

```

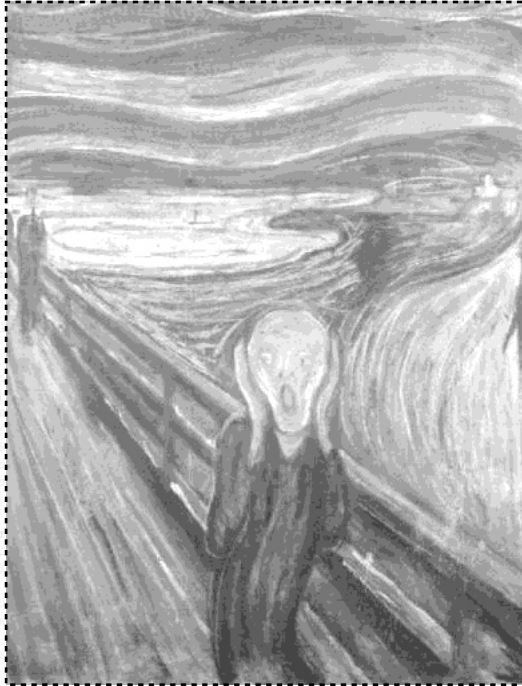
$self->connect($sym_id, $token, $, $current_package_id, undef, undef, CONN_PACPAK);
$current_package_id = $sym_id;
next TOKENS;
}
} elsif ($token->content eq
next TOKENS;
}
}
# Handle use and require de
if ($spre && $spre->isa('PFI:
my $name = $token->content
$sym_id = CPANXN::Database
$self->connect($sym_id, $
my $pkg_id = $sym_id;
if ($spre->content eq 'use
# continue til ;
my $check = $tokens->{
while ($check && !($che
if ($check->isa('PFI:
) elsif ($check->isa(
my ($spregv, $sparse);
my $spregv len = len
chop $sparse;

my $lines = split('\
$check->{ $panxr}->
for my $line ($line
while($line -- w/
my $simp_name =
my $sprefix = su
unless(grep ( $
my $spos = pos
if($sprefix --
$func($simp_na
my $sym_id =
my $offset =

if($name eq '
$self->conn
) else {
$self->conn
}
}

$check->{ $panxr}->{1} = $;
$check->{ $panxr}->{2} = $;

```



Odds and ends of Perl cross-referencing

```

($post->isa('PFI::Token::Bareword') ||
$post->isa('PFI::Token::Number')) {
next TOKENS;
}
}

```

Cross-referencing? huh?

```

}
# Handle use and require declarations
if ($pre && $pre->isa{ PPI::Token::Bareword } && $pre->content =~ /use|require|no$/ ) {
    my $name = $token->content;
    $sym_id = CPANAN::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use' ) {
        # continue till ;
        my $check = $tokens->[++$idx];
        my $lines = $pre->lines;
        my $prefix = $check->content =~ /(?:\s*\$)?\s*(\w+)$/;
        my $lines = $pre->lines;
        $check->{ cpanxr }{ 1 } = $pre->lines;
        my $tmp_name = $?;
        my $prefix = substr($tmp_name, 0, 1);
        unless(grep { $prefix eq $_ } @PREFIX) {
            $tmp_name = $?;
            $sym_id = CPANAN::Database->insert_symbol($tmp_name);
            $offset = $pos - length($tmp_name) + length($prefix);
            if($name eq 'base' ) {
                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
            } else {
                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_DEF);
            }
        }
        $check->{ cpanxr }{ 1 } = 0;
        $check->{ cpanxr }{ 0 }++;
    }
    $check = $tokens->[++$idx];
}
next TOKENS;
} elsif ($token->content =~ /use|require|no$/ &&
    ($post->isa{ PPI::Token::Bareword } ||
     $post->isa{ PPI::Token::Number }) ) {
    next TOKENS;
}
}

```

- From source code find definitions and references and put it into a database
- Query database and create a report
- Present the report as a hypertext document or a graph
- Really cool and really useful

References

```

}
# Handle use and require declarations
if ($pre && $pre->isa{ PPI::Token::Bareword } && $pre->content =~ /^use(require|no)?/ ) {
    my $name = $token->content;
    $sym_id = CPANX::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !$check->isa{ PPI::Token::Structure } && $check->content eq ';') {
            $check = $tokens->[$check->next_token];
        }
        my $preqv, $sparse = $check->content =~ /pvs*(.*)?$/;
        my $preqv_len = length($preqv);
        chop $sparse;
        my $preqv_len = length($preqv);
        for my $line (@lines) {
            while($line =~ s/(\s*)(\S+)(\s*)//gc) {
                my $simp_name = $1;
                my $pos = pos($line);
                if($prefix =~ /[A-Za-z_0-9]/) { $prefix = "" } else { $simp_name = substr($simp_name, 1); }
                my $pkg_id = $sym_id;
                my $length = length($prefix);
                my $symbol($simp_name);
            }
        }
        if($name eq 'base') {
            $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_BER);
        }
        $check->{_cpanxr}->[1] = 0;
        $check->{_cpanxr}->[0]++;
    }
    $check = $tokens->[++$idx];
}
next TOKENS;
} elsif ($token->content =~ /^use(require|no)? &&
($post->isa{ PPI::Token::Bareword } ||
$post->isa{ PPI::Token::Number }) {
    next TOKENS;
}
}

```

- Imports (use, require)
- Function calls
- Method calls
- CPP Macro expansion
- Variable

