

# Testování webových aplikací

Ing. Martin Dostal, Ph.D.

# Typy testování z pohledu životního cyklu sw

- Testování v průběhu vývoje
  - průběžné **výpisy** přes **echo**, **printr**, nebo zápis do souboru (**log**)
  - **ruční kontrola** aplikace
  - **ruční testy** - jednoduchý .php soubor, který zavolá nějaký objekt, předá vstup a kontroluje výstup. Mohou i nemusí používat frameworky.
  - **automatické testy** - spouští se průběžně a automaticky a pokud nějaký test skončí chybou, tak zařvou - mohou být pouze lokální
- Testování před uvolněním nové verze
  - **automatické testy** - využití např. kompletních nočních buildů, většinou na serveru a komunikují s git repozitářem.
  - vhodný je **“beta” server** pro simulaci ostrého provozu
- Testování v ostrém provozu
  - ruční kontrola - zejména klíčových funkcí aplikace
  - automatické testování celé webové aplikace - např. AB testy

# Testování při vývoji - vývoj a ruční kontrola

- základem jsou kontrolní výpisy echo, printr, kterými kontrolujeme zejména obsah proměnných. Např. je to opravdu pole? Jaké mám klíče?
- aplikaci můžeme kdykoliv ukončit příkazem exit; nebo die;
  - exit
    - destruktory objektů nebo shutdown funkce budou zavolané vždy
    - parametrem může být kód chyby (0-244) do logu nebo textový řetězec, který bude vypsán
  - die - ekvivalent k exitu, je lepší používat exit, ale fungují stejně. Časem může být deprecated.

# Testování při vývoji - ruční testy

```
class kalkulacka {  
  
    function secti($a, $b)  
    {  
        $c = $a + $b;  
        // echo $c; exit;  
        return $c;  
    }  
  
}
```

```
test_kalkulacky.php  
  
kalk = new kalkulacka();  
$a = 5;  
$b = 3;  
$c = 8;  
  
$vysledek = kalk->secti($a, $b);  
  
if ($vysledek == $c) echo "OK";  
else echo "ERROR";
```

# Testování při vývoji - automatické testy

- Testování - hledání chyb:
  - PHPUnit (<https://phpunit.de/>) - známý a používaný testovací framework
  - NetteTester - český nástroj ...
- Debugování - odstraňování chyb:
  - Tracy - český nástroj, česká dokumentace, malá komunita
  - FirePhp - plugin do Firefoxu
  - XDebug - velmi dobrá volba díky velké popularitě

# PHPUnit - vytvoření testu

```
use PHPUnit\Framework\TestCase;                                // použití namespace
class StackTest extends TestCase
{
    public function testPushAndPop(){
        $stack = [];
        $this->assertEquals(0, count($stack));                    // opravdu je zásobník prázdný?
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);    // mam tam foo?
        $this->assertEquals(1, count($stack));                    // je velikost zásobníku 1?
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEquals(0, count($stack));
    }
}
```

# PHPUnit - spuštění testů z příkazové řádky

phpunit ArrayTest

PHPUnit 5.7.0 by Sebastian Bergmann and contributors.

..

// tečka = spuštěný test, OK

Time: 0 seconds

OK (2 tests, 2 assertions)

- bude hledat soubor ArrayTest.php v aktuálním adresáři, načte ho a očekává class ArrayTest. Poté spustí všechny její metody.

