MooseX::Declare

I'm here today to talk about MooseX::Declare. Hopefully everyone here knows what Moose is, but perhaps you don't know about MooseX::Declare.

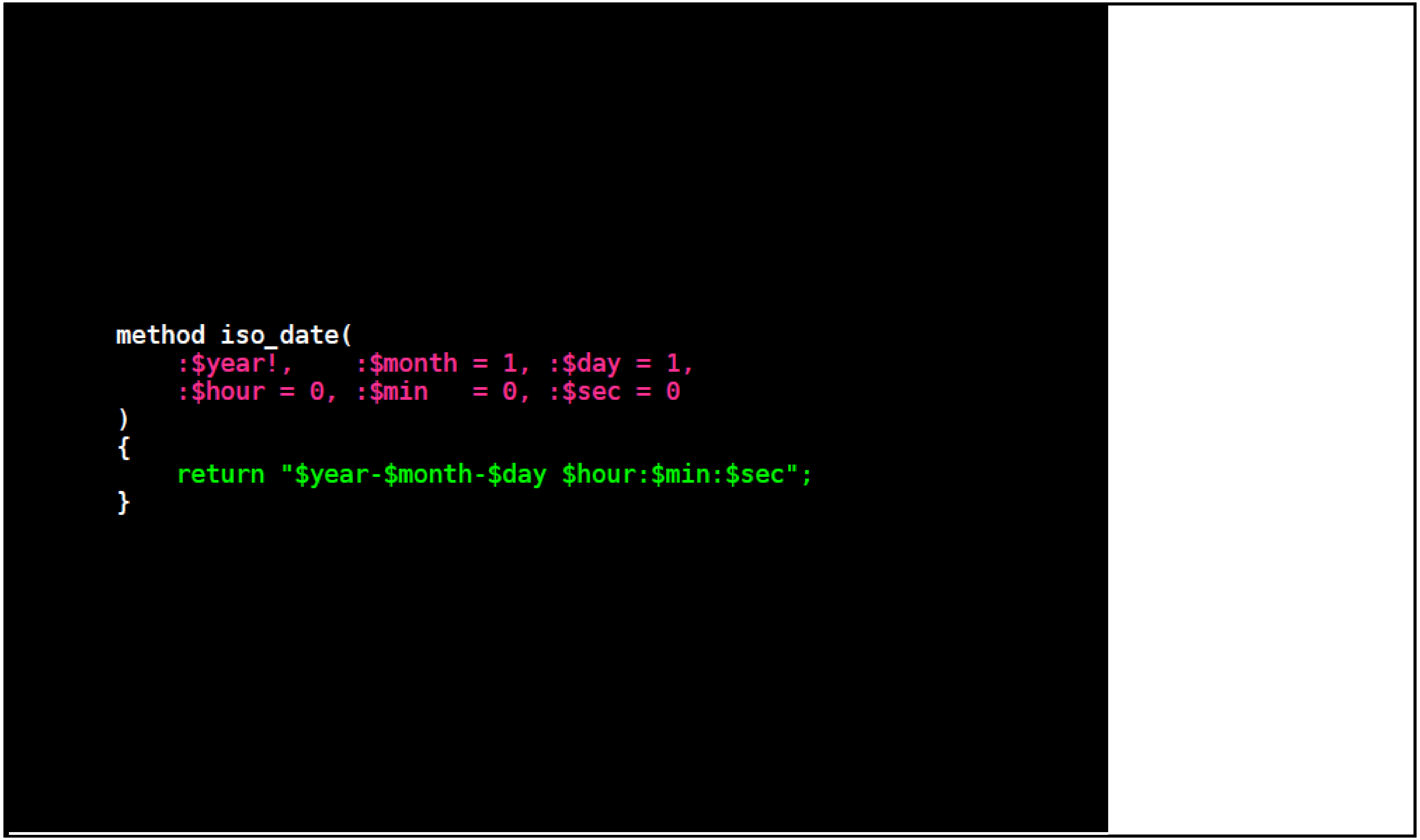
```perl
package MyCompany::Bar;

use Moose;
use MooseX::Params::Validate;

use namespace::autoclean;

extends 'MyCompany::Foo';

sub adjusted_rate
{
    my ($self, $type, $discount) = validated_list(
            \@_,
            type        =>  { isa => 'Str' },
            discount    =>  { isa => 'Int', default => 0 }
    );

    return $self->rate($type) * (1 - $discount / 100);
}

around 'total' => sub
{
    my $orig = shift;
    my $self = shift;
    my ($rate_type, @costs) = pos_validated_list(
            \@_,
            { isa => 'Str' },
            MX_PARAMS_VALIDATE_ALLOW_EXTRA => 1,
    );

    my $total =  $self->$orig(@costs)
            * $self->adjusted_rate(
                    type        =>  $rate_type,
                    discount    =>  $self->current_discount
            );
    return $total;
};

__PACKAGE__->meta->make_immutable;

1;
```

Here's some Moose code.

Now, if you've ever had to roll your own OO in Perl, this is pretty nice. Especially if you've had to roll a lot of it. But could it be nicer?

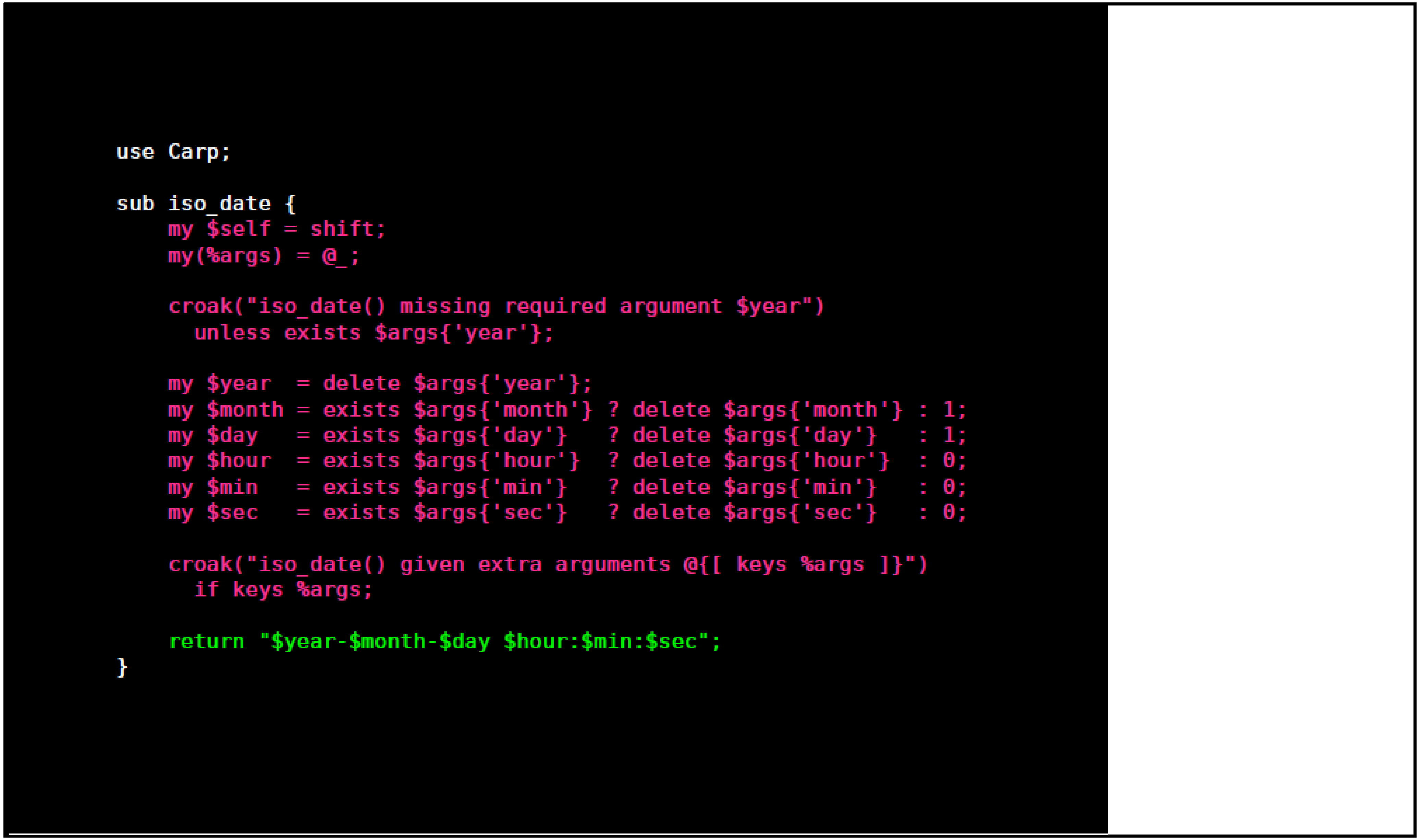
```perl
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

This is the same code in MooseX::Declare. Everything's a lot clearer and more concise. It's more obvious here that we're declaring a Moose class and that it's a subclass, and it's obvious where the class declaration ends. The "magical Moose incantations" of `namespace::autoclean` and `make_immutable` are gone, because MooseX::Declare does those things for you. But possibly the most awesome thing is that we have method signatures. Now, these signatures are based on the Perl 6 method signatures syntax: it's not 100% compatible, but it's pretty darn close. And in trying to explain how cool method signatures are, I can't really do any better than these slides that I stole from schwern's presentation on Method::Signatures—Method::Signatures has basically the same signature syntax as MooseX::Declare, but it's designed for non-Moose code.