Database

```

}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/ {
    my $name = $token->content;
    $sym_id = CPANAR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, $CONN_INCLUDE);
    if ($pre->content =~ /(?:\s+|(?=[\s\w\:\@\/\#\&'\*\-\+\=\>\<\|\&#x2D;])/) {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check->isa('Perl::Token::Structure') && $check->content eq ';' ) {
            $check->isa('Perl::Token::Single') {
                $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, $CONN_INCLUDE);
            }
            $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, $CONN_INCLUDE);
        }
        my ($preqv, $parse) = $check->content =~ /^(?!(\w|'|\s|\/|\#|\&|\*|\-|\+|=|>|<|\|&#x2D;)/;
        my $preqlen = length($preqv);
        my $lines = split("\n", $preqv);
        $check->{cpanxr}--[1] += $preqlen;
        for my $line ($lines) {
            my $symbol=$preqv;
            my $tmp_name = $preqv;
            my $prefix = substr($tmp_name, 0, 1);
            unless($preqv =~ /^$prefix/) {
                $preqv = $preqv.substr($preqlen, 1);
            }
            $preqlen = length($preqv);
            my $sym_id = CPANAR::Database->insert_symbol($tmp_name);
            my $offset = $pos - length($tmp_name) + length($prefix);
            if($name eq 'base' : {
                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, $CONN_ISA);
            }
            else {
                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, $CONN_BER);
            }
        }
        $check->{cpanxr}--[1] = 0;
    }
    $check = $tokens->[++$idx];
}

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword') ||
    $post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

- **module (group of files)**
symbolic name, path
- **files (source)**
path, module, type
- **symbols**
value
- **data (references and declarations)**
symbol, file, line, column, type, [other]

Report

```

}

# Handle use and require declarations
if ($pre && $pre->isa{ PPI::Token::Bareword } && $pre->content =~ /^use|require|no$/ {
    my $name = $token->content;
    my $id = CPAN::Database->insert_symbol($name);
    $self->connect($id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $id;
    if ($pre->content =~ /use/ {
        use HTML::Parser ();
        use strict;
    }
    while ($check->isa{ PPI::Token::Structure } && $check->content eq '{') {
        if ($check->isa{ PPI::Token::Quote::Single } {
            sub start { print "open:  $_[0]\n"; }
            sub end   { print "close: $_[0]\n"; }

            # Create parser object
            $p = HTML::Parser->new( api_version => 3,
                my $tmp_name = $id;          start_h => [\&start, "tagname, attr"],
                my $prefix = substr($tmp_name, unless(grep { $prefix eq $_ } $tokens), end_h => [\&end, "tagname"],
                );
                if($prefix =~ /[\A-Za-z_]/) { $prefix = "" } else { $tmp_name = substr($tmp_name, 1); }
                $tmp($tmp_name) = $pkg_id;
                my $id = CPAN::Database->insert_symbol($tmp_name);
            # Parse document text chunk by chunk
            $p->parse($chunk1);
            $p->parse($chunk2);
            # ...
            $self->connect($id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_INCLUDE);
            $p->eof;          # signal end of document

            $check->{_cpanxr}->{1} = 0;
            $check->{_cpanxr}->{0}++;
        }
        $check = $tokens->[++$idx];
    }

    next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa{ PPI::Token::Bareword } ||
    $post->isa{ PPI::Token::Number }) {
    next TOKENS;
}
}

```

Report

```

}

# Handle use and require declarations
if ($pre && $pre->isa{ PPI::Token::Bareword } && $pre->content =~ /^use|require|no$/ {
    my $name = $token->content;
    $sym_id = CPAN::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content =~ /use|require/ {
        1: use HTML::Parser ();
        2: use strict;
        3: while ($check->isa{ PPI::Token::Structure } && $check->content eq '{') {
            4: if ($check->isa{ PPI::Token::Quote::Single } {
                4: sub start { print "open:  $_[0]\n"; }
                5: sub end   { print "close: $_[0]\n"; }
            6:
            7: # Create parser object
            8: $p = HTML::Parser->new( api_version => 3,
                9:             my $tmp_name = $?;          start_h => [\&start, "tagname, attr"],
                10:            unless{grep { $prefix eq $_ } } end_h   => [\&end,   "tagname"],
                11: );
                12:             if($prefix =~ /[\A-Za-z ]/) { $prefix = "" } else { $tmp_name = substr($tmp_name, 1); }
                13:             $func($tmp_name) = $pkg_id;
                14:             my $sym_id = CPAN::Database->insert_symbol($tmp_name);
                15:             $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_ISA);
                16:             # ...
                17:             $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
                17: $p->eof; # signal end of document

                $check->{_cpanxr}->{1} = 0;
                $check->{_cpanxr}->{0}++;

                $check = $tokens->{++$idx};
            }

            next TOKENS;
        } elsif ($token->content =~ /^use|require|no$/ &&
            ($post->isa{ PPI::Token::Bareword } ||
             $post->isa{ PPI::Token::Number }) {
            next TOKENS;
        }
    }
}

```

Background

