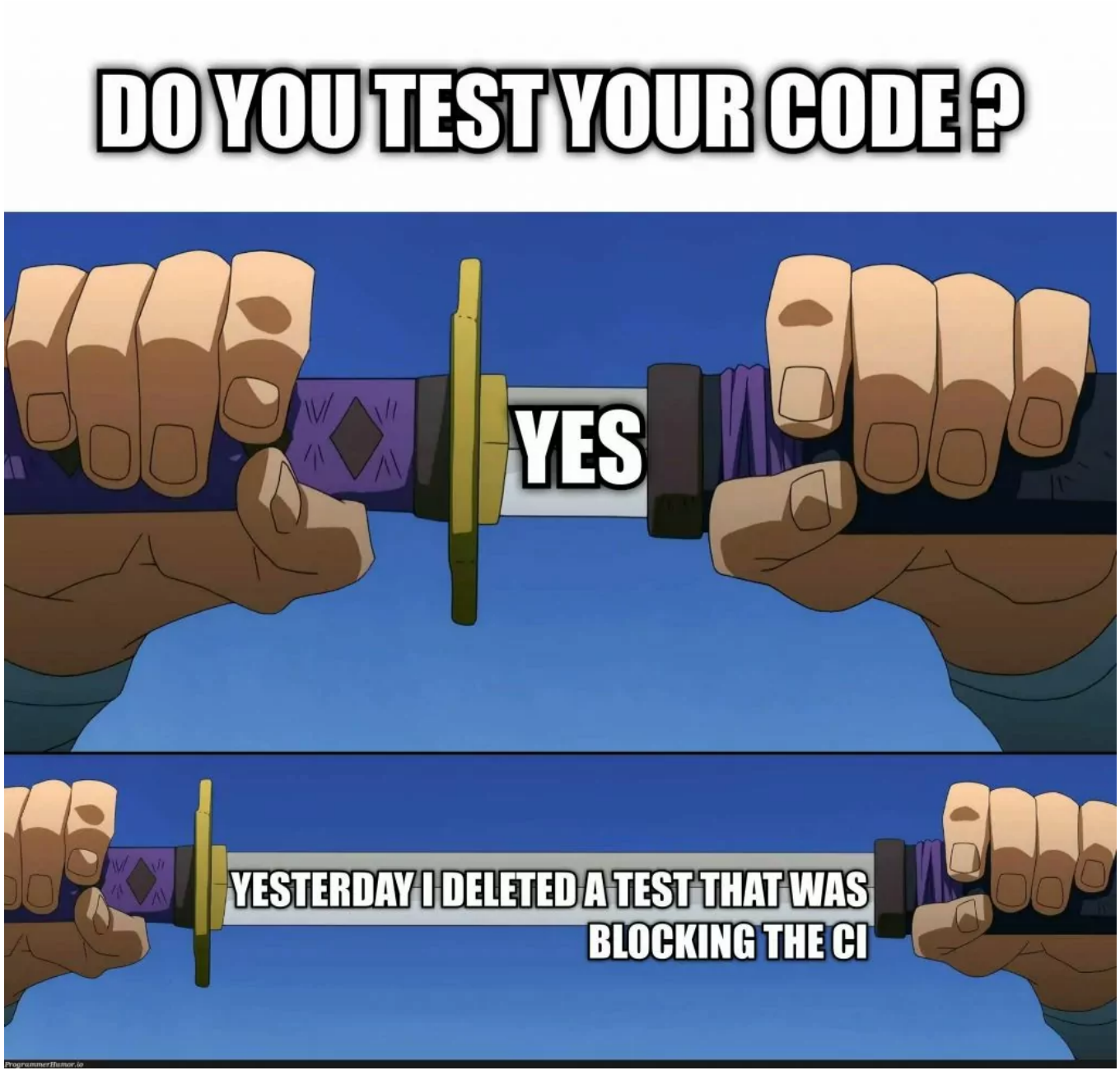
DELTA TopGun

(23) Testování softwaru

Luboš Zápotočný

2024

Obsah


DO YOU TEST YOUR CODE ?
YES
YESTERDAY I DELETED A TEST THAT WAS
BLOCKING THE CI

Co je to testování softwaru?

