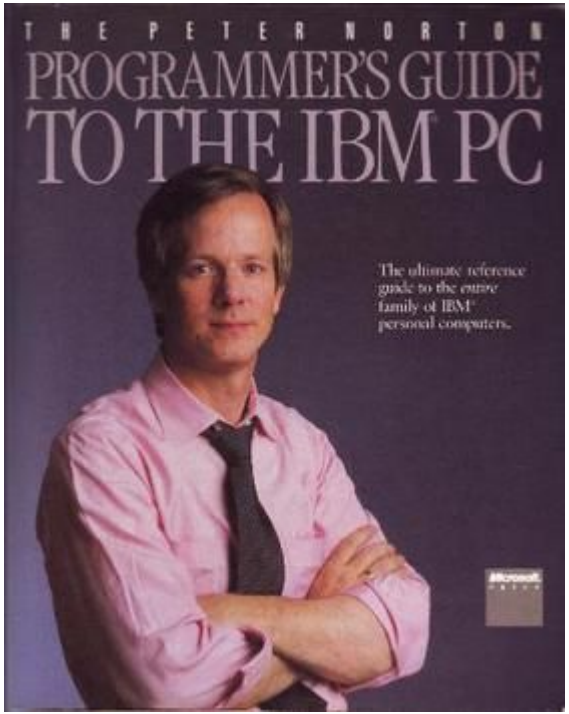
Sviluppo di applicazioni professionali in PHP 8
(parte II)

Dott. Enrico Zimuel

Seminario del 13 Maggio 2022, Eventi CLEII - CLEBA
Università degli Studi "G.D'Annunzio" di Chieti-Pescara

Sommario

- Il programmatore professionista
- Gli strumenti:
  - Framework di sviluppo
  - Composer
  - PHPUnit
  - Git
  - PHPStan
- Le metodologie:
  - KISS / SOLID
  - Dependency injection
  - Design pattern / MVC



Programmatore

- Il programmatore (o sviluppatore) è chi produce applicazioni o sistemi software
- “Uno sviluppatore software è un programmatore che si prende cura di uno o più aspetti del ciclo di vita del software, che è un qualcosa di più ampio della vera programmazione in sé” Wikipedia