```

}

# Handle use and require declarations
if ($pre && $pre->isa{ Perl::Token::Bareword } && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANX::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->{++$idx};
        while ($check && !($check->isa{ Perl::Token::Structure } && $check->content eq ';')) {
            $check->isa{ Perl::Token::Quote::Single } {
                my $preqx_len = length($preqx);
                $preqx = substr($preqx, $preqx_len);
            } else {
                $check->{ _cpanxr }->{ | } *- $preqx_len;
                for my $line (@lines) {
                    $line =~ s/^\s+//;
                    $name = $preqx;
                    my $prefix = substr($imp_name, 0, 1);
                    unless(grep { $prefix eq $_ } %prefix) {
                        my $pos = pos($line);
                        if ($prefix =~ s/^\s+//) { $prefix = "" } else { $imp_name = substr($imp_name, 1); }
                        $base->insert_symbol($imp_name);
                        my $offset = $pos - length($imp_name) + length($prefix);
                    }
                    if ($name =~ s/^\s+//) {
                        $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_ISA);
                        $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
                    }
                }
                $check->{ _cpanxr }->{ | } = 0;
                $check->{ _cpanxr }->{ | }++;
            }
            $check = $tokens->{++$idx};
        }
    }

    next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa{ Perl::Token::Bareword } ||
    $post->isa{ Perl::Token::Number }) {
    next TOKENS;
}
}

```

- Born at a London.pm emergency social meeting in August 2003
- Goals
 - Entire CPAN
 - Tool for in-house use

Plan C - CPANXR

```

}
# Handle use and require declarations
if ($pre && $pre->isa('PPI::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANXR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('PPI::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('PPI::Token::Quote::Single')) {
                } elsif ($check->isa('PPI::Token::Quote::Words')) {
                    $pre->content .= $check->content;
                }
            }
        }
        my @lines = split/\n/, $pre->content;
        $check->{cpanxr}->{l} += $pre->len;
        my $imp_name = $pre->content;
        my $imp_name = $pre->content;
        my $prefix = substr($imp_name, 0, 1);
        unless($pre->{cpanxr}->{l} > 0) {
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_ISA);
        } else {
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
        }
    }
    $check->{cpanxr}->{l} = 0;
    $check->{cpanxr}->{c}++;
}
$check = $tokens->[++$idx];
}
next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
($post->isa('PPI::Token::Bareword') ||
$post->isa('PPI::Token::Number')) {
    next TOKENS;
}
}

```

- Uses PPI for parsing (Parse::Perl::Isolated)
- Parser is good
- Doesn't execute code
- Doesn't require installation of modules

Description

```

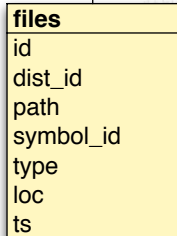
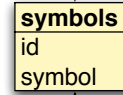
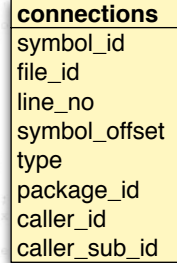
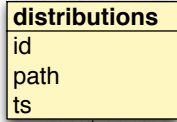
}
# Handle use and require declarations
if ($pre && $pre->isa{ PPI::Token::Bareword } && $pre->content =~ /^use|require|no$/ ) {
    my $name = $token->content;
    my $sym_id = CPAN::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, $CONN_INCLUDE);
    # continue till ;
    my $check = $tokens->[++$idx];
    while($check && ($check->isa{ PPI::Token::Structure } && $check->content eq '{') {
        my $scope = $check->scope;
        my $quote = $scope->quote;
        my $preqv = $preqv;
        my $preqv_sparse = $check->content =~ /^(qv's*|{.*}|\$?/;
        my $preqv_len = length($preqv);
        chop $preqv;
        my $line = $preqv;
        for my $line (@lines) {
            while($line =~ s/^(.*)(\s+symbol)/get/; {
                my $pos = pos($line);
                if($prefix =~ /^[A-Za-z_0-9]/) { $prefix = "" } else { $simp_name = substr($simp_name, 1); }
                my $sym_id = CPAN::Database->insert_symbol($simp_name);
                $self->connect($sym_id, $token, $pos - length($simp_name) + length($prefix);
                if($name eq 'base' : {
                    $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, $CONN_ISA);
                    $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, $CONN_BER);
                }
            }
        }
        $check = $tokens->[++$idx];
    }
}

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa{ PPI::Token::Bareword } ||
     $post->isa{ PPI::Token::Number }) {
    next TOKENS;
}
}

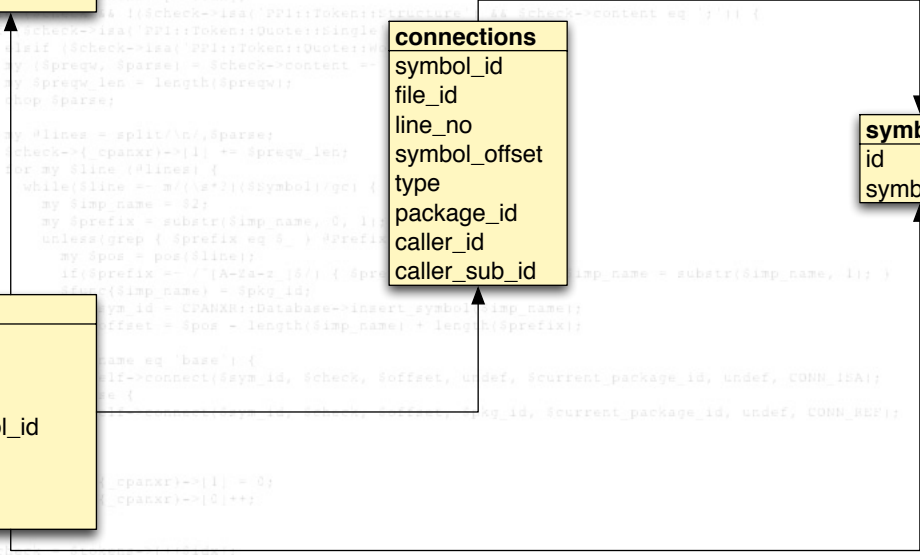
```

- Source code analyzer for Perl and XS files
- SQL database
- mod_perl front-end
 - module and file listings
 - file view
 - search function
 - Inheritance tree viewer

Database schema