```
method iso_date(
    :$year!,    :$month = 1, :$day = 1,
    :$hour = 0, :$min   = 0, :$sec = 0
)
{
    return "$year-$month-$day $hour:$min:$sec";
}
```

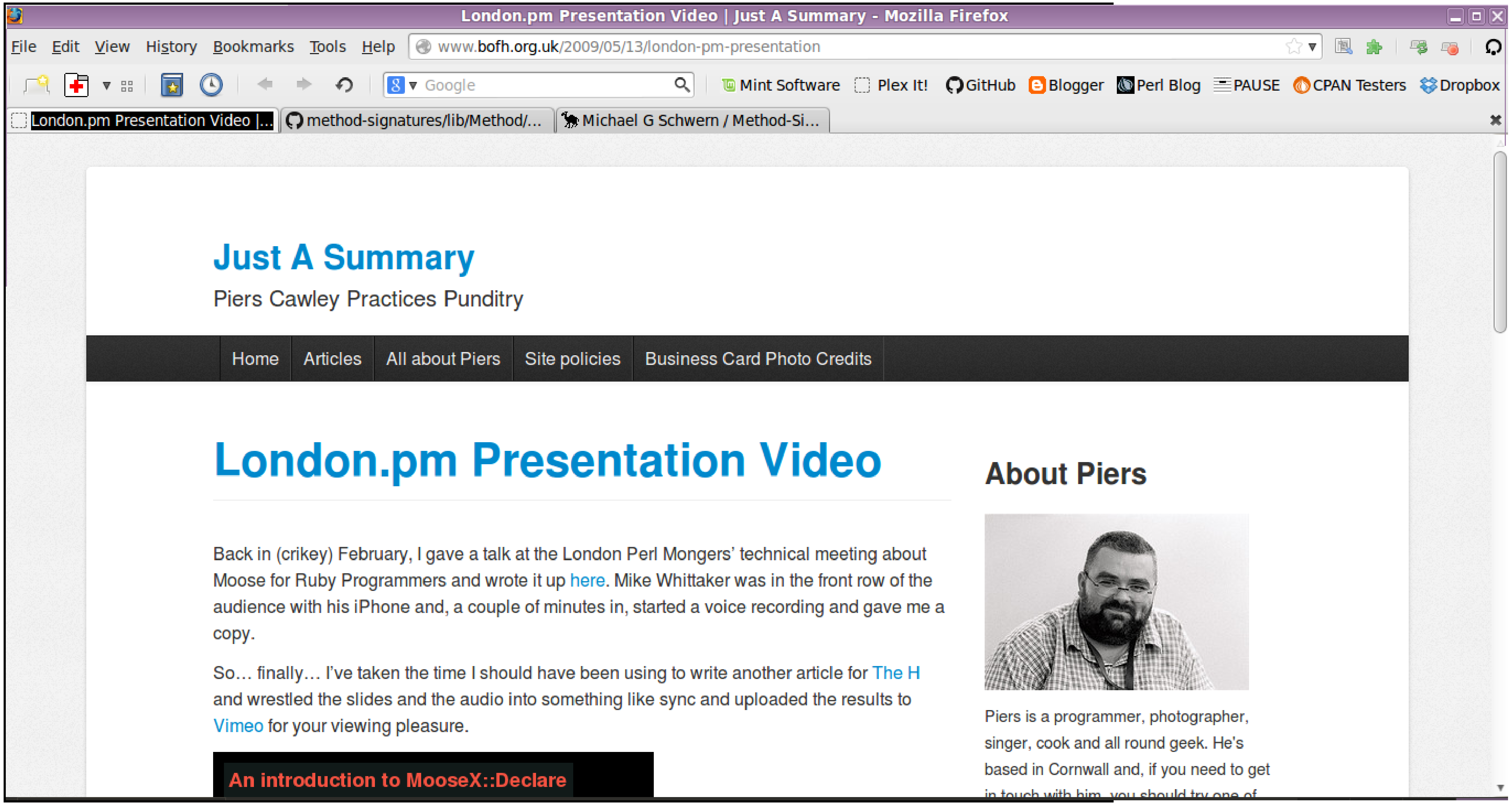
Anyway, as schwern points out, here's some code using a method signature.



```
use Carp;

sub iso_date {
    my $self = shift;
    my(%args) = @_;

    croak("iso_date() missing required argument $year")
      unless exists $args{'year'};

    my $year  = delete $args{'year'};
    my $month = exists $args{'month'} ? delete $args{'month'} : 1;
    my $day   = exists $args{'day'}   ? delete $args{'day'}   : 1;
    my $hour  = exists $args{'hour'}  ? delete $args{'hour'}  : 0;
    my $min   = exists $args{'min'}   ? delete $args{'min'}   : 0;
    my $sec   = exists $args{'sec'}   ? delete $args{'sec'}   : 0;

    croak("iso_date() given extra arguments @{[ keys %args ]}")
      if keys %args;

    return "$year-$month-$day $hour:$min:$sec";
}
```

And here's the same code w ithout the signature.

As you can see, the real w ork of the function just gets overw helmed by the parameter validation code. But if you still need more convincing about just how cool MooseX::Declare is, you should check out


London.pm Presentation Video | Just A Summary - Mozilla Firefox
File Edit View History Bookmarks Tools Help
www.bofh.org.uk/2009/05/13/london-pm-presentation
Google
Mint Software Plex It! GitHub Blogger Perl Blog PAUSE CPAN Testers Dropbox
London.pm Presentation Video |...
method-signatures/lib/Method/...
Michael G Schwern / Method-Si...
Home
Articles
All about Piers
Site policies
Business Card Photo Credits

Just A Summary

Piers Cawley Practices Punditry

London.pm Presentation Video

Back in (crikey) February, I gave a talk at the London Perl Mongers' technical meeting about Moose for Ruby Programmers and wrote it up here. Mike Whittaker was in the front row of the audience with his iPhone and, a couple of minutes in, started a voice recording and gave me a copy.

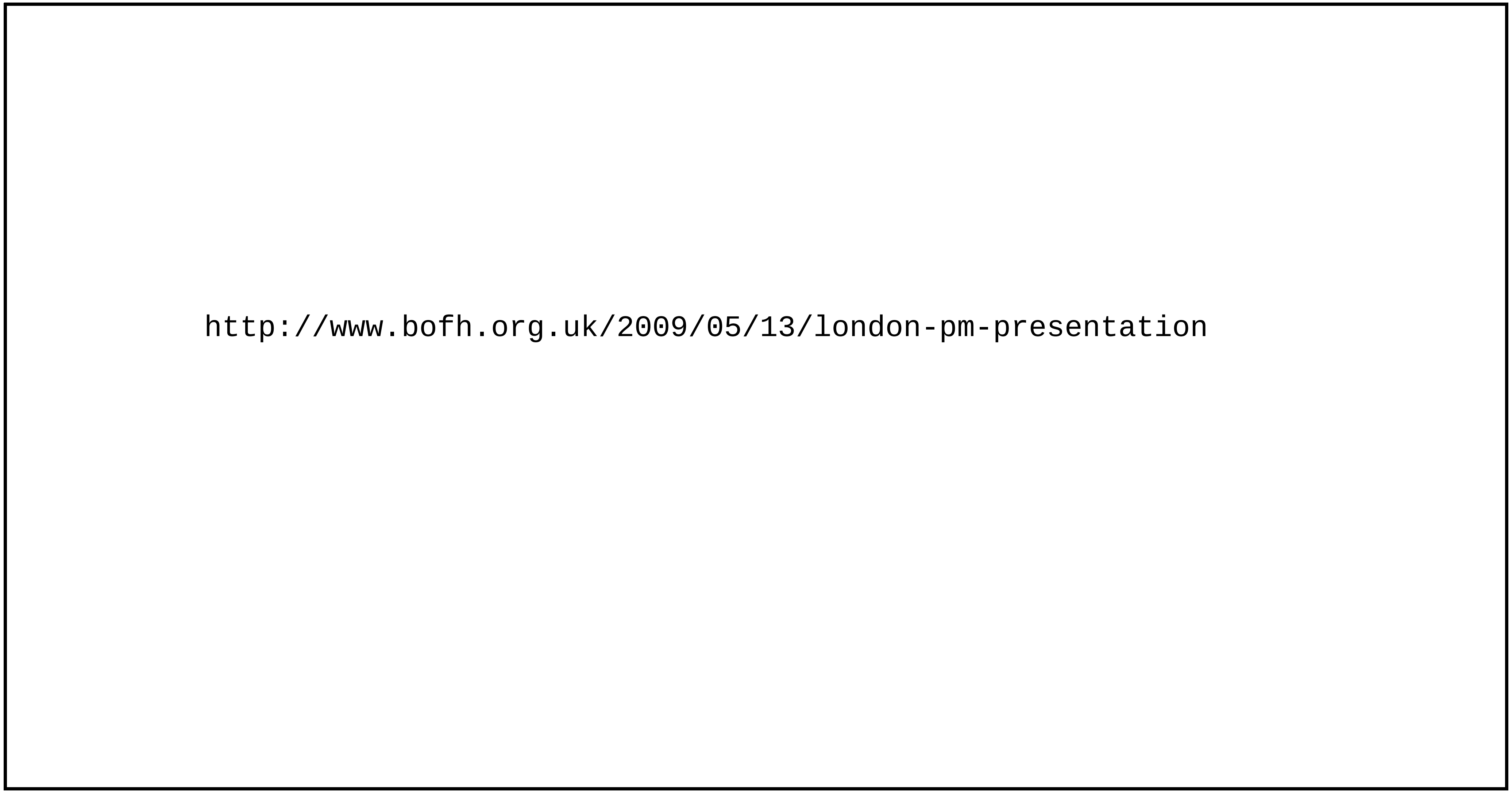
So… finally… I've taken the time I should have been using to write another article for The H and wrestled the slides and the audio into something like sync and uploaded the results to Vimeo for your viewing pleasure.

An introduction to MooseX::Declare

About Piers

Piers is a programmer, photographer, singer, cook and all round geek. He's based in Cornwall and, if you need to get

this talk that Piers Cawley gave at a London Perl Mongers group; he explains better than I ever could how this is declarative programming, and why that's more readable and more maintainable. So I highly recommend you



`http://www.bofh.org.uk/2009/05/13/london-pm-presentation`

check it out (except don't crank it up too loud at work, 'cause he drops a few F-bombs in there) and see for yourself. Because I'm not actually here to answer the question "How cool is MooseX::Declare?" I'm here to answer the question: "If this thing is so damn cool, how come nobody actually uses it?"

Now, there's a perception that MooseX::Declare is still experimental. But it's been out for nearly three years now. Of course, it's not really just the passage of time that makes someting less experimental: it's the amount of market penetration. In other words, it's not the case that people aren't using it because it's still experimental—it's rather that it's still experimental because not enough people are using it. So there must be something else going on here.

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

So, what if I told you there *might* be a bug in this code? You might think I was being coy: challenging you to stare at this code and try to pick out a bug that might or might not be there. But that's not what I'm saying at all. There's *definitely* a bug in this code ... but only sometimes. If you run this code—or, more accurately, if you run some code that *calls* this code—on, let's say, Perl 5.8.9, it works fine. Or, possibly, Perl 5.10.0: works fine there too. But if you run it on, let's say, Perl 5.12.3 ...