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

- Manuální testování

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

- Manuální testování
- Jednotkové testování (unit testing)

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

- Manuální testování
- Jednotkové testování (unit testing)
- Integrační testování (integration testing)

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

- Manuální testování
- Jednotkové testování (unit testing)
- Integrační testování (integration testing)
- System testování (system testing / end-to-end testing)

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

- Manuální testování
- Jednotkové testování (unit testing)
- Integrační testování (integration testing)
- System testování (system testing / end-to-end testing)
- Akceptační testování (acceptance testing)

Co je to testování softwaru?

Testování softwaru je **proces ověřování**, zda software funguje správně a splňuje požadavky.

- Zajišťuje kvalitu a spolehlivost softwaru.
- Pomáhá odhalit chyby a problémy.
- Zvyšuje důvěru uživatelů.

Typy testování

- Manuální testování
- Jednotkové testování (unit testing)
- Integrační testování (integration testing)
- System testování (system testing / end-to-end testing)
- Akceptační testování (acceptance testing)
- ...

Unit testy

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.
- Snadnější refaktorování kódu.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.
- Snadnější refaktorování kódu.
- Snadnější spolupráce v týmu.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.
- Snadnější refaktorování kódu.
- Snadnější spolupráce v týmu.
- Snadnější nasazení a udržování.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.
- Snadnější refaktorování kódu.
- Snadnější spolupráce v týmu.
- Snadnější nasazení a udržování.
- Snadnější dokumentace a pochopení kódu.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.
- Snadnější refaktorování kódu.
- Snadnější spolupráce v týmu.
- Snadnější nasazení a udržování.
- Snadnější dokumentace a pochopení kódu.
- Snadnější návrh a implementace nových funkcí.

Unit testy

Jednotkové testy (unit tests) jsou testy, které ověřují správnost jednotlivých částí kódu (funkcí, metod).

- Testují se tzv. **jednotky** kódu.
- Jednotkové testy by měly být **nezávislé** na sobě a na prostředí.
- V PHP se jednotkové testy obvykle píšou pomocí knihovny PHPUnit.

Výhody jednotkových testů

- Rychlejší odhalení chyb.
- Snadnější refaktorování kódu.
- Snadnější spolupráce v týmu.
- Snadnější nasazení a udržování.
- Snadnější dokumentace a pochopení kódu.
- Snadnější návrh a implementace nových funkcí.
- **Rychlé** spouštění a opakování testů.

```php
<?php
class Calculator
{
    public function add(int $a, int $b): int
    { return $a + $b; }

    public function subtract(int $a, int $b): int
    { return $a - $b; }

    public function multiply(int $a, int $b): int
    { return $a * $b; }

    public function divide(int $a, int $b): float
    { return $a / $b; }
}
```

Co by na této třídě mělo být otestováno?

Nejdříve základní operace sčítání a odčítání.

```php
<?php
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase
{
    public function testAdd(): void
    {
        $calculator = new Calculator();
        $result = $calculator->add(2, 3);
        $this->assertSame(5, $result);
    }

    public function testSubtract(): void
    {
        $calculator = new Calculator();
        $result = $calculator->subtract(5, 3);
        $this->assertSame(2, $result);
    }
}
```

Nyní násobení

```php
<?php
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase
{
    ...

    public function testMultiply(): void
    {
        $calculator = new Calculator();
        $result = $calculator->multiply(2, 3);
        $this->assertSame(6, $result);
    }
}
```

A nakonec dělení

```php
<?php
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase
{
    ...

    public function testDivide(): void
    {
        $calculator = new Calculator();
        $result = $calculator->divide(6, 3);
        $this->assertSame(2, $result);
    }
}
```

Co ale dělení nulou?

```php
<?php
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase
{
    ...

    public function testDivideByZero(): void
    {
        $calculator = new Calculator();
        $this->expectException(InvalidArgumentException::class);
        $calculator->divide(6, 0);
    }
}
```

Test spadnul

There was 1 failure:
1) CalculatorTest::testDivideByZero Failed asserting that exception of type "DivisionByZeroError" matches expected exception "InvalidArgumentException".