# PHPUnit - symboly pro jednotlivé testy

.	// OK - test prošel
F	// chyba - test neprošel, porovnání přes assertEquals nesouhlasí
E	// chyba při spouštění testovací metody,
R	// test označen jako riskantní
S	// test přeskočen
I	// test je označen jako nekompletní nebo ignorovaný



# DbUnit - balíček pro testování databází

```
class MyGuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    // ... připojení, naplnění testovacími daty a mnoho dalšího

    public function testGetRowCount()
    {
        $this->assertEquals(2, $this->getConnection()->getRowCount('guestbook'));
    }
}
```

**Závěr:** poměrně komplikovaný nástroj. Doporučoval bych raději vlastní ruční test typu obnovení dat ze zálohy + testování modelu nad prázdnou databází. (2x zavolám Insert a očekávám 2 řádky v tabulce)

# Nette Tester - česká “kopie” PHPUnit

```
use Tester\Assert;
require ....;           // několik souborů
// Upraví chování PHP a zapne vlastnosti Testeru
Tester\Environment::setup();
```

```
// vytvoření naší třídy
$o = new Greeting;
```

```
// očekáváme shodu
Assert::same( 'Hello John', $o->say('John') );
```

```
class Greeting
{
    public function say($name)
    {
        if (!$name) {
            throw new InvalidArgumentException ('Jm');
        }
        return "Hello $name";
    }
}
```

Závěr: PHPUnit je lepší volbou - jedná se o standard integrovaný do mnoha různých nástrojů a IDE.

# Tracy - další český projekt

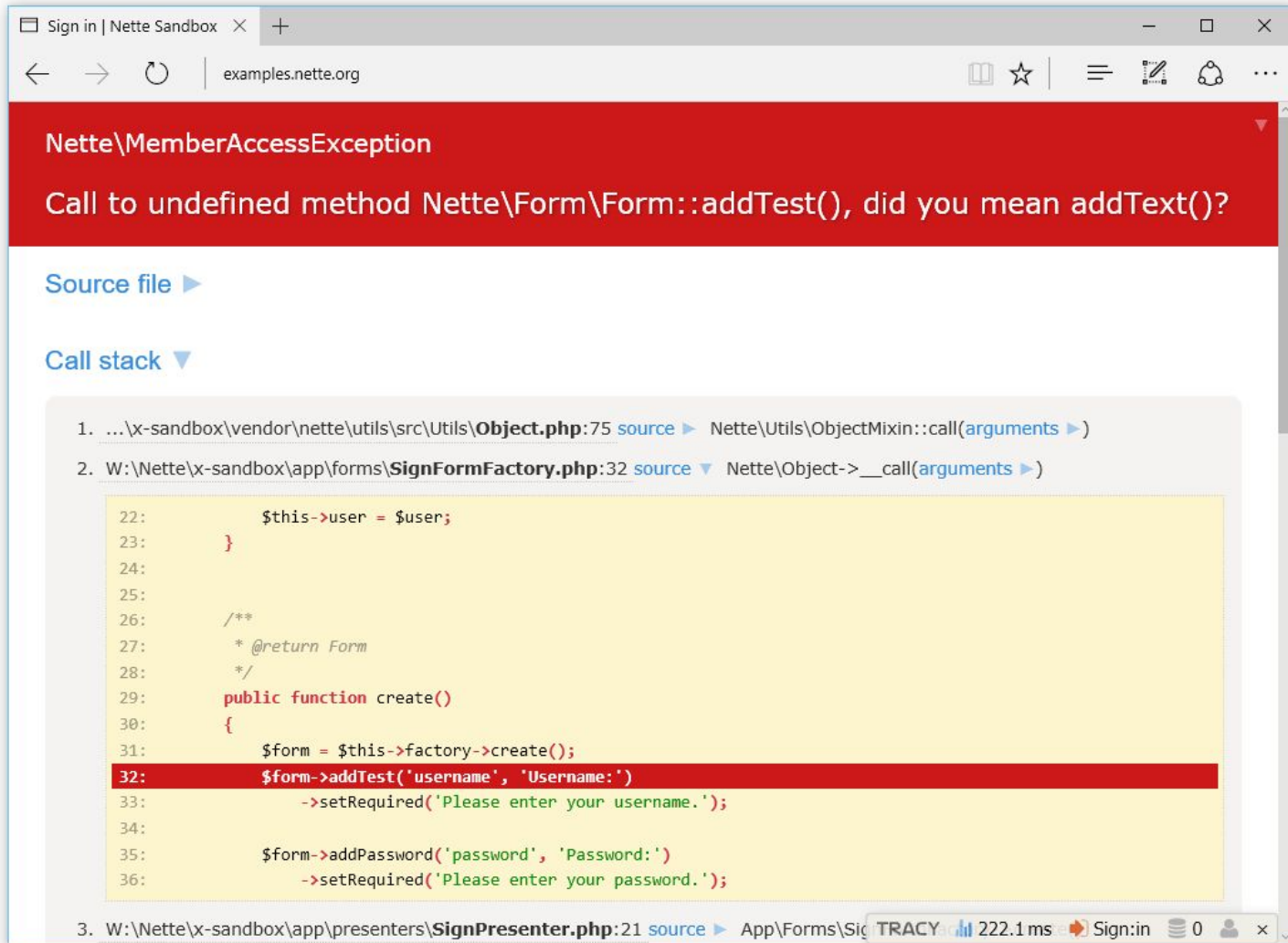
- funguje jen na **localhostu**
- v rámci produkce by se neměla zobrazit, pokud se nevynutí !!!
- na produkční server by se neměla raději ani kopírovat => velmi nebezpečné
- použití:

```
use Tracy\Debugger;  
Debugger::enable();
```