```
Can't locate object method "adjusted_rate" via package "MyCompany::Bar" at ./bro
ken.pl line 7.
```

... kaboom. Now, I can show you how to fix this bug—or, technically, how to change this code so it no longer triggers the bug.

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

So I'm going to show you that now: ready? Pay attention now. Shift ... J.

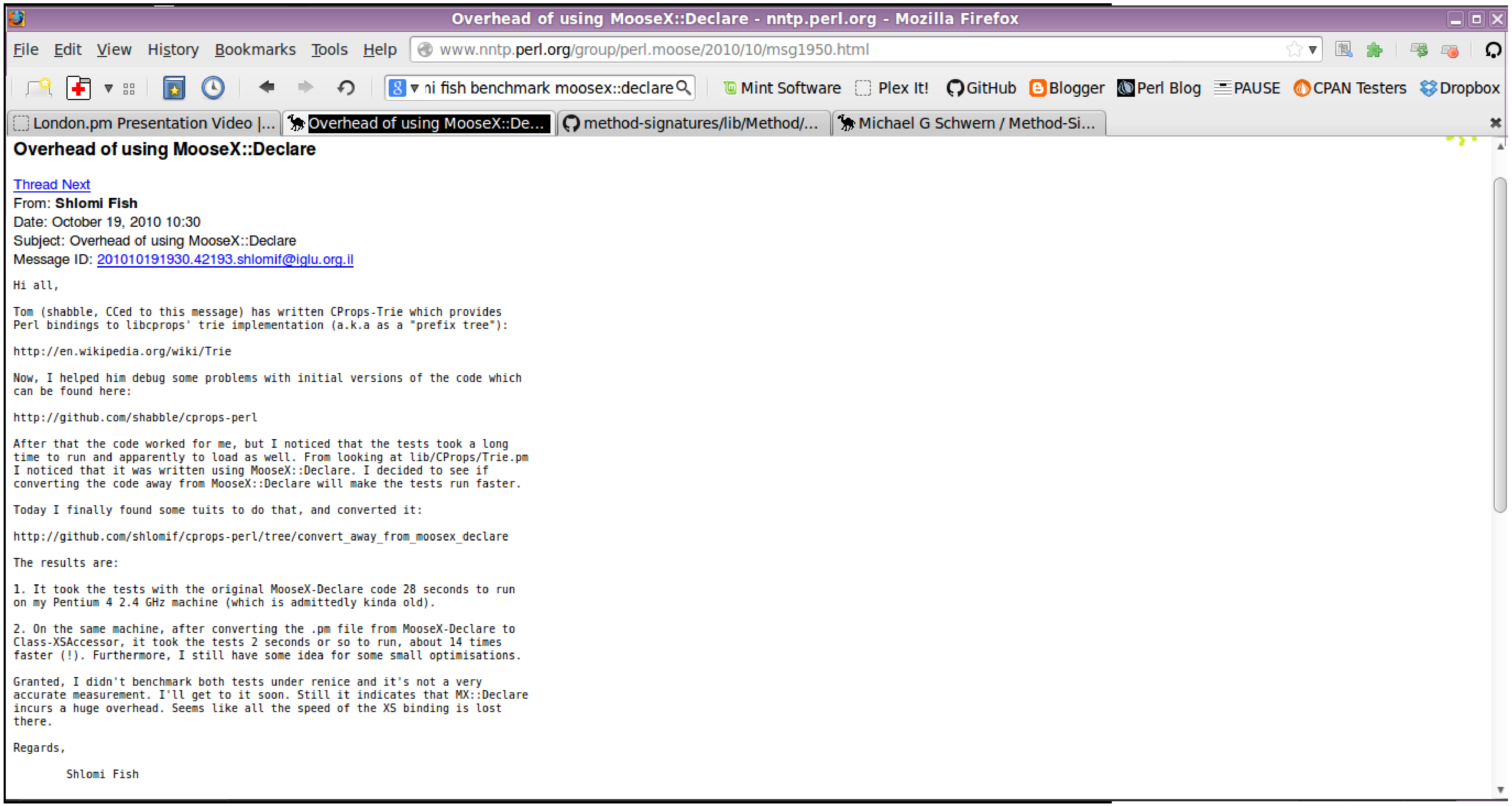
```perl
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0) {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

Done! Yes, that's right: this is a bug that's not only Perl-version-specific, but also whitespace-dependent. And if you didn't just shudder ... Well, you probably should have.

1. Stability Problems

So it's buggy. But MooseX::Declare was around for a year and a half before Perl 5.12 came out, and obviously no one was using it even then. I say "obviously" because, if anyone *had* been using it, the bug would have been fixed by now. So there must be something else going on here.

This


Overhead of using MooseX::Declare - nntp.perl.org - Mozilla Firefox
File Edit View History Bookmarks Tools Help
www.nntp.perl.org/group/perl.moose/2010/10/msg1950.html
ni fish benchmark moosex::declare
Mint Software Plex It! GitHub Blogger Perl Blog PAUSE CPAN Testers Dropbox
London.pm Presentation Video |...
Overhead of using MooseX::De...
method-signatures/lib/Method/...
Michael G Schwern / Method-Si...

Overhead of using MooseX::Declare

Thread Next
From: **Shlomi Fish**
Date: October 19, 2010 10:30
Subject: Overhead of using MooseX::Declare
Message ID: 201010191930.42193.shlomif@iglu.org.il

```
Hi all,

Tom (shabble, CCed to this message) has written CProps-Trie which provides
Perl bindings to libcprops' trie implementation (a.k.a as a "prefix tree"):

http://en.wikipedia.org/wiki/Trie

Now, I helped him debug some problems with initial versions of the code which
can be found here:

http://github.com/shabble/cprops-perl

After that the code worked for me, but I noticed that the tests took a long
time to run and apparently to load as well. From looking at lib/CProps/Trie.pm
I noticed that it was written using MooseX::Declare. I decided to see if
converting the code away from MooseX::Declare will make the tests run faster.

Today I finally found some tuits to do that, and converted it:

http://github.com/shlomif/cprops-perl/tree/convert_away_from_moosex_declare

The results are:

1. It took the tests with the original MooseX-Declare code 28 seconds to run
on my Pentium 4 2.4 GHz machine (which is admittedly kinda old).

2. On the same machine, after converting the .pm file from MooseX-Declare to
Class-XSAccessor, it took the tests 2 seconds or so to run, about 14 times
faster (!). Furthermore, I still have some idea for some small optimisations.

Granted, I didn't benchmark both tests under renice and it's not a very
accurate measurement. I'll get to it soon. Still it indicates that MX::Declare
incurs a huge overhead. Seems like all the speed of the XS binding is lost
there.

Regards,

        Shlomi Fish
```

is a post on the Moose mailing list from Shlomi Fish talking about the performance of MooseX::Declare. And it's a very common complaint: you can find dozens of similar posts by people saying something like "I benchmarked method calls in MooseX::Declare against plain Moose and it's 4,000 times slower!" And this leads to discussions of various lengths (and intensities) about the slippery nature of interpreting benchmarks. Are you using MooseX::Params::Validate—which is the canonical answer on mailing lists to doing parameter validation outside of MooseX::Declare—in your "plain" Moose code? No? Then you're comparing apples to oranges. What do the methods in your benchmarks actually do? Nothing? So that means that 100% of the work of your methods is validating their parameters. That's not even remotely realistic. So at the end of all this back-and-forth, it's generally agreed that no, MooseX::Declare is not 4,000 times slower than plain Moose, it's only 40 times slower. Which is much better, right?

And this is what the people who actually *like* MooseX::Declare will tell you. This, they will say, is just the price you pay for syntactical sugar. There's no way around it.

```
1. Stability Problems
2. Performance Problems
```

So it's buggy, *and* it's slow. Well, that's probably enough nails in the coffin right there. But wait! there's more!

```
1. Stability Problems
2. Performance Problems
3. ???
```

What is the point of parameter validation anyway? Are we discriminating against certain parameters? Non-integers need not apply? No, of course not. The point of parameter validation is to catch mistakes. Because sooner or later, some value coming into that function is going to be something it shouldn't, because someone—not you, of course, but *someone*—made a mistake and stuck something in there that they shouldn't have. And the sooner you find that out, the better chance you have of being able to fix it. So if parameter validation doesn't help us find mistakes, there's no point in doing it at all. In other words, what it does when you pass parameters correctly is actually less interesting than what it does when you pass them *incorrectly*.

So since MooseX::Params::Validate is **the** way to do parameter validation outside of MooseX::Declare, let's see what happens when you pass, say, a floating point value to something expecting an integer.

```
The 'discount' parameter ("8.5") to MyCompany::Bar::adjusted_rate did not pass th
e 'checking type constraint for Int' callback
 at /usr/local/share/perl5/MooseX/Params/Validate.pm line 118
     MooseX::Params::Validate::validated_list('ARRAY(0x2d7d690)', 'type', 'HA
SH(0x2260d80)', 'discount', 'HASH(0x227de28)') called at MooseExample.pm line 13
     MyCompany::Bar::adjusted_rate(undef, 'type', 'sales', 'discount', 8.5) c
alled at ./good_err.pl line 15
     main::foo() called at ./good_err.pl line 9
```

Well, this is not too bad, as error messages go. It tells us what method we called, what parameter we screwed up, and how we screwed it up. Now, it could certainly be better—the full stack trace means we have to go hunting for where we actually called the problematic method, and the text of the error is not as clear as it might be—but, overall, this mostly gets the job done.

How about the same thing in MooseX::Declare?

```
Validation failed for 'Tuple[Tuple[Object],Dict[type,Str,discount,Optional[Int]]
]' with value [ [ MyCompany::Bar=HASH(0x107f008) ], { discount: 8.5, type: "sale
s" } ], Internal Validation Error is:
 [+] Validation failed for 'Dict[type,Str,discount,Optional[Int]]' with value {
discount: 8.5, type: "sales" }
  [+] Validation failed for 'Optional[Int]' with value 8.5 at /usr/local/share/p
erl5/MooseX/Method/Signatures/Meta/Method.pm line 435
     MooseX::Method::Signatures::Meta::Method::validate('MooseX::Method::Sign
atures::Meta::Method=HASH(0x2987c60)', 'ARRAY(0x2963178)') called at /usr/local/
share/perl5/MooseX/Method/Signatures/Meta/Method.pm line 151
     MyCompany::Bar::adjusted_rate('MyCompany::Bar=HASH(0x107f008)', 'type',
'sales', 'discount', 8.5) called at ./bad_err.pl line 15
     main::foo() called at ./bad_err.pl line 9
```

This is a little more imposing. Let's see if we can parse it. "Validation failed" ... a promising beginning. Then there's some stuff about tuples ... I thought that was a database thing? And then the parameters, but it's all of them, so you can't tell which one was the bad one. And then the same for the values: all of them. Ah, perhaps this will help: "Internal Validation Error is" ... "Validation failed." It's like we're trapped on the Mobius strip of error messages. For purposes of helping us figure out what went wrong, this is an epic fail. And that was the entire point of parameter validation.

