

WHY YOUR CODE WANTS TO
KILL YOU
(AND HOW YOU CAN SURVIVE)



VICTOR NICOLLET

<http://www.nicollet.net>

Programming Language Design

Static Analysis, Compiler Theory, Objective Caml

LAMP / jQuery Guru*

MySQL, PHP, jQuery

Start-Up Co-Founder

CouchDB, Objective Caml, jQuery

*:self-appointed

FAILURE CAUSES

Obvious syntax error or typo

Type errors, wrong functions

Faulty or incomplete logic

Unhandled corner case

Good code, bad design

Act of God

FAILURE CAUSES

Obvious error or typo

USE A COMPILER

Type errors, wrong functions

Faulty or incomplete logic

Unhandled corner case

Good code, bad design

Act of God

FAILURE CAUSES

Obvious error or typo

USE A COMPILER

Type error conditions

TYPE SYSTEM

Faulty or incomplete logic

Unhandled corner case

Good code, bad design

Act of God

FAILURE CAUSES

Obvious error or typo

USE A COMPILER

Type error conditions

TYPE SYSTEM

Faulty state logic

UNIT TESTS

Unhandled corner case

Good code, bad design

Act of God

FAILURE CAUSES

Obvious error or typo

USE A COMPILER

Type error or conditions

TYPE SYSTEM

Faulty or illogical logic

UNIT TESTS

Unhandled cases

CODE COVERAGE

Good code, bad design

Act of God

FAILURE CAUSES

Obvious error or typo

USE A COMPILER

Type error or conditions

TYPE SYSTEM

Faulty state logic

UNIT TESTS

Unhandled cases

CODE COVERAGE

Good design

BE SMARTER ;-)

Act of God

The lower on the list,
the harder to find.

Software architecture moves
potential errors up or down.

Find the error :

```
public void write (Banana banana)
{
    this.write (banana.contents);
}
```

Find the error :

```
public void write (Banana banana)
{
    this.write (banana.contents);
}
```

banana could be **null**



Must not be null:

```
method write_banana banana =  
  self # write (banana # contents)
```

(compiler-enforced)

May be null:

```
method write_banana = function  
  | Some banana ->  
    self # write (banana # contents)  
  | None -> (* ... *)
```

(compiler warning if unhandled)

SHORT EXAMPLE

Obvious syntax error or typo

Type errors, wrong functions

Faulty or incomplete logic

Unhandled corner case

God

Using OCaml
moves the error up

Act of God



JSON + JS or XML + XSLT ?

JS: lets you shoot yourself in the foot

Server-side or Client-side ?

Server: your software, your hardware

Configure or Develop ?

Config: less flexible but safer

Exceptions or Return Codes?

Exceptions don't appear in function signatures

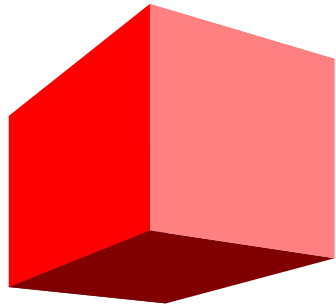
We're used to dealing with
hostile clients.

We always validate user input.
(Being paranoid doesn't make me wrong!)

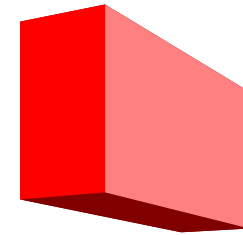
But what about ...

- ... the database ?
- ... the file system ?
- ... the cache ?

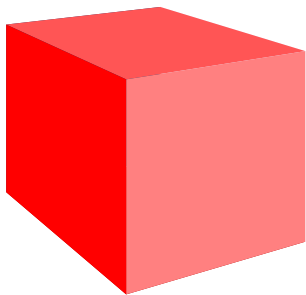
Persistence is the enemy
of **change**.



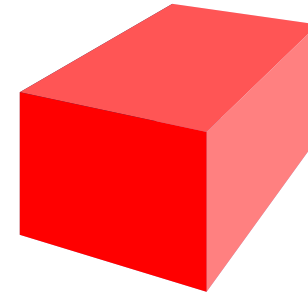
schema
1.0



application
1.0



schema
2.0



application
2.0



SQL says : ALTER

- Annoys the DBAs
- Might cause downtime
- Not under source control

NoSQL says : extend

(keep both 1.0 and 2.0 around)

- Annoys the developers
- More special cases
- Harder to document

THE WORLD HATES YOU

Obvious syntax error or typo

Type errors, wrong functions

Faulty or incomplete logic

Unhandled corner case

God

Using NoSQL
moves the error down

Act of God



Testing can only prove
the existence of errors,
not their absence.

How to **prove** these?

- Only friends can see this
- Only the owner can edit this
- This plane autopilot will not crash

Testing can only prove the existence of errors, not their absence.

How to **prove** these?

- Only friends can see this
- Only the admin can edit this
- The plane autopilot will not crash

Move them up enough

Using types to prove things

C++ Singleton

```
class Singleton
{
private:
    Singleton() {}
public:
    static Singleton &instance()
    {
        static Singleton me;
        return me;
    }
};
```

Using types to prove things

C++ Singleton

Private constructor means
`instance()`
is the **only** way to
get a value of type
`Singleton`

Editable Page

Page is represented by two types :

Read-Only - Can only read
- Used almost everywhere
- Returned by "get" functions

Read-Write - Can read and write
- Not returned by anything

One function tests ownership to turn Read-Only into Read-write

Editable Page

The **only** way to get a Read-Write is to run the ownership test.

Even if you do it in a different file!

This proof is encoded in the type, and **verified** by the compiler.

OCaml has parametric types
(also known as generics)

```
[ 1 , 2 ] : int list  
[ "a" , "b" ] : string list  
[ [1] , [2] ] : int list list
```

Use the parameter to encode part of the proof

```
type 'proof id

val read :
  [<`rw|`ro] id -> page

val write :
  [`rw] id -> page -> unit

val can_write :
  'any id -> user -> [`rw] id option
```

... using abstract types

```
type 'proof id = int
```

All ids actually have the same type...

```
let read id =  
  Db.read_page id
```

```
let write id page =  
  Db.write_page id page
```

```
let can_write id user =  
  if User.is_owner user id then  
    Some id  
  else  
    None
```

... which makes conversion easy

Assuming that `Page.id` turns a string into a read-only page identifier:

```
let id      = Page.id (request "page") in
let data   = request "data" in

Page.write id data
```

This expression has type `[`ro] Page.id`
but type `[`rw] Page.id` was expected

SHORT EXAMPLES

```
let id      = Page.id (request "page") in
let data    = request "data" in

let rwid =
  Page.can_write id User.current
in

Page.write rwid data
```

This expression has type [``rw`] `Page.id option`
but type [``rw`] `Page.id` was expected

SHORT EXAMPLES

```
let id      = Page.id (request "page") in
let data = request "data" in

match
  Page.can_write id User.current
with
  | Some rwid -> Page.write rwid data
  | None -> respond "Cannot edit!"
```

This is correct: no unauthorized writes, no unhandled corner cases.

MPVC ARCHITECTURE

PERSISTENCE LAYER

SQL, NoSQL...

STATELESS MODEL

Any technology

PROOF LAYER

OCaml, SML, Haskell, Scala, F# ...

VIEW

CONTROLLER

Online management tools for
Associations, Communities, Unions, Groups



December 2010
www.runorg.com