Oprava dělení nulou

```php
<?php
class Calculator
{
    ...

    public function divide(int $a, int $b): float
    {
        if ($b === 0) {
            throw new InvalidArgumentException('Division by
            zero');
        }

        return $a / $b;
    }
}
```

Testy pro dělení nulou

..... 5 / 5 (100%)
Time: 00:00.008, Memory: 8.00 MB
OK (5 tests, 5 assertions)

Unit testy

- Jednotkové testy se píší v samostatných třídách.
- Každá testovací metoda by měla testovat jednu vlastnost nebo chování.
- Testovací metody by měly být **nezávislé** na sobě.
- Testovací metody by měly být **nezávislé** na prostředí.
- Testovací metody by měly být **nezávislé** na pořadí.
- Testovací metody by měly být **nezávislé** na stavu.
- Testovací metody by měly být **nezávislé** na čase.
- Testovací metody by měly být **nezávislé** na konfiguraci.
- Testovací metody by měly být **nezávislé** na databázi.
- Testovací metody by měly být **nezávislé** na souborech.

Co **mohou** jednotkové testy ověřit?

- Správnost výpočtů a operací.

Co **mohou** jednotkové testy ověřit?

- Správnost výpočtů a operací.
- Správnost návratových hodnot.

Co **mohou** jednotkové testy ověřit?

- Správnost výpočtů a operací.
- Správnost návratových hodnot.
- Správnost chování v různých situacích.

Co **mohou** jednotkové testy ověřit?

- Správnost výpočtů a operací.
- Správnost návratových hodnot.
- Správnost chování v různých situacích.
- Správnost zacházení s chybami a výjimkami.

Co **mohou** jednotkové testy ověřit?

- Správnost výpočtů a operací.
- Správnost návratových hodnot.
- Správnost chování v různých situacích.
- Správnost zacházení s chybami a výjimkami.
- Správnost interakce s jinými třídami a objekty.

Co jednotkové testy **nemohou** ověřit?

- Správnost uživatelského rozhraní.

Co jednotkové testy **nemohou** ověřit?

- Správnost uživatelského rozhraní.
- Správnost výkonu a škálovatelnosti.

Co jednotkové testy **nemohou** ověřit?

- Správnost uživatelského rozhraní.
- Správnost výkonu a škálovatelnosti.
- Správnost integrace s jinými systémy.

Co jednotkové testy **nemohou** ověřit?

- Správnost uživatelského rozhraní.
- Správnost výkonu a škálovatelnosti.
- Správnost integrace s jinými systémy.
- Správnost chování v reálném prostředí.

Co jednotkové testy **nemohou** ověřit?

- Správnost uživatelského rozhraní.
- Správnost výkonu a škálovatelnosti.
- Správnost integrace s jinými systémy.
- Správnost chování v reálném prostředí.
- Správnost chování v různých prohlížečích a zařízeních.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.
- Testujte **všechny možné vstupy** a výstupy.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.
- Testujte **všechny možné vstupy** a výstupy.
- Testujte **všechny možné cesty** kódu.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.
- Testujte **všechny možné vstupy** a výstupy.
- Testujte **všechny možné cesty** kódu.
- Testujte **hranice** a **okrajové případy**.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.
- Testujte **všechny možné vstupy** a výstupy.
- Testujte **všechny možné cesty** kódu.
- Testujte **hranice** a **okrajové případy**.
- Testujte **správnost chování** v různých situacích.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.
- Testujte **všechny možné vstupy** a výstupy.
- Testujte **všechny možné cesty** kódu.
- Testujte **hranice** a **okrajové případy**.
- Testujte **správnost chování** v různých situacích.
- Testujte **správnost chování** v různých prostředích.

Jak psát dobré jednotkové testy?

- Testujte **pouze veřejné rozhraní** tříd a metod.
- Testujte **všechny možné vstupy** a výstupy.
- Testujte **všechny možné cesty** kódu.
- Testujte **hranice** a **okrajové případy**.
- Testujte **správnost chování** v různých situacích.
- Testujte **správnost chování** v různých prostředích.
- Testujte **správnost chování** v různých konfiguracích.

```php
<?php

interface Mailer {
    public function send(
        string $email,
        string $message
    ): bool;
}
```

```php
<?php

class User {
    public function __construct(
        private readonly Mailer $mailer
    ) {}

    public function register(string $email): bool {
        // Registration logic here...

        // Send a confirmation email
        return $this->mailer->send($email, 'Welcome to our
        website');
    }
}
```

Mockování a falešné objekty

Jak psát jednotkové testy pro třídu `X`, která ale využívá jinou (nebo i více jiných) tříd `Y`?

