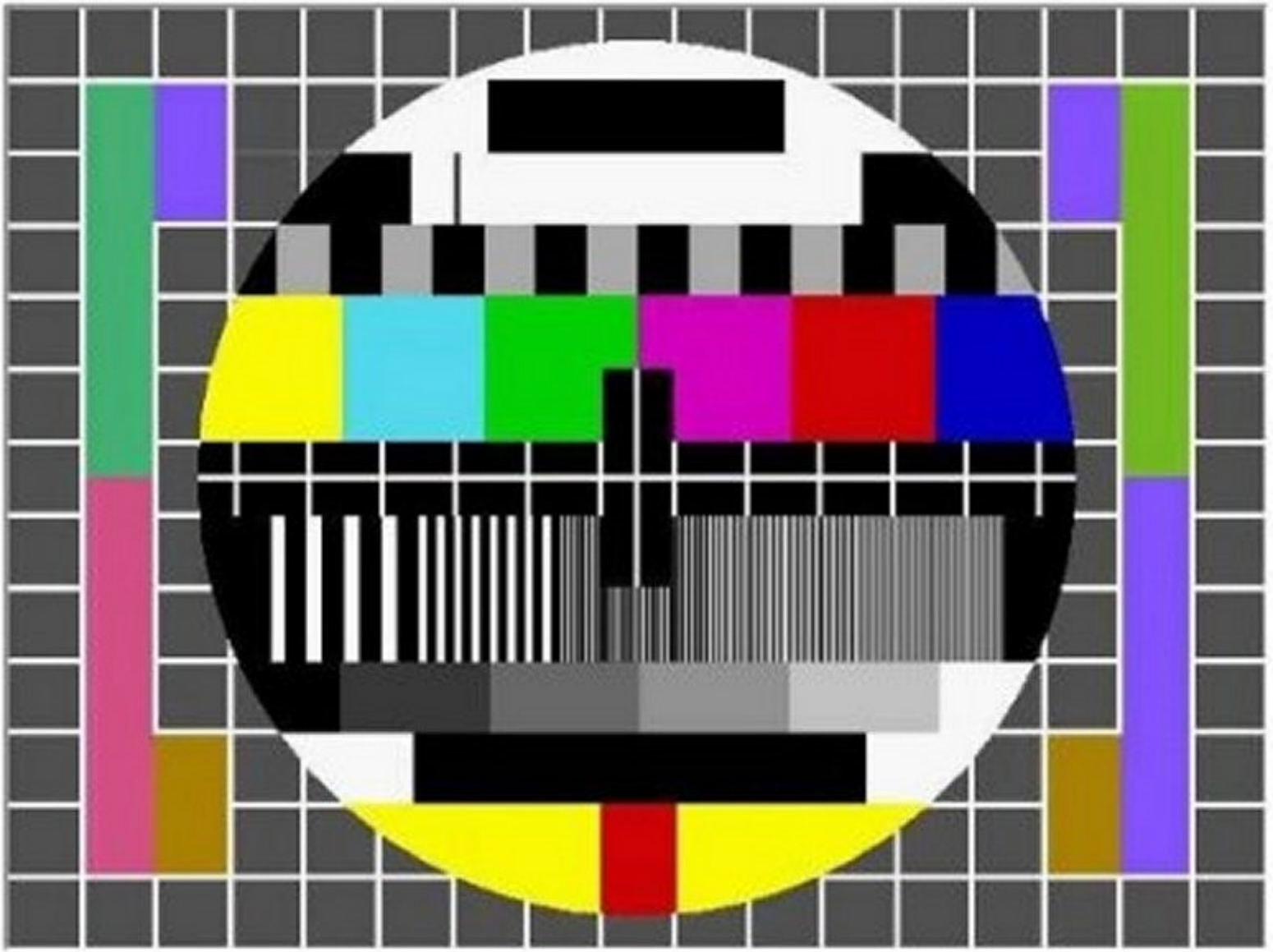
Test::Am::Meisten



Test::More::More

Testen für Fortgelaufene

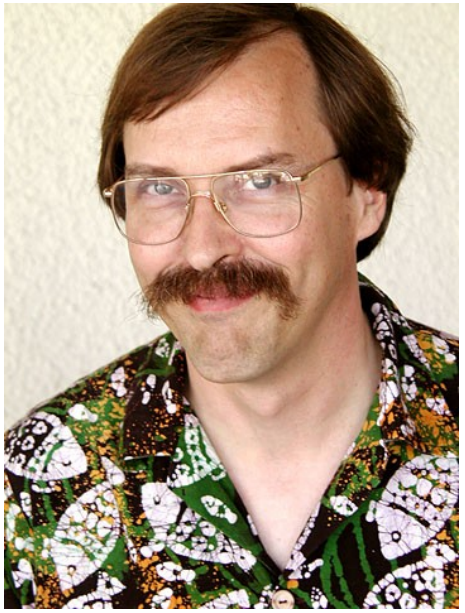
Test::Am::Meisten

Am Anfang

Am Anfang

1987: Perl 1.0

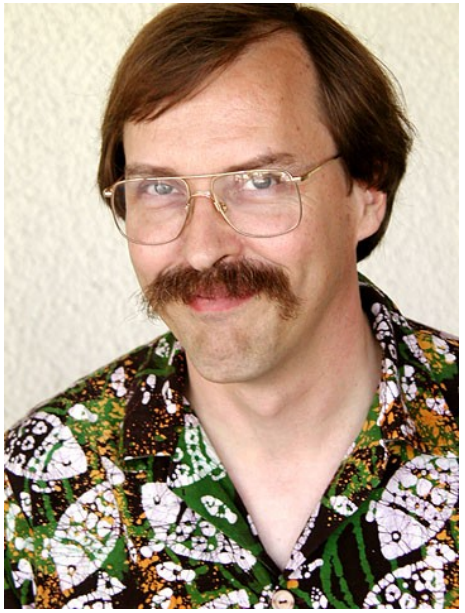
starke Testsuite



1987: Perl 1.0

starke Testsuite

Anfänge von TAP

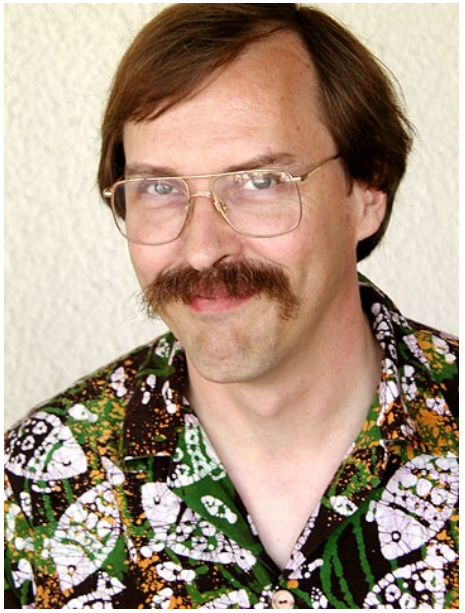


1987: Perl 1.0