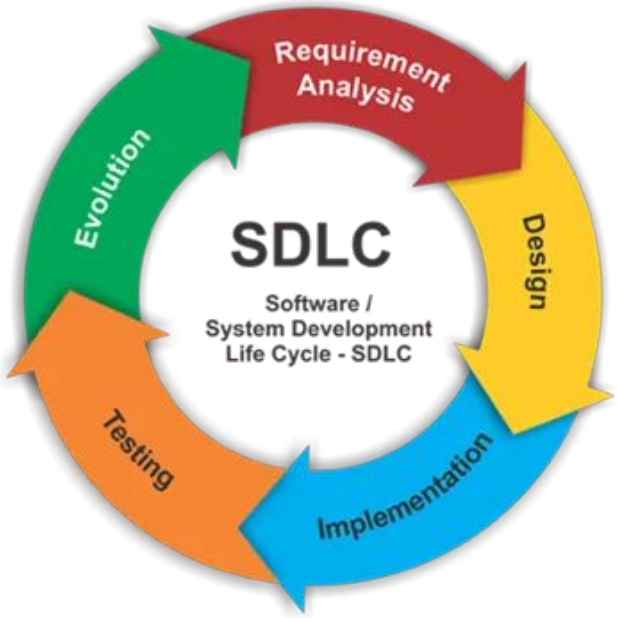
Software

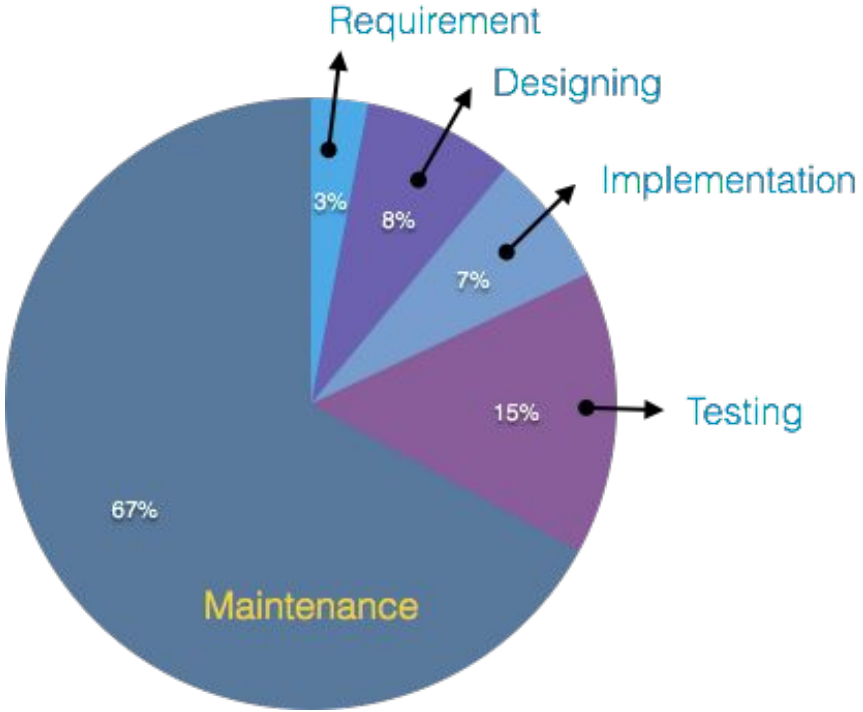
“Il software è come l'entropia. É difficile da afferrare, non pesa nulla e obbedisce alla seconda legge della termodinamica: aumenta sempre.” *Norman R. Augustine*

Il software è un sistema complesso

- Il numero di possibili interazioni/flussi all'interno di un software cresce in maniera esponenziale
- Il numero variabile di utenti contemporanei di un'applicazione (es. applicazione web)
- La gestione di un team di sviluppo, con la crescita esponenziale delle comunicazioni