Mockování a falešné objekty

Jak psát jednotkové testy pro třídu `X`, která ale využívá jinou (nebo i více jiných) tříd `Y`?

- **Mockování** je technika, která umožňuje vytvořit **falešné objekty** (mocks) pro testování.

Mockování a falešné objekty

Jak psát jednotkové testy pro třídu `X`, která ale využívá jinou (nebo i více jiných) tříd `Y`?

- **Mockování** je technika, která umožňuje vytvořit **falešné objekty** (mocks) pro testování.
- Mockování umožňuje **simulovat** chování a vlastnosti reálných objektů.

Mockování a falešné objekty

Jak psát jednotkové testy pro třídu `X`, která ale využívá jinou (nebo i více jiných) tříd `Y`?

- **Mockování** je technika, která umožňuje vytvořit **falešné objekty** (mocks) pro testování.
- Mockování umožňuje **simulovat** chování a vlastnosti reálných objektů.
- Mockování umožňuje **ovládat** chování a výstupy falešných objektů.

Mockování a falešné objekty

Jak psát jednotkové testy pro třídu `X`, která ale využívá jinou (nebo i více jiných) tříd `Y`?

- **Mockování** je technika, která umožňuje vytvořit **falešné objekty** (mocks) pro testování.
- Mockování umožňuje **simulovat** chování a vlastnosti reálných objektů.
- Mockování umožňuje **ovládat** chování a výstupy falešných objektů.
- Mockování umožňuje **testovat** třídy a metody **nezávisle** na ostatních třídách a objektech.

```php
<?php
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase {
    public function testRegister(): void {
        // Create a mock for the Mailer interface.
        $mailer = $this->createMock(Mailer::class);
        // Set up the expectation for the send() method
        // to be called only once and with the email
        'user@example.com' and
        // the message 'Welcome to our website'.
        $mailer->expects($this->once())
               ->method('send')
               ->with(
                   $this->equalTo('user@example.com'),
                   $this->equalTo('Welcome to our website')
               )
               ->willReturn(true);
        ...
    }
}
```

```php
<?php
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase {
    public function testRegister(): void {
        ...
        // Create a User object with the mocked Mailer.
        $user = new User($mailer);
        // Call the register method with 'user@example.com'.
        $result = $user->register('user@example.com');
        // Assert that the result is true (email sent
        successfully).
        $this->assertTrue($result);
    }
}
```

Souvislost s DI (Dependency Injection)

Pokud je aplikace psaná s ohledem na DI, je mockování mnohem jednodušší.

Souvislost s DI (Dependency Injection)

Pokud je aplikace psaná s ohledem na DI, je mockování mnohem jednodušší.
Mockovat volání `new` je totiž mnohem složitější než vytvořit mock a předat ho jako parametr konstruktoru.

Integrační testování

Integrační testování

Integrační testování (integration testing) je testování, které ověřuje správnost interakce mezi jednotlivými částmi systému.

Integrační testování

Integrační testování (integration testing) je testování, které ověřuje správnost interakce mezi jednotlivými částmi systému.
Hlavní rozdíl oproti jednotkovým testům je v tom, že integrační testy testují **více tříd a objektů najednou**.

Integrační testování

Integrační testování (integration testing) je testování, které ověřuje správnost interakce mezi jednotlivými částmi systému.
Hlavní rozdíl oproti jednotkovým testům je v tom, že integrační testy testují **více tříd a objektů najednou**.
Často také mají **větší rozsah** a **delší dobu běhu** než jednotkové testy.

Integrační testování

Integrační testování (integration testing) je testování, které ověřuje správnost interakce mezi jednotlivými částmi systému.
Hlavní rozdíl oproti jednotkovým testům je v tom, že integrační testy testují **více tříd a objektů najednou**.
Často také mají **větší rozsah** a **delší dobu běhu** než jednotkové testy.
Mohou také vyžadovat **reálná data** a **reálné prostředí**, tedy například přístup k databázi nebo k síti.