- debugger bar - zřejmě z jiného open source projektu:



# Tracy



The screenshot shows a web browser window with the address bar displaying "examples.nette.org". The main content area has a red header with the text "Nette\MemberAccessException" and "Call to undefined method Nette\Form\Form::addTest(), did you mean addText()?". Below the header, there are two links: "Source file" and "Call stack". The "Call stack" section is expanded, showing a list of three items. The second item is highlighted, showing the file "SignFormFactory.php" at line 32. The code snippet for this line is highlighted in red. The code shows a public function "create()" that creates a form object and calls "addTest()" on it. The error message indicates that the "addTest()" method is undefined.

Sign in | Nette Sandbox × +

examples.nette.org

Nette\MemberAccessException

Call to undefined method Nette\Form\Form::addTest(), did you mean addText()?

Source file ▶

Call stack ▼

1. ...x-sandbox\vendor\nette\utils\src\Utils\Object.php:75 source ▶ Nette\Utils\ObjectMixin::call(arguments ▶)
2. W:\Nette\x-sandbox\app\forms\SignFormFactory.php:32 source ▼ Nette\Object->\_\_call(arguments ▶)
3. W:\Nette\x-sandbox\app\presenters\SignPresenter.php:21 source ▶ App\Forms\SignFormFactory->create()

```
22:         $this->user = $user;
23:     }
24:
25:
26:     /**
27:      * @return Form
28:      */
29:     public function create()
30:     {
31:         $form = $this->factory->create();
32:         $form->addTest('username', 'Username:');
33:         $form->setRequired('Please enter your username.');
```