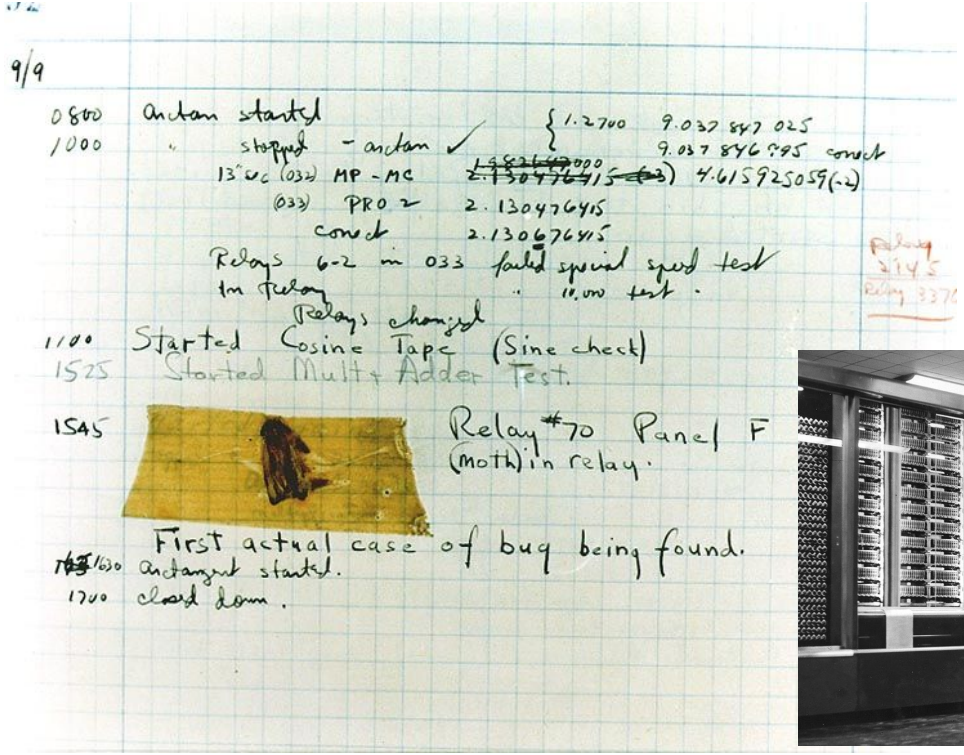
Il ciclo di vita e il costo





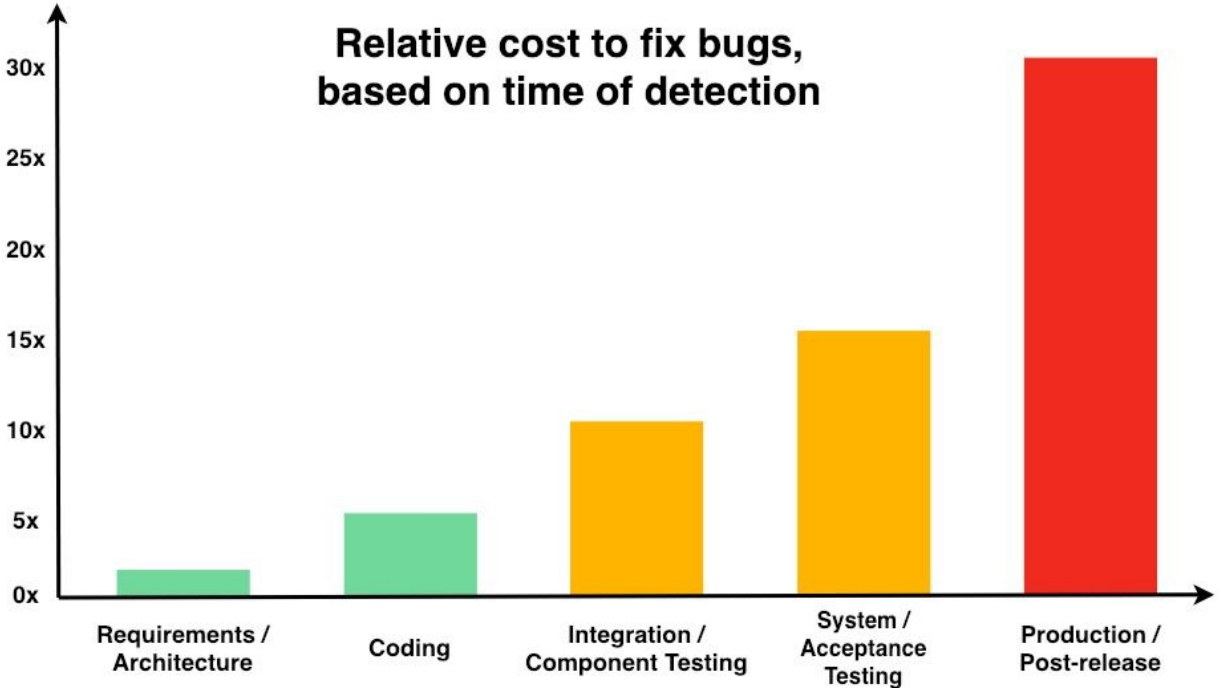
Fonte: Jussi Koskinen, Software Maintenance Costs, University of Eastern Finland, 2015