Rozšíření jednotkových testů do integračních

V našem příkladu bychom mohli vytvořit integrační test, který ověří, že `User` pošle e-mail.

Rozšíření jednotkových testů do integračních

V našem příkladu bychom mohli vytvořit integrační test, který ověří, že `User` pošle e-mail.
Můžeme využít testování SMTP bránu jako například `Mailhog` a ověřit, že e-mail dorazil.

Rozšíření jednotkových testů do integračních

V našem příkladu bychom mohli vytvořit integrační test, který ověří, že `User` pošle e-mail.
Můžeme využít testování SMTP bránu jako například `Mailhog` a ověřit, že e-mail dorazil.
Můžeme tedy rozšířit náš test o zavolání metody `mail()` s připojením na `Mailhog`. Následně pomocí `Mailhog API` ověřit, že e-mail dorazil.

```php
<?php

use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

final class MailhogMailer implements Mailer
{
    public function __construct(
        private readonly string $host,
        private readonly int    $port
    )
    {
    }

    public function send(
        string $email,
        string $message
    ): bool
    {
    ...
    }
}
```

```php
<?php
...
    public function send(
        string $email,
        string $message
    ): bool
    {
        $mail = new PHPMailer(true);

        try {
            // Server settings
            $mail->isSMTP();
            // Send using SMTP
            $mail->Host = $this->host;
            // Set the SMTP server to send through
            $mail->SMTPAuth = false;
            // Disable SMTP authentication
            $mail->Port = $this->port;
            // Set the SMTP port to connect to - 1025 for
            mailhog
...
```

```php
<?php
...
            // Recipients
            $mail->setFrom('from@example.com', 'Mailer');
            $mail->addAddress($email);
            // Add a recipient
            $mail->addReplyTo('info@example.com', 'Information');

            // Content
            $mail->isHTML();
            // Set email format to HTML
            $mail->Subject = 'Here is the subject';
            $mail->Body = 'This is the HTML message body <b>in
            bold!</b>';
            $mail->AltBody = 'This is the body in plain text for
            non-HTML mail clients';

            $mail->send();

            return true;
        } catch (Exception $e) {
            return false;
        }
...
```

Naše třída tedy nemusí nadále používat mockovaný `Mailer`, ale může použít skutečný `MailhogMailer`.

Naše třída tedy nemusí nadále používat mockovaný `Mailer`, ale může použít skutečný `MailhogMailer`.
Tento (integrační) testy tedy bude kontrolovat dvě třídy najednou a zároveň ověří, že se e-mail doopravdy dokázal odeslat.

Naše třída tedy nemusí nadále používat mockovaný `Mailer`, ale může použít skutečný `MailhogMailer`.
Tento (integrační) testy tedy bude kontrolovat dvě třídy najednou a zároveň ověří, že se e-mail doopravdy dokázal odeslat.
Pozor! Tento test vyžaduje běžící `Mailhog` server.
Lze ho spustit například pomocí

```
docker run -d -p 8025:8025 -p 1025:1025
mailhog/mailhog
```

```php
<?php
use PHPUnit\Framework\TestCase;

final class UserIntegrationTest extends TestCase {
    public function testRegister(): void {
        $mailer = new MailhogMailer('127.0.0.1', 1025);
        $user = new User($mailer);
        // Check, if the mailhog is empty
        $messages =
        $this->check_mailhog_for_email('user@example.com');
        $this->assertEmpty($messages);
        // Call the register method with 'user@example.com'.
        $result = $user->register('user@example.com');
        $this->assertTrue($result);
        // Check if the email was sent
        $messages =
        $this->check_mailhog_for_email('user@example.com');
        $this->assertNotEmpty($messages);
        // Remove the email from MailHog
        $this->remove_email_from_mailhog('user@example.com');
    }
...
}
```