starke Testsuite

Anfänge von TAP

Testscripte benutzten print



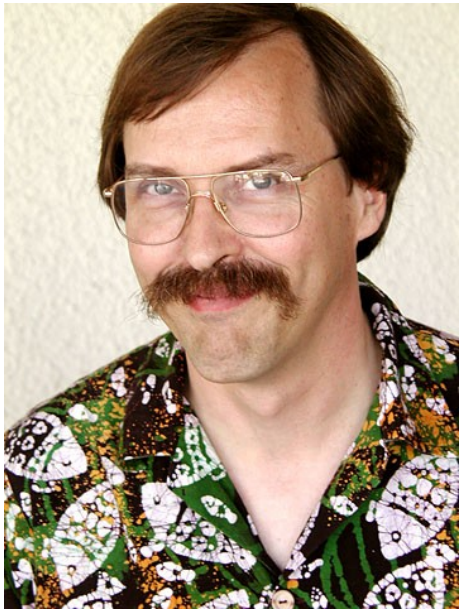
1987: Perl 1.0

starke Testsuite

Anfänge von TAP

Testscripte benutzten print

TEST fasste zusammen



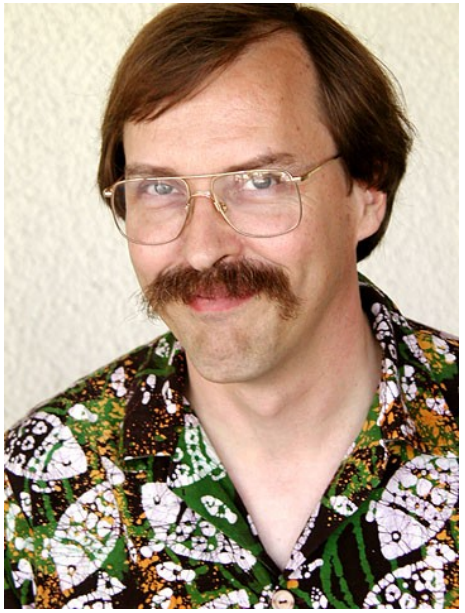
1987: Perl 1.0

starke Testsuite

Anfänge von TAP

Testscripte benutzten print

TEST => Test::[Harness|Run]



Test

```
use Test;