Bug



Primo “bug” della storia, Harvard Mark II, 9 Settembre 1947

Costo di un bug



Fonte: Sanket, The exponential cost of fixing bugs, Deepsource, 2019

Stipendio di un programmatore in Italia

Senior

DA
42.000€

A
62.000€

Mid

DA
28.000€

A
46.000€

Junior

DA
24.000€

A
30.000€

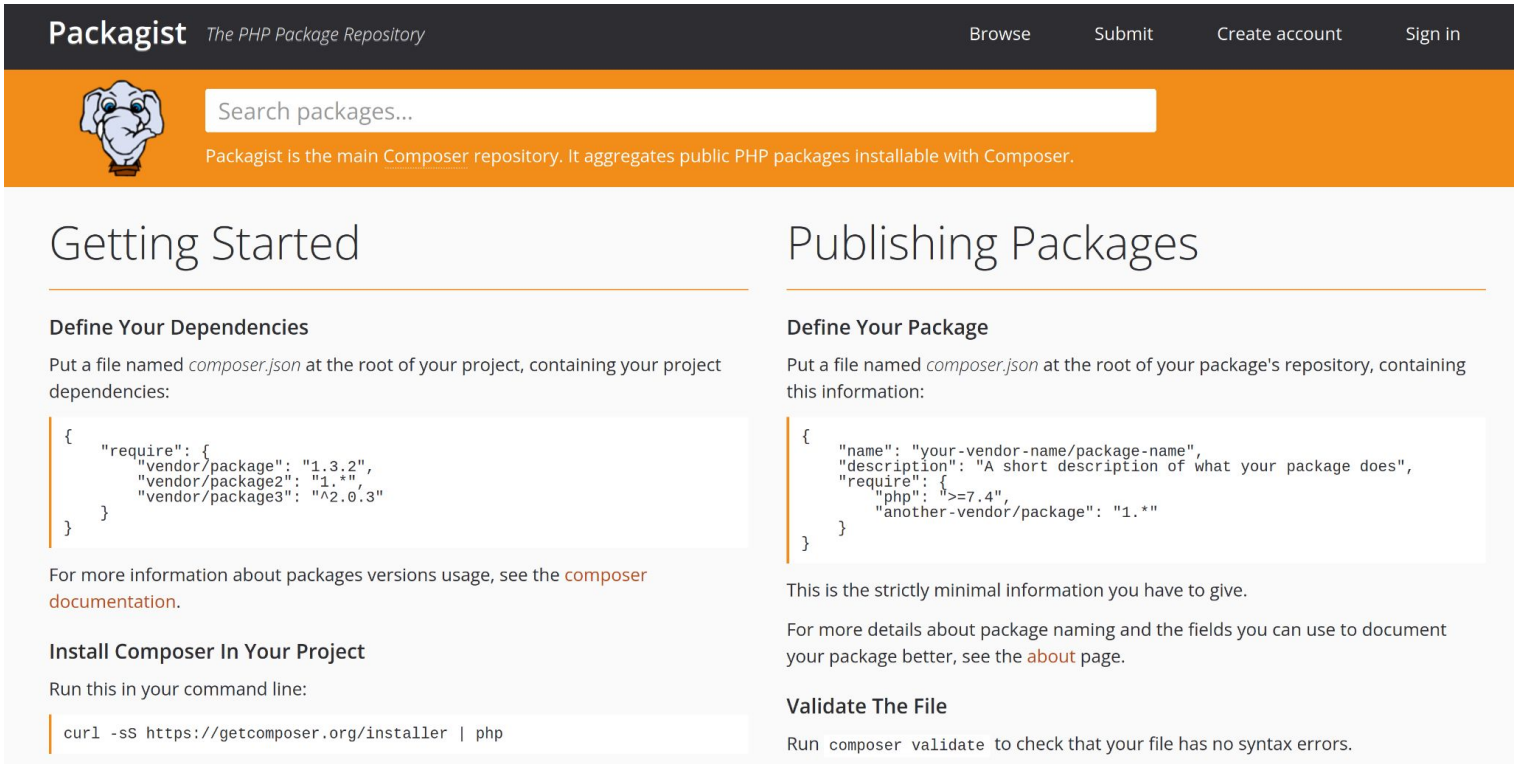
Fonte: https://crebs.it/stipendi/backend-developer/

Strumenti (PHP)

Composer

- Utility a linea di comando per l'installazione di librerie di terze parti
- Gestione dell'autoloading delle classi
- E' uno standard de facto in PHP
- Sito ufficiale: getcomposer.org

