

A beginner's guide to refactoring

- Refactoring in Perl
- A quick look at tackling common refactoring techniques

Christian Brink (grep)

What is Refactoring

- Simplifying code without changing purpose
- More than just "Cleaning Up"
- Gained popularity from XP

Why Refactor?

- Improve maintainability
- Self-documenting code
- Increase extensibility
- Easier to debug
- Next guy won't hate you
- Won't hate yourself (in a month)

Why Wouldn't You Refactor?

or how do I explain this to my boss?

- Spending time on code that works (Bottom Line)
 - Reduces cost in the long run
 - Time and costs rise as maintainability declines
 - Reduces errors
 - Code almost always evolves
- Might break something
 - Testing is integral to refactoring
- Possible performance penalty
 - "Premature optimization is the root of all evil" - Donald Knuth

Where Do I Start?

- Reformat Code
- Automated Tests and Code Versioning
- Self Documenting Code
- Code Smell, Red Flags, and Anti-Patterns

Tools to Help

- CVS, Git, SVN, Darcs,...
- Perltidy
- Coding standards (perlstyle, PBP)
- Test::Simple, Test::More, prove
- Perlmonks.com, Perl.com
- Derive satisfaction from quality, NOT your code

Reformat Code

- Just cleaning up code helps readability
- Structure code with indentation and proper whitespace
- Use a standard. Any standard, just make it standard.
- Hide other languages

```
# Actual code submitted to perlmonks.com
```

```
my $client_job_spoil
    = Client_Actuals->retrieve(
        reference      =>
            $lodgement->{reference},
        cj_job_ref_no  =>
            $job->{cj_job_ref_no},
        qcs_sequence_no =>
            $_->qcs_sequence_no,
    );
```

```
##
```

```
## After perltidy
```

```
##
```

```
my $client_job_spoil = Client_Actuals->retrieve(
    reference      => $lodgement->{reference},
    cj_job_ref_no  => $job->{cj_job_ref_no},
    qcs_sequence_no => $_->qcs_sequence_no,
);
```

Automated Testing and Code Versioning

Eliminating the fear of changing your code

Testing

- Automated tests are easy to run and validate that nothing is broken
- Testable code is generally well-factored
- You should be doing it already

Code Versioning

- You can always go back
- Branching
- You should be doing it already

Self-Documenting Code

- Don't write cool code
- Improve variable and sub names
- Use named parameters not positional params
- Use temporary variables to clarify code

Self-Documenting Code – Cool Code

```
my $foo = join'. ',map{ucfirst}split/\. /,lc;s/\bi\b/I/g;
```

```
my $foo = $_=lc;s/^\|\bi\b|\.\s*./\U$&/g
```

```
my $foo = lc;s/^\|\bi\b|\.\s*./\U$&;g;
```

```
my $foo = s/(.+?)(\.\s*)/\u\L$1$2/gs;s/\bi\b/I/gs
```

```
my $foo = pop;for(;;){warn$f,map(int rand 10,1..$n),$/;++$i%1e3 or$f.=int rand 10 and$n--or  
last}
```

```
@p=(1,1);map@p=(@p,$p[-1]+$p[-2]),A..R;$\=$,="\n";print@p
```

```
my $foo = length(join('; ',reverse(split //,++$a))) + $#a + 50 ;
```

Self-Documenting Code – Better Naming

```
sub update_plus {  
#-----  
# only retry 10 times  
$count++;  
...  
...  
...  
if ($count >= 10) {  
    next;  
}  
}
```

```
#-----  
$brate = shift;  
$erate = shift;  
$date_1 = shift;  
$date_2 = shift;
```

```
sub update_product_ids {  
#-----  
$retry_count++;  
...  
...  
...  
if ($retry_count >= 10) {  
    next;  
}  
}
```

```
#-----  
my $payrate;  
$payrate->{begin} = shift;  
$payrate->{end} = shift;  
@date_range = @_[0..1]
```

Self-Documenting Code – Named Params

```
$foo->update( $bdate, $store, $data)
```

```
sub update {  
  my $self = shift;  
  my $date = shift;  
  my $store_id = shift;  
  my $data = shift;  
  
  ....  
}
```

```
$foo->update_status({  
  date      => $date,  
  store_id => $store_id,  
  status    => $status  
})
```

```
);  
  
sub update_status {  
  my $self = shift;  
  my $params = shift;  
  ...  
}
```