1. Stability Problems
2. Performance Problems
3. Hard to Debug

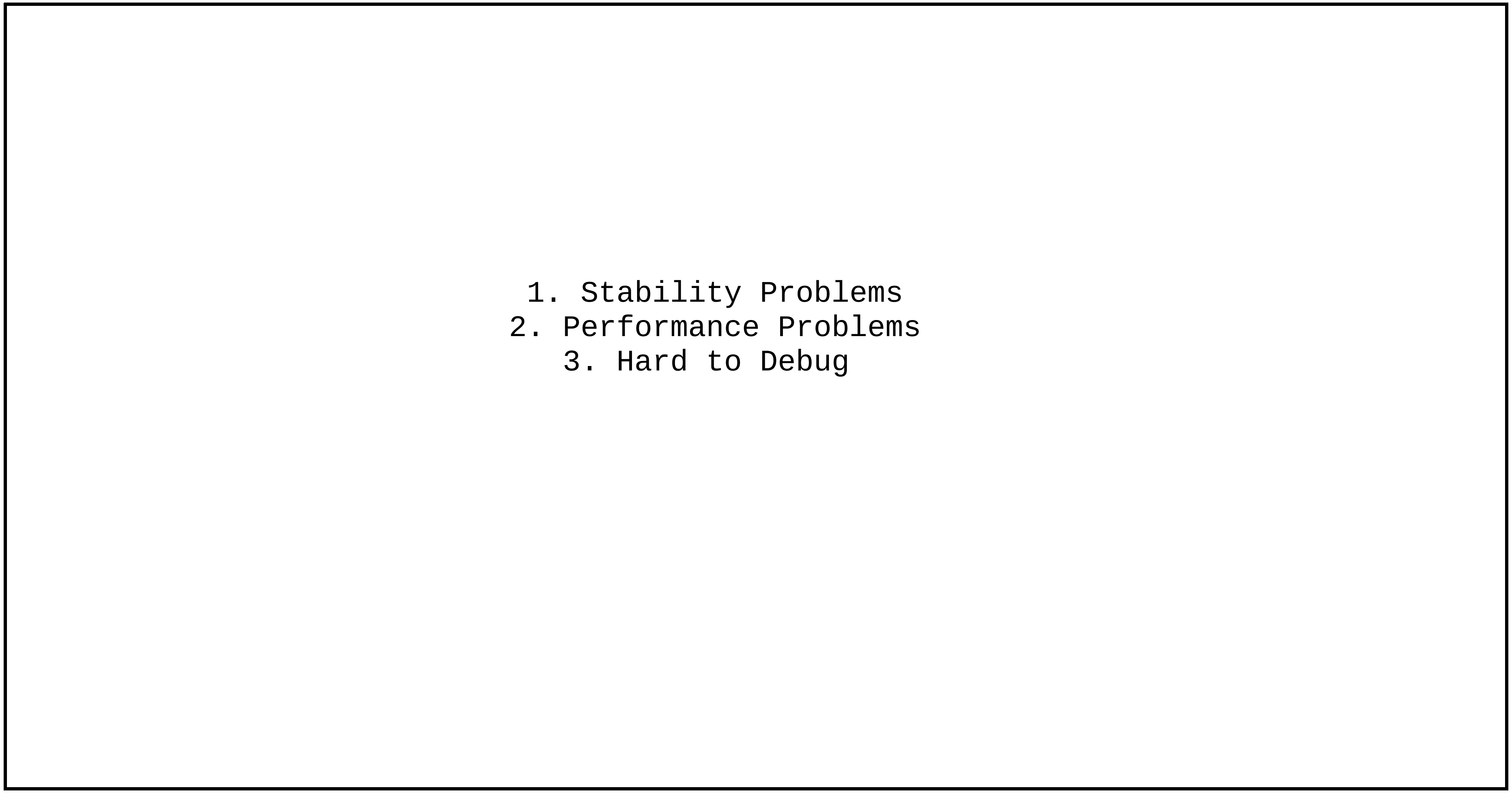
So it's buggy, it's slow, *and* it has crappy error messages. There's really only one thing you can do.

```
Conclusion:
Never use MooseX::Declare.
```

No wonder no one uses it. You're crusing along the Internet, and you see this:

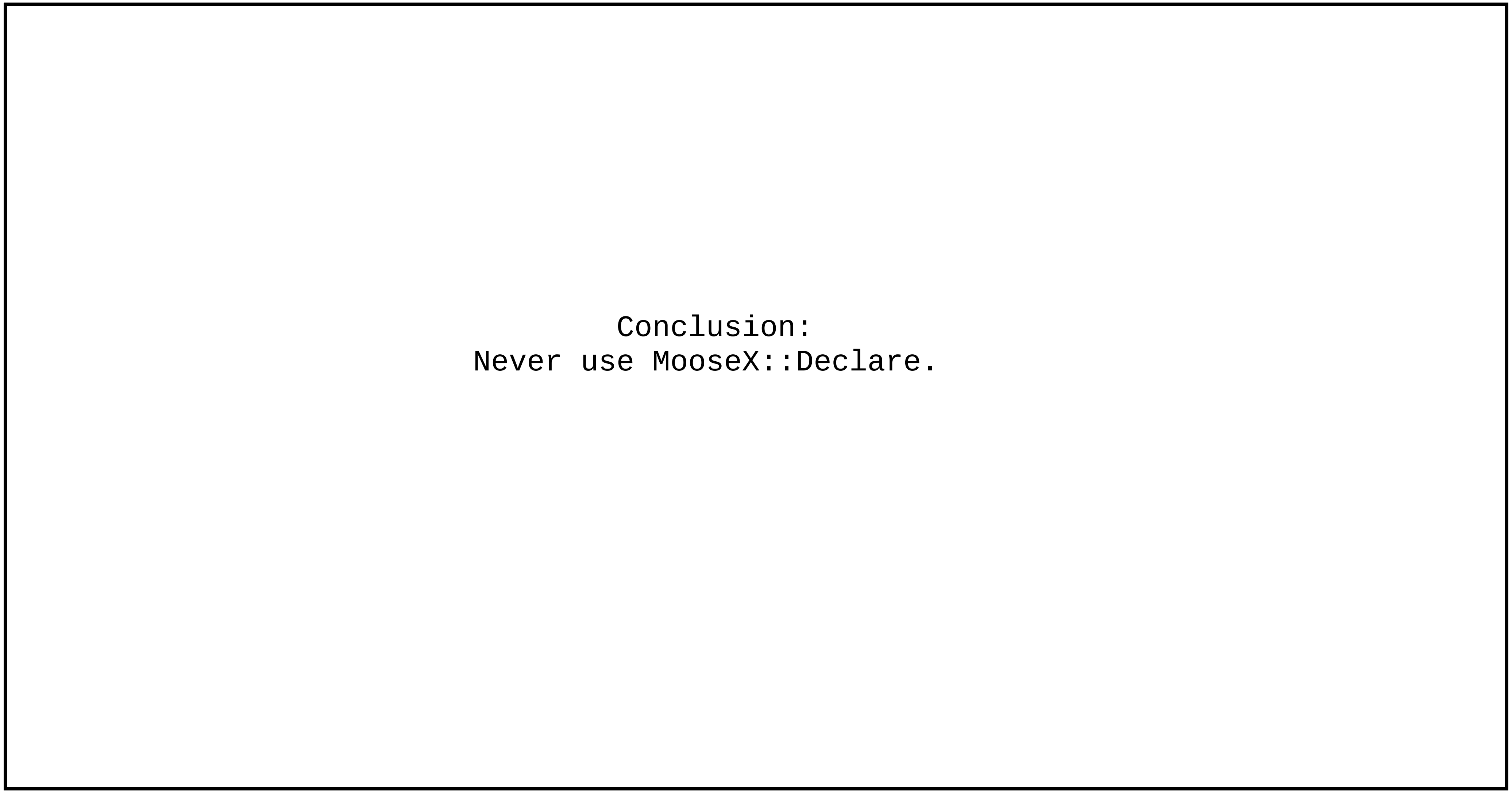
```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

and you say "Hey, that looks pretty nifty" and, being an intelligent programmer, you spend a little time Googling, maybe talk to a few people, and you find this:



1. Stability Problems
2. Performance Problems
3. Hard to Debug

which leads you to this:



```
Conclusion:
Never use MooseX::Declare.
```

and you congratulate yourself on a headache averted. That's what you do, I say, if you're intelligent programmer. If you're, on the other hand, let's say, *me*, you see this:

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

and you say "ooooh, preeetty cooooode" and you immediately start using it. At work. In production.

Which is maybe not so bad, because perhaps production is still using Perl 5.8, so there's no bug, and it's not performance-critical code, so if it's a little slower than it might be, it's not really noticeable, and as for the crappy error messages ... well, you just train yourself to read them. And for everyone else at work,

<insert wiki page image here>

you make a wiki page. And so everything is fine and dandy until one day your boss—or, hypothetically speaking, your boss's boss—decides that's it's high time you upgraded to Perl 5.12, so he fires up a sandbox, slaps 5.12 on it, starts running the test suite, and

```
Can't locate object method "adjusted_rate" via package "MyCompany::Bar" at ./bro
ken.pl line 7.
```

kaboom. So, being an intelligent programmer, he goes out on the 'Net, and he finds this:

1. Stability Problems
2. Performance Problems
3. Hard to Debug

which leads him to this:

```
Conclusion:
Never use MooseX::Declare.
```

and he comes to you and says: "Dude, WTF?" To which you articulately reply: "but, but, but ..."

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

"... preeetty cooode ..."

So, yeah, that's what happened to me. Now, maybe I'm not an intelligent programmer, but I do have one thing going for me: I'm stubborn. Well, let's put it this way: Larry Wall tells us that one of the three great programmerly virtues is hubris. So I'm not stubborn; I'm ... hubrisy. Hubrisious? Hubrisical? Whatever; I'm that. So I said: "No! I will not give up my pretty code." This is Open Source World, right? If there's a problem, we fix it!