Packagist


Packagist
The PHP Package Repository
Browse
Submit
Create account
Sign in
Search packages...
Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer.
Getting Started
Define Your Dependencies
Put a file named composer.json at the root of your project, containing your project dependencies:
{
"require": {
"vendor/package": "1.3.2",
"vendor/package2": "1.*",
"vendor/package3": "^2.0.3"
}
}
For more information about packages versions usage, see the composer documentation.
Install Composer In Your Project
Run this in your command line:
curl -sS https://getcomposer.org/installer | php
Publishing Packages
Define Your Package
Put a file named composer.json at the root of your package's repository, containing this information:
{
"name": "your-vendor-name/package-name",
"description": "A short description of what your package does",
"require": {
"php": ">=7.4",
"another-vendor/package": "1.*"
}
}
This is the strictly minimal information you have to give.
For more details about package naming and the fields you can use to document your package better, see the about page.
Validate The File
Run composer validate to check that your file has no syntax errors.

Come installare una libreria

- Esempio, installazione della libreria Monolog per la gestione del file di log (standard PSR-3):
  - **composer require monolog/monolog**
- Composer installa l'ultima versione stabile della libreria monolog e crea (o aggiorna) il file **composer.json** con queste informazioni:

```
{
    "require": {
        "monolog/monolog": "^2.6"
    }
}
```

Come utilizzare composer da PHP

- E’ necessario configurare il sistema di autoloading delle classi nella sezione “autoload” del file composer.json:

```
"autoload": {
   "psr-4": {
     "MioNamescape\\": "src/"
   }
 },
```

- Includere il file **vendor/autoload.php** per caricare composer nel proprio progetto

Testing del codice

- Ci sono fondamentalmente due modi per testare un software:
  - Test manuale
  - Test automatico
- Test manuale:
  - Vantaggi:
    - Di facile implementazione
  - Svantaggi:
    - Lento
    - Costoso
    - Soggetto ad errori
    - Noioso!

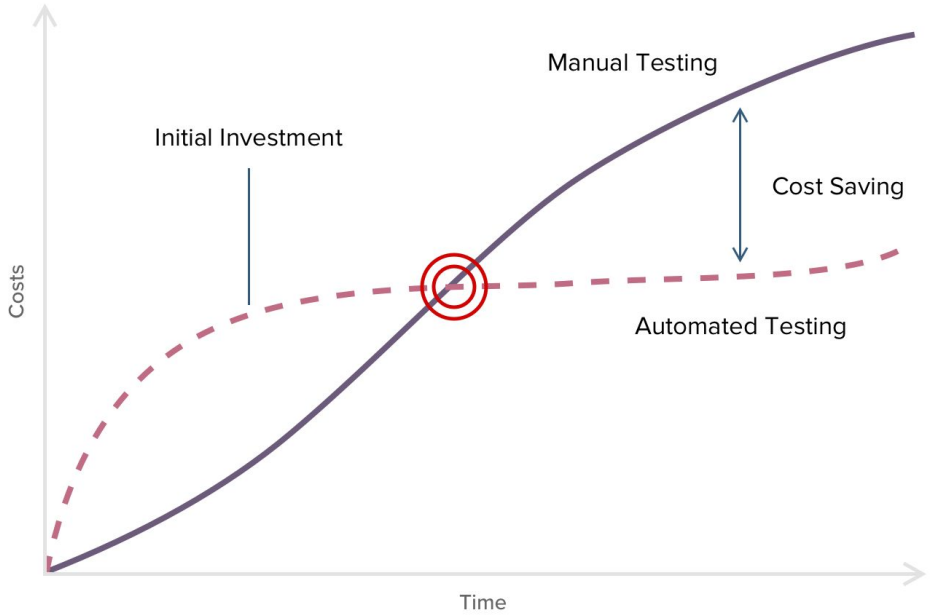
Test automatico

- Test del software tramite scrittura di un programma che utilizza il software da testare
- I test automatici sono una collezione di **asserzioni**
- Esempio: testare una funzione che implementi la somma di due numeri interi: **sum(X,Y)**
- Un’asserzione è una dichiarazioni del tipo se **X = 5** e **Y = 2** allora mi aspetto che il risultato **sum(X,Y) = 7**