Self-Documenting Code – Temp Vars

```
$days{$store_id}{$weekday}{total_pay} =  
  ( ( $sql->{hours_worked} - ( $clerks{ $sql->{clerk_num} }{hours_worked} - 40 ) ) * $regular )  
  + ( ( $clerks{ $sql->{clerk_num} }{hours_worked} - 40 ) * $overtime );
```

```
#-----
```

```
my $overtime_hours = $clerks{ $sql->{clerk_num} }{hours_worked} - 40;  
my $regular_hours  = $sql->{hours_worked} - $overtime_hours;
```

```
$days{$store_id}{$weekday}{total_pay} =  
  ( $regular_hours * $regular_rate ) + ( $overtime_hours * $overtime_rate )
```

Code Smell, Red Flags, and Anti-Patterns

- A lot of comments (except POD)
- Duplicate or similar code
- Large subs/methods
- Gnarly if/else's - Arrow Anti-Pattern
- Many more but these are most common/specific

Code Smell, Red Flags and Anti-Patterns

Comments

```
#calculate daily totals
my $daylies = DailyTotals->new( $in, $self->{dbh}, my $daily_totals = DailyTotals->calc({
    $store, $date, $hours_totals->get(),      value_names => $value_name,
    $proj->get(), 1 );                        store_id    => $store_id,
                                             date        => $date,
                                             hour_totals => $hour_totals->get_daily_totals(),
                                             projections => $projections->get_daily_projections(),
                                             for_single_day => 1
                                             }
);

#-----
#-----

# get the start date
$self->{from_dt} =~ m/(\d\d\d\d)-(\d\d)-(\d\d)/;  $self->{start_date} =~ m/(\d\d\d\d)-(\d\d)-(\d\d)/;
```

Code Smell, Red Flags and Anti-Patterns

Comments

```
# regular == <= 40 per week per employee
if ( $clerks{ $sql->{clerk_num} }{hours_worked} + $sql->{hours_worked} <= 40 ) {
    $days{$store_id}{$weekday}{regular_hours} += $sql->{hours_worked};
    $clerks{ $sql->{clerk_num} }{hours_worked} += $sql->{hours_worked};
    $days{$store_id}{$weekday}{total_pay} += ( $sql->{hours_worked} * $regular );
}
else {
    # overtime == > 40 per week per employee
    # if hours worked is less than 40 before we get to here
    if ( $clerks{ $sql->{clerk_num} }{hours_worked} < 40 ) {
        $clerks{ $sql->{clerk_num} }{hours_worked} += $sql->{hours_worked};
        $days{$store_id}{$weekday}{overtime_hours} += ( $clerks{ $sql->{clerk_num} }{hours_worked} - 40 );
        $days{$store_id}{$weekday}{regular_hours} += ( $sql->{hours_worked} - $ot );
        $days{$store_id}{$weekday}{total_pay} += ( $ot * $overtime );
        $days{$store_id}{$weekday}{total_pay} += ( $reg * $regular );
    }
    else {
        $clerks{ $sql->{clerk_num} }{hours_worked} += $sql->{hours_worked};
        $days{$store_id}{$weekday}{overtime_hours} += $sql->{hours_worked};
        $days{$store_id}{$weekday}{total_pay} += ( $sql->{hours_worked} * $overtime );
    }
}

#-----

my $clerk_num = $sql->{clerk_num};
my $hours_this_shift = $sql->{hours_worked};
my $overtime_hours_this_shift = 0;
my $regular_hours_this_shift = $hours_this_shift;

if ( $clerks{ $clerk_num }{hours_worked} + $hours_this_shift > 40 ) {
    if ( $clerks{ $clerk_num }{hours_worked} < 40 ) {
        $overtime_hours_this_shift = ( $clerks{ $clerk_num }{hours_worked} + $hours_this_shift ) - 40;
        $regular_hours_this_shift = $hours_this_shift - $overtime_hours_this_shift;
    }
    else {
        $overtime_hours_this_shift = $hours_this_shift;
        $regular_hours_this_shift = 0;
    }
}

$days{$store_id}{$weekday}{overtime_hours} += $overtime_hours_this_shift;
$days{$store_id}{$weekday}{regular_hours} += $regular_hours_this_shift;
$clerks{ $clerk_num }{hours_worked} += $regular_hours_this_shift;
```

