The need for Speed ERM Testing

Marcus Börger

International PHP Conference 2006

The need for Testing

- ☑ Why Testing
- ☑ Introduction to phpt Testing

Why Testing

- ☑ Programming often comes along with code re-use
  - ☑ Code re-use comes along with code changes
  - ☑ Code changes are changes
- ☑ Even for a few codelines - looking is not enough
  - ☑ Names can misslead
  - ☑ Code may have non obvious side effects
- ☑ Sometimes code is designed for a limited domain
  - ☑ Increasing/Changing that domain is error prone
- ☑ Code interaction is often underestimated
  - ☑ A bugfix in one function may affect other functions

How to test

- ☑ Testing after test log
  - ☑ Record problematic input actions and replay them
- ☑ Automated testing
  - ☑ Integration/System testing
  - ☑ Function testing
  - ☑ Unit testing
  - ☑ Acceptance/Requirements testing
  - ☑ Regression Testing

Integration testing

- ☑ Not only a particular pieces but the whole
  - ☑ Major is to verify all parts work together
  - ☑ When working on real data it can detect system issues
- ☑ Often requires multiple test systems
  - ☑ A manual or automated log is required
  - ☑ Usually performed/organized by QA

Does the system work?

Function testing

- ☑ Execute parts of API
- ☑ Use common data (domain API is designed for)
- ☑ Use code from observed bugs

Does the API work?

Unit testing

- ☑ Execute testing on code
  - ☑ From single routines, to parts (usually not the whole)
  - ☑ Test private stuff
  - ☑ Analyse untouched code to write more tests
- ☑ Analytically find test data
- ☑ Use code from observed bugs

Does the code work?

Acceptance testing

- ☑ Requirements engineering
  - ☑ Develop tests from requirments

Does it do what the customer wants?

Regression testing

- ☑ Backwards compatibility test
  - ☑ Verify input against expected output

Does it still work as expected?

Non functional testing

- ☑ Performance
- ☑ Stability
- ☑ Usabilty
- ☑ Stress-Testing

Test driven development

- ☑ Think what you want or review specs
- ☑ Write tests
- ☑ Develop code and test
- ☑ Write more tests if you figure any weakness

What is phpt-Testing

- ☑ Easy 1 PHP script test system (run-tests.php)
- ☑ Everything goes into one file (*.phpt)
- ☑ Capable of testing any aspect of PHP
- ☑ Regression testing with pattern & regex matching
- ☑ Integrates with memcheck
- ☑ Used on http://gcov.php.net

Test file names

- ☑ Tests for bugs
  - bug<bugid>.phpt — bug17123.phpt
- ☑ Tests for functions
  - <functionname>.phpt — dba_open.phpt
- ☑ General tests for extensions
  - <extname><no>.phpt — dba_003.phpt
- ☑ Do not use any .php files for includes or alike

Getting started with phpt

- Each test consists of several sections
  - Name
  - Input
  - Expected output

```
--TEST--
Hello World
--FILE--
Hello World
--EXPECT--
Hello World
```

Always output something that can be verified.

Getting started with phpt

- ☑ Each test consists of several sections
- ☑ The input is usually a php snippet
- ☑ An additional empty line makes cvs happy

```
--TEST--
Hello World
--FILE--
<?php echo "Hello World"; ?>
--EXPECT--
Hello World
```

Use only the long version of the php script tag.

Getting started with phpt

- ☑ Each test consists of several sections
- ☑ The input is usually a php snippet
- ☑ The expected out must not be fixed
  - ☑ Scanf-like expressions

```
--TEST--
Hello World
--FILE--
<?php echo "Hello World
--EXPECTF--
Parse error: syntax error, unexpected $end in %s.php on line %d
```

Do not check directories in error messages.

Getting started with phpt

- ☑ Each test consists of several sections
- ☑ The input is usually a php snippet
- ☑ The expected out must not be fixed
  - ☑ Scanf-like expressions

```
--TEST--
Hello World
--FILE--
<?php echo "Hello World
--EXPECTF--
Parse error: syntax error, unexpected $end in %s.php on line %d
```

When executed, the test file has .php ending.

Getting started with phpt

- ☑ Each test consists of several sections
- ☑ The input is usually a php snippet
- ☑ The expected out must not be fixed
  - ☑ Scanf-like expressions
  - ☑ Regular expressoins

```
--TEST--
Hello World
--FILE--
<?php echo "Hello World"
--EXPECTREGEX--
Parse error: (parse|syntax) error, unexpected $end in .* on .*
```

You can - but don't drop too much: It is "on line".

