Adding automated tests to existing projects

Adding automated tests to existing projects

Problems of programming

Code is buggy

- Human testing doesn't scale
- Human time is too expensive
- We test manually, intermittently, or not at all

Human testing doesn't scale

Bugs keep reappearing.

We're afraid to change the code.

Changes in code here
break code over there.

New code takes
too long to write.

Our code
is poorly designed
(especially the APIs).

Promises

- Faster coding
- Fewer bugs
- Prevent regressions
- Improved confidence in the code
- Refactor with impunity
- Documentation & examples for free

Test-first bug fixing

- Don't reach for the debugger
- *prove* was made for test-first
- Once the bug is fixed, it's unlikely to regress

Overview of Perl testing

- What did we just use?
- Lightweight
  - No Java-like structure necessary
- Test::Harness, Test::More & TAP
- prove

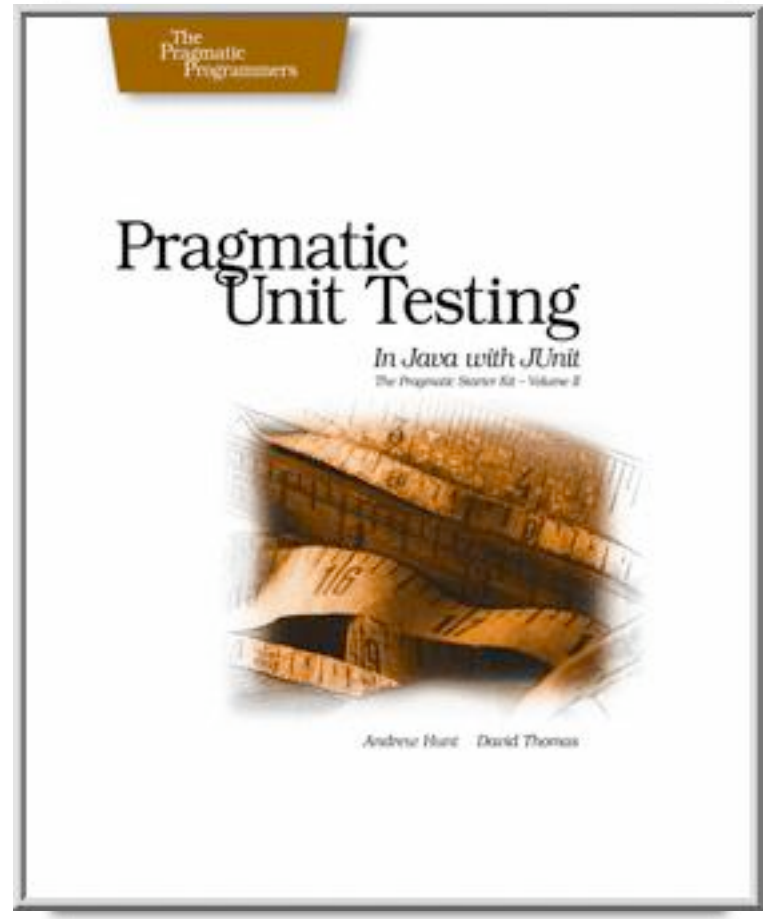
Test-first new code

- Logical extension of bug fixing
- Write docs as we write new functionality
- Define the API
- Code/docs/tests must all agree
- Test anti-examples

What makes a good test?

Right-BICEP

- Is it **right**?
- **B**oundaries?
- **I**nverse conditions?
- **C**ross-checking
- **E**rror conditions?
- **P**erformance



Ten cool tests you can write today

No default passwords

- Make sure there are no defaults

```
use constant USER => 'SYS';
use constant PASS => 'CHANGE_ON_INSTALL';

my $dbh = DBI->connect( $mydb, USER, PASS );
my $sth = $dbh->prepare( "select 1 from dual" );
my $rc = $sth->execute;
isnt( $rc, 0, 'SYS user doesn't have default PW' );
```

All ISBNs valid

- Get all ISBNs and run them through a validator

```
my $sth = $dbh->prepare( "select ISBN from BOOK" );
$sth->execute();
my $bad;
while ( my $row = $sth->fetch ) {
    $isbn = new Business::ISBN( $row->[0] );
    if ( !$isbn->is_valid ) {
        fail( "Invalid ISBN(s) found" )
            unless $bad++;
        diag( "$row->[0] is invalid" );
    }
}
pass( "No bad ISBNs found" ) unless $bad;
```

Use warnings/strict

- First find all our files

```
# Find all Perl files, but don't look in CVS

my $rule = File::Find::Rule->new;
$rule->or(
  $rule->new->directory->
        name('CVS')->prune->discard,
  $rule->new->file->name( '*.pl','*.pm','*.t' ));

my @files = $rule->in( $base );
check( $_ ) for @files;
```

