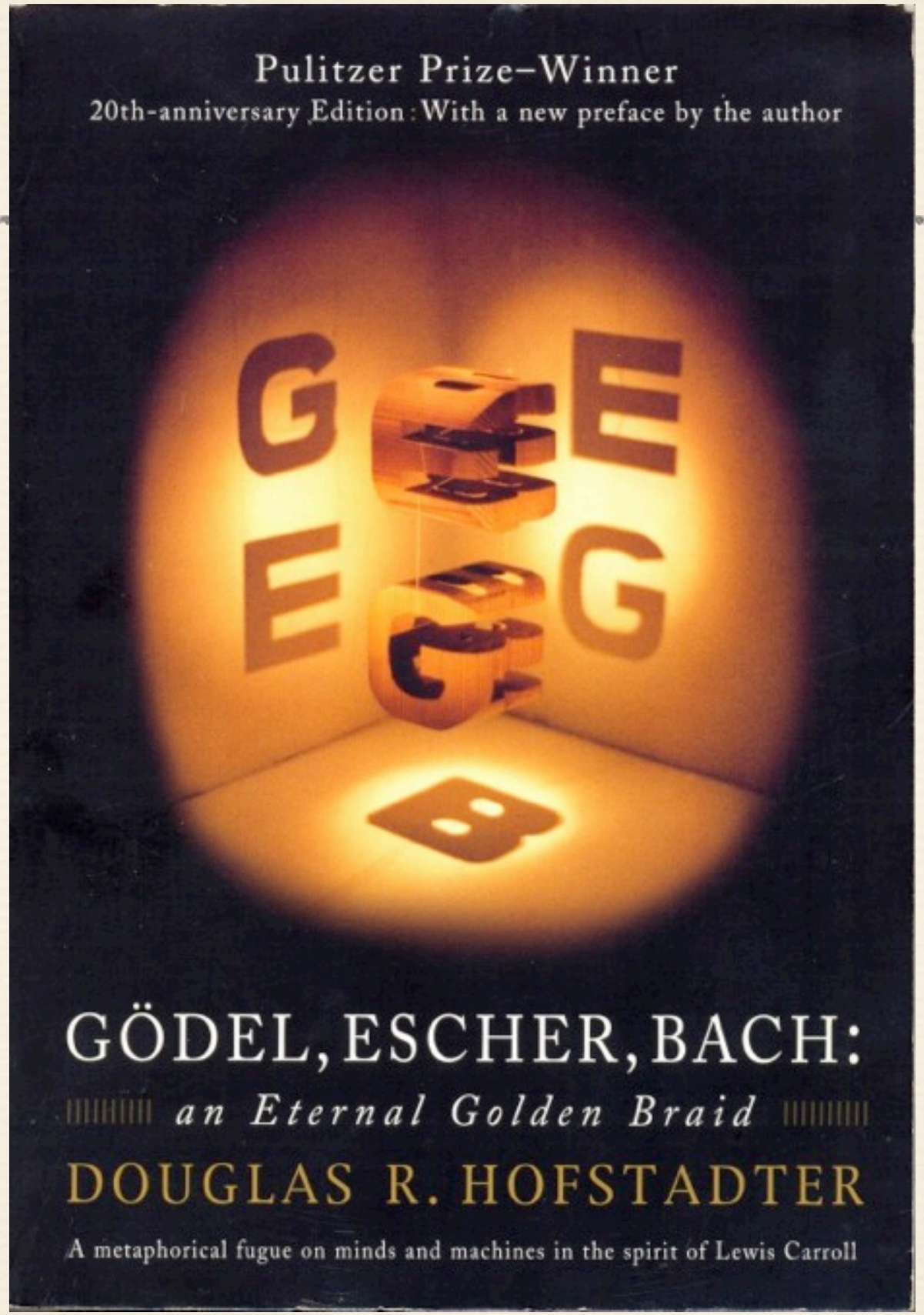
An Anatomie and Historie of Coding Joy

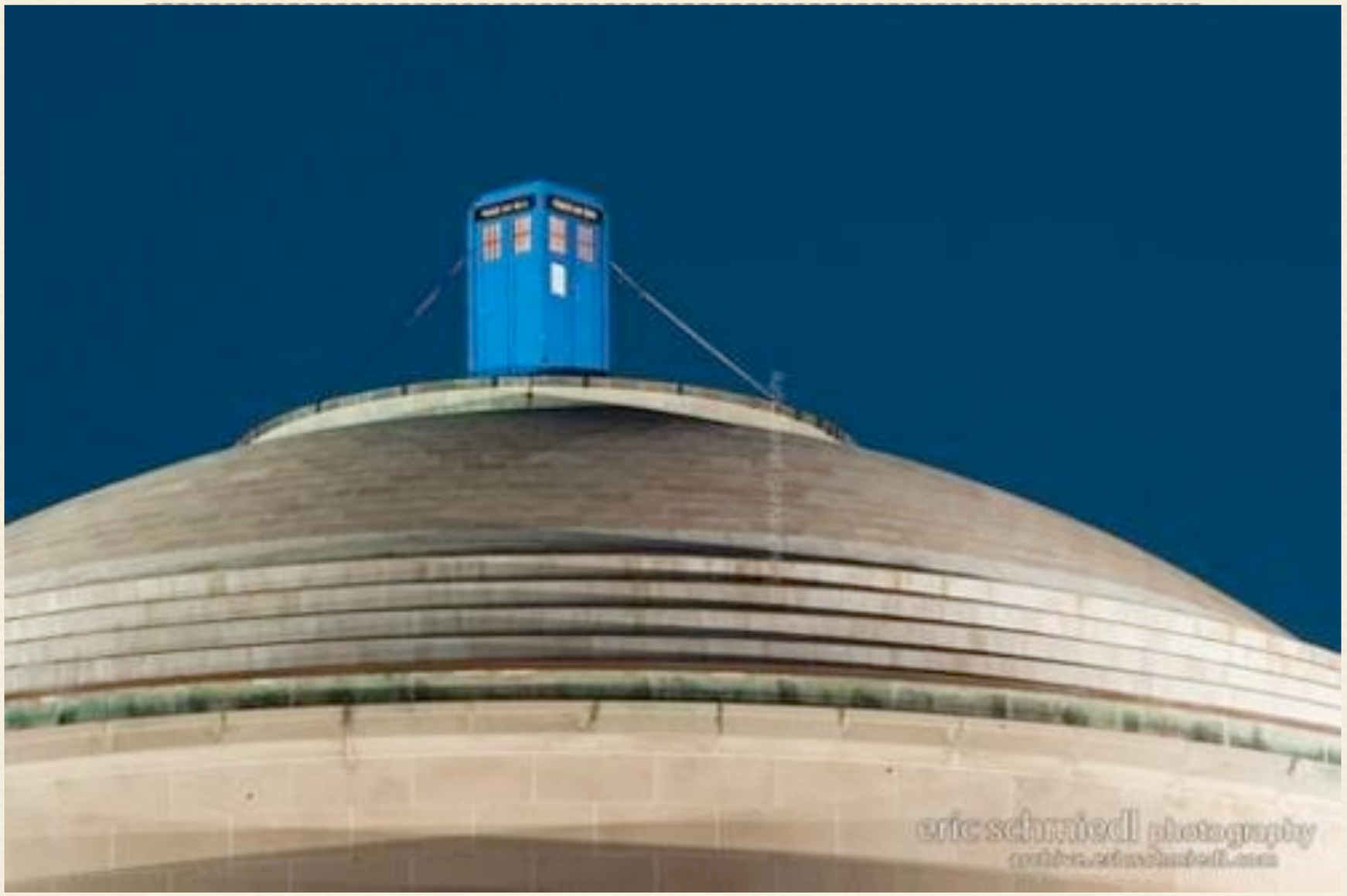
Michael Feathers

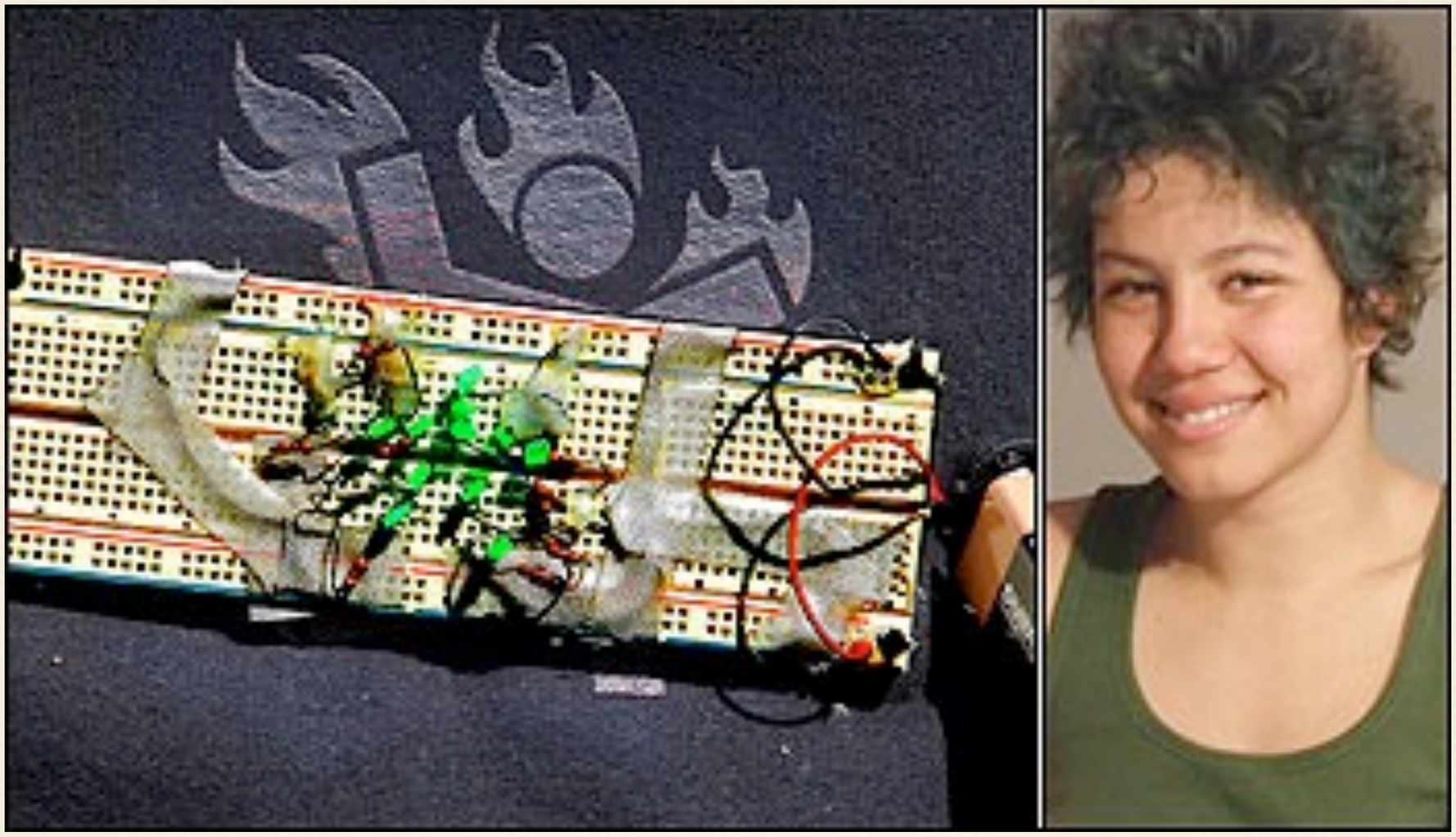



Vi
The Ubiquitous Editor


REAC


Pulitzer Prize–Winner
20th-anniversary Edition : With a new preface by the author
G
E
E
G
B
GÖDEL, ESCHER, BACH:
an Eternal Golden Braid
DOUGLAS R. HOFSTADTER
A metaphorical fugue on minds and machines in the spirit of Lewis Carroll


eric schmiedl photography



Generalizing Overloading for C++2000

Bjarne Stroustrup

AT&T Labs, Florham Park, NJ, USA

Abstract

This paper outlines the proposal for generalizing the overloading rules for Standard C++ that is expected to become part of the next revision of the standard. The focus is on general ideas rather than technical details (which can be found in AT&T Labs Technical Report no. 42, April 1,1998).