Test automatico (2)

- Vantaggi:
  - Veloce
  - Economico (a lungo termine)
  - Completo (copertura del codice)
  - Aumenta la confidenza dello sviluppatore
  - Migliora la fase di analisi
- Svantaggi:
  - Costoso (nelle fasi iniziali)
  - Curva di apprendimento (una tantum)

ROI dei test automatici



Fonte: The True ROI of Test Automation

PHPUnit

- PHPUnit è un framework per lo sviluppo di test unitari (o di integrazione)
- E’ un progetto open source ideato da Sebastian Bergman che fa parte della famiglia di xUnit, ispirato da JUnit
- Standard de facto in PHP

PHPUnit: esempio

```php
namespace App;
class Filter
{
    public function isEmail(string $email): bool {
        // @todo
    }
}
```

```php
namespace App\Test;
use PHPUnit\Framework\TestCase;
use App\Filter;
class FilterTest extends TestCase
{
    public function testValidEmail() {
        $filter = new Filter();
        $this->assertTrue($filter->isEmail('foo@bar.com'));
    }
    public function testInvalidEmail() {
        $filter = new Filter();
        $this->assertFalse($filter->isEmail('foo'));
    }
}
```

Code coverage

- E’ possibile ottenere la copertura del codice con PHPUnit (tramite Xdebug):
  - **vendor/bin/phpunit --coverage-text**

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		35.36%	389 / 1100		18.73%	65 / 347		27.45%	14 / 51
Controller		90.23%	231 / 256		78.26%	18 / 23		71.43%	10 / 14
Entity		9.74%	41 / 421		11.33%	29 / 256		0.00%	0 / 18
EventListener		6.25%	5 / 80		9.09%	1 / 11		0.00%	0 / 3
Form		100.00%	14 / 14		100.00%	3 / 3		100.00%	1 / 1
Model		38.46%	10 / 26		21.43%	3 / 14		0.00%	0 / 2
Repository		81.82%	36 / 44		75.00%	3 / 4		75.00%	3 / 4
Security		93.94%	31 / 33		75.00%	6 / 8		0.00%	0 / 1
Serializer		9.32%	11 / 118		7.14%	1 / 14		0.00%	0 / 4
Service		7.07%	7 / 99		7.69%	1 / 13		0.00%	0 / 3
Util		33.33%	3 / 9		0.00%	0 / 1		0.00%	0 / 1
AppBundle.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