tests =>, todo=>, onfail =>

ok();
skip();
plan();                              # Joshua Nathaniel Pritikin
```

sofort austauschen

```
# use Test;
use Test::Legacy;
```

Test::Legacy::More

```
# use Test;
use Test::Legacy;
```

Test::Legacy

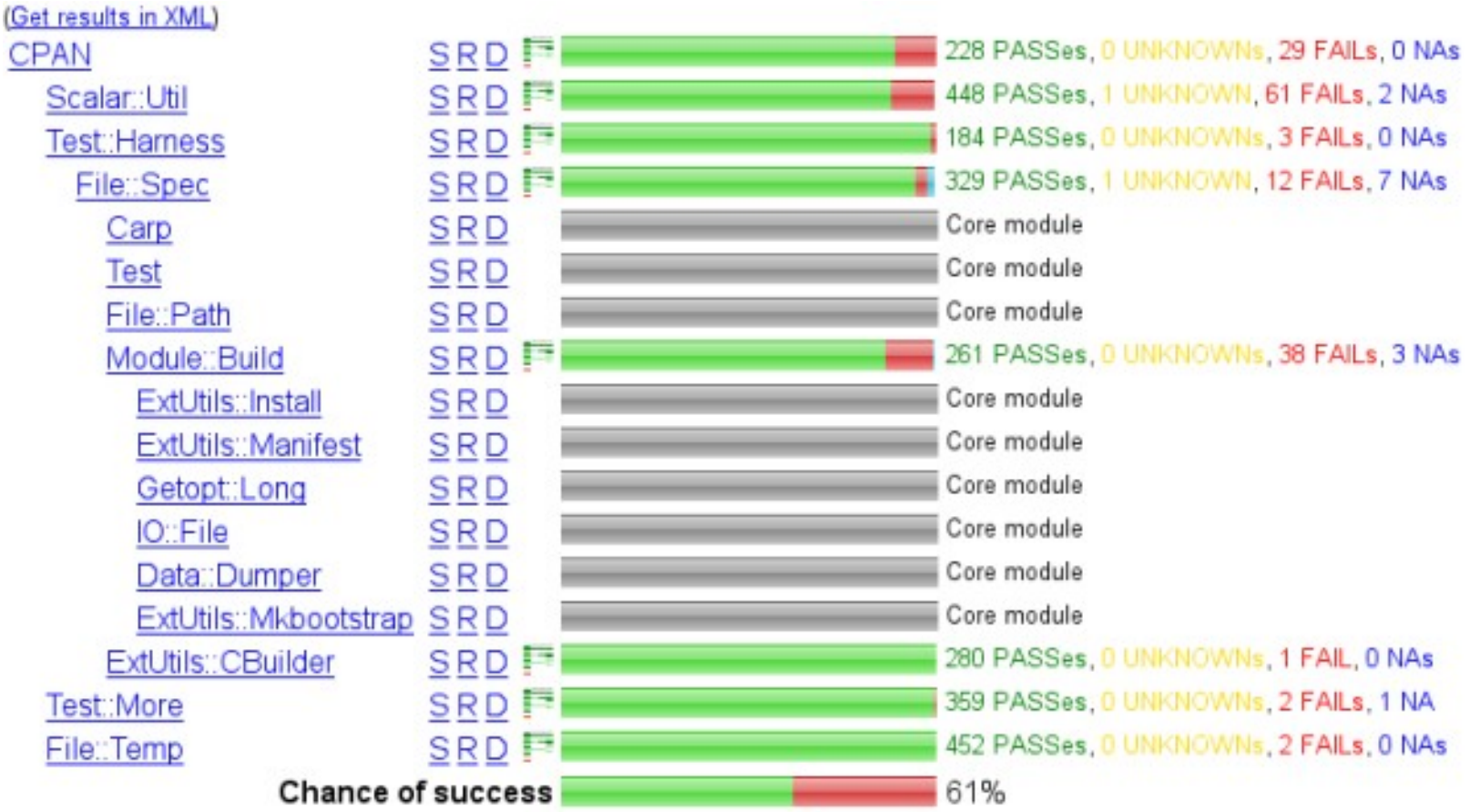
```
use Test::Legacy;
use Test::Legacy::More;

# use Test::More import => [ qw(!ok !skip !plan)];
```