Use warnings/strict

- Check for warnings & strict

```
sub check {
  my $filename = shift;

  my $dispname =
    File::Spec->abs2rel( $filename, $base );

  local $/ = undef;
  open( my $fh, $filename ) or
    return fail( "Couldn't open $dispname: $!" );
  my $text = <$fh>;
  close $fh;

  like( $text, qr/use strict;/,
    "$dispname uses strict" );
  like( $text, qr/use warnings;|perl -w/,
    "$dispname uses warnings" );
} # check()
```

All .pm have .t

- Get a list of .pm files and then...

```
sub check {
  my $filename = shift;

  my $tname = $filename;
  $tname =~ s/ \.pm \Z / .t /x
    or die "Only send me .pm files, please";
  ok( -s $tname, "$filename has a test file" );
} # check()
```

All HTML valid

- Get a list of HTML files, and...

```
for my $filename ( @files ) {
  open( my $fh, $filename ) or
    fail( "Couldn't open $filename" ), next;

  my $text = do { local $/ = undef; <$fh> }
  local $/ = undef;
  close $fh;

  my $lint = HTML::Lint->new;
  $lint->only_types( HTML::Lint::Error::STRUCTURE );
  html_ok( $lint, $text, $dispname );
}
diag( "$html HTML files" );
```

Installed modules

- Create a hash of modules you must have

```
my %requirements = (            # 0 means we don't care
    'Business::ISBN'          => 0,
    'Carp::Assert'            => '0.17',
    'Carp::Assert::More'      => '1.10',
    'Date::Calc'              => 0,
    'Date::Manip'             => 0,
    'DateTime'                => '0.20',
    'DB_File'                 => '1.808',
    'Exporter'                => '5.562',
    'File::Spec'              => '0.82',
    'File::Temp'              => '0.13',
    ...
);
```

Installed modules

- ... and then check for them.

```
for my $module ( sort keys %requirements ) {
    my $wanted = $requirements{ $module };
    if ( use_ok( $module ) ) {
        if ( $wanted ) {
            my $actual = $module->VERSION;
            cmp_ok( $actual,'>=',$wanted, $module );
        }
        else {
            pass( "$module loaded" );
        }
    }
    else {
        fail( "Can't load $module" );
    }
} # for keys %requirements
```

sprintf works

- PHP broke sprintf

```
// sprintf broke between PHP 4.2.3 and 4.3.0

require( "Test.php" );
plan( 4 );
diag( "PHP Version " . phpversion() );

$masks = Array( "%-3.3s", "%.3s", "%-.3s" );
$str = "abcdefg";
foreach ( $masks as $mask ) {
    $result = sprintf( "[$mask]", $str );
    is( $result, "[abc]", "[$mask]" );
}
```

No embedded tabs

- Find and print any embedded tab files

```
sub check {
    my $fname = shift;
    open( my $fh, "<", $fname )
        or die "$fname: $!\n";

    my $bad;
    while ( my $line = <$fh> ) {
        if ( $line =~ /\t/ ) {
            fail( "$fname has tabs" ) unless $bad++;
            diag( "$.: $line" );
        }
    } # while

    pass( "$fname is tab-free" ) unless $bad;
} # check
```

All POD is OK

- Test::Pod makes it terribly simple

```
use Test::More;
use Test::Pod 1.00;

all_pod_files_ok();
```

All functions have POD

- Test::Pod::Coverage makes it terribly simple

```
use Test::More;
use Test::Pod::Coverage 1.04;

all_pod_coverage_ok();
```

For the managers

- Make tests & docs part of code standards
  - Code without tests is not complete
- Tests & docs are part of code reviews
  - All three are reviewed at once.

For the managers

- Track and post metrics
  - Test counts over time are reassuring for you and for programmers
  - Trends matter
  - Individual numbers don't

For the managers

- Make testing part of hiring
  - "What experience do you have with automated testing?"
  - "What could go wrong with this code? How could you test for it?"

Best practices

- Code/tests/docs must all agree
- Continuous integration
- Don't worry about test execution time
- Do worry about programmer time
- It's an investment, albeit a pretty cheap one.

Best practices

- Bug tracking system & source control
  - Tie tests & commits to specific tickets
- Module::Starter to get things going
- Devel::Cover to check test coverage

Best practices

- Test anything that ever goes wrong
- Add tests every time you fix a bug
- Treat failed tests like an oil light

Recommended resources

- Perl Testing: A Developer's Notebook, by Ian Langworth & chromatic
- Perl Best Practices, by Damian Conway
- Pragmatic Unit Testing In Java With JUnit, by Andrew Hunt & Dave Thomas
- Test-Driven Development, by Kent Beck
- Refactoring, by Martin Fowler

Thanks for coming