Legend

Low: 0% to 50% **Medium**: 50% to 90% **High**: 90% to 100%

PHPStan

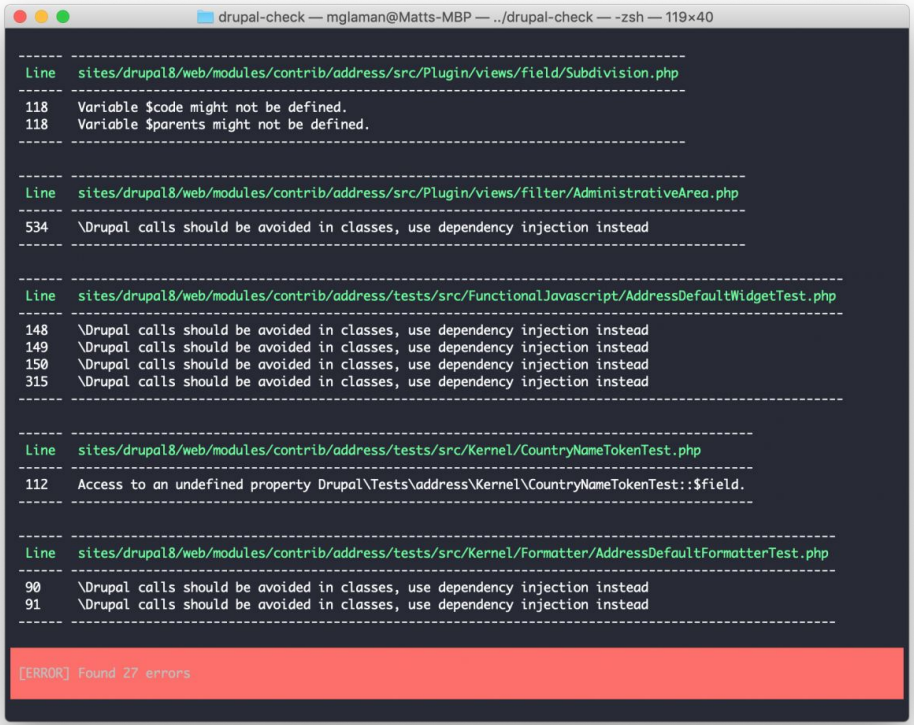
- PHPStan è uno strumento di analisi statica del codice utilizzato per evidenziare (potenziali) errori in un progetto PHP senza eseguirlo
- Installazione:

```
composer require --dev phpstan/phpstan
```

- Esecuzione:

```
vendor/bin/phpstan analyse src tests
```

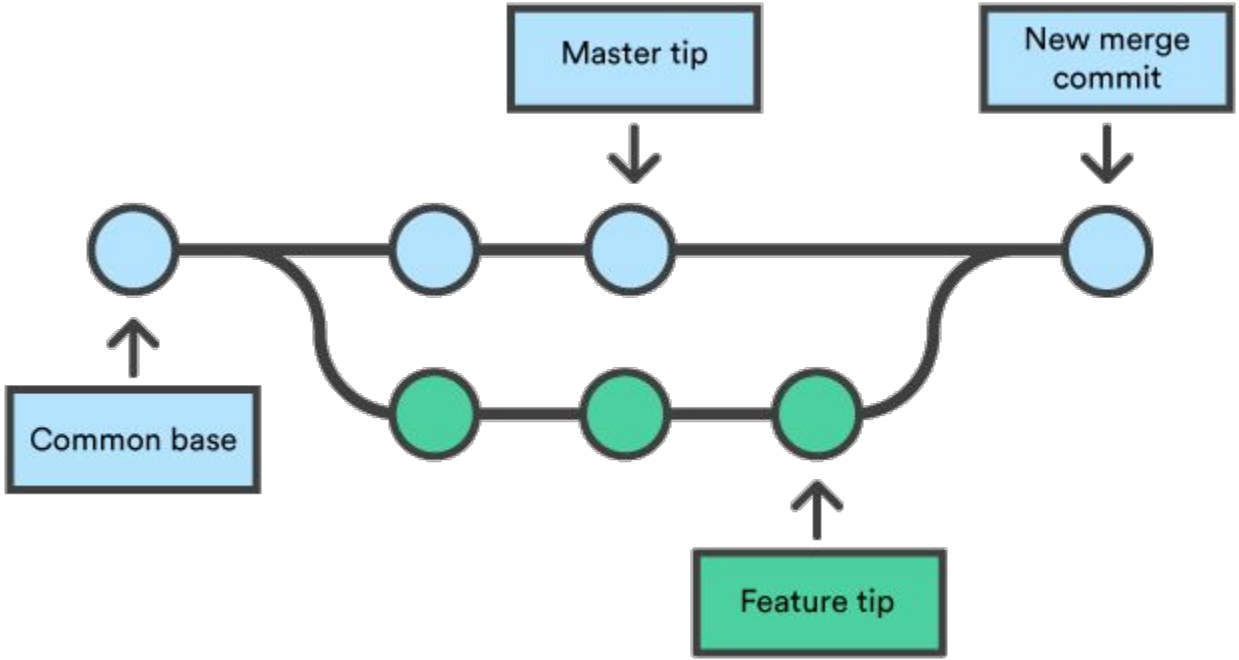
PHPStan esempio di output



Git

- Git è un sistema open source distribuito per il versionamento dei sorgenti ideato per gestire progetti di qualsiasi dimensioni
- Basato sull'idea del **branching** e del **merging**
- Ogni copia locale dei sorgenti detiene tutta la storia delle modifiche dei sorgenti (**commit**)
- Originariamente sviluppato da Linus Torvalds nel 2005