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.
- Musíme zajistit, že `Mailhog` na v průběhu testu doopravdy dorazil jeden email.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.
- Musíme zajistit, že `Mailhog` na v průběhu testu doopravdy dorazil jeden email.
- Musíme zajistit, že `Mailhog` na konci testu neobsahuje žádné emaily od daného uživatele, aby byl test opakovatelný.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.
- Musíme zajistit, že `Mailhog` na v průběhu testu doopravdy dorazil jeden email.
- Musíme zajistit, že `Mailhog` na konci testu neobsahuje žádné emaily od daného uživatele, aby byl test opakovatelný.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.
- Musíme zajistit, že `Mailhog` na v průběhu testu doopravdy dorazil jeden email.
- Musíme zajistit, že `Mailhog` na konci testu neobsahuje žádné emaily od daného uživatele, aby byl test opakovatelný.

Celkově jsme tedy provedli mnoho API volání a složitých operací, abychom ověřili, že se e-mail doopravdy odeslal.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.
- Musíme zajistit, že `Mailhog` na v průběhu testu doopravdy dorazil jeden email.
- Musíme zajistit, že `Mailhog` na konci testu neobsahuje žádné emaily od daného uživatele, aby byl test opakovatelný.

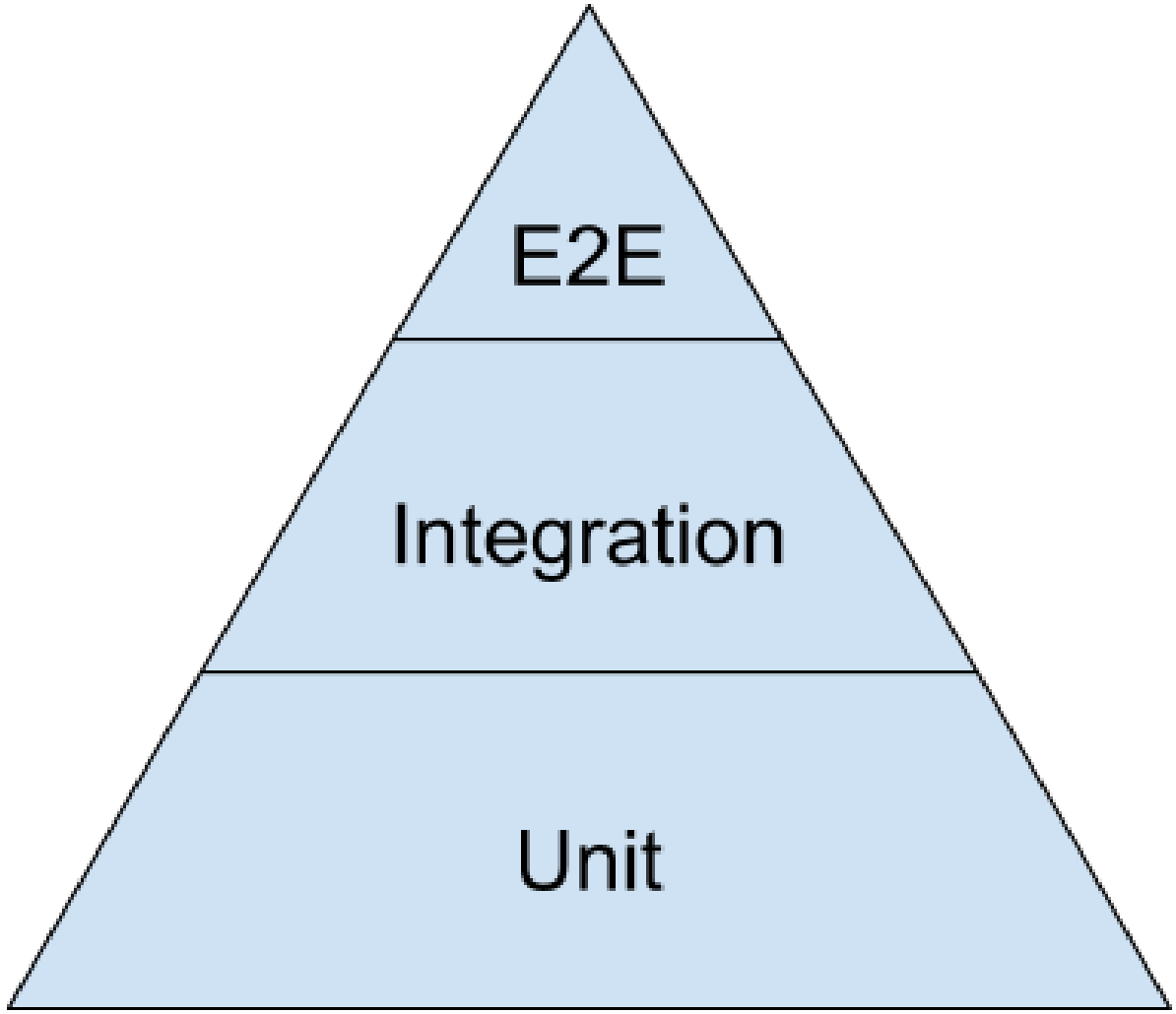
Celkově jsme tedy provedli mnoho API volání a složitých operací, abychom ověřili, že se e-mail doopravdy odeslal.
Tato volání a „příprava a úklid“ testu mohou být velmi časově náročné a složité.

