Warum modern perl toll ist

sECuRE beim NoName e.V.

powered by LaTeX, of course

20. August 2009

Inhalt

- Vorurteile
- Code-Reuse / CPAN
- Moose
- ein paar Module
- Testcases

Vorurteile

- Perl ist tot
- Perl-code ist unlesbar
- Perl-code kann man nicht maintainen
- Perl 5 stirbt, vielleicht Perl 6 mal anschauen...

Ein Negativbeispiel

```
if (length ($ENV{'QUERY_STRING'}) > 0) {
      $buffer = $ENV{'QUERY_STRING'};
      @pairs = split(/&/, $buffer);
      foreach $pair (@pairs) {
           ($name, $value) = split(/=/, $pair);
           $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C",
                hex($1))/eg;
           $in{$name} = $value;
      }
}

print Dumper($in);
```

Wie es eigentlich ginge:

```
use CGI;

my $query = CGI->new;
print Dumper($query->param);
```

CODE-REUSE

- wird bei Perl großgeschrieben :-)
- sofern man kein Experte ist, macht man Fehler
- je mehr Leute den Code nutzen, desto schneller werden Fehler gefunden
- Beispiel: URL parsen

URL parsen (ganz einfach?)

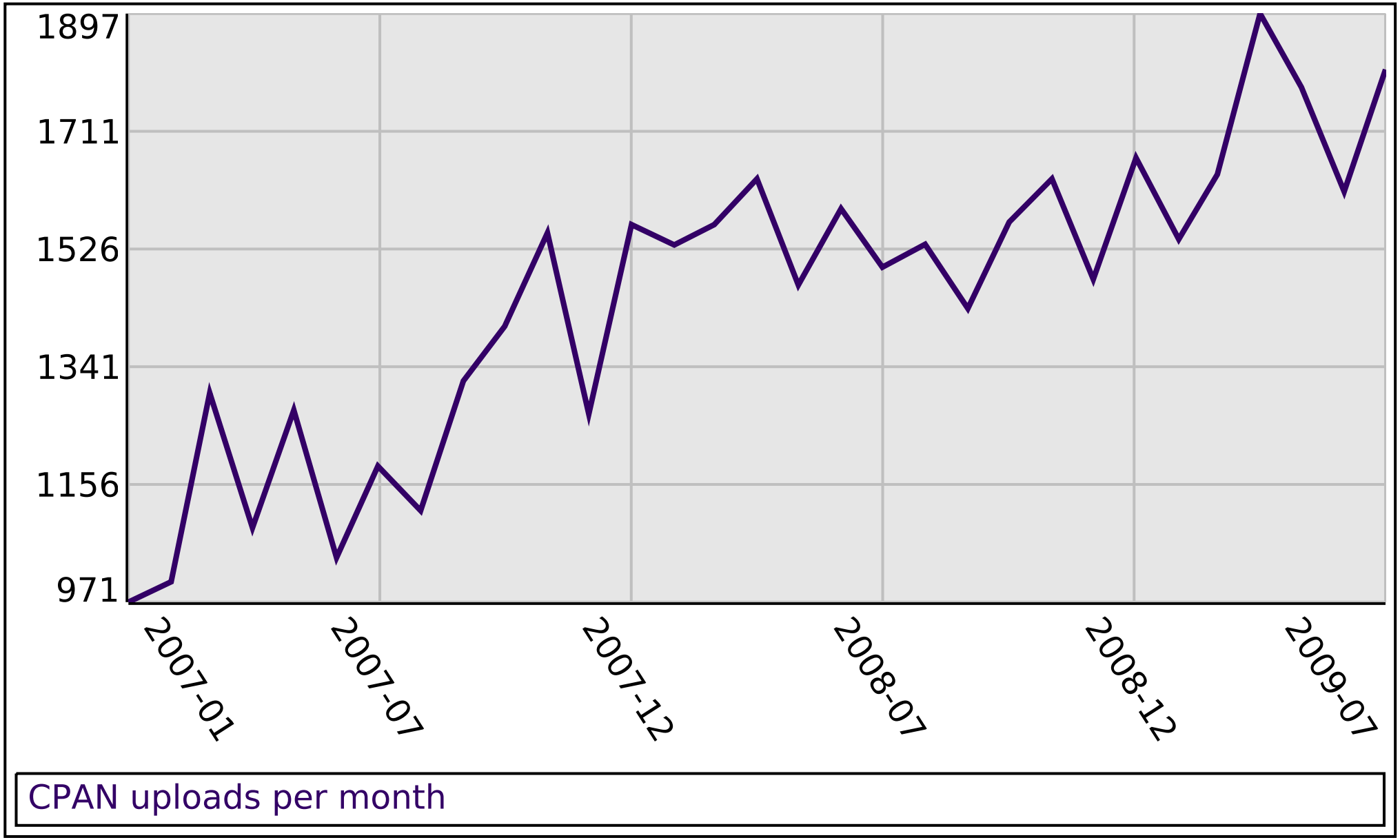
- `http://www.ebay.de/` →
  `my ($proto, $host) = /([a-z]*):\/\/([a-z0-9.]*)\//;`
- `http://www.ebay.de:80/` →
  `my ($proto, $host, $port) =`
  `/([a-z]*):\/\/([a-z0-9.]*)(|:[0-9]{1-5})\//;`
- `http://www.ebay.de:http/`
- `http://[2001:4860:b003::84]/`
- `http://[2001:4860:b003::84]:80/`
- `http://[fe80::21c:c0ff:fe7e:4776%eth0]:80/`

```
 use URI;
my $u = URI->new('http://www.perl.com:80');
my ($proto, $host, $port) = ($u->scheme, $u->host, $u->port);
```