Introduction

With the acceptance of the ISO C++ standard, the time has come to consider new directions for the C++ language and to revise the facilities already provided to make them more complete and consistent.
A good example of a current facility that can be generalized into something much more powerful and useful is overloading. The aim of overloading is to accurately reflect the notations used in application areas. For example, overloading of + and * allows us to use the conventional notation for arithmetic operations for a variety of data types such as integers, floating point numbers (for built-in types), complex numbers, and infinite precision numbers (user-defined types). This existing C++ facility can be generalized to handle user-defined operators and overloaded whitespace.

enterprise

- github.com/tenderlove/enterprise/

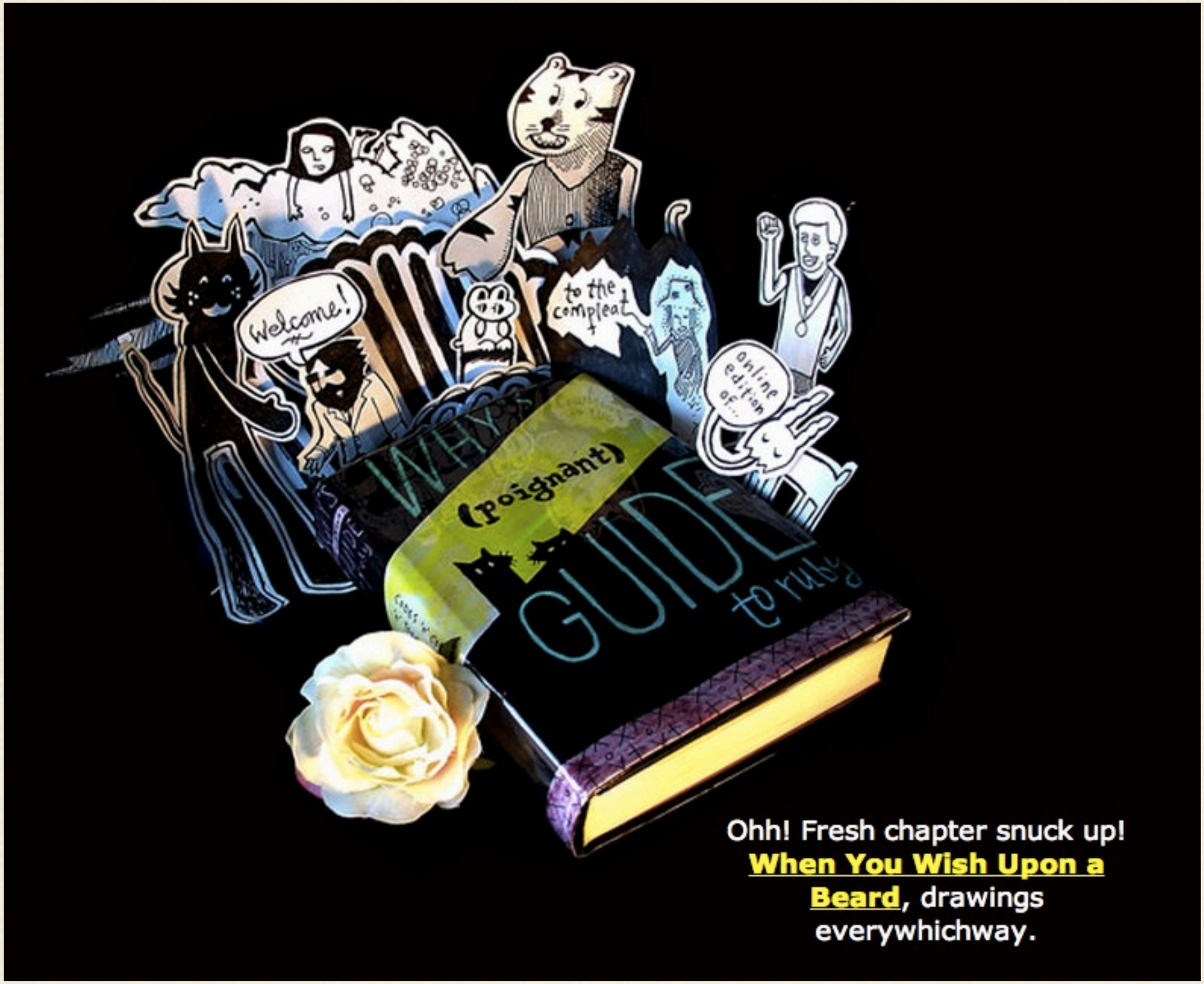
DESCRIPTION:

Wish you could write your Ruby in XML? Has the fact that Ruby is not "enterprise" got you down? Do you feel like your Ruby code could be made to be more "scalable"? Well look no further my friend. You've found the enterprise gem. Once you install this gem, you too can make Rails scale, Ruby faster, your code more attractive, **and** have more XML in your life.

I'm sure you're asking yourself, "how can the enterprise gem promise so much?". Well the answer is easy, through the magic of XML! The enterprise gem allows you to write your Ruby code in XML, therefore making your Ruby and Rails code scale. Benefits of writing your code in XML include:

- It's easy to read!
- It scales!
- Cross platform
- TRANSFORM! your code using XSLT!
- Search your AST with XPath **or** CSS!


Hello! This is
@tenderlove,
who is this?



Ohh! Fresh chapter snuck up! **When You Wish Upon a Beard**, drawings everywhichway.

FIU

FLORIDA INTERNATIONAL UNIVERSITY

FIU

FLORIDA INTERNATIONAL UNIVERSITY

```
send(to, from, count)
register short *to, *from;
register count;
{
        register n = (count + 7) / 8;
        switch(count % 8) {
        case 0: do {    *to = *from++;
        case 7:         *to = *from++;
        case 6:         *to = *from++;
        case 5:         *to = *from++;
        case 4:         *to = *from++;
        case 3:         *to = *from++;
        case 2:         *to = *from++;
        case 1:         *to = *from++;
                } while(--n > 0);
        }
}
```

```
#include <iostream>
using namespace std;

template<int n> struct Fib {
enum { val = Fib<n-1>::val + Fib<n-2>::val };
};

template<> struct Fib<1> { enum { val = 1 }; };

template<> struct Fib<0> { enum { val = 0 }; };

int main() {
cout << Fib<5>::val << endl; // 6
cout << Fib<20>::val << endl; // 6765
} ///:~
```

```
/*
 * A simple, lambda-calculus-based functional language encoded in C++
 * templates.  The language supports simple type-level literals,
 * conditionals and higher-order functions.
 *
 * We use an Eval/Apply-style interpreter as described in SICP.
 *
 * Author: Matthew Might
 * Site:   http://matt.might.net/
 *
 */

#include <stdlib.h>
#include <iostream>
#include <assert.h>



/* A type-level Pressburger encoding of natural numbers. */

// A type for 0:
struct Zero
{
  enum { value = 0 } ;
} ;

// 1 == Succ<Zero>
// 2 == Succ<Succ<Zero> >
// ...
template <typename N>
struct Succ
{
  enum { value = N::value + 1 } ;
} ;
```

Brainfuck	Ook!	Description
>	Ook. Ook?	Move the pointer to the right
<	Ook? Ook.	Move the pointer to the left
+	Ook. Ook.	Increment the memory cell under the pointer
-	Ook! Ook!	Decrement the memory cell under the pointer
.	Ook! Ook.	Output the character signified by the cell at the pointer
,	Ook. Ook!	Input a character and store it in the cell at the pointer
[	Ook! Ook?	Jump past the matching `Ook? Ook!` if the cell under the pointer is 0
]	Ook? Ook!	Jump back to the matching `Ook! Ook?`

