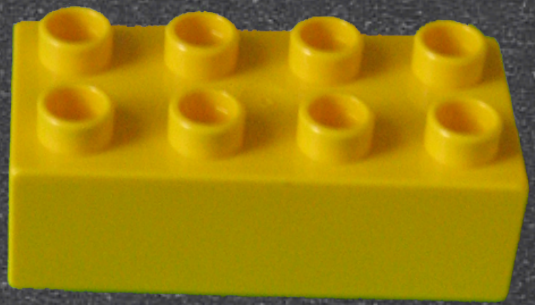
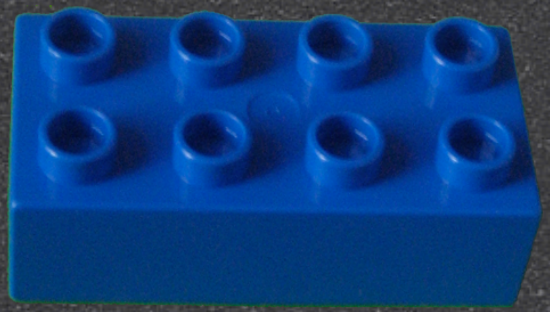
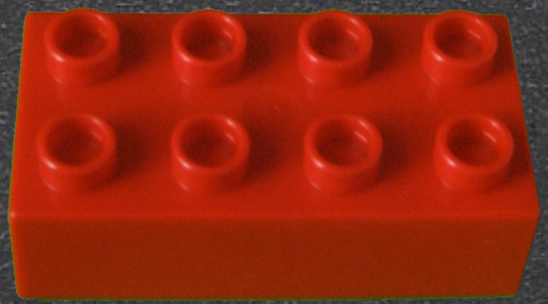
Business Rules with Brick

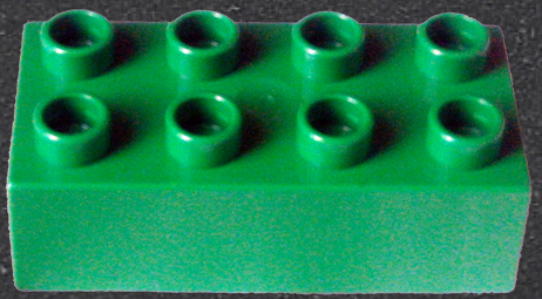




brian d foy

Nordic Perl Workshop 2007





Field validation is too low-level

Business rules are high-level

Code is for programmers

Connect coders and business

Field validation is too simple

```
is_number( $age );

cookie_expired( $cookie );

amount_ok( $n + $m + $o );

required( @fields );
```

Errors too vague

“Number is out of range”

“Password has invalid characters”

“Field foo is missing”

Helpful messages

“Number was %s but needs to be %s”

“Password can only be alphabetic, but I found %s”

“Field bar requires field foo, which was blank”

