AOP in 15 (or tw3nty) minutes

- Module: Aspect

There's a module by Marcel Grünauer
But he didn't invent this

http://search.cpan.org/~adamk/Aspect-1.01/lib/Aspect.pm

If you're using Moose...

you've already got most of this.

- Speaker: Matthew B. Gray

<himself [at] matthew {dot} geek {dot} nz>

There's a website there, but it's down :)

What Does it Solve?

- NOTHING! AOP doesn't add anything to your app
- However, it will help you keep your sanity
- Usually implemented by a Library
  - Supported in Perl
  - Javascript
  - COBOL
- AOP helps manage Cross Cutting Concerns
- Cross Cutting Concerns are 'aspects' of a program that cannot be modularised.

Design Mentaility

- Programs can be broken into components already
- Components reference or call each other implicitly or explicitly
- Analogus: calls between components are like messages
- It's not OO!

But what can you use it for?

- Logging
- Security
- Database Transactions
- Proxy Components
- Lazy Initalisation
- Locking / Connection Pooling
- Crosscutting Business/Application Logic

Functions Provided by Aspect

- Perl Time! use Aspect;

```
# Run some code "Advice" before a particular function
before {
    print "About to call create\n";
} call 'Person::create';
```

- Logging
- Security
- Proxy Components
- Lazy Initalisation
- Locking / Connection Pooling
- Crosscutting Business/Application Logic

 # Run Advice conditionally based on multiple factors

```
before {
    print "Calling a get method in void context within Tester::run_tests";
} wantvoid
& ( call qr/^Person::get_/ & ! call 'Person::get_not_trapped' )
& cflow 'Tester::run_tests';
```

- ...more of the same :)

 # Run Advice after several methods and hijack their return values

```
after {
    print "Called getter/setter " . $_->sub_name . "\n";
    $_->return_value(undef);
} call qr/^Person::[gs]et_/;
```

- Logging
- Crosscutting Business/Application Logic

 # Catch and handle unexpected exceptions in a function into a formal object

```
after {
    $_->exception(
        Exception::Unexpected->new($_->exception)
    );
} throwing()
& ! throwing('Exception::Expected')
& ! throwing('Exception::Unexpected');
```

- ...More of the same :)

 # Context-aware runtime hijacking of a method if certain condition is true

```
around {
    if ( $_->self->customer_name eq 'Adam Kennedy' ) {
        # Ensure I always have cash
        $_->return_value('One meeeelion dollars');
    } else {
        # Take a dollar off everyone else
        $_->proceed;
        $_->return_value( $_->return_value - 1 );
    }
} call 'Bank::Account::balance';
```

- Combines before and after.

```
# Run Advice only on the outmost of a recursive series of calls
around {
    print "Starting recursive child search\n";
    $_->proceed;
    print "Finished recursive child search\n";
} call 'Person::find_child' & highest;
```

Moooooooose

- before $name|@names|\@names|qr/.../ => sub { ... }
- after $name|@names|\@names|qr/.../ => sub { ... }
- around $name|@names|\@names|qr/.../ => sub { ... }

http://search.cpan.org/~doy/Moose-2.0010/lib/Moose.pm

Questions

TADA!

Questions?