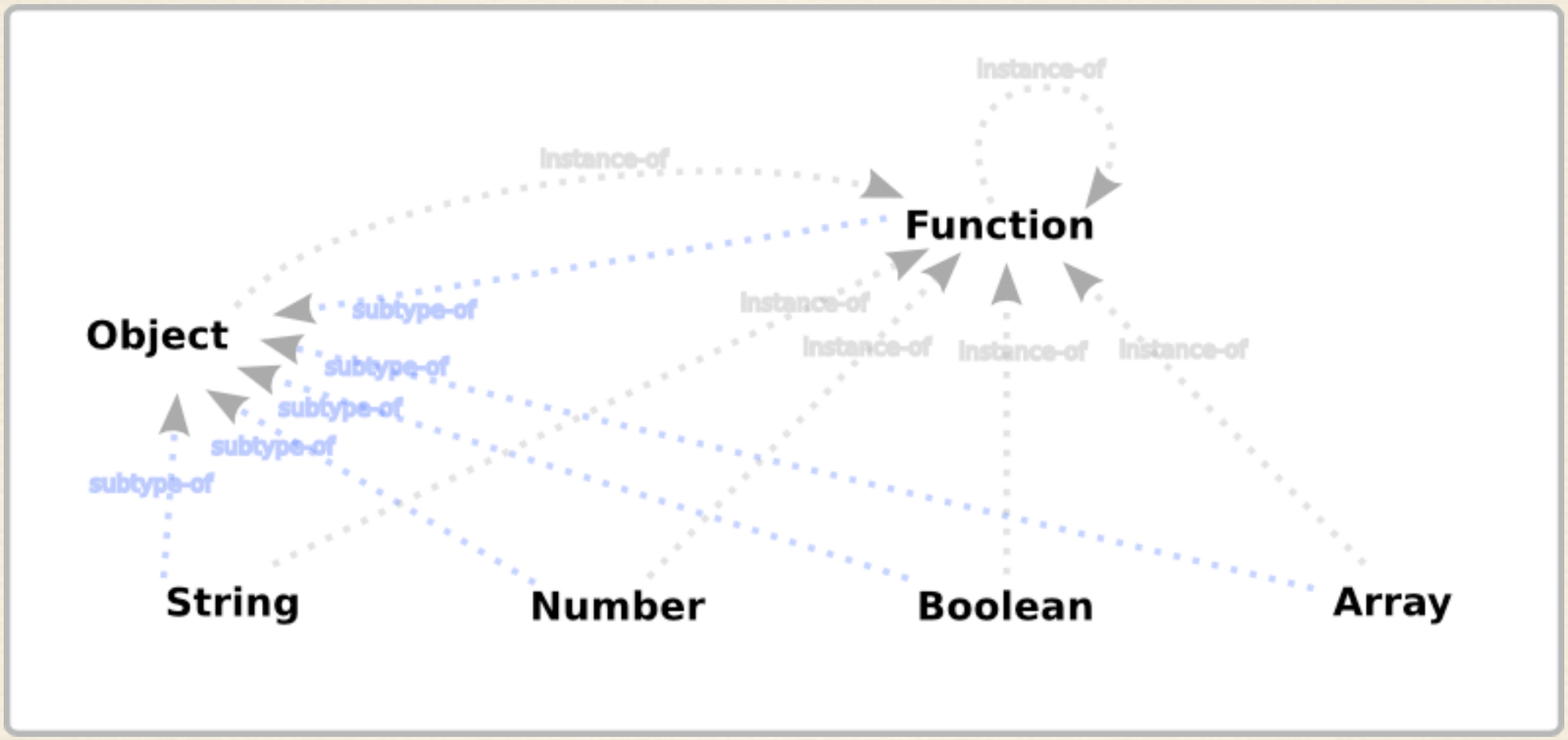
Hello, world! program [edit]