Warum?

TAP : Test Anything Protocol

Wichtig!


(Get results in XML)
CPAN S R D 228 PASSes, 0 UNKNOWNs, 29 FAILs, 0 NAs
Scalar::Util S R D 448 PASSes, 1 UNKNOWN, 61 FAILs, 2 NAs
Test::Harness S R D 184 PASSes, 0 UNKNOWNs, 3 FAILs, 0 NAs
File::Spec S R D 329 PASSes, 1 UNKNOWN, 12 FAILs, 7 NAs
Carp S R D Core module
Test S R D Core module
File::Path S R D Core module
Module::Build S R D 261 PASSes, 0 UNKNOWNs, 38 FAILs, 3 NAs
ExtUtils::Install S R D Core module
ExtUtils::Manifest S R D Core module
Getopt::Long S R D Core module
IO::File S R D Core module
Data::Dumper S R D Core module
ExtUtils::Mkbootstrap S R D Core module
ExtUtils::CBuilder S R D 280 PASSes, 0 UNKNOWNs, 1 FAIL, 0 NAs
Test::More S R D 359 PASSes, 0 UNKNOWNs, 2 FAILs, 1 NA
File::Temp S R D 452 PASSes, 0 UNKNOWNs, 2 FAILs, 0 NAs
Chance of success 61%

Test::Simple

```
use Test::Simple 'no_plan';

ok( $dies eq $das, 'dies gleich das' );
```

Test::Simple

```
package Test::Simple;

use 5.006;
use strict;

our $VERSION = '0.94';

use Test::Builder::Module;
our @ISA    = qw(Test::Builder::Module);
our @EXPORT = qw(ok);

my $CLASS = __PACKAGE__;
```

Test::Between

```
package Test::Between;

use base 'Exporter';
our @EXPORT = qw( is_between);

use Test::Builder;
my $test = Test::Builder->new();

sub is_between($$$;$) {
    my ($item, $lower, $upper, $name) = @_;
    return ( $test->ok(...) || $test->diag(„…“)
    );
}1;
```