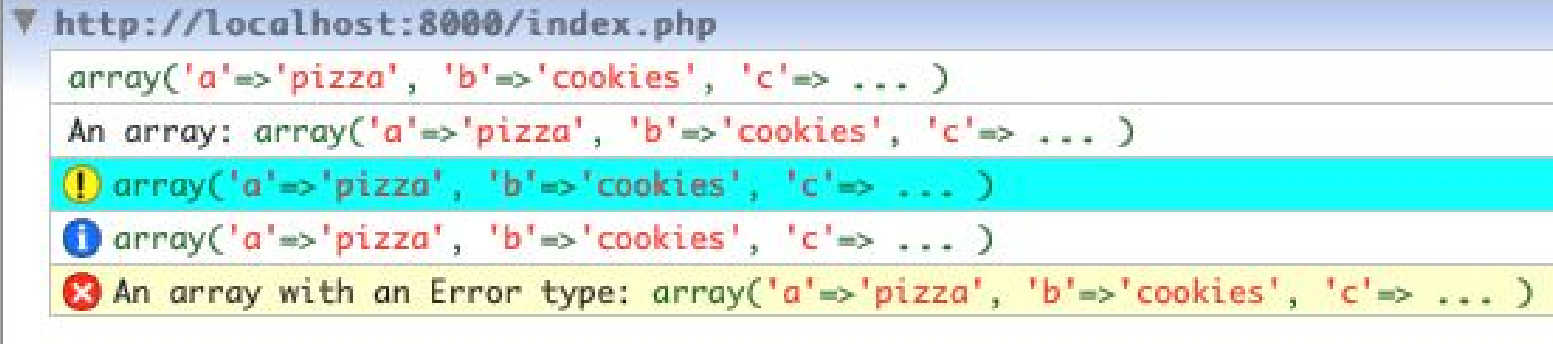
TRACY 222.1 ms Sign:in 0

# FirePhp + FireBug

1. instalace na server
2. include FirePhp: `require_once('FirePHPCore/fb.php');`
3. zapnout output buffering: `ob_start();`
  - nic se nepošle uživateli ihned, čeká se na zpracování ze strany FirePhp
  - FirePhp posílá logovací data v hlavičkách http protokolu

# FirePhp

```
$var = array('a'=>'pizza', 'b'=>'cookies', 'c'=>'celery');  
fb($var);  
fb($var, "An array");  
fb($var, FirePHP::WARN);  
fb($var, FirePHP::INFO);  
fb($var, 'An array with an Error type', FirePHP::ERROR);
```



The screenshot shows a web browser window with the address bar displaying `http://localhost:8000/index.php`. Below the address bar, the FirePHP output is visible in the console area. It consists of five lines of output, each preceded by a colored icon in a circle:

- A blue downward-pointing triangle icon followed by the text `array('a'=>'pizza', 'b'=>'cookies', 'c'=> ... )`.
- A blue circle icon with a white 'i' followed by the text `An array: array('a'=>'pizza', 'b'=>'cookies', 'c'=> ... )`.
- A yellow circle icon with a black exclamation mark followed by the text `array('a'=>'pizza', 'b'=>'cookies', 'c'=> ... )`. This line is highlighted with a yellow background.
- A blue circle icon with a white 'i' followed by the text `array('a'=>'pizza', 'b'=>'cookies', 'c'=> ... )`.
- A red circle icon with a white 'x' followed by the text `An array with an Error type: array('a'=>'pizza', 'b'=>'cookies', 'c'=> ... )`. This line is highlighted with a red background.

# XDebug (<https://xdebug.org/>) - doporučuji !!!

- funkce:
  - zobrazení stacku (zásobníku), přidělování paměti
  - ochrana proti nekonečným smyčkám
  - a mnoho dalšího
- používá DBGp
  - Common DeBugGer Protocol, pouze pro lokální testování
- rozšíření:
  - XDebug pro Atom: <https://atom.io/packages/php-debug>
  - XDebug pro PhpStorm - viz. seznam pluginů
  - rozšíření XDebug pro Chrome:  
<https://chrome.google.com/webstore/detail/xdebug-helper/eadndfjplgieldjbigjakmdgkmoaaaoc>

# Testování s využitím napojení na Git

- Travis CI (<https://travis-ci.org/>)
  - spouštění automatických testů s využitím např. PHPUnitu
  - zdarma pro opensource projekty a tedy i repozitáře
  - vhodné zejména pro GitHub díky skvělé integraci
- David (<https://david-dm.org/>) - kontrola závislostí v JS
- další odznáčky pro další služby

## Bootstrap





# Travis CI (<https://travis-ci.org/>) - použití

1. provedu fork jejich testovacího repozitáře travis-broken-example
2. přihlásím se do Traviše, povolím přístup k danému repozitáři
3. provedu nějakou změnu + commit a push, starší commity Travis netestuje!
4. Travis CI - vypíše chybu a pošle jí emailem