```
Brilliant Idea #1:
Fix MooseX::Declare
```

So I started looking at the problems.

1. Stability Problems
2. Performance Problems
3. Hard to Debug

First the bug. Now, remember in this code

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0) {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

where I said you had to put this curly brace on the previous line to avoid the bug? What about this curly brace? Do you have to "fix" that one too?

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0) {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs) {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

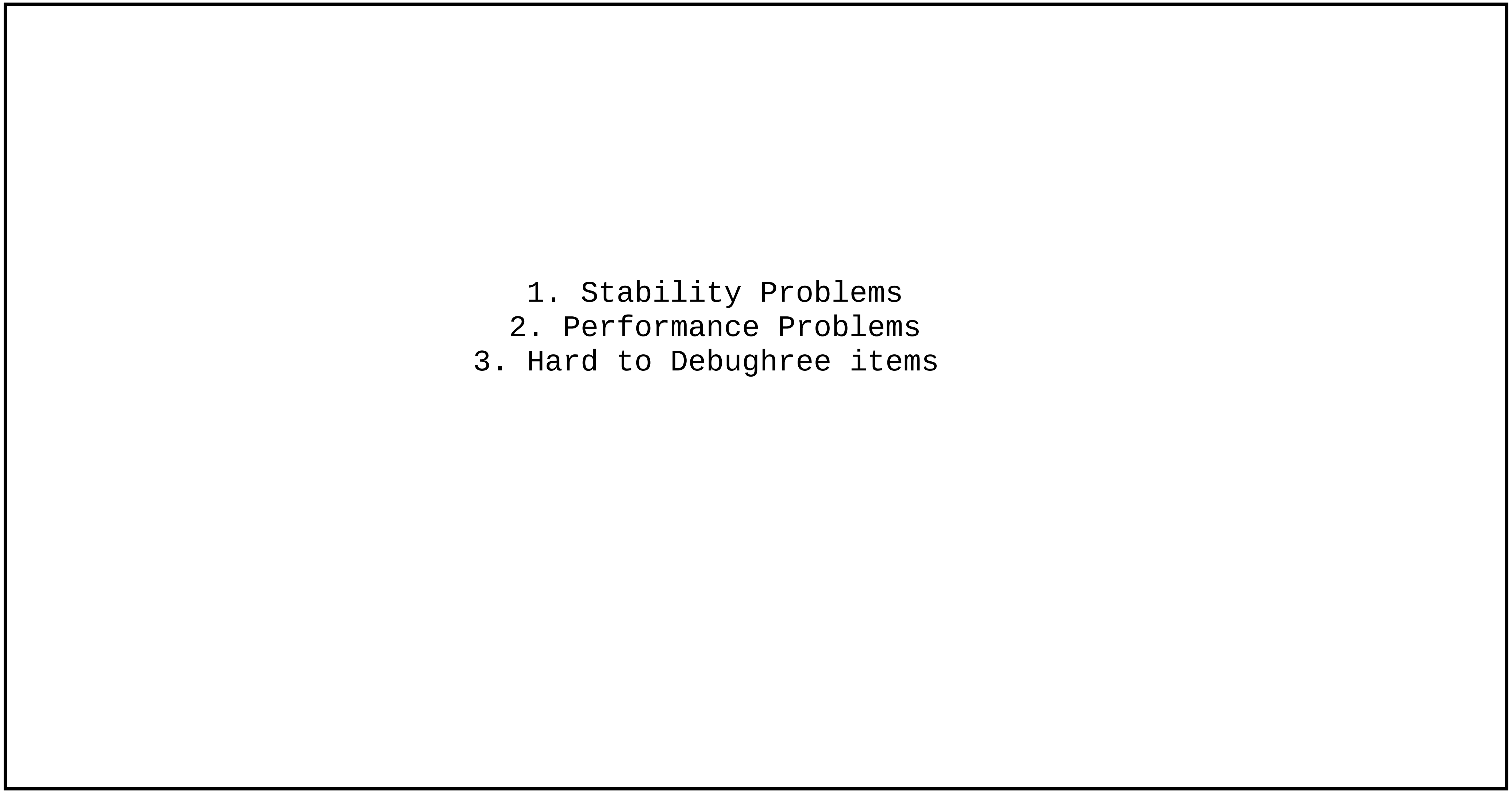
As it turns out, yes, you do. Well, what about *this* curly brace? Do you have to put that one up here?

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo {
    method adjusted_rate (Str :$type!, Int :$discount = 0) {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs) {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

As it turns out, no. You don't.

```
use MooseX::Declare;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0) {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs) {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

It seems that this whitespace problem only applies to lines that have a method signature on them.



```
1. Stability Problems
2. Performance Problems
3. Hard to Debughree items
```

What about the performance problems? Well, it turns out that all these posts on the Internet are about benchmarking calls to methods ... with method signatures. And where are the crappy error messages coming from? From failing a type check on a method ... with a method signature.

I'm starting to see a pattern here.

And it turns out that MooseX::Declare doesn't even **do** method signatures. It farms out that work to another module called

MooseX::Method::Signatures

(a.k.a. MXMS)

MooseX::Method::Signatures, or MXMS for short. So, as it turns out, there's NOTHING WRONG WITH MOOSEX::DECLARE. All the problems are with MXMS. So ... next brilliant idea.

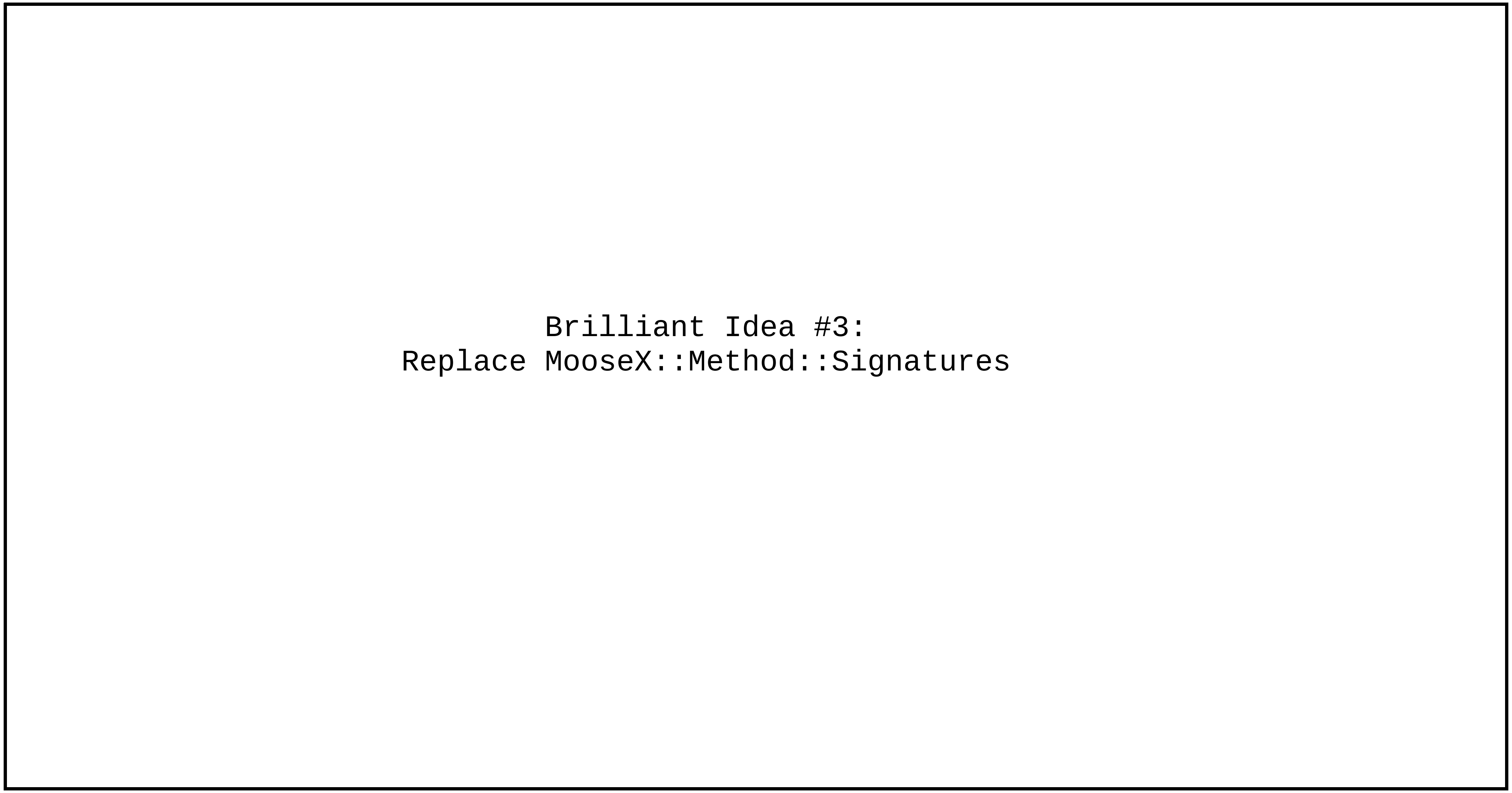
Brilliant Idea #2:
Fix MooseX::Method::Signatures

Fix MXMS. Now, I gotta tell you: I tried. I really did. But this code was written partially by Matt Trout, of DBIC fame, and partially by Florian Ragwitz, one of the big Moose guys, and these guys are just plain brillianter than me. So, after beating my head against this code for the equivalent of a few days (spread out over a few weeks), I came to this conclusion: I have faith that, given enough time—say, a year, maybe two at the most—I **could** understand this code well enough to fix the bug. But there's two problems with that.

First, one of the other of the three great programmerly virtues that Larry tells us about is impatience: I just don't have that kind of time.

And the second problem is, the bug showed up with Perl 5.12. But the performance problems were there before that. That means that fixing the bug is almost certainly not going to help the performance. And performance problems are tricky: sure, sometimes you find that one magical line that's slowing the whole thing down, and you fix that, and then everything's fine. But, more often, you find that the slowness is an endemic problem, baked into the design. And that means that I might spend all this time trying to fix the bug and still be in the position of needing to completely rewrite MXMS. And what's the last of the great programmerly virtues? That's right: laziness.