Code Smell, Red Flags and Anti-Patterns

Duplicate Code

```
sub get_adult_drink_count {
    my $self = shift;
    my $sql = qq{
        SELECT items_count, report_id
        FROM register_items
        WHERE items_name = ?
    };

    my $sth = $self->{dbh}->prepare($sql);
    $sth->execute( 'adult_drink' );
    my $row = $sth->fetchrow_hashref();
    return $row->{ items_count } || 0;
}
```

```
sub get_child_drink_count {
    my $self = shift;
    my $sql = qq{
        SELECT items_count, report_id
        FROM register_items
        WHERE items_name = ?
    };

    my $sth = $self->{dbh}->prepare($sql);
    $sth->execute( 'child_drink' );
    my $row = $sth->fetchrow_hashref();
    return $row->{ items_count } || 0;
}
```

```
sub get_adult_drink_count {
    my $self = shift;
    return $self->_get_item_count('adult_drink') || 0;
}

sub get_child_drink_count {
    my $self = shift;
    return $self->_get_item_count('child_drink') || 0;
}

sub _get_item_count {
    my $self = shift;
    my $value = shift || die "no item given\n";
    return $self->_get_from_table(
        'items', 'count', $value );
}

sub _get_from_table {
    my $self = shift;
    my $table = shift || die "No table given\n";
    my $type = shift || die "no type given\n";
    my $value = shift || die "no value given\n";

    my $field_name = $table . '_' . $type;
    my $value_field = $table . '_name';
    my $sql = qq{
        SELECT $field_name, report_id
        FROM register_$table
        WHERE $value_field = ?
    };

    my $sth = $self->{dbh}->prepare($sql);
    $sth->execute( @reports, $value );
    return $self->_return_hash_from_sql(
        $field_name, $sth );
}
```

Code Smell, Red Flags and Anti-Patterns

Large Subs

```
sub wsa {
  my $date_1 = shift;
  my $date_2 = shift;
  my $store = shift;

  #check date_1 or use today
  #check date_2
  #check date_1 against date_2

  #open database handle
  #sql for store lookup
  #execute sql
  #retrieve record
  #check for errors

  #sql for looking up employees
  #execute sql
  #retrieve records
  #check for errors

  #sql for looking up sales
  #execute sql
  #retrieve records
  #check for errors

  #calculate pay for employees during date range

  #calculate pay for employees during date range

  #calculate sale / labor cost

  return ($labor_cost, $net_sales,
          $gross_sales, $sales_tax,
          $labor_percentage)
}
```

```
sub weekly_sales_analysis {
  my $begin_date = shift;
  my $end_date = shift;
  my $store_id = shift;
  my $date_range = validate_dates(...);
  my $labor_cost = get_labor_for_date_range(...);
  my $sales = get_sales_for_date_range(...);
  my $labor_percentage =
    get_labor_cost_percentage_of_sales(...);

  return {
    labor_cost => $labor_cost,
    net_sales => $sales->{net_sales},
    gross_sales => $sales->{gross_sales},
    labor_percentage => $labor_percentage
  };
}
```

Code Smell, Red Flags and Anti-Patterns

Gnarly if/else's

```
if ( $foo == 42 ) {
  if ( $bar < 1 ) {
    if ( $baz == 1) {
      if ( $blah > 6 ) {
        return "Blart!";
      }
      else {
        return "Beep!";
      }
    }
  }
  elseif ( $bar > 1 ) {
    if ( $blah < 6 ) {
      return "B00!";
    }
    else {
      return "w00t!";
    }
  }
}
elseif ( $foo == ( $six * $nine ) ) {
  return "The meaning of life"
}
else {
  return;
}
```

```
my $status = {
  foo => $foo,
  bar => $foo * $baz,
  baz => $foo ** 2,
  blah => $brand_of_whiskey
};
return "The meaning of life"
  if ( $foo == ( $six * $nine ) );

return "Blart!" if is_blart($status);
return "Beep!"  if is_beep($status);
return "B00!"   if is_boo($status);
return "w00t!"  if is_woot($status);
```

Other Common Large Scope Anti-Patterns

- Spaghetti code – unstructured, hard to follow, lots of "Magic"
 - Get rid of globals
 - Pull the magic out (action at a distance)
- Big Ball of Mud – no architecture
 - Start small and modularize
 - Tests lead the way
 - Think about data structures – not necessarily objects (at least in the beginning)