Test::More

```
use Test::More 'no_plan'; # or skip_all => $reason;
```

ok, is, isnt, like, unlike, cmp_ok, is_deeply

use_ok, require_ok, can_ok, isa_ok, new_ok

SKIP, TODO, BAIL_OUT, subtest

diag, note, explain, pass, fail, done_testing

Test::Most

Devel::Cover

```
$ cover -test
Deleting database /home/pjcj/g/perl/Shell-Source-0.01/cover_db
PERL_DL_NONLAZY=1 /usr/local/pkg/cover/c1/perl-5.8.7/bin/perl "-
MExtUtils::Command::MM" \
  "-e" "test_harness(0, 'blib/lib', 'blib/arch')" t/*.t
t/bash....ok
...
All tests successful.
Files=6, Tests=24, 17 wallclock secs (16.16 cusr +  1.01 csys = 17.17 CPU)
Reading database from /home/pjcj/g/perl/Shell-Source-0.01/cover_db

---------------------------------- ------ ------ ------ ------ ------ ------ ------
File                                stmt   bran   cond    sub    pod   time  total
---------------------------------- ------ ------ ------ ------ ------ ------ ------
blib/lib/Shell/Source.pm           96.6   65.0   66.7   90.9    n/a  100.0   86.1
Total                              96.6   65.0   66.7   90.9    n/a  100.0   86.1
---------------------------------- ------ ------ ------ ------ ------ ------ ------

Writing HTML output to /home/pjcj/g/perl/Shell-Source0.01/cover_db/coverage.html
…
done.
```

Devel::Cover

uc			
Criterion	*Covered*	*Total*	*%*
statement	4	9	100.0
branch	2	4	100.0
condition	2	6	100.0
subroutine	0	1	100.0
pod			n/a
total	8	20	100.0

Test::Pod::Coverage

```
use Test::Spelling;
use Test::More;


all_pod_coverage_ok();
```

Test::Pod

```
use Test::More;

eval {
    require Test::Pod;
    import Test::Pod;
};

all_pod_files_ok();
```

Test::Spelling

```
use Test::Spelling;
use Test::More;


pod_file_spelling_ok( $file, "POD file spelling OK" );
```

Test::Kwalitee

```
use Test::More;

eval {
    require Test::Kwalitee;
    Test::Kwalitee->import()
};

plan( skip_all => 'Test::Kwalitee not installed; skipping' )
    if $@;
```

Test::Kwalitee

- extractable, no_symlinks
- has_readme, have_manifest, have_meta_yml
- have_buildtool, have_changelog, have_tests
- use_strict, proper_libs
- no_pod_errors, have_test_pod, have_test_pod_coverage

Test::Perl::Critic

```
use Test::Perl::Critic;
use Test::More tests => 1;

critic_ok($file);

all_critic_ok($dir_1, $dir_2, $dir_N);

all_critic_ok();
```

Test::Most

```
use Test::More;

use Test::Differences;

use Test::Deep;

use Test::Exception;

use Test::Warn;
```

mehr ...

Test::Most

```
use Test::More;

use Test::Differences;

use Test::Deep;

use Test::Exception;

use Test::Warn;
```

mehr ...

Test::Most

- die_on_fail bail_on_fail
- restore_fail set_failure_handler

- explain # prove -v
- show

- all_done

Test::Most

```
use Test::More;

use Test::Differences;

use Test::Deep;

use Test::Exception;

use Test::Warn;
```

mehr ...