Use var_dump()

- ☑ Usually output variables are verified by var_dump
  - ☑ Allows to check for exact type
  - ☑ Allows to check for private/protected properties

```
--TEST--
Var_dump
--FILE--
<?php
var_dump(NULL);  Var_dump(0);
Var_dump(false); Var_dump("");
?>
--EXPECT--
NULL
int(0)
bool(false)
string(0) ""
```

When checking object IDs, use scanf/regex.

More scanf matching

- ☑ Allows matching blocks of output

%s	Any string	%i	Integers (includes "–")
%d	Numbers	%f	Floating point values
%c	Single characters	%x	Hexadecimal values
%w	Any amount of Whitespace	%e	DIRECTORY_SEPARATOR ('\' or '/').

- ☑ Cannot verify complex output

```
--TEST--
More Testing
--FILE--
<?php
$s = '123';
var_dump(str_shuffle($s));
var_dump($s);
?>
--EXPECTF--
string(3) "%s"
string(3) "123"
```

Don not use %d for string length, unless youhave to.

More regex matching

- ☑ Regex matching requies escaping
- ☑ Full regex support

```
--TEST--
More Testing
--FILE--
<?php
$s = '123';
var_dump(str_shuffle($s));
var_dump($s);
?>
--EXPECTREGEX--
string\(3\) "[123]{3}"
string\(3\) "123"
```

Be as precise as possible in matching expressions.

More output matching

- ☑ Huge output can be verified indirectly using md5
- ☑ When using files delete them before and after

```
--TEST--
Output validation using md5
--FILE--
<?php
$dest = dirname(__FILE__) . '/bug22544.png';
@unlink($dest);
imagePng(imageCreateTruecolor(640, 100), $dest);
Var_dump(md5_file($dest));
@unlink($dest);
?>
--EXPECT--
String(32) "10a57d09a2c63fad87b85b38d6b258d6"
```

Use dirname(__FILE__) as temporary directory.

More output matching

- ☑ Huge output can be verified indirectly using md5
- ☑ When using files delete them before and after
- ☑ Move clean-up code into a special section

```
--TEST--
Output validation using md5
--FILE--
<?php
$dest = dirname(__FILE__) . '/bug22544.png';
@unlink($dest);
imagePng(imageCreateTruecolor(640, 100), $dest);
Var_dump(md5_file($dest));
?>
--CLEAN--
<?php @unlink(dirname(__FILE__) . '/bug22544.png'); ?>
--EXPECT--
String(32) "10a57d09a2c63fad87b85b38d6b258d6"
```

Hide potential notices using the @ operator.

When tests get bigger

- The special section ===DONE=== ends the test
  - Only available in --FILE--
  - Anything below that will be ignored

```
--TEST--
More Testing
--FILE--
<?php
$s = '123';
var_dump(str_shuffle($s));
var_dump($s);
?>
===DONE===
<?php exit(0); ?>
--EXPECTF--
string(3) "%s"
string(3) "123"
```

With exit() in tests, no memleaks get reported.

Stopping the compiler

- ☑ Some --EXPECT-- prevent from running the phpt
- ☑ Use pseudo function __HALT_COMPILER()

```
--TEST--
SimpleXML: Attribute creation
--FILE--
<?php
$xml = '<?xml version="1.0" encoding="ISO-8859-1" ?><foo/>';
$sxe = simplexml_load_string($xml);
$sxe["attr"] = "value";
echo $sxe->asXML();
__HALT_COMPILER();
?>
--EXPECT--
<?xml version="1.0" encoding="ISO-8859-1"?>
<foo attr="value"/>
```

Here the '<?' in the output woult prevent execution.

An alternative to --FILE--

- ☑ Very specific to Bug #35382

```
--TEST--
Bug #35382 (Comment in end of file produces fatal error)
--FILEEOF--
<?php
eval("echo 'Hello'; // comment");
echo " World";
//last line comment
--EXPECT--
Hello World
```

Here the 't' of 'comment' is the very last test file byte.

Preconditions

- ☑ Tests may have several preconditions
- ☑ Include files are good for common preconditions
- ☑ Output "skip" if a precondition is not met
- ☑ Usefull: function_exists, extension_loaded, compare_version+phpversion

```
--TEST--
Check for exif_read_data, unusual IFD start
--SKIPIF--
<?php if (!extension_loaded('exif')) die('skip exif n/a');?>
--FILE--
<?php
$e=exif_read_data(dirname(__TEST__).'/test.jpg','',true,false);
var_dump($e['IFD0'][0],$e['IFD0'][1]);
?>
--EXPECT--
string(11) "Ifd00000009"
string(19) "2002:10:18 20:06:00"
```

Use die() and an explanation in –SKIPIF--.