```
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /use|require|no/) {
    $content;
    Database->insert_symbol($name);
    $sym_id, $token, $, undef, $current_package_id, undef, $CONN_INCLUDE;
    $id;
    eq 'use' {
        ;
        $tokens->{++$idx};
        $A { $check->isa('Perl::Token::Bareword') && $pre->content eq '}' } {
    }
    $check->isa('Perl::Token::Quote::Single')
    }
    $check->isa('Perl::Token::Quote::Double')
    }
    $pregv, $parse) = $check->content =~
    }
    $pregv len = length($pregv);
    $op $parse;
    ;
    $lines = split(/\n/, $parse);
    $check->{ $panxr->[1] } += $pregv len;
    $r my $line ($lines) {
        while($line =~ w{(?:\s+)}{0} {
            my $simp_name = $2;
            my $prefix = substr($simp_name, 0, 1);
            unless(grep { $prefix eq $_ } %Prefix) {
                my $pos = pos($line);
                if($prefix =~ /[A-Za-z_0-9]/) { $pre->insert_symbol($simp_name = substr($simp_name, 1));
                    $sym_id = CPANXR->Database->insert_symbol($simp_name);
                    $offset = $pos - length($simp_name) + length($prefix);
                    $name eq 'base' {
                        $if->connect($sym_id, $check, $offset, undef, $current_package_id, undef, $CONN_ISA);
                    }
                    $if->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, $CONN_DEF);
                }
                $panxr->[1] = 0;
                $panxr->[0]++;
            }
        }
    }
    next TOKENS;
} elsif ($token->content =~ /use|require|no/ &&
($post->isa('Perl::Token::Bareword') ||
$post->isa('Perl::Token::Number')) {
    next TOKENS;
}
```



Interface

```
}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $saym_id = CPANXR::Database->insert_symbol($name);
    $self->connect($saym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $saym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('Perl::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('Perl::Token::Quote::Single')) {
            } elsif ($check->isa('Perl::Token::Quote::Words')) {
                my ($preqw, $parse) = $check->content =~ /^(qw/' . '.*' . '$/);
                my $preqw_len = length($preqw);
                chop $parse;

                my @lines = split/\n/, $parse;
                $check->[_cpanxr]->[1] += $preqw_len;
                for my $sline (@lines) {
                    while($sline =~ w/(\s*)(\S+)/gc) {
                        my $simp_name = $2;
                        my $sprefix = substr($simp_name, 0, 1);
                        unless(grep { $sprefix eq $_ } @tokens) {
                            my $spos = pos($sline);
                            if($sprefix =~ /^[A-Za-z_0-9]/) { $sprefix = "" } else { $simp_name = substr($simp_name, 1); }
                            $func($simp_name) = $pkg_id;
                            my $saym_id = CPANXR::Database->insert_symbol($simp_name);
                            my $offset = $spos - length($simp_name) + length($sprefix);

                            if($name eq 'base') {
                                $self->connect($saym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
                            } else {
                                $self->connect($saym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
                            }
                        }
                    }
                }
                $check->[_cpanxr]->[1] = 0;
                $check->[_cpanxr]->[0]++;
            }
        }
        $check = $tokens->[++$idx];
    }
}

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword')) ||
    ($post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}
```

LIVE DEMO

Problems

```

}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $says_id = CPANANR::Database->insert_symbol($name);
    $self->connect($says_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $says_id;
    while ($check = $tokens->[++$idx]) {
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('Perl::Token::Structure') && $check->content eq '}') {
            if ($check->isa('Perl::Token::Quote::Single') {
                my $pre = $pre->content;
                my $spare = $pre->content;
                chop $spare;
                my $token = $check->content;
                for (split //, $token) {
                    my $symbol = $_;
                    my $tmp_name = $_;
                    my $prefix = substr($tmp_name, 0, 1);
                    $symbol = ($prefix eq ' ') ? $prefix : ($prefix eq '-' ? '-' . substr($symbol, 1) : $symbol);
                    $tmp_name = substr($tmp_name, 1);
                    my $says_id = CPANANR::Database->insert_symbol($tmp_name);
                    my $offset = $pos - length($tmp_name) + length($prefix);
                    $self->connect($says_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
                    $self->connect($says_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_BEP);
                }
                $check = $tokens->[++$idx];
            }
            $check = $tokens->[++$idx];
        }
    }
}
next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword') ||
    $post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

- Parsing

- Split source into an array of tokens

- A token is an thing such as bareword, number, operator, structural element etc.