Test::Differences

Test::Differences

```
use Test::More;

is_deeply([1,2],[1,2,[3]], '… nicht gleich');
```

Test::Differences

```
use Test::More;

is_deeply([1,2],[1,2,[3]], '… nicht gleich');
```

```
#   Failed test 'is_deeply'
#   at t\02-more.t line 12.
#     Structures begin differing at:
#          $got->[3] = ARRAY(0xae3704)
#     $expected->[3] = Does not exist
```

Test::Differences

```
use Test::More tests => 1;
use Test::Differences;

eq_or_diff([1,2],[1,2,[3]], '… nicht gleich');
```

Test::Differences

```
use Test::Most tests => 1;

eq_or_diff([1,2],[1,2,[3]], '… nicht gleich');
```

Test::Differences

```
#   Failed test '...nicht gleich'
#   at t\04-diff.t line 8.
# +---+------+---+------------+
# |  Elt|Got   |  Elt|Expected |
# +---+------+---+------------+
# |    0|[     |    0|[         |
# |    1|   1, |    1|  1,      |
# *    2|   2  *    2|  2,      *
# |     |      *    3|  [       *
# |     |      *    4|    3     *
# |     |      *    5|  ]       *
# |    3|]     |    6|]         |
# +---+------+---+------------+
```

```
eq_or_diff([1,2],[1,2,[3]]
```

Test::Differences

4 Anzeigestile:

* table_diff; #(the default)
* unified_diff;
* oldstyle_diff;
* context_diff;

Test::Differences

```
#   Failed test '...nicht gleich'
#   at t\04-diff.t line 8.
# --- Got
# +++ Expected
# @@ -1,4 +1,7 @@
#  [
#    1,
# -  2
# +  2,
# +  [
# +    3
# +  ]
```

```
unified_diff;
eq_or_diff([1,2],[1,2,[3]]
```

Test::Differences

```
#   Failed test '...nicht gleich'
#   at t\04-diff.t line 8.
# 3c3,6
# <   2
# ---
# >   2,
# >   [
# >     3
# >   ]
```

```
oldstyle_diff;
eq_or_diff([1,2],[1,2,[3]]
```

Test::Differences

```
# *** Got
# --- Expected
# ***************
# *** 1,4 ****
#     [
#         1,
# !       2
#     ]
# --- 1,7 ----
#     [
#         1,
# !       2,
# !       [
# !           3
# !       ]
#     ]
```

```
context_diff;
eq_or_diff([1,2],[1,2,[3]]
```

Test::Differences

```
use Test::More;                 # oder Test
use Test::Differences;          # use Text::Diff;
```

```
eq_or_diff($gegeben, $gesucht, $meldung);
# autoselect:
eq_or_diff_data($gegeben, $gesucht, $meldung);
eq_or_diff_text($gegeben, $gesucht, $meldung);
```

Test::More

```
use Test::More;

is_deeply([1,2],[1,2,[3]], '… nicht gleich');
```

Test::More

```
use Test::More;

is_deeply([1,2],[1,2,[3]], '… nicht gleich');

eq_array();
eq_hash();
eq_set();
```

Test::More

```
use Test::More;

is_deeply([1,2],[1,2,[3]], '… nicht gleich');

eq_array();
eq_hash();
eq_set();
```

Test::Deep

```
use Test::More 'no_plan';
use Test::Deep;

cmp_deeply($got, $expected, $name);    # is_deeply
```

```
not ok 1 – test name
#   Failed test 'test name'
#   at t\03-deep.t line 7.
# Compared $data->[3][2][0]
#    got : '2'
# expect : '1'
```

Test::Deep

```
use Test::Most 'no_plan';


cmp_deeply($got, $expected, $name); # is_deeply
eq_deeply($got, $expected);                # no test
cmp_bag(\@got, \@bag, $name);      # Reihenfolge egal
cmp_set(\@got, \@set, $name);      # Mehrfache mögl.
cmp_methods(\@got, \@methods, $name);
```

Test::Deep

```
use Test::Most 'no_plan';


cmp_deeply($got, $expected, $name); # is_deeply
eq_deeply($got, $expected);                 # no test
cmp_bag(\@got, \@bag, $name);       # bag wie in P6
cmp_set(\@got, \@set, $name);       # eq_set
cmp_methods(\@got, \@methods, $name);
```