Redirected tests

- ☑ Some extensions are drivers to others (e.g. PDO)
- ☑ The --REDIRECTTEST-- section replaces --FILE—
  - ☑ It gets evaluated and must return an array
  - ☑ Entry ENV contains the environment
  - ☑ Entry TESTS contains the test directory/files

```
--TEST--
SQLite
--SKIPIF--
<?php # vim:ft=php
if (!extension_loaded('pdo_sqlite')) print 'skip';
?>
--REDIRECTTEST--
// no start tag needed
return array(
     'ENV' => array(
         'PDOTEST_DSN' => 'sqlite::memory:'),
     'TESTS' => 'ext/pdo/tests');
```

There is no --FILE-- section in redirect tests.

Optional Input sections

- ☑ `--POST--`
  POST variables to be passed to the test script.
- ☑ `--POST_RAW--`
  RAW POST data (doesn't set the Content-Type).
- ☑ `--GET--`
  GET variables to be passed to the test script.
- ☑ `--STDIN--`
  Data to be fed to the test script's standard input.
- ☑ `--INI--`
  php.ini settings (use one line per setting e.g. foo=bar).
- ☑ `--ARGS--`
  A single line defining the arguments passed to PHP.
- ☑ `--ENV--`
  Configures the environment to be used for PHP.

Running the tests

- ☑ Execute the script run-tests.php
- ☑ Pass any number of directories or *.phpt files
- ☑ Without any option all tests in current dir are run

```
$> php run-tests.php

$> php run-tests.php tests sapi ext

$> php run-tests.php mytest.phpt
```

For help use:
php run-tests.php -h

Running the tests

- ☑ Execute the script run-tests.php
- ☑ Pass any number of directories or *.phpt files
- ☑ Without any option all tests in current dir are run
- ☑ You can create a list of failed tests for later use

```
$> php run-tests.php

$> php run-tests.php tests sapi ext

$> php run-tests.php –w myerr.lst mytest.phpt

$> php run-tests.php –l myerr.lst
```

For help use:
php run-tests.php -h

Running the tests

- ☑ Execute the script run-tests.php
- ☑ Pass any number of directories or *.phpt files
- ☑ Without any option all tests in current dir are run
- ☑ You can create a list of failed tests for later use

```
$> php run-tests.php

$> php run-tests.php tests sapi ext

$> php run-tests.php –w myerr.lst mytest.phpt

$> php run-tests.php –l myerr.lst
```

There is also –r, -a and –w to work with lists.

Running the tests

- ☑ Use –n to suppress INI usage
- ☑ Use –d <foo>=<bar> to specify INI entries
- ☑ Use –q to be quiet – do not ask questions
- ☑ Use –s to write result to a file
- ☑ Use –m to run tests through valgrind (very slow)

```
$> php run-tests.php -n

$> php run-tests.php –d zend.ze1_compatibility_mode=1

$> php run-tests.php –q

$> php run-tests.php –s mytest.res

$> php run-tests.php -m
```

Files and dirs should be right from options.

INI overwrites

- ☑ Some INI entries are hardcoded

output_handler=
open_basedir=
safe_mode=0
disable_functions=
output_buffering=Off
error_reporting=8191
display_errors=1
log_errors=0
html_errors=0

track_errors=1
report_memleaks=1
report_zend_debug=0
docref_root=
docref_ext=.html
error_prepend_string=
error_append_string=
auto_prepend_file=
auto_append_file=',

The environment

☑ TEST_PHP_EXECUTABLE	The test executable
☑ TEST_PHP_CGI_EXECUTABLE	When --GET-- is used
☑ TEST_PHP_USER	User directories
☑ TEST_PHP_ARGS	Arguments to use
☑ TEST_PHP_LOG_FORMAT	Output files to create

```
$> export TEST_PHP_EXECUTABLE=/path/to/my/php

$> export TEST_PHP_CGI_EXECUTABLE=/usr/bin/php-cgi

$> export TEST_PHP_USER=/my/test/file/dir

$> export TEST_PHP_ARGS="-n –q"

$> export TEST_PHP_LOG_FORMAT=""

$> make test
```

All environtment variables can used together.

Output files

- ☑ Use TEST_PHP_LOG_FORMAT to select output files
  - L Log file, all information in one file
  - E Expected output (--EXPECT--)
  - O Actual output
  - D Difference from expected and actual output
- ☑ Sometimes it helps to use diff command
  - ☑ diff –u test.exp test.out
- ☑ Use --keep-[all|php|skip|clean] to keep temp files

THANK YOU

- ☑ This Presentation
  http://somabo.de/talks/
- ☑ PHPT Documentation
  http://qa.php.net/write-test.php
- ☑ PHPUnit
  http://sebastian-bergmann.de/talks/2006-11-02-PHPUnit.pdf
- ☑ SimpleTest
  http://www.lastcraft.com/simple_test.php
- ☑ Power PHP Testing
  http://brainbulb.com/power-php-testing.pdf