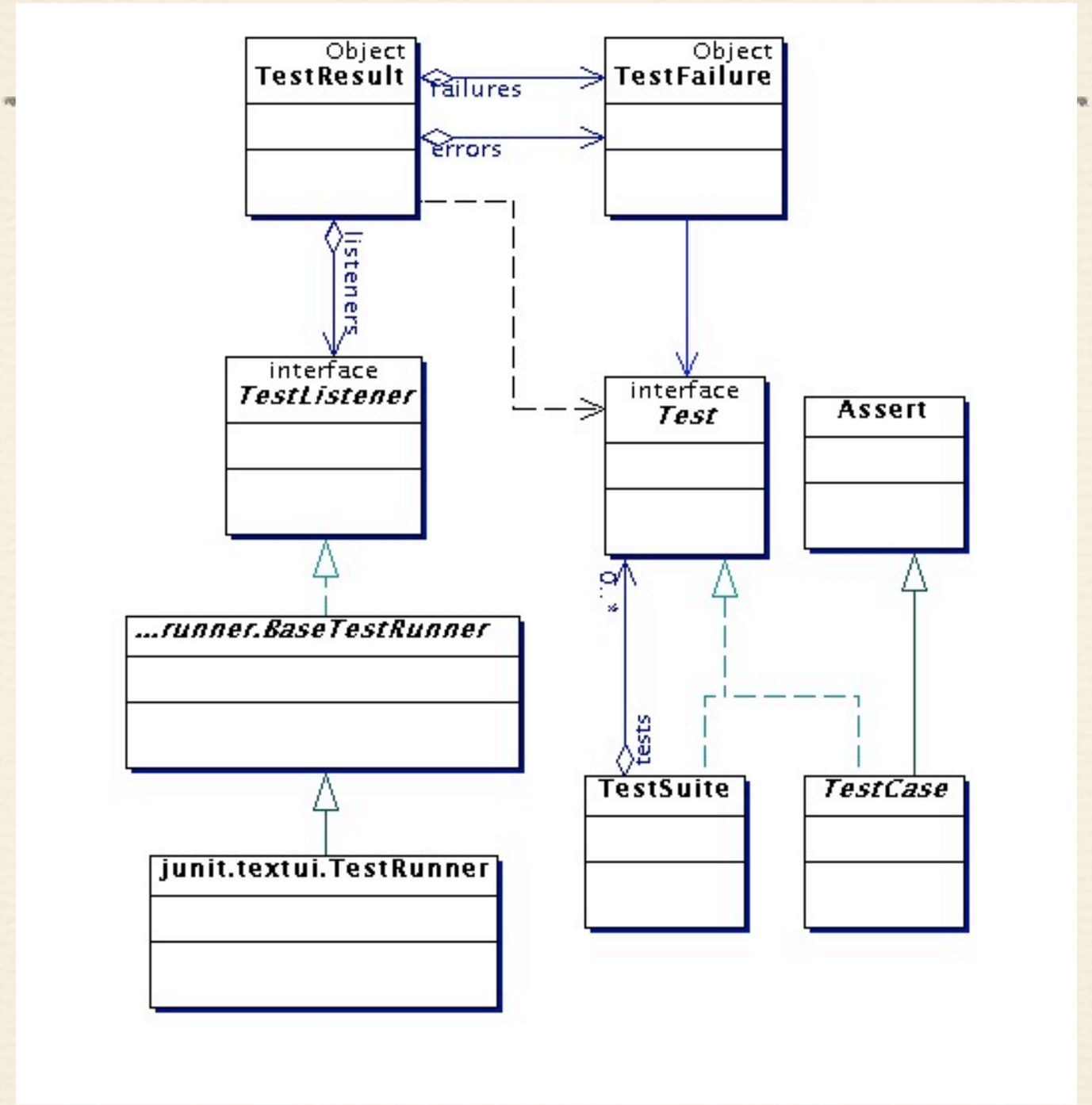
```
Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook.
Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook?
Ook! Ook! Ook? Ook! Ook? Ook. Ook. Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook.
Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
Ook? Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook. Ook! Ook.
```


D8


SINGAPORE

```
quicksort [] = []
quicksort (x:xs) = quicksort small ++ (x : quicksort large)
  where small = [y | y <- xs, y <= x]
        large = [y | y <- xs, y > x]
```


Function
Object
String
Number
Boolean
Array
instance-of
instance-of
instance-of
instance-of
instance-of
instance-of
subtype-of
subtype-of
subtype-of
subtype-of
subtype-of


Object
TestResult
failures
errors
Object
TestFailure
listeners
interface
TestListener
interface
Test
Assert
...runner.BaseTestRunner
junit.textui.TestRunner
tests
TestSuite
TestCase

com.googlecode.refit.test.fixture.CityColumnFixture		
city	getCountry()	isCapital()
Berlin	Germany	true
Hamburg	Germany	false *expected* --- true *actual*
Madrid	Spain *expected* --- Italy *actual*	true

```
mix = unwords . map unwords . transpose .  map (concat . repeat . words)
```