✗ **master** added record to NewUser.txt

✗ **#1 failed**

↻ Restart build

🐙 Commit 5139fbb

🕒 Elapsed time 38 sec

🐙 Compare c041083..5139fbb

🕒 Total time 1 min 20 sec

🐙 Branch master

📅 3 minutes ago

🔌 Martin Dostal authored 🔌 GitHub committed

# Travis CI - zobrazení na GitHubu - skvělé !!!

- do README.md přidáme řádek:  
[![Build Status](<https://travis-ci.org/madostal/travis-broken-example.svg?branch=master>)]  
(<https://travis-ci.org/madostal/travis-broken-example>)



# Testování **celé** webové aplikace

# Základní typy testování celé web. aplikace

- ruční - ruční otestování “všeho”
  - programátor
  - tester
  - testovací skupina uživatelů - viz. např. beta testéři nebo AB testy
- automatické - např. automatické proklikání webu robotem, kontrola odkazů ...
  - Selenium (<http://www.seleniumhq.org/>)
    - testy je možné psát v mnoha různých jazycích - C#, Php, Java ...

# Testování web. aplikace - ruční (1)

- **funkčnost aplikace** - kontrola odkazů, formulářů, připojení do databáze, validita html + css
- **použitelnost aplikace**
- **kompatibilitu** - různé prohlížeče, operační systémy, mobilní zařízení a tisk, pokud má smysl
- **výkon aplikace**
  - doba načtení webu na různých typech připojení,
  - zátěž na serveru a spotřeba **RAM**
    - např. Moodle 1 potřebuje 1 GB pro 50 paralelních uživatelů !!!,
    - např. Moodle 2 potřebuje 1 GB pro 20 paralelních uživatelů !!!!!!!
  - zátěžové testování - chování při překročení limitů aplikace a serveru, co se stane? Kdy nastane?

# Testování web. aplikace - ruční (2)

- **bezpečnost aplikace** - hádání URL, formuláře, SQL injection ...
  - tedy je to často o dobrém návrhu aplikace než přímo o testování
- **zálohování** - opravdu se zálohuje web i databáze?, jak dlouho?, lze to ze zálohy obnovit?

# Testování web. aplikace s využitím testerů (lidí)

- Na tuto pozici se využívají lidé technického i humanitního zaměření.
- Humanisté jsou levnější, ale některé technické věci nejsou schopní testovat.
- Proč?
  - Je to spolehlivější a levnější než vývoj univerzálního testu, který testuje úplně všechno.
  - UX (user experience) nelze automaticky testovat.
- Většinou se vyžaduje zaškolení - jak správně “klikat” a jak správně popsat chyby.
- Nemůže to dělat každý. Vyžaduje se zručnost a aspoň základní technické povědomí.
- Jedinou výjimkou je tester “blbec”, který se snaží simulovat chování nejhoršího možného uživatele, který na web přijde.

# Selenium (seleniumhq.org)

- celý test je možné naprogramovat v několika různých jazycích
- naklikat s využitím Selenium IDE - plugin do Firefoxu
  - prohlížeč pak celý web proklikává sám a vyplňuje požadované formuláře, uživateli stačí vše sledovat
- příklad testu:
  1. Otevři stránky [www.google.com](http://www.google.com)
  2. Napiš heslo: "Wikipedia" do elementu s id "gbqfq"
  3. Počkej až se zobrazí výsledky vyhledávání
  4. Klikni na odkaz s textem: "Wikipedie, otevřená encyklopedie" a počkej, než se stránka načte



# Selenium - zdrojový kód testu

```
<body> ....
```

```
<tbody>
```

```
<tr>
```

```
    <td>open</td><td>http://www.google.cz</td><td></td>
```

```
</tr>
```

```
<tr>
```

```
    <td>type</td><td>id=gbqfq</td><td>Wikipedia</td>
```

```
</tr>
```