Aliase

```
use Test::Deep;


cmp_bag(\@got, \@bag, $name);
cmp_deeply(\@got, bag(@exp), $name);

cmp_set(\@got, \@set, $name);
cmp_deeply(\@got, set(@exp), $name);

cmp_methods($obj, \@exp, $name);
cmp_deeply($obj, methods(%exp), $name);
```

kombinierbare Befehle

```
cmp_deeply(\@got, bag(@exp), $name);

cmp_deeply(\@got, set(@exp), $name);

cmp_deeply($obj, methods(%exp), $name);
```

kombinierbare Befehle

bag(@exp)

set(@exp)

methods(%exp)

kombinierbare Befehle

ignore()
any(@exp) / all(@exp)
bag(@exp)
set(@exp)
super|sub . hash|bag|set . of ()
array_each
str / num / bool / code
re(@exp)
shallow($ref)
methods(@exp)
listmethods(@exp)
noclass(@exp)
useclass(@exp)
isa(@exp)

Test::Deep::NoTest

```
use Test::Deep::NoTest;
```

Test::Exception

```
use Test::Exception;

throws_ok( {...}, $ausnahmeklasse, $name);
throws_ok( {...}, $regex, $name);
lives_ok( {…}, $name);
dies_ok({...}, $name);
```

Test::Warn

```
# Welche Warnungen wurden ausgegeben?

use Test::Warn;

warning_is( {...} $warnung, $name);
warnings_are(  {...} @warnungen, $name);

warning_like();
```

Test::Most

use Test::More;

use Test::Differences;

use Test::Deep;

use Test::Exception;

use Test::Warn;

mehr …

fertig

Bundle::Test

Test
Test::Simple
Test::More
Test::Harness
Test::MockObject
Test::Pod
Test::Builder
Test::Warn
Test::Inline
Test::Strict
Test::Distribution
Test::Perl::Critic
Test::Kwalitee
Pod::Spell
Test::Spelling
Devel::Cover
...

Test::NoWarnings

```
use Test::NoWarnings;
```

Test::NoWarnings

```
use Test::NoWarnings;
```

```
# $plan += 1
```

```
>ok $plan - no warnings
```

Devel::Hide

```
use Devel::Hide qw[Test::Most];

use Tests::Most;   # error
```

Test::Script

```
use Test::Script;

script_compiles();

script_runs();
```

Test::Cmd

```
use Test::Cmd;

my $cmd = Test::Cmd->new ( prog => 'prog.pl',
                           interpreter => 'script_interpreter',
                           string  => 'identifier_string',
                           workdir  => '',
                           subdir  => 'dir',
                           match_sub  => $code_ref,
                           verbose  => 1, ...);

$cmd->write(  # Datei schreiben
$cmd->run(    # $? == 0
```

Test::ManyParams

```
use Test::ManyParams;


all_ok { tannu(shift) }, @para, $nachricht;
all_ok { tuwas(@_) }, [ @a, @b], $nachricht;

qw(all any) X~ qw(ok is isnt)
most_ok{ sub(@_) }, [ @a, @b] => $nr, $nachricht;
```

Test::Inline

```
=begin testing label
  ....
  ....
=end testing
```

Test::Inline

```
=begin testing label
  ....
  ....
=end testing
```

inline2test

Test::Unit

```
use Test::Unit;
```

Test::Unit

```
use Test::Unit;                                    # 2001
```

Test::Unit::Lite

```
use Test::Unit;                                    # 2008
```

Test::Class

```
package Meine::Test::Klasse;
use base qw(Test::Class);


sub eats : Test(3) {
```

```
Meine::Test::Klasse->runtests
```

Test::Class

```
package Meine::Test::Klasse;


sub eats : Test(3) {


package main;
Meine::Test::Klasse->runtests;
```

Test::Class

```
BEGIN {
    chdir 't' if -d 't';
    require Meine::Test::Klasse;
}

Meine::Test::Klasse->runtests;

# Meine/Test/Klasse.pm
```

Test::Class

```
sub tischlein_deck_dich : Test (startup) {
sub knueppel_aus_dem_sack : Test (shutdown) {

sub neues_tischtuch : Test (setup => 2 ) {
sub abraeumkommando : Test (teardown) {
```