Well, if you can't fix it ...



Brilliant Idea #3:
Replace MooseX::Method::Signatures

... replace it.

So now all I need is something that does everything that MXMS does. And I already mentioned something, back at the beginning of the presentation:

Method::Signatures

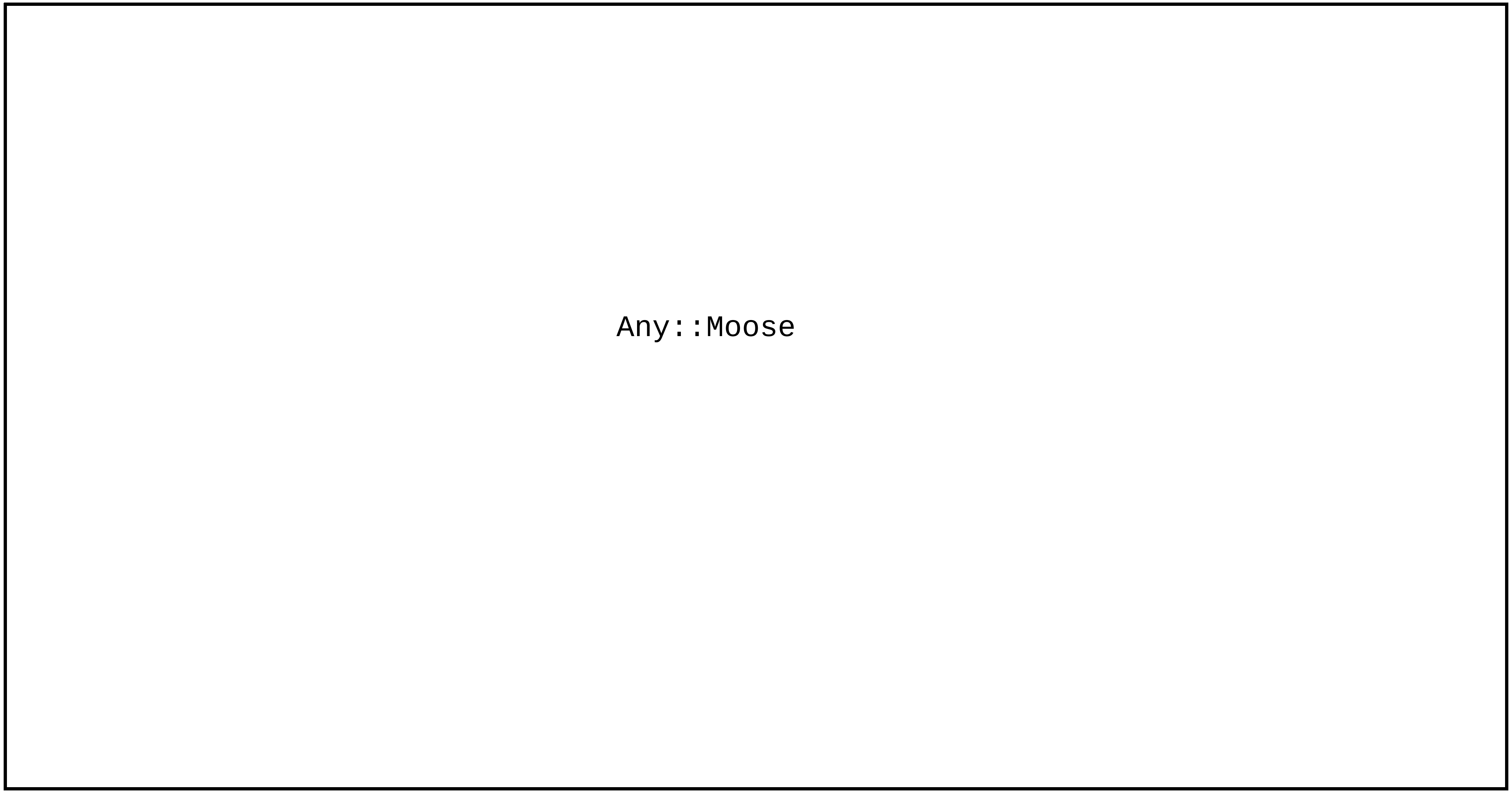
(by schwern)

Method::Signatures, by schwern. It has almost all the features that MXMS has. There are a few, more obscure, features that it's lacking, like parameter aliasing and where constraints, but I wasn't using any of those anyway. There is one big difference between MXMS and Method::Signatures though: types. MXMS was designed for Moose, so of course it understands Moose types. Method::Signatures was designed for non-Moose code, so of course it doesn't know anything about Moose types.

So I decided to write schwern an email. Never met the man in person, but I'd exchanged posts with him on various mailing lists, and he seemed like a nice enough guy. So I wrote him and said, basically, "Hey, dude, I'm thinking about extending Method::Signatures, perhaps with a subclass, so that it can do type checking. And I was looking at your code, and it looks like there's a couple of places where I might need to submit some patches for Method::Signatures to make it work. So I was wondering if you'd be into that." And, two days later, he's writing me back saying "Screw the subclass! Just put the type checking directly into Method::Signatures. In fact, I've already added the type *parsing* to the code on GitHub. All you have to do is put the actual type checking in there."

So I said: "Cool!"

And we talked about how to make sure that people who don't want type checking don't have to pay for it. The type checking would all be done at run-time, so, if your method signatures don't use any types, you don't add any extra overhead. And we decided that, to avoid a module dependency on Moose, we'd use



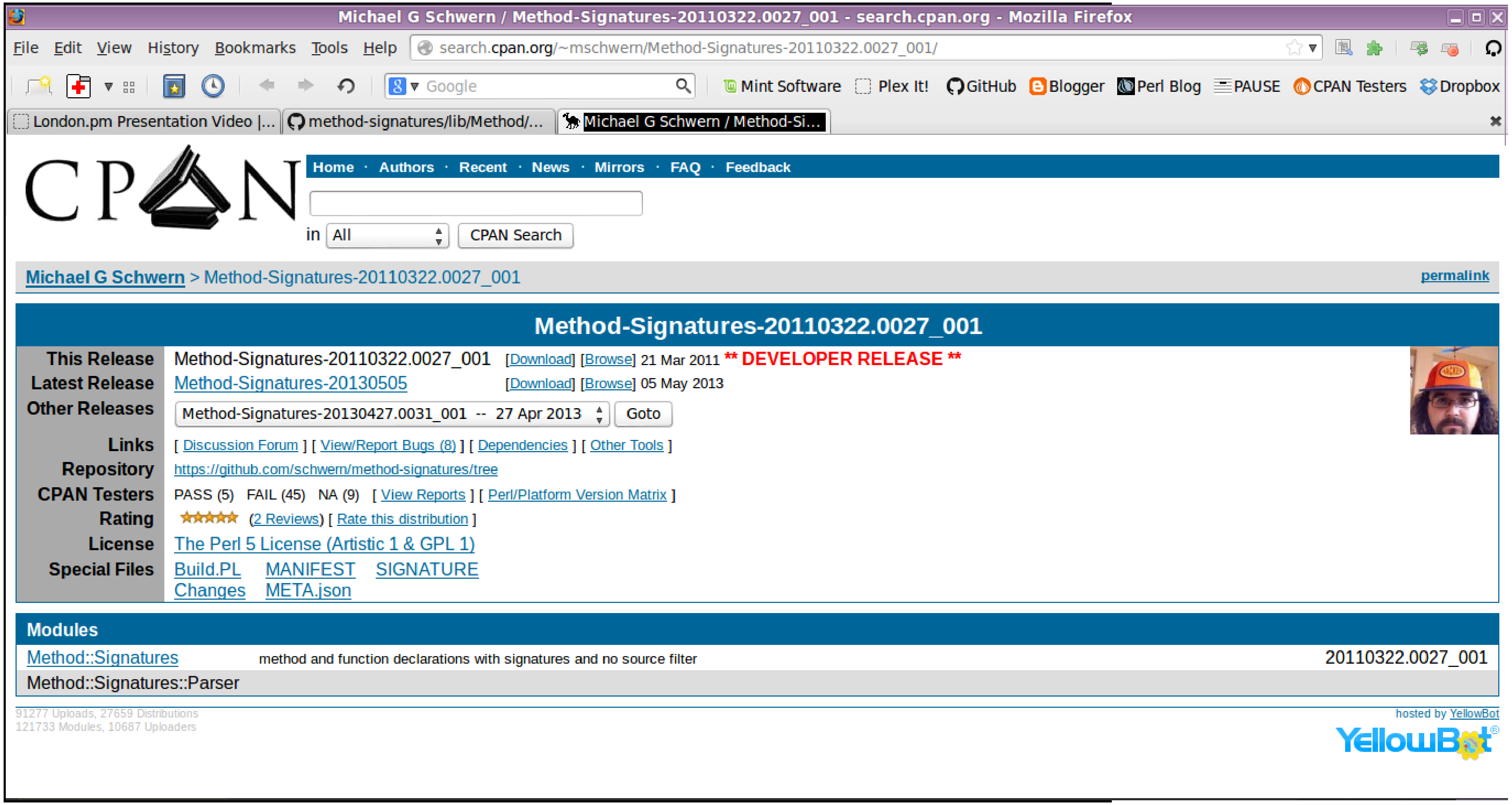
`Any::Moose`

Any::Moose. If you're not familiar with that, what it does is, when you load it up, it pokes its head up and says, "Is Moose already loaded? If so, I'll use Moose. If not, I'll load Mouse."

Mouse

And if you're not familiar with Mouse, it's like Moose light. For a visual approximation, try this: Imagine a moose. Now imagine a mouse. See? So it's actually like Moose super-light. It doesn't do everything Moose does, but it does a lot of it, and it's faster and uses less memory. So it's small dependency, which is great for when you don't actually need everything that Moose does.

So I implemented all the type checking, and then we did a code review, using the collaboration tools on GitHub (highly recommended, by the way), and now Method::Signatures does type checking.