- Analysis

- Iterate over the the token array and look for patterns

- Such patterns can be a bareword followed by another bareword

Parsing

```

}

# Handle use and require declarations
if ($pre && $pre->isa('PPI::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANX::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('PPI::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('PPI::Token::Quote::Single') {
                } elsif ($check->isa('PPI::Token::Quote::Words') {
                    my ($preqv, $parse) = $check->content =~ /^(q|w|s)*\{.*\}$/;
                    my $preqv_len = length($preqv);
                    chop $parse;
                    $preqv = $preqv . $parse;
                    for my $tick ($preqv =~ /'/'g) {
                        $preqv = substr($preqv, 0, $tick);
                    }
                    if ($preqv =~ /-sa-2 /) { $preqv = $preqv; } else { $tmp_name = substr($tmp_name, 1); }
                    $preqv = $preqv . $tmp_name;
                }
            }
            $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
        } else {
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
        }
    }
    $check->{_cpanxr}->{1} = 0;
    $check->{_cpanxr}->{0}++;
}
$check = $tokens->[++$idx];

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('PPI::Token::Bareword') ||
     $post->isa('PPI::Token::Number')) {
    next TOKENS;
}
}

```

- In the words of Chaim Frenkel: *"Perl's grammar can not be reduced to BNF. The work of parsing perl is distributed between yacc, the lexer, smoke and mirrors."*

Subroutines

```

}

# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANAR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->{++$idx};
        while ($check && !($check->isa('Perl::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('Perl::Token::Quote::Single')) {
                } elsif ($check->isa('Perl::Token::Quote::Words')) {
                    my $pre = $check->content =~ /^(gv's*|'|".*")$/;
                    $preqvl;
                }
            }

            my $lines = split/\n/, $pre;
            $check->{ 'package' } = $lines[0];
            my $simp_name = $pre;
            my $prefix = substr($simp_name, 0, 1);
            while ($prefix eq ' ' || $prefix eq '-') { $prefix = substr($simp_name, 1); }
            $func($simp_name, $pkg_id, $prefix);
            my $sym_id = CPANAR::Database->insert_symbol($simp_name);
            my $offset = $pos - length($simp_name) + length($prefix);
            $self->connect($sym_id, $check, $offset, $sym_id, $current_package_id, undef, CONN_ISA);
        } else {
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
        }
    }

    $check->{_cpanxr}->{1} = 0;
    $check->{_cpanxr}->{0}++;
}

$check = $tokens->{++$idx};

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword') ||
     $post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

- sub NAME
- sub NAME(PROTOTYPE)
- sub NAME [:ATTR]*
- sub NAME(PROTOTYPE) [:ATTR]*

Subroutines cont.

```

)
# Handle tokens
sub foo { }
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /"use|require|no"/) {
    my $name = $token->content;
    $sym_id = CPANXR::Database->insert_symbol($name);
    $self->sub sub { } 0, undef, $current_package_id, undef, 'CONN_INCLUDE';
    my $pkg_id = CPANXR::Database->insert_package($name);
    if ($pre->content eq 'use' ) {
        # continue till :
        my $check = $pre->content;
        while ($check && $check->isa('Perl::Token::Bareword') && $check->content eq ':') {
            if ($check->isa('Perl::Token::Quote::Single') ) {
                $self->sub($check->isa('Perl::Token::Quote::Words') ) {
                    my $preqv, $sparse = $check->content =~ /"(q|s|'|\.\.)*"/;
                    my $preqv_len = length($preqv);
                    chomp $sparse;
                    my $lines = split/\n/, $sparse;
                    sub foo :attr1 :attr2(not a clue?) {
                        for my $line ($lines) {
                            $self->sub($line) {
                                my $simp_name = $line;
                                my $prefix = substr($simp_name, 0, 1);
                                unless(grep { $_ eq $prefix } @PREFIXES) {
                                    *foo = sub($$) : attr { } else { $simp_name = substr($simp_name, 1); }
                                    $func($simp_name) = $pkg_id;
                                }; my $sym_id = CPANXR::Database->insert_symbol($simp_name);
                                my $offset = $pos - length($simp_name) + length($prefix);
                                if($name eq 'base' ) {
                                    $svar = (reverse __PACKAGE__ ) . ">::oof"; 'CONN_ISA';
                                    *$svar = sub {
                                        print "called $svar";
                                    };
                                }
                                $check->($cpanxr->{1}) = 0;
                                $chk->($cpanxr->{0})++;
                            }
                        }
                    }
                }
            }
            $check = $tokens->{**$idx};
        }
    }
    sub whitespace # Comment
    next TOKENS;
    :attr {
        } elsif ($token->content =~ /"use|require|no"/) &&
        } $ost->isa('Perl::Token::Bareword') ||
        } $ost->isa('Perl::Token::Number') || {
    next TOKENS;
}
)

```

Imports

```

}
# Handle use and require declarations
if ($pre && $pre->isa{ Perl::Token::Bareword } && $pre->content =~ /^use(require|no)?/ ) {
    my $name = $token->content;
    $sym_id = CPANXZ::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->{++$idx};
        while ($check && !($check->isa{ Perl::Token::Structure } && $check->content eq ';')) {
            my $nr = $check->Quote::Single{};
            my $tokens = $tokens->Quote::Words{};
            my $preqv = $check->content =~ /^(qv's*|.|.*|$/;
            my $preqv_len = length($preqv);
            chop $preqv;
            my $preqv_pos = $preqv_pos + $preqv_len;
            for my $line ($lines) {
                while ($line =~ s/^(.*?)(\s+)$//) {
                    unless ($preqv_pos < $preqv_pos + $preqv_len) {
                        my $pos = pos($line);
                        if ($preqv_pos < $preqv_pos + $preqv_len) {
                            $preqv_pos = $preqv_pos + $preqv_len;
                        } else {
                            $preqv_pos = $preqv_pos + $preqv_len;
                        }
                    }
                }
            }
            if ($name eq 'base') {
                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
            } else {
                $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_BER);
            }
        }
        $check->{$_panxr}->{1} = 0;
        $check->{$_panxr}->{0}++;
    }
    $check = $tokens->{++$idx};
}

next TOKENS;
} elsif ($token->content =~ /^use(require|no)?/ &&
    ($post->isa{ Perl::Token::Bareword } ||
    $post->isa{ Perl::Token::Number }) {
    next TOKENS;
}
}
```

- use Module
- sub Module LIST
- use Module VERSION
- use Module VERSION LIST
- use VERSION

Imports cont.