Loose coupling

Remove business logic from code

Avoid lock-in to technology

Separate architecture

Data::FormValidator

Perfectly fine for simple things

Based on fields

Relationships tough to specify

Poor error reporting

Tried to subclass

Tried to refactor

Easy for programmers

presence

right format

allowed value

one-to-one

ignore business

Hard for business

Many-to-many relationships

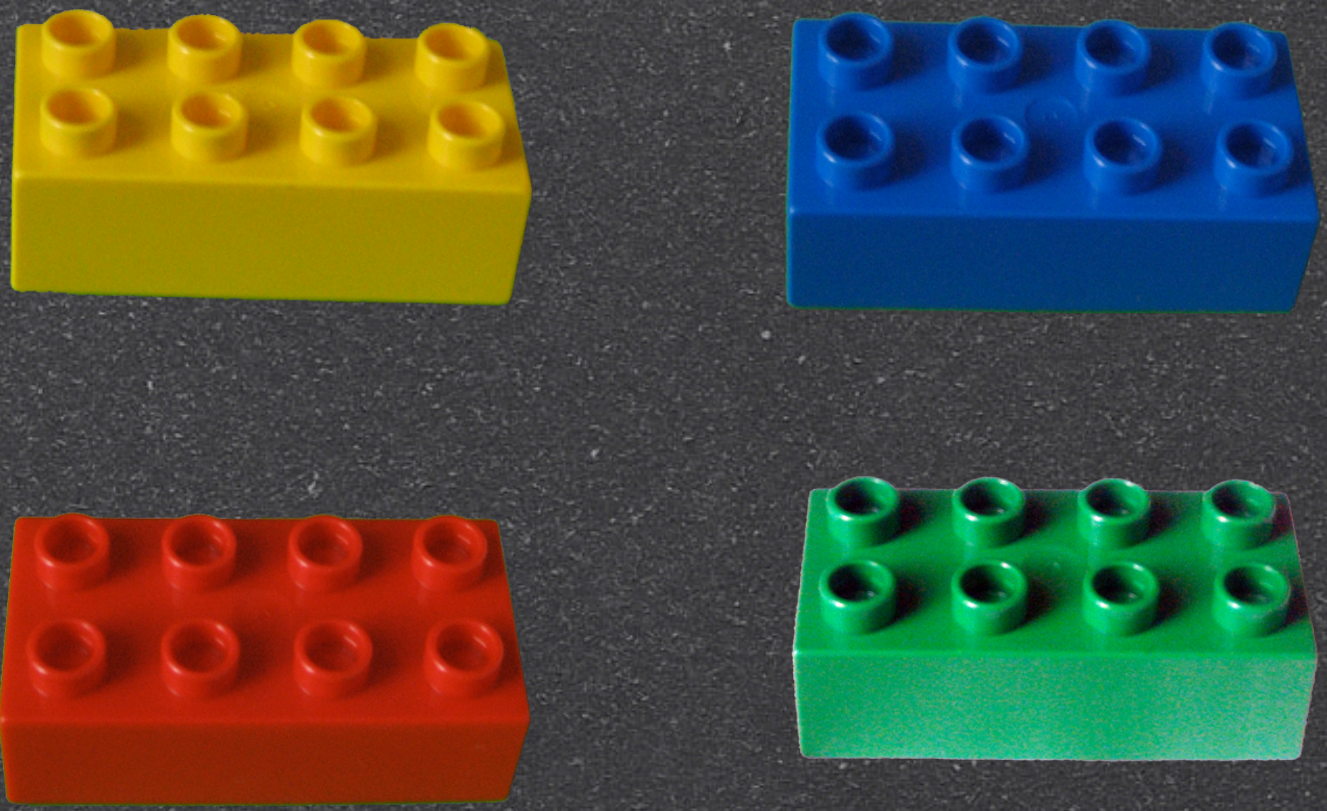
Out-of-band information

Legacy rules

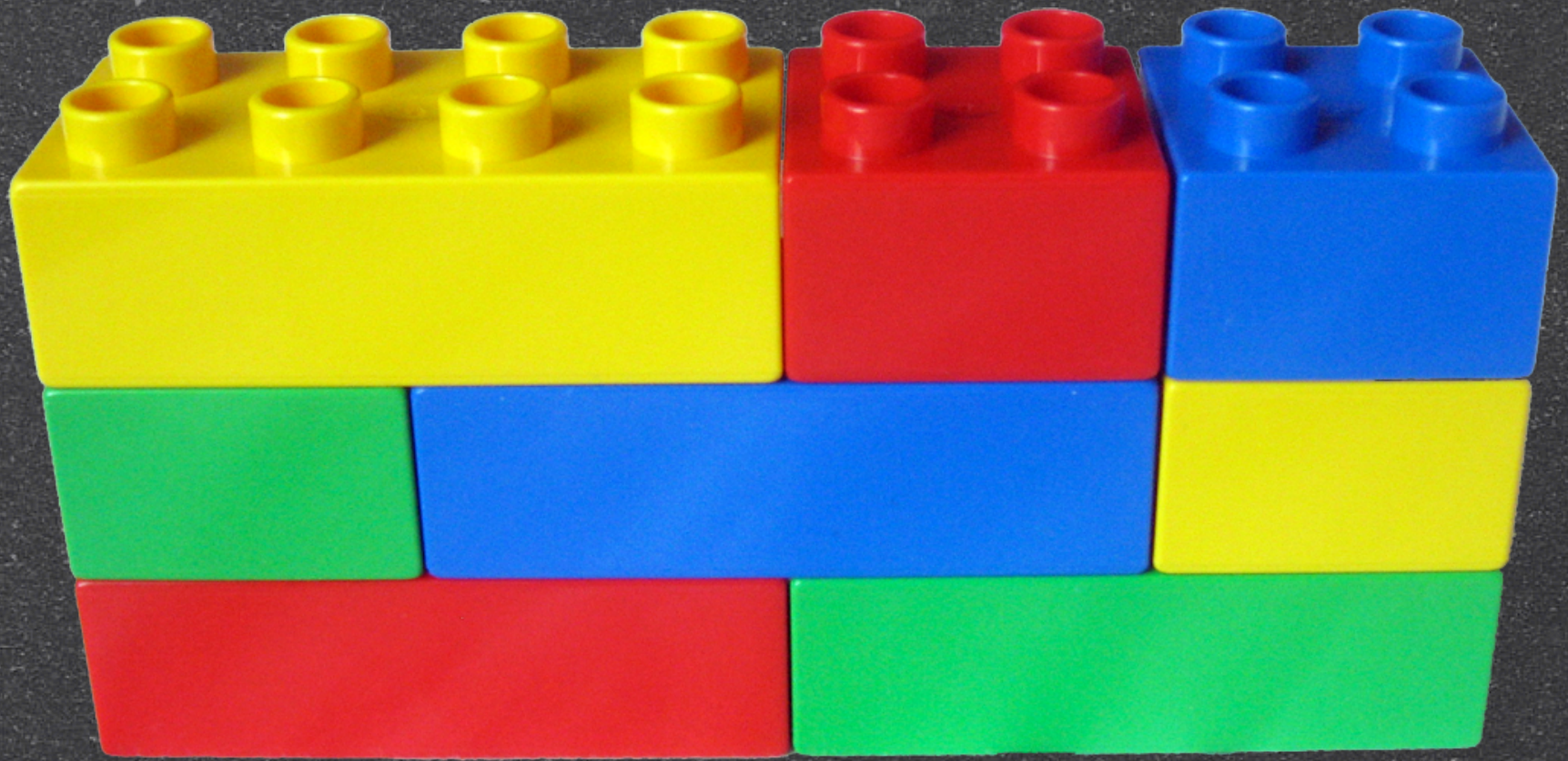
Exceptions

Don’t know Perl

Programmers think...



Business is...



Full validation

Presence

Format

Valid Value

Relationships

Right Value

Programmers write code

No one else does

Programmers read code