Test::Class::Sugar

```
use Test::Class::Sugar;
```

Test::Able

Test::Able

Test::Able

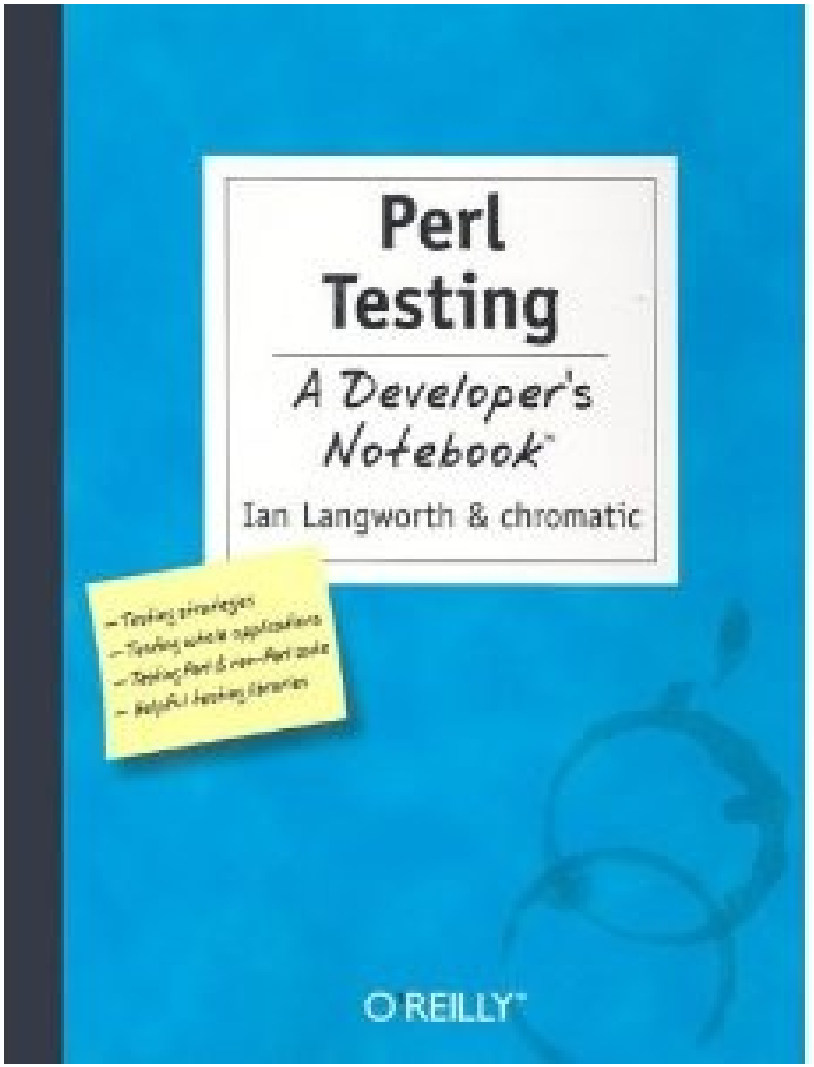
```
package MyTest;

use Test::Able;
use Test::More;

startup        some_startup  => sub { ... };
setup          some_setup    => sub { ... };
test plan => 1, foo          => sub { ok( 1 ); };
test           bar           => sub {
    my @runtime_list = 1 .. 42;
    $_[ 0 ]->meta->current_method->plan( scalar @runtime_list );
    ok( 1 ) for @runtime_list;
};
teardown       some_teardown => sub { ... };
shutdown       some_shutdown => sub { ... };

MyTest->run_tests;
```

Gutes Buch


Perl
Testing
A Developer's Notebook
Ian Langworth & chromatic
O'REILLY

Links

http://testanything.org

http://www.cpantesters.org/

http://qa.perl.org/testing/

Glückliches Ende