```

)
# Handle use and require declarations
if ($pre-isa PFI::Token::Bareword) {
  my $name = $token->content;
  use MyModule;
  $sym_id = $name;
  $self->add_import($name, $current_package_id, undef, CONN_INCLUDE);
  my $pkg_id = $sym_id;
  if ($pre->content =~ use {
    # cont
    use MyModule qw(this that);
    my $check = $token->next;
    while ($check-isa PFI::Token::Structure) {
      if ($check->content =~ use {
        use MyModule qw(:these);
      } {
        my ($preqv, $sparse) = $check->content =~ /(qv|s)*\{.*\}/;
        my $preqv_len = length($preqv);
        use Module this => that;
      }
      my $lines = split/\n/, $sparse;
      sub that { "these"; }
      use Module this => that;
      my $prefix = $token->next;
      unless (grep { $prefix eq $_ } @PREFIX) {
        use Inline Java => <<'END', AUTOSTUDY => 1 ;
        import java.util.* ;
        class Pod_10 {
          public Pod_10() {}
          public HashMap get_hm() {
            HashMap hm = new HashMap() ;
            return hm ;
          }
        }
      }
    }
  }
}
next TOK;
END
} elsif ($token->content =~ /use|require|no|/ {
  ($post->isa PFI::Token::Bareword) ||
  ($post->isa PFI::Token::Number) ||
  next TOKEN;
}
)
```

Function calls

```

}

# Handle use and require declarations
if ($pre && $pre->isa{PPI::Token::Bareword} && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $say_id = CPANXR::Database->insert_symbol($name);
    $self->connect($say_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $say_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa{PPI::Token::Structure} && $check->content eq ';')) {
            if ($check->isa{PPI::Token::Quote::Single}) {
                }
                foo("What's going on here?");
            my $preq = $pre->next;
            my $preq_len = length($preq);
            chop $sparse;
            my $lines = split/\n/, $sparse;
            foo "What's going on here?";
            for my $line (<$lines) {
                while($line =~ m/(\w+)(\s+)$/g) {
                    my $simp_name = $1;
                    my $prefix = substr($simp_name, 0, 1);
                    &foo("What's going on here?");
                    if($prefix =~ /[A-Za-z_]/) { $prefix = "" } else { $simp_name = substr($simp_name, 1); }
                    $func($simp_name) = $pkg_id;
                    my $say_id = CPANXR::Database->insert_symbol($simp_name);
                    my $offset = $pos - length($simp_name) + length($prefix);
                    use Carp qw(croak);
                    croak "Argh!"
                    $self->connect($say_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
                    $self->connect($say_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
                }
            }
            $check->{cpanxr}->{C}++;
        }
        $check = $tokens->[++$idx];
    }

    next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
        ($post->isa{PPI::Token::Bareword} ||
         $post->isa{PPI::Token::Number}) {
    next TOKENS;
}
}

```

Method calls

```
}
# Handle use and require declarations
if ($pre && $pre->isa('PPI::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $say_id = CPANXR::Database->insert_symbol($name);
    $self->connect($say_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $say_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('PPI::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('PPI::Token::Quote::Single')) {
                # skip
            } elsif ($check->isa('PPI::Token::Quote::Words')) {
                my $indirect_method = "syntax";
                my $sval = $self->parse($check->content);
                chop $sparse;
                my $lines = split("\n", $sparse);
                $check->[_cpanxr]->[1] += $sparse . "\n";
                $svar = MyClass->new(check => \&rox);
                my $simp_name = $sval->parse($check->content);
                my $sval = $self->parse($check->content);
                $sval->parse($sval->source => "/tmp/foo");
                my $base = $self->parse($sval->source => "/tmp/foo");
                $base->report($base->HTML => tempfile);
                $check->[_cpanxr]->[1] = 0;
                $check->[_cpanxr]->[C]++;
            }
            $check = $tokens->[++$idx];
        }
    }
    next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
($post->isa('PPI::Token::Bareword') ||
$post->isa('PPI::Token::Number')) {
    next TOKENS;
}
}
```

```

    $self->connect($sym_id, $token, 0, $current_package_id, undef, undef, CONN_INCLUDE);
    $current_package_id = $sym_id;
    next TOKENS;
}
}
} elsif ($token->content eq "package" && $post->isa('PFI::Token::Bareword')) {
    next TOKENS;
}
}

# Handle use and require declarations
if ($pre && $pre->isa('PFI::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANXR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue til ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('PFI::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('PFI::Token::Quote::Single')) {
            } elsif ($check->isa('PFI::Token::Quote::Words')) {
                my ($preqv, $parse) = $check->content =~ /^(q\w*"|'|\s*)$/;
                my $preqv_len = length($preqv);
                chop $parse;

                my @lines = split/\n/, $parse;
                $check->{content} = $preqv . $parse;
            }
        }
    }
}

```

Future

```

    $curr($imp_base) = $pkg_id;
    my $sym_id = CPANXR::Database->insert_symbol($imp_name);
    my $offset = $pos - length($imp_name) + length($prefix);

    if($name eq 'base') {
        $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
    } else {
        $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
    }
}
}

    $check->{_cpanxr}->{1} = 0;
    $check->{_cpanxr}->{C}++;
}
}

    $check = $tokens->[++$idx];
}
}

    next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('PFI::Token::Bareword') ||
    $post->isa('PFI::Token::Number')) {
    next TOKENS;
}
}
}

```


CPANXR cont.