Michael G Schwern / Method-Signatures-20110322.0027_001 - search.cpan.org - Mozilla Firefox

search.cpan.org/~mschwern/Method-Signatures-20110322.0027_001/

Michael G Schwern > Method-Signatures-20110322.0027_001 permalink

Method-Signatures-20110322.0027_001

This Release	Method-Signatures-20110322.0027_001 [Download] [Browse] 21 Mar 2011 ** DEVELOPER RELEASE **
Latest Release	Method-Signatures-20130505 [Download] [Browse] 05 May 2013
Other Releases	Method-Signatures-20130427.0031_001 -- 27 Apr 2013 Goto
Links	[Discussion Forum] [View/Report Bugs (8)] [Dependencies] [Other Tools]
Repository	https://github.com/schwern/method-signatures/tree
CPAN Testers	PASS (5) FAIL (45) NA (9) [View Reports] [Perl/Platform Version Matrix]
Rating	★★★★★ (2 Reviews) [Rate this distribution]
License	The Perl 5 License (Artistic 1 & GPL 1)
Special Files	Build.PL MANIFEST SIGNATURE Changes META.json

Modules

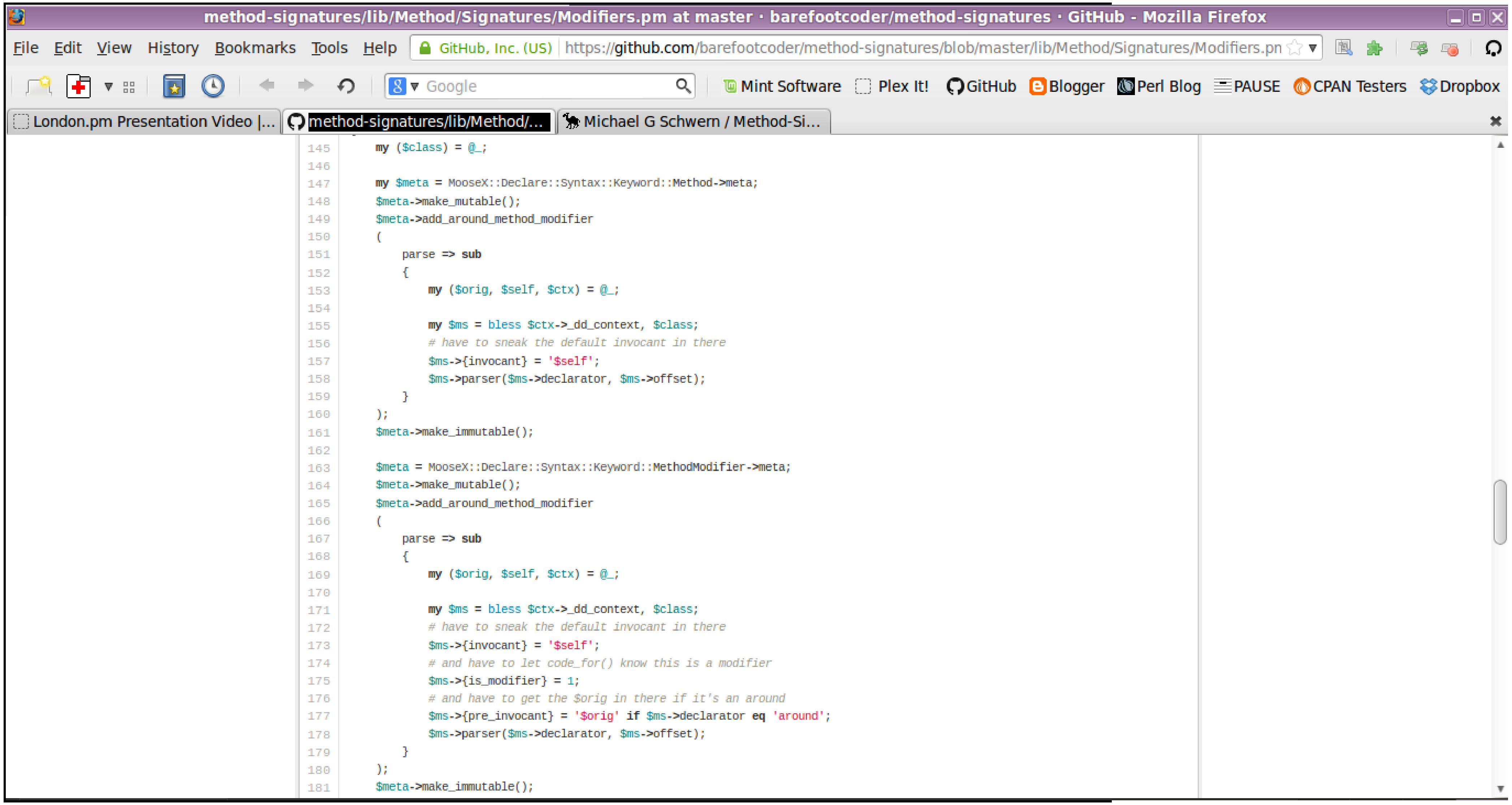
Method::Signatures	method and function declarations with signatures and no source filter	20110322.0027_001
Method::Signatures::Parser		

91277 Uploads, 27659 Distributions
121733 Modules, 10687 Uploaders

hosted by YellowBot

This is available right now on CPAN, by the way, as a developer release.

So the next challenge was getting MooseX::Declare to use Method::Signatures instead of MXMS. And that was actually harder, for a couple of reasons. First of all, MXMS is glued into MooseX::Declare pretty firmly, and second of all, MooseX::Declare has to be able to handle method modifiers: the "around", and "before", and "after", and so forth, that Moose offers. These are declared differently, and also "around" in particular has a parameter that comes in *before* $self, which is particularly tricky.


method-signatures/lib/Method/Signatures/Modifiers.pm at master · barefootcoder/method-signatures · GitHub - Mozilla Firefox
File Edit View History Bookmarks Tools Help
GitHub, Inc. (US) https://github.com/barefootcoder/method-signatures/blob/master/lib/Method/Signatures/Modifiers.pn
Google
Mint Software Plex It! GitHub Blogger Perl Blog PAUSE CPAN Testers Dropbox
London.pm Presentation Video |...
method-signatures/lib/Method/...
Michael G Schwern / Method-Si...

```perl
    my ($class) = @_;

    my $meta = MooseX::Declare::Syntax::Keyword::Method->meta;
    $meta->make_mutable();
    $meta->add_around_method_modifier
    (
        parse => sub
        {
            my ($orig, $self, $ctx) = @_;

            my $ms = bless $ctx->_dd_context, $class;
            # have to sneak the default invocant in there
            $ms->{invocant} = '$self';
            $ms->parser($ms->declarator, $ms->offset);
        }
    );
    $meta->make_immutable();

    $meta = MooseX::Declare::Syntax::Keyword::MethodModifier->meta;
    $meta->make_mutable();
    $meta->add_around_method_modifier
    (
        parse => sub
        {
            my ($orig, $self, $ctx) = @_;

            my $ms = bless $ctx->_dd_context, $class;
            # have to sneak the default invocant in there
            $ms->{invocant} = '$self';
            # and have to let code_for() know this is a modifier
            $ms->{is_modifier} = 1;
            # and have to get the $orig in there if it's an around
            $ms->{pre_invocant} = '$orig' if $ms->declarator eq 'around';
            $ms->parser($ms->declarator, $ms->offset);
        }
    );
    $meta->make_immutable();
```

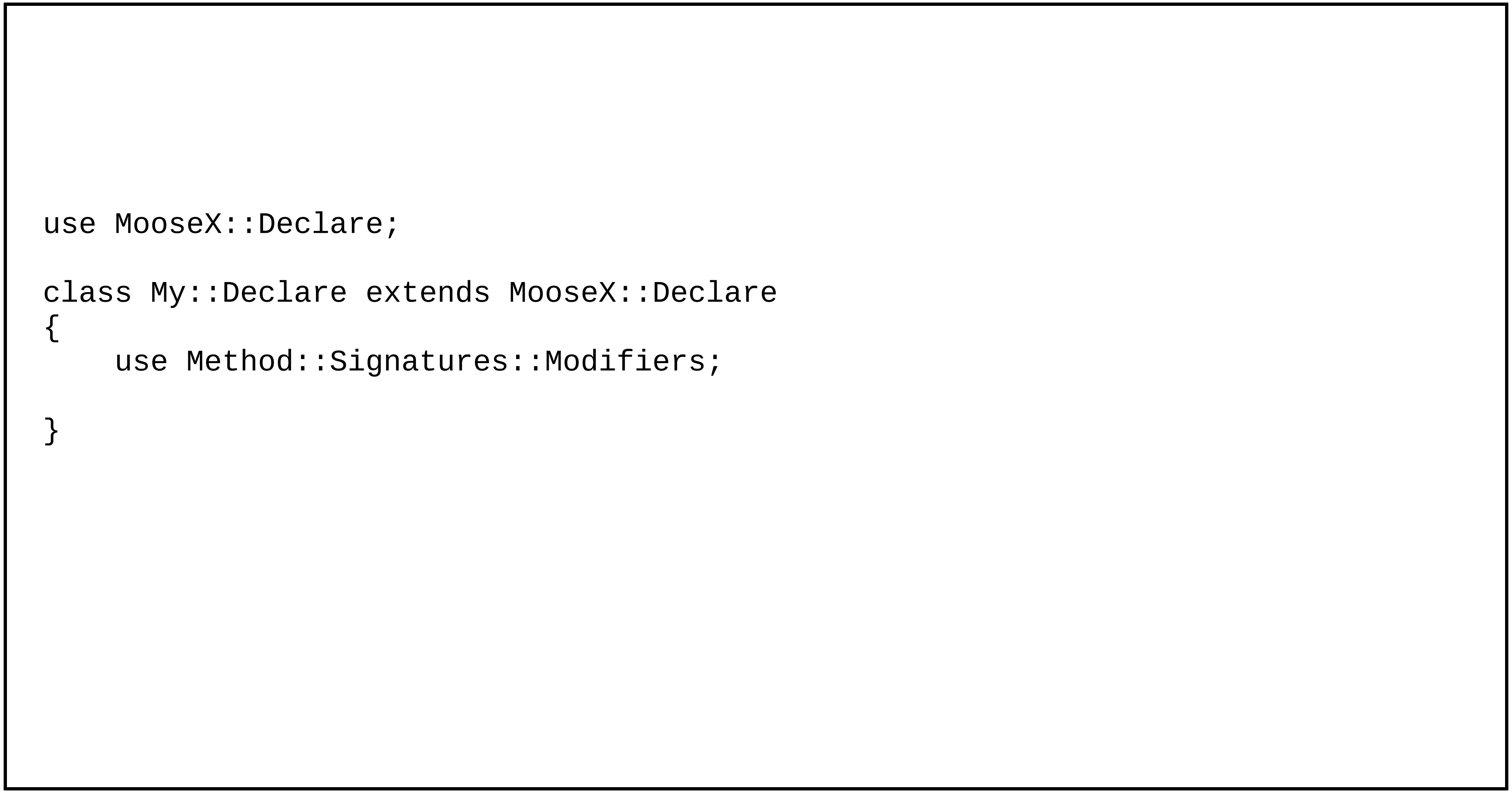
So I created a Method::Signatures::Modifiers that looks like this.