CPAN

- Zentraler (!) Anlaufpunkt für Perl-Module
- `search.cpan.org`
- `sudo cpan WWW::Mechanize`
- Über 14000 distributions (enthalten 54000 Module)
- Für nahezu alles, was man sich ausdenken könnte, gibt es Module: HTTP, SMTP, IRC, XMPP, Linux::DVB, DBI, Parse::*, Algorithm::*, XML::*, WWW::Mechanize, IO::All, ...
- (nahezu) alles dokumentiert
- (nahezu) überall Testcases vorhanden, automatisches testen

CPAN-Uploads



(powered by `Chart::Clicker`, data from `http://stats.cpantesters.org/mreports.html`)

Moose

- Modernes Object-Framework für Perl
- Erleichtert das Schreiben von Modulen, die OO nutzen
- Ein Beispiel:

OO in Perl (Beispiel)

```
use WWW::Mechanize;
use File::Basename;

my $mech = WWW::Mechanize->new;

$mech->get('https://elearning.uni-heidelberg.de/login');
$mech->submit_form(
        form_number => 2,
        fields => {
                username => 'Michael',
                password => 'geheim',
        },
);

$mech->get('https://elearning.uni-heidelberg.de/slides/info1/');
for my $link ($mech->links) {
        next unless $link->url =~ /\.pdf$/;
        $mech->get($link->url,
        ':content_file' => basename($link->url));
}
```

Was wir benutzen wollen

```
my $torrent = Net::BitTorrent::Builder->new(
        name => '/tmp/openoffice.deb'
);

my $hash = $torrent->info_hash;

print "Created torrent file with info_hash $hash\n";
```

OO mit Moose

```
package Net::BitTorrent::Builder;

use Moose;
use Digest::SHA1;
use Bencode;

has 'name' => (is => 'ro', isa => 'Str', required => 1);
has 'info_hash' => (is => 'ro', isa => 'Str', lazy_build => 1);
# has 'pieces' ...
# has 'piece_size' ...

sub _build_info_hash {
        my $self = shift;
        my %info = (
                length => $self->size,
                name => $self->name,
                'piece length' => $self->piece_size,
                pieces => join('', $self->pieces),
        );

        return sha1_hex(bencode(\%info));
}
```

Module

- IO::All
- AnyEvent, generische Event-Library
- List::Util, List::MoreUtils
- autodie

IO::All

```
use IO::All;

# Datei schreiben
"foo bar!" > io('/tmp/output.txt');

# Datei lesen
my $content = io('/tmp/output.txt')->slurp;
print "File contains: $content\n";

# Directory durchgehen
my $dir = io('/tmp');
for my $path ($dir->all) {
        print "File in /tmp: $path\n";
}

# Datei hochladen (FTP)
io('/tmp/output.txt') > io->('ftp://foo.example.net/');

# Website laden
my $html = io->http('www.google.de');
```

List::Util, List::MoreUtils

```
my @list = (4, 5, 6);

print "First item > 4: " . first { $_ > 4 } @list;
# First item > 4: 5

if (all { $_ > 0 } @list) {
        print "All items positive\n";
}

my @double = apply { $_ *= 2 } @list;
# @double = (8, 10, 12);

my @list = (2, 2, 2);
print "There is only: " . Dumper(uniq @list) . "\n";
# There is only: 2
```

autodie

```
my $fh;
open($fh, '<', '/tmp/foo.txt') or
    die "Could not open /tmp/foo: $!\n";

# besser:
use autodie;

my $fh;
open($fh, '<', '/tmp/foo.txt');
```

Testcases

- . . . testen kleine Teile des Programms
- $\rightarrow$ begünstigen Wiederverwendbarkeit, gutes Design
- möglichst auf corner cases auslegen
- sehr einfach zu schreiben dank `Test::*` (insb. `Test::More`, `Test::Exception`, `Test::Deep`), sehr schöne Anleitung in `Test::Tutorial` [1]

[1] `http://search.cpan.org/~rgarcia/perl-5.10.0/lib/Test/Tutorial.pod`

Test::More

```
use Test::More tests => 3;
use File::Temp qw(tempfile);

BEGIN {
        use_ok('Net::BitTorrent::Builder') or
        BAIL_OUT('Could not load Net::BitTorrent::Builder');
}

my ($fh, $filename) = tempfile(UNLINK => 1);
print $fh, "foo";

my $builder = Net::BitTorrent::Builder->new(name => $filename);
isa_ok($builder, 'Net::BitTorrent::Builder');

my $hash = "f1d2d2f924e986ac86fdf7b36c94bcdf32beec15";
is($builder->info_hash, $hash, "info hash ok");
```

Test::Exception, Test::Deep

```
# Anfang wie auf der letzten Folie

throws_ok { Window->new(type => 'fnord') }
   qr/Invalid window type/,
   'Invalid window type throws';

my $original_rect = Rect->new(x => 0, y => 0, w => 30, h => 30);
my $window = Window->new(type => 'utility', rect => $original_rect);
ok(eq_deeply($window->rect, $original_rect),
   "rect unmodified before mapping");

$window->map;
my $new_rect = $window->rect;
ok(!eq_deeply($new_rect, $original_rect),
   "window got repositioned");
```

EOF

Fragen?