No one else does

Business people know the rules

No one else does

Connect both sides

Describe the validation

Turn it into code

Explain the validation

Apply it to input data

Explain the results


Brick

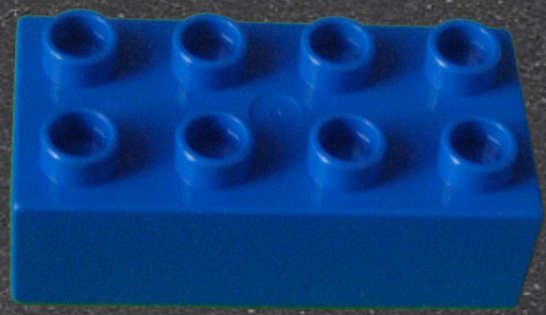
Business

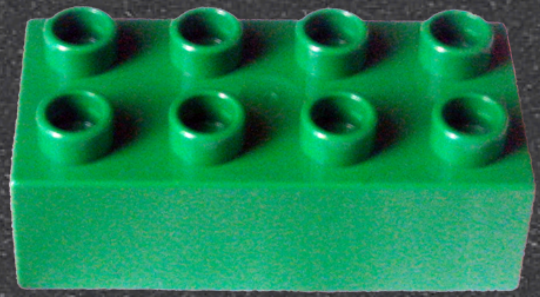
Rules

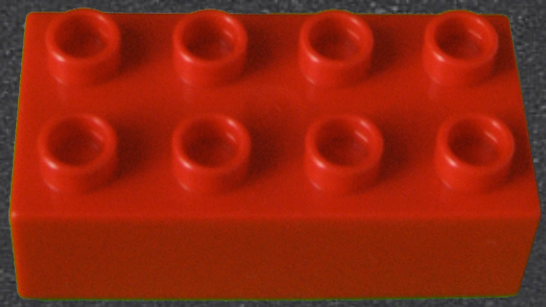
in

Closures,

‘**K**ay







A rule is simple

Complex rules compose simple rules

Rules divorced from input fields

Re-useable rules close over setup

Still alpha

In active use at a major client

Detailed, user-defined error messages

Describe the situation

Make it look less like code

```
@Description = (

   [ label => method_name => \%setup ],

    );
```