Now, this code does several things that I'm not happy with. It does a rebless, and it calls a private accessor, and, though you can't really tell, it's pulling two values out of the context and passing them to the parser() method, which promptly sticks them right back into the context, which is totally redundant. Also, MooseX::Declare is still going to *load* MXMS; it just never calls it. But it does one thing that I *am* happy with: it works.

```
use MooseX::Declare;
use Method::Signatures::Modifiers;

class MyCompany::Bar extends MyCompany::Foo
{
    method adjusted_rate (Str :$type!, Int :$discount = 0)
    {
        return $self->rate($type) * (1 - $discount / 100);
    }

    around total (Str $rate_type, Int @costs)
    {
        my $total =  $self->$orig(@costs)
                * $self->adjusted_rate(
                        type        =>  $rate_type,
                        discount    =>  $self->current_discount
                );
        return $total;
    }
}

1;
```

Here's how you use it:

One extra line. That's it.



```
use MooseX::Declare;

class My::Declare extends MooseX::Declare
{
    use Method::Signatures::Modifiers;

}
```

Or you can create a "policy" module, like so.

1. Stability Problems
2. Performance Problems
3. Hard to Debug

So, have we solved all the problems?

Well, Method::Signatures doesn't suffer from the bug in Perl 5.12, so that's a check. To check the performance, I put together some benchmarks. Here's the "plain" Moose version:

```
package PlainMoose;

use Moose;
use MooseX::Params::Validate;


sub doit
{
    my ($self, $count, $msg) = validated_list( \@_,
            count => { isa => 'Int' }, msg => { isa => 'Str' } );

    open(OUT, '>/dev/null') or die("can't open output");
    for (1..$count)
    {
        print OUT "$msg\n" for 1..10;
    }
    close(OUT);
}


1;
```

Note that I'm using MooseX::Params::Validate, to insure that I'm comparing apples to apples. Also note that my method is actually doing some work. In fact, by varying the integer I pass to it, I can prove that age-old Mark Twain adage that there are lies, there are damn lies, and then there are statistics. So, I can set it to, say, zero:

```
               Rate fancy_moose plain_moose
fancy_moose  7916/s          --        -40%
plain_moose 13277/s         68%          --
```

and say "Look at how awful the performance of MooseX::Declare is! Plain Moose is nearly twice as fast!" Or, I could set it to, maybe, 10 thousand:

```
             Rate fancy_moose plain_moose
fancy_moose 580/s          --         -6%
plain_moose 616/s          6%          --
```

and say "See? only a few percent difference. That's nothing!" You know, depending on what sort of mood I'm in.

But let's shoot for something reasonable and set it to 10, which will actually output 100 lines:

```
                  Rate fancy_moose plain_moose
fancy_moose  6785/s          --         -35%
plain_moose 10381/s         53%           --
```

So we can see that MooseX::Declare is incurring a moderate performance penalty, even when the parameter validation is a pretty small percentage of what the method is doing. Next, I tried using plain Moose without MooseX::Params::Validate, but using the old Method::Signatures. Now, this is not truly a fair comparison, because, while Method::Signatures is doing *some* parameter validation, it's not doing any type checking, so the other options are doing more work. But what it does do is give me an upper bound. It tells me that, if I could implement type checking such that it took zero time, this is how fast it would be. Now obviously, I can't do that, but this gives me something to shoot for.

```
                Rate fancy_moose plain_moose   sigs_only
fancy_moose  6521/s          --        -36%        -81%
plain_moose 10208/s         57%          --        -71%
sigs_only   34686/s        432%        240%          --
```

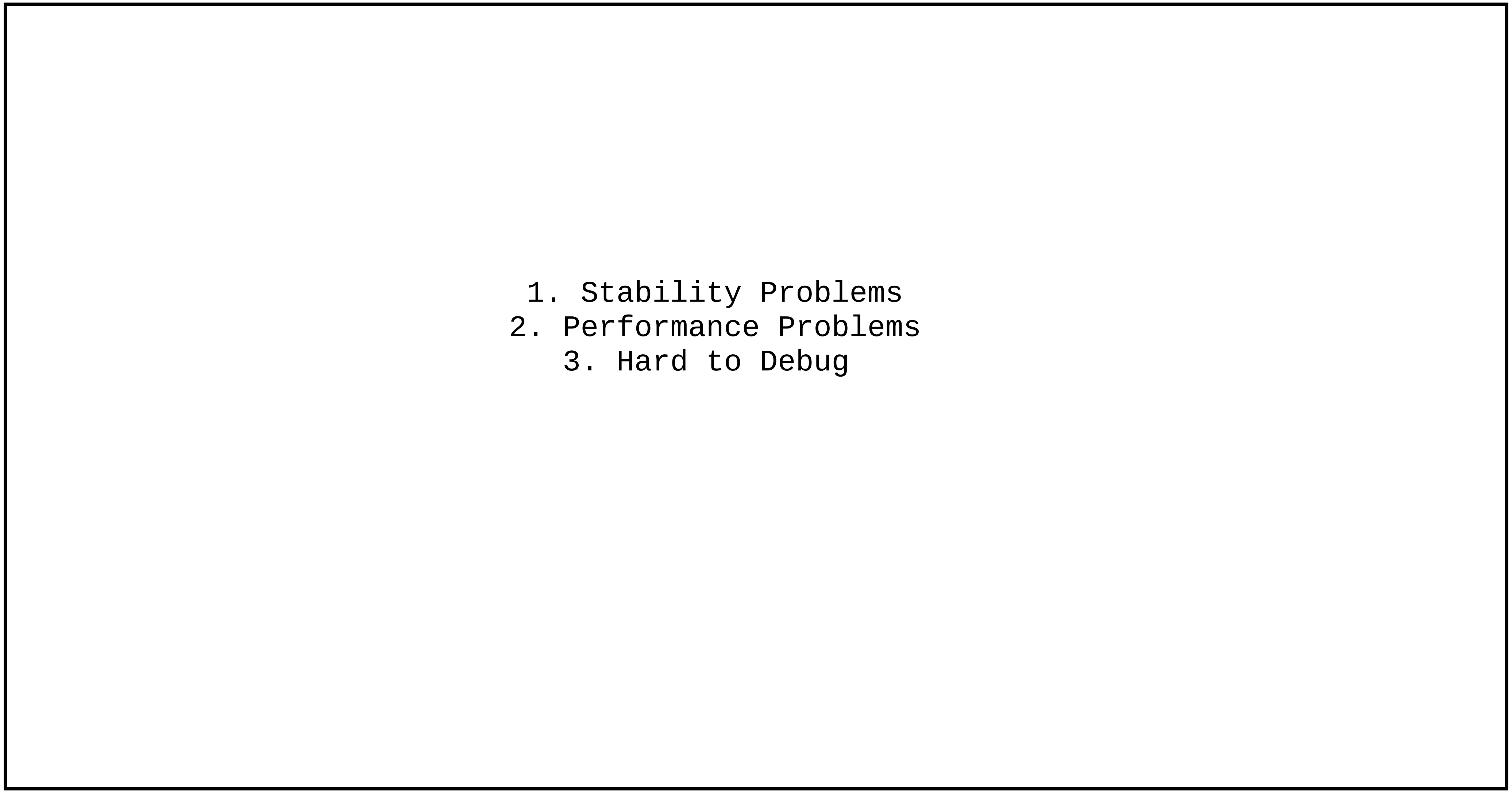
So that's a good deal faster. Now, how does using Method::Signatures::Modifiers compare?

```
               Rate fancy_moose plain_moose     mxd_msm   sigs_only
fancy_moose  6729/s          --        -35%        -72%        -81%
plain_moose 10275/s         53%          --        -58%        -71%
mxd_msm     24301/s        261%        137%          --        -30%
sigs_only   34892/s        419%        240%         44%          --
```

Pretty impressive, eh? And, if you use the "policy" module approach, it's even a tiny bit faster,

```
                Rate fancy_moose plain_moose     mxd_msm      policy   sigs_only
fancy_moose  6760/s          --        -35%        -72%        -72%        -81%
plain_moose 10476/s         55%          --        -56%        -56%        -70%
mxd_msm     23781/s        252%        127%          --         -0%        -32%
policy      23895/s        253%        128%          0%          --        -32%
sigs_only   35224/s        421%        236%         48%         47%          --
```

for some reason that I haven't been able to figure out. But the primary point is, this is even faster than using MooseX::Params::Validate, which is supposed to be what you do when MooseX::Declare is too slow for you. So whenever you see someone saying that slowness is just the price you pay for syntactical sugar, you can call bullshit on that.



```
 1. Stability Problems
2. Performance Problems
   3. Hard to Debug
```

So that's performance taken care of. How about those crappy error messages? Well, here's what you get out of Method::Signatures::Modifiers:

```
In call to MyCompany::Bar::adjusted_rate(), the 'discount' parameter ("8.5") is
not of type Int at ./better_err.pl line 15.
```

And if that's not cool enough for you, you can always write your own:

```
package CustomError;
use base qw< Method::Signatures::Modifiers >;

sub type_error
{
    my ($class, $type, $value, $name) = @_;

    $class->signature_error("your types suck!");
}
```

```
In call to MyCompany::Bar::adjusted_rate(), you really suck, you know that? at .
/best_err.pl line 15.
```

Method::Signatures

(or: How I Learned to Stop Worrying
and Love MooseX::Declare)

So that's how I managed to make MooseX::Declare work for me. Hopefully it can work for you too.

Now, this code isn't on CPAN yet, but if you want to play with it and you can't wait, you can get it from my fork on GitHub:

barefootcoder/method-signatures

THANKS:

schwern, for Method::Signatures

Nick Perez (nperez), for MooseX::Declare integration ideas

And of course I have to thank schwern, for being a great collaborator, and Nick Perez, one of the Moose guys, who gave me the basic idea on how to substitute Method::Signatures::Modifiers into MooseX::Declare.

Questions?

Questions?