Jak je vidět, test už musí obsahovat mnohem více kódu a je mnohem složitější.

- Musíme zajistit, že `Mailhog` běží.
- Musíme zajistit, že `Mailhog` na začátku testu neobsahuje žádné emaily od daného uživatele.
- Musíme zajistit, že `Mailhog` na v průběhu testu doopravdy dorazil jeden email.
- Musíme zajistit, že `Mailhog` na konci testu neobsahuje žádné emaily od daného uživatele, aby byl test opakovatelný.

Celkově jsme tedy provedli mnoho API volání a složitých operací, abychom ověřili, že se e-mail doopravdy odeslal.
Tato volání a „příprava a úklid“ testu mohou být velmi časově náročné a složité.
Proto je důležité správně zvolit, které testy budou jednotkové a které integrační.

Testovací pyramida



Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Větší softwarové projekty by měly mít větší podíl jednotkových testů a menší podíl systémových testů. Například 75% jednotkových testů, 20% integračních testů a 5% systémových testů.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Větší softwarové projekty by měly mít větší podíl jednotkových testů a menší podíl systémových testů. Například 75% jednotkových testů, 20% integračních testů a 5% systémových testů.

- Jednotkové testy jsou rychlejší, levnější a snadnější.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Větší softwarové projekty by měly mít větší podíl jednotkových testů a menší podíl systémových testů. Například 75% jednotkových testů, 20% integračních testů a 5% systémových testů.

- Jednotkové testy jsou rychlejší, levnější a snadnější.
- Jednotkové testy odhalí většinu chyb.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Větší softwarové projekty by měly mít větší podíl jednotkových testů a menší podíl systémových testů. Například 75% jednotkových testů, 20% integračních testů a 5% systémových testů.

- Jednotkové testy jsou rychlejší, levnější a snadnější.
- Jednotkové testy odhalí většinu chyb.
- Jednotkové testy umožňují rychlejší vývoj a nasazení.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Větší softwarové projekty by měly mít větší podíl jednotkových testů a menší podíl systémových testů. Například 75% jednotkových testů, 20% integračních testů a 5% systémových testů.

- Jednotkové testy jsou rychlejší, levnější a snadnější.
- Jednotkové testy odhalí většinu chyb.
- Jednotkové testy umožňují rychlejší vývoj a nasazení.
- Jednotkové testy umožňují snadnější refaktorování a údržbu.

Testovací pyramida

- **Jednotkové testy** (unit tests) tvoří základ testovací pyramidy.
- **Integrační testy** (integration tests) tvoří střed testovací pyramidy.
- **Systémové testy** (system tests) tvoří vrchol testovací pyramidy.

Větší softwarové projekty by měly mít větší podíl jednotkových testů a menší podíl systémových testů. Například 75% jednotkových testů, 20% integračních testů a 5% systémových testů.

- Jednotkové testy jsou rychlejší, levnější a snadnější.
- Jednotkové testy odhalí většinu chyb.
- Jednotkové testy umožňují rychlejší vývoj a nasazení.
- Jednotkové testy umožňují snadnější refaktorování a údržbu.
- Jednotkové testy umožňují snadnější spolupráci a **dokumentaci**.