```

}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $saym_id = CPANXR::Database->insert_symbol($name);
    $self->connect($saym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $saym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->{++$idx};
        elsif ($check->isa('Perl::Token::Structure') && $check->content eq '{') {
            elsif ($check->isa('Perl::Token::Single') {
            } elsif ($check->isa('Perl::Token::Quote::Words') {
                my ($pregv, $parse) = $check->content =~ /^(gv's*'|'.*'|$?/;
                $pregv_len = length($pregv);
                my $lines = split/\n/, $parse;
                $check->{ cpanxr }{ 1 } += $pregv_len;
                my $cpanxr_name = $pregv;
                my $simp_name = $cpanxr_name;
                unless(grep { $_ eq $cpanxr_name } @prefix) {
                    my $prefix = substr($simp_name, 0, 1);
                    unless(grep { $_ eq $prefix } @prefix) {
                        $prefix = $prefix . " ";
                    } else { $simp_name = substr($simp_name, 1); }
                    $func($simp_name) = $pkg_id;
                    my $saym_id = CPANXR::Database->insert_symbol($simp_name);
                    $pos = length($simp_name) + length($prefix);
                }
                $base : {
                    $self->connect($saym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
                } else {
                    $self->connect($saym_id, $check, $offset, $orig_id, $current_package_id, undef, CONN_REF);
                }
            }
            $check->{ cpanxr }{ 1 } = 0;
            $check->{ cpanxr }{ 0 }++;
        }
        $check = $tokens->{++$idx};
    }
}

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
($post->isa('Perl::Token::Bareword') ||
$post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

- **Code::Analysis**

General interface for writing code analysis tools

- **Module::CrossRef** (name not decided)

Generates cross-ref reports

- **CPANXR**

Interface to Module::CrossRef reports

Code::Analysis

```

}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANAR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, 'CONN_INCLUDE');
    my $pkg_id = $sym_id;
    if ($pre->content =~ /;/) {
        my $check = $token->content;
        while ($check && !($check->isa('Perl::Token::Structure') && $check->content eq '}' )) {
            if ($check->isa('Perl::Token::Quote::Single') ||
                $check->isa('Perl::Token::Quote::Double') ||
                $check->isa('Perl::Token::Text')) {
                my $spare;
                if ($pre->content =~ /;/) {
                    $check->{ '_spanx' }--(1);
                    for my $line (@lines) {
                        while ($line =~ /(?:\s+|;)/g) {
                            my $pos = pos($line);
                            my $prefix = substr($line, 0, $pos);
                            unless (grep { $prefix eq $_ } @Prefix) {
                                my $base = substr($line, $pos, 1);
                                $succ($base) { $prefix = "" } else { $succ($base) { $prefix = substr($line, $pos, 1); }
                                $succ($base) = $pkg_id;
                                my $sym_id = CPANAR::Database->insert_symbol($base);
                                my $offset = $pos - length($base) + length($prefix);
                                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, 'CONN_ISA');
                            } else {
                                $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, 'CONN_DEF');
                            }
                        }
                    }
                }
                $pre->{ '_spanx' }--(1);
            }
        }
    }
}
next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword') ||
    $post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

- HTML::Parser like interface but for code
- Calls handlers when it encounters an interesting piece of code
 - function declaration
 - block start and end
 - comments
 - method/function calls
 - imports
 - class declarations

Code::Analysis

```

}

# Handle use and require declarations
if ($pre && $pre->isa{ Perl::Token::Bareword } && $pre->content =~ /^use|require|no$/ ) {
    my $name = $token->content;
    $saym_id = CPANXR::Database->insert_symbol($name);
    $self->connect($saym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $saym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->{++$idx};
        while ($check && !$check->isa{ Perl::Token::Structure } && $check->content eq ';') {
            $check = $tokens->{++$idx};
        }
        my $preq = $pre->content;
        my $preq_len = length($preq);
        chop $preq;
        for my $line (@lines) {
            while($line =~ w{(?:\s|;)}($symbol)?gc) {
                my $simp_name = $2;
                my $prefix = substr($simp_name, 0, 1);
                unless(grep { $prefix eq $_ } %Prefix) {
                    my $pos = pos($line);
                    if($prefix =~ /^[A-Za-z_0-9]/) { $prefix = "" } else { $simp_name = substr($simp_name, 1); }
                    $simp_name =~ s/^\s+//;
                    my $saym_id = CPANXR::Database->insert_symbol($simp_name);
                    my $offset = $pos - length($simp_name) + length($prefix);
                }
            }
        }
        $check = $tokens->{++$idx};
    }
}

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa{ Perl::Token::Bareword } ||
    $post->isa{ Perl::Token::Number }) {
    next TOKENS;
}
}

```

- Prio 1: Perl analyzer

- Prio 2: More languages

XS

C/C++

Java

C#

Python

Module::CrossRef