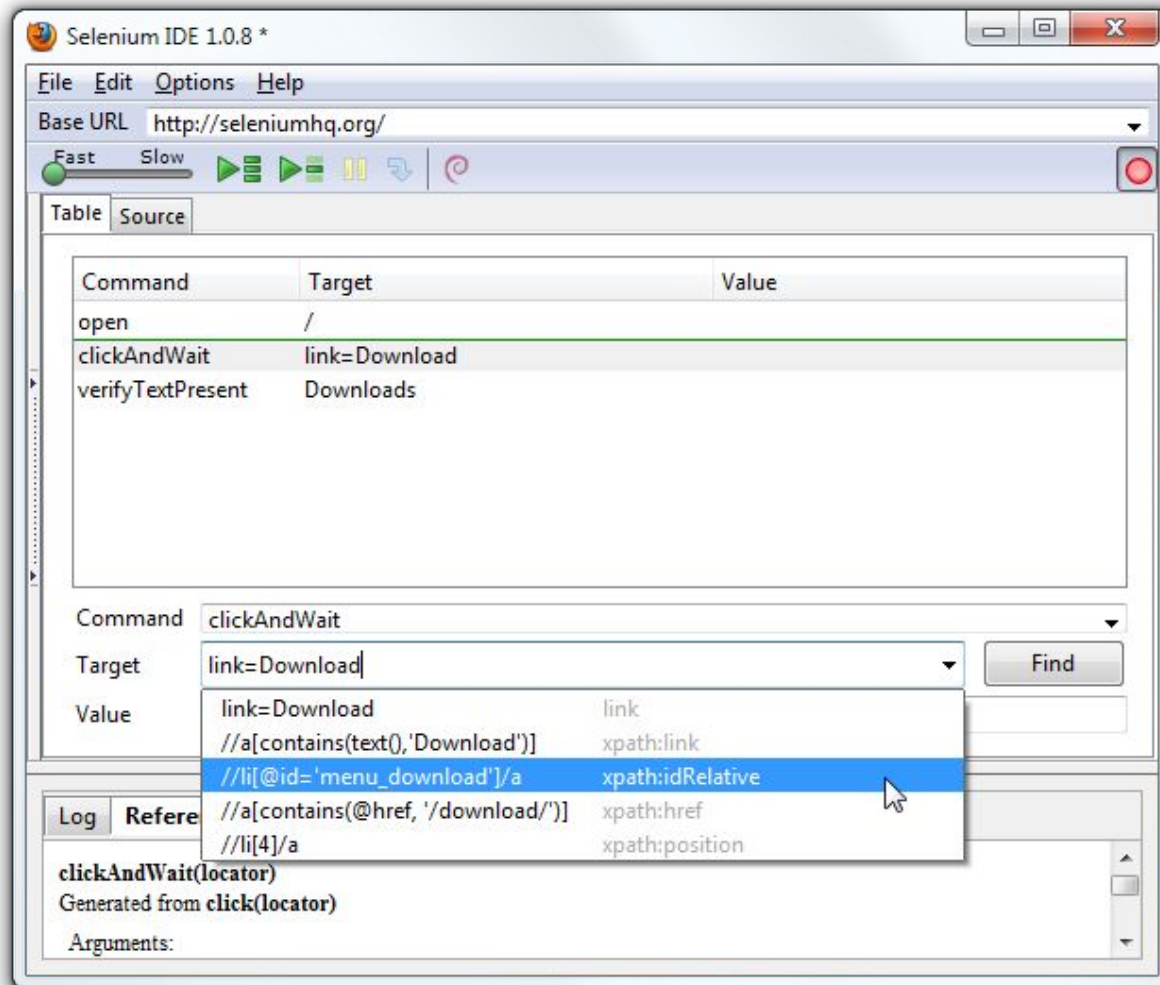
```
<tr>
```

```
    <td>clickAndWait</td><td>link=Wikipedie, otevřená encyklopedie</td>
```

```
<td></td>
```

```
</tr>
```

# Selenium IDE



# Závěr

**Testovat, testovat a ještě jednou testovat !**

Čím později chybu odhalíte, tím ...

... je její oprava dražší.

... přijdete o více zákazníků.

... více poškodíte svojí firmu.