Testing Legacy Code

Milton Keynes Perl Mongers, March 14 2006

Nik Clayton, nikc@cpan.org

Testing with Perl

```
% prove -r t          # or “make test”
t/03commandline....ok
t/02basics.........ok
t/01compile........ok
t/pod..............ok
All tests successful.
Files=4, Tests=34,  3 wallclock secs
( 1.55 cusr +  0.77 csys =  2.31 CPU)
```

200+ Test::* modules

- Test::Simple
- Test::More
- Test::Differences
- Test::Benchmark
- Test::HTML::Content
- ...

Great if....

- You write tests first, or
- The code you're testing is not crack-fuelled

- Some of you may see where I'm going with this

GEMSdb

```
% ls | wc -l
      85

% grep ‘use strict’ * | wc -l
      0

% grep ‘use warnings’ * | wc -l
      0

% grep ‘my’ * | wc -l
      18
```

Some of my favourites...

```
if($testing eq ‘TRUE’) { ... }

if($testing eq ‘FALSE’) { ... }

&get_rollup_info(*rolloc,*roolflr,*roldnm,*rolmcd,
*rollhrg,*roliig);

foreach $csrits (keys %incrs) {
next if ($incrs{$csrits} eq "UNDEFINED");
if ($incrsq{$csrits} ne "TRUE") { $cntcrsm++; push
(@CRSM, $crsline{$csrits})
; } }
```

The serious flaws are masked by the superficial flaws

Obvious fixes

- “use strict; use warnings”
- Sane indentation style
- Replace code with CPAN equivalents where possible
- Remove redundant code

Chicken / Egg

- Existing code is hostile to unit testing
- Can’t fix code without making changes
- How do I know I’ve not introduced other bugs?

Hypothesis

- Given the same input, two runs of the same program should produce the same output
- Problem: I don't know this code
- Solution: Other engineers/ops team do

Have them write a test plan

- Input files:
  gemsdb
- Output files:
  aliases.ssmb.<doy>.<pid>
  virtual.ssmb.<day>
  ....

- Make a test tree

```
mkdir -p /tmp/test/input
mkdir -p /tmp/test/new
mkdir -p /tmp/test/old
```

- Take snapshot of input file

```
cp $INPUT_FILE /tmp/test/input
```

- Run old program

```
/path/to/old/prog.pl -i /tmp/test/input/gemsdb 2> /tmp/test/old/stderr \
  > /tmp/test/old/stdout
```

- Snapshot output to /tmp/test/old
- Run new program

```
/path/to/new/prog.pl -i /tmp/test/input/gemsdb 2> /tmp/test/new/stderr \
  > /tmp/test/new/stdout
```

- Snapshot output to /tmp/test/new
- “diff -ur /tmp/test/old /tmp/test/new”

Test plan → driver

- Take test plan, write Perl script that implements it
- Each plan becomes a new script
- Most of these scripts look very similar
- Refactor common code in to new script, test plan becomes a config file

Overriding Perl

- Some code is still extremely hostile to testing
- Perl’s ability to override core functionality is very useful

Generating output file list

- open(FILE,“>$foo$bar$baz.$sfx”) or die “dead”;
- What’s the filename?
- Perl can tell us

t.pl

```
#!/usr/bin/perl

use warnings;
use strict;

open(F, '<', '/etc/motd') or die "$!\n";
print while(<F>);
close(F);

open(my $fh, '/etc/motd') or die "$!\n";
print while(<$fh>);
close($fh);
```

OpenWithLogging.pm

```perl
package OpenWithLogging;

use strict;
use warnings;

sub import {
    *CORE::GLOBAL::open = \&open_with_logging;
}

# ... continued overleaf
```

```
sub open_with_logging (*;$@) {
    my($pkg, $filename, $line) = caller();
    print STDERR "$filename:$line:open('", join("', '", @_), "')\n";

    if(defined($_[0])) {
        use Symbol ();
        my $fh = Symbol::qualify($_[0], $pkg);
        no strict 'refs';

        if(@_ == 1) {
            return CORE::open($fh);
        } elsif(@_ == 2) {
            return CORE::open($fh, $_[1]);
        } else {
            return CORE::open($fh, $_[1], @_[2..$#_]);
        }
    } else {
        if(@_ == 1) {
            return CORE::open($_[0]);
        } elsif(@_ == 2) {
            return CORE::open($_[0], $_[1]);
        } else {
            return CORE::open($_[0], $_[1], @_[2..$#_]);
        }
    }
}
1;
```

Results

```
% ls
OpenWithLogging.pm          t.pl

% perl -MOpenWithLogging ./t.pl > /dev/null
./t.pl:6:open('F', '<', '/etc/motd')
./t.pl:12:open('', '/etc/motd')
```

Other uses for this

- Force open() to fail, to test that error handling code works
- Change paths on the fly
  - Poor man’s chroot(8)

Overriding system()

- Many of the programs spawn external commands
- Download data files using /usr/bin/ftp
- Plays havoc with automated tests

t2.pl

```
#!/usr/bin/perl

use warnings;
use strict;

system qw(echo hello world!);  # Canonical greeting
system qw(ftp ftp://ftp.internal/path/to/file);
```

MySystem.pm

```
package MySystem;

use strict;
use warnings;

sub import {
    *CORE::GLOBAL::system = \&my_system;
}

# ... continued overleaf
```

```perl
use File::Copy;

my %local_from_url = (
  'ftp://ftp.internal/path/to/file' => '/etc/motd',
);

sub my_system {
    if($_[0] =~ /ftp/) {
        print 'Overriding FTP command: "',
            join(' ', @_), "\"\n";

        File::Copy::copy($local_from_url{$_[1]}, '.');

        $? = 0;       # Set return value explicitly
    } else {
        CORE::system(@_);
    }
}

1;
```

Results

```
% ls
MySystem.pm         t2.pl

% perl -MMySystem t2.pl
hello, world!
Overriding FTP command: "ftp ftp://ftp.internal/path/
to/file"

% ls
MySystem.pm         motd            t2.pl
```

See Also

- Test::MockObject
- Sub::Override

Conclusions

- Still an ongoing project - many tests written, many more to write
- Legacy code is difficult to test modularly
- Step back, test at a higher level
- Perl's ability to easily redefine core functionality is very useful

Thanks for listening

Any questions?

Bonus Slides

Subroutine prototypes

- Generally a bad idea
- Necessary to override *CORE::GLOBAL::open
- perl -e ‘print prototype “CORE::open” *;$@

http://www.perlmonks.org/?node_id=124339

&some_sub;

- These two calls are identical:*

```
&some_sub(); # Empty arg. list
some_sub();  #  ditto
```

- These two calls are not:

```
&some_sub;   # some_sub(@_);
some_sub;    # some_sub()
```

* Almost. & also disables prototype checking

Args to import

```
% perl -MSome::Module=foo,bar foo.pl

package Some::Module;

sub import {                    # args passed in @_ as normal
    print $_[0], “\n”;    # ‘foo’
    print $_[1], “\n”;    # ‘bar’
}
```