Might come from a config file

Explain profile

```
some_label
    __compose_AND
    __compose_ONE_OF
                __fields_are_something
                __compose_AND
                    __compose_AND
                        _value_length_is_equal_to_greater_than
                        _value_length_is_equal_to_less_than
                    _is_only_decimal_digits
                        _is_only_decimal_digits
            __compose_ONE_OF
                __fields_are_something
                __compose_AND
                    _is_YYYYMMDD_date_format
                    _is_valid_date
            __compose_ONE_OF
                __fields_are_something
                __compose_AND
                    _is_YYYYMMDD_date_format
                    _is_valid_date
```

Putting it together

```
@Description = (
   [ label => constraint_name => \%setup ],
     );
my $Brick = Brick->new();
my $profile =
   $Brick->profile_class->new( \@Description );
my $result = $Brick->apply( $profile, \%Input );
```

Results object

Tree data structure

Brick::Result can play with it

```
@Results = (
   [ label => [ 1 | 0 ] => \%errors ],
    );
```

Error Hash

```
$errors = [
   { handler => $method1, message => ...,
      errors => [ ... ] },
   { handler => $method2, message => ...,
      errors => [ ... ] },
   ... ];
```

Describe what happened

```
just_right: passed three_digit_odd_number

too_long: failed three_digit_odd_number
    long_number: _value_length: [12345] isn't 3 or fewer characters

too_short: failed three_digit_odd_number
    short_number: _value_length: [13] isn't 3 or more characters

even_number: failed three_digit_odd_number
    even_number: _matches_regex: [24] did not match the pattern
    even_number: _value_length: [24] isn't 3 or more characters

two_fields: failed twofer
    even_number: _matches_regex: [24] did not match the pattern
    short_number: _value_length: [13] isn't 3 or more characters
```

The brick interface

Closes over setup data

Has access to all input

True if everything is okay

die with a reference if it isn't

A validation routine

```
my $sub = sub {
   my $input = shift;
   return 1 if exists $input->{cat};
   die { # result error message
      handler      => 'Cat key check',
      failed_field => 'cat'
      message      => "No field named 'cat'",
      };
   }
```

Add to bucket

Put it in the communal bucket

Use the brick in different relationships

```
$brick = $bucket->add_to_bucket( {
  name        => 'cat key checker',
  description => "Has field named 'cat'",
  code        => $sub
  } );
```

Compose bricks

```
sub _us_postal_code_format
  {
  my( $bucket, $setup ) = @_;

  $setup->{exact_length} = 5;

  my $composed = $bucket->__compose_satisfy_all(
    $bucket->_value_length_is_exactly( $setup ),
    $bucket->_is_only_decimal_digits( $setup ),
    );
  }
```

Make trees

```
my $postal  = $brick->_postal_code_format( { ... } );
my $street  = $brick->_address_format( { ... } );
my $usps    = $brick->_usps_check( { ... } );

my $address = $brick->__compose_satisfy_all(
  $postal, $street, $usps );

my $basket  = $brick->__compose_satisfy_all( ... );


my $order = $brick->__compose_satisfy_all(
  $address, $basket, ... );
```

Validation profile

```
some_label
__compose_AND
  __compose_ONE_OF
                  __fields_are_something
                  __compose_AND
                          __compose_AND
                                  _value_length_is_equal_to_greater_than
                                  _value_length_is_equal_to_less_than
                          _is_only_decimal_digits
                                  _is_only_decimal_digits
          __compose_ONE_OF
                  __fields_are_something
                  __compose_AND
                          _is_YYYYMMDD_date_format
                          _is_valid_date
          __compose_ONE_OF
                  __fields_are_something
                  __compose_AND
                          _is_YYYYMMDD_date_format
                          _is_valid_date
```

Get the results

```
foreach my $item ( @profile ) {
   my $label  = $item->[0];
   my $method = $item->[1];
   my $result =
      eval{ $brick->$method->( $input ) }
   my $eval_error = $@;
   $result = 0 if ref $eval_error;
   push @results,
      [ $label, $method, $result, $@ ];
   }
```

How to use Brick

Plug-in validation (MVC)

Subclass to adapt

Store all business logic separately

Didn’t cover...

Filters

Selectors

Subclasses

Configuration as code

Conclusion

Many-to-many relationships

Descriptive error messages

Replay validation