```

}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANAN::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till:
        # check for @ISA inheritance
        if ($check->isa('Perl::Token::Quote::Single') {
            # check for @ISA inheritance
        } elsif ($check->isa('Perl::Token::Quote::Words') {
            # check for @ISA inheritance
            my $lines = split/\n/, $pre->content;
            $check->{cpanxr}--[1] += $pre->len;
            my $tmp_name = $name;
            my $prefix = substr($tmp_name, 0, 1);
            unless(grep { $prefix eq $_ } @prefix) {
                $tmp_name = substr($tmp_name, 1);
            }
            my $sym_id = CPANAN::Database->insert_symbol($tmp_name);
            my $offset = $pos - length($tmp_name) + length($prefix);
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_ISA);
        } else {
            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_REF);
        }
        $check->{cpanxr}--[1] = 0;
        $check->{cpanxr}--[0]++;
    }
}

next TOKEN;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword') ||
     $post->isa('Perl::Token::Number')) {
    next TOKEN;
}
}

```

- Generates cross-reference reports
 - YAML (XREF.yml) or some other format
- Uses Code::Analysis to figure out
 - Subroutine declarations
 - Function and method calls
 - Imports of other modules
 - @ISA inheritance

CPANXR

```

}

# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANXR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('Perl::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('Perl::Token::Quote::Single')) {
                } elsif ($check->isa('Perl::Token::Quote::Words')) {
                    my ($pre_w, $words) = $check->content =~ /^"(\w+)"$/;
                    my $lines = split/\n/, $words;
                    $check->{ cpanxr }->{ i } += $pre_w . len;
                    for my $line ($lines) {
                        my $sym_name = $line;
                        my $tmp_name = $line;
                        my $prefix = substr($tmp_name, 0, 1);
                        unless (grep { $prefix eq $_ } @prefix) {
                            my $sym_id = CPANXR::Database->insert_symbol($tmp_name);
                            my $offset = $pos - length($tmp_name) + length($prefix);
                            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_ISA);
                        } else {
                            $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_DEF);
                        }
                    }
                    $check->{ cpanxr }->{ i } = 0;
                    $check->{ cpanxr }->{ C }++;
                }
            }
            $check = $tokens->[++$idx];
        }
    }

    next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
        ($post->isa('Perl::Token::Bareword') ||
         $post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

- Loads Module::CrossRef reports into database
- Front ends
 - web interface like current
 - CPANPLUS integration
 - emacs?

Module::Checkstyle

```

}
# Handle use and require declarations
if ($pre && $pre->isa('Perl::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $saym_id = CPANXN::Database->insert_symbol($name);
    $self->connect($saym_id, $token, 0, undef, $current_package_id, undef, 'CONN_INCLUDE');
    my $pkg_id = $saym_id;
    if ($pre->content eq 'use') {
        # continue till ;
        my $check = $tokens->{++$idx};
        while ($check->content =~ /(?:\s|'"/>
```

- **Naming conventions**
 - subroutines must match `qr/^(?:_?[a-z]+)*$/`
 - packages must match `qr/^[A-Z][a-z]+$/`
- **Blocks**
 - { must not be on a single line
 - } must be on a single line
- **Complexity**
 - Not more than 8 nested conditional blocks

```

    }
    $check = $tokens->{++$idx};
}
next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('Perl::Token::Bareword') ||
     $post->isa('Perl::Token::Number')) {
    next TOKENS;
}
}

```

```

$self->connect($sym_id, $token, 0, $current_package_id, undef, undef, CONN_INCLUDE);
$current_package_id = $sym_id;
next TOKENS;
}
} elsif ($token->content eq "package" && $post->isa('PFI::Token::Bareword')) {
    next TOKENS;
}
}

# Handle use and require declarations
if ($pre && $pre->isa('PFI::Token::Bareword') && $pre->content =~ /^use|require|no$/) {
    my $name = $token->content;
    $sym_id = CPANXSR::Database->insert_symbol($name);
    $self->connect($sym_id, $token, 0, undef, $current_package_id, undef, CONN_INCLUDE);
    my $pkg_id = $sym_id;
    if ($pre->content eq 'use') {
        # continue til ;
        my $check = $tokens->[++$idx];
        while ($check && !($check->isa('PFI::Token::Structure') && $check->content eq ';')) {
            if ($check->isa('PFI::Token::Quote::Single')) {
            } elsif ($check->isa('PFI::Token::Quote::Words')) {
                my ($preqw, $parse) = $check->content =~ /^(qw/' . '\s*' . ')$/;
                my $preqw_len = length($preqw);
                chop $parse;

                my @lines = split(/\n/, $parse);
                $check->[_cpanxr]->[1] += $preqw_len;
                for my $line (@lines) {
                    while($line =~ w/((?:\s|\\|\/)*)/gc) {
                        my $simp_name = $2;
                        my $prefix = substr($simp_name,
                            unless(grep { $prefix eq $_ } @lines)) {
                            my $pos = pos($line);
                            if($prefix =~ /^[A-Za-z_0-9]/) { $prefix = "" } else { $simp_name = substr($simp_name, 1); }
                            $func($simp_name) = $pkg_id;
                            my $sym_id = CPANXSR::Database->insert_symbol($simp_name);
                            my $offset = $pos - length($simp_name) + length($prefix);

                            if($name eq 'base') {
                                $self->connect($sym_id, $check, $offset, undef, $current_package_id, undef, CONN_ISA);
                            } else {
                                $self->connect($sym_id, $check, $offset, $pkg_id, $current_package_id, undef, CONN_REF);
                            }
                        }
                    }
                }
                $check->[_cpanxr]->[1] = 0;
                $check->[_cpanxr]->[C]++;
            }
        }
        $check = $tokens->[++$idx];
    }
}

next TOKENS;
} elsif ($token->content =~ /^use|require|no$/ &&
    ($post->isa('PFI::Token::Bareword')) ||
    $post->isa('PFI::Token::Number')) {
    next TOKENS;
}
}
}

```

Thank you...