Branch & merge



Github

- Github è un provider per la gestione dei sorgenti e dei progetti software tramite Git
- Offre gratuitamente il servizio per progetti sia pubblici che privati (con qualche limitazione sulle risorse)
- E’ anche questo uno standard de facto
- I recruiter utilizzano le metriche di Github per selezionare i migliori programmatori

Framework di sviluppo

- CodeIgniter
- Laravel
- Slim
- Symfony
- Laminas (ex Zend Framework)

Metodologie

KISS

- **Keep It Simple, Stupid** (KISS)
- Principio di progettazione ideato nel 1960 da U.S. Navy
- Ideato dall'Ing. Kelly Johnson
- Ripreso nella filosofia di Unix:
  - Write programs that do one thing and do it well
  - Write programs to work together
  - Make it easy to write, test, and run programs

Fai una sola cosa ma bene

```
$ head -n 3 /var/log/syslog

May 19 07:19:51 XPS15 rsyslogd:  [origin software="rsyslogd"...
May 19 07:19:52 XPS15 rtkit-daemon[1771]: Supervising 8 threads...
May 19 07:19:53 XPS15 rtkit-daemon[1771]: message repeated 3 times...
```

```
$ tail -n 3 /var/log/syslog

May 19 22:03:16 XPS15 kernel: [29379.492699] mce: CPU7: Package...
May 19 22:03:16 XPS15 kernel: [29379.492700] mce: CPU9: Package...
May 19 22:03:20 XPS15 wpa_supplicant[883]: wlp59s0: WPA: Group...
```

Pipe: restituire la decima riga dall'alto di un file: **head -n 10 file** | **tail -n 1**

SOLID

- **S**ingle-responsibility principle
- **O**pen–closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

TDD

- Il **Test Driven Design** (**TDD**) è una metodologia che ribalta le fasi sviluppo/test in test/sviluppo
- Si parte dalla definizione del test del programma per procedere poi all'implementazione del codice per fare in modo che il test passi
- Focalizza l'attenzione sulle specifiche del progetto
- Aiuta a chiarire le specifiche lavorando su dei casi concreti di utilizzo del software

Dependency Injection

- La **Dependency Injection (DI)** è un design pattern che esplicita le dipendenze tra oggetti, iniettandole in costruzione (o tramite setter)

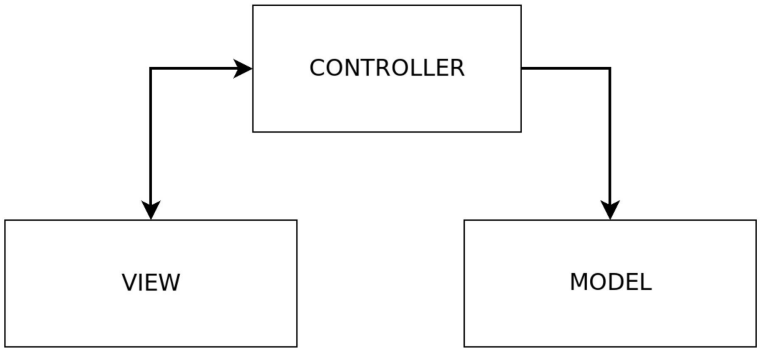
Esempio con e senza DI

```php
class Cart
{
    public function __construct()
    {
        $this->pdo = new PDO(/* dsn */);
    }
}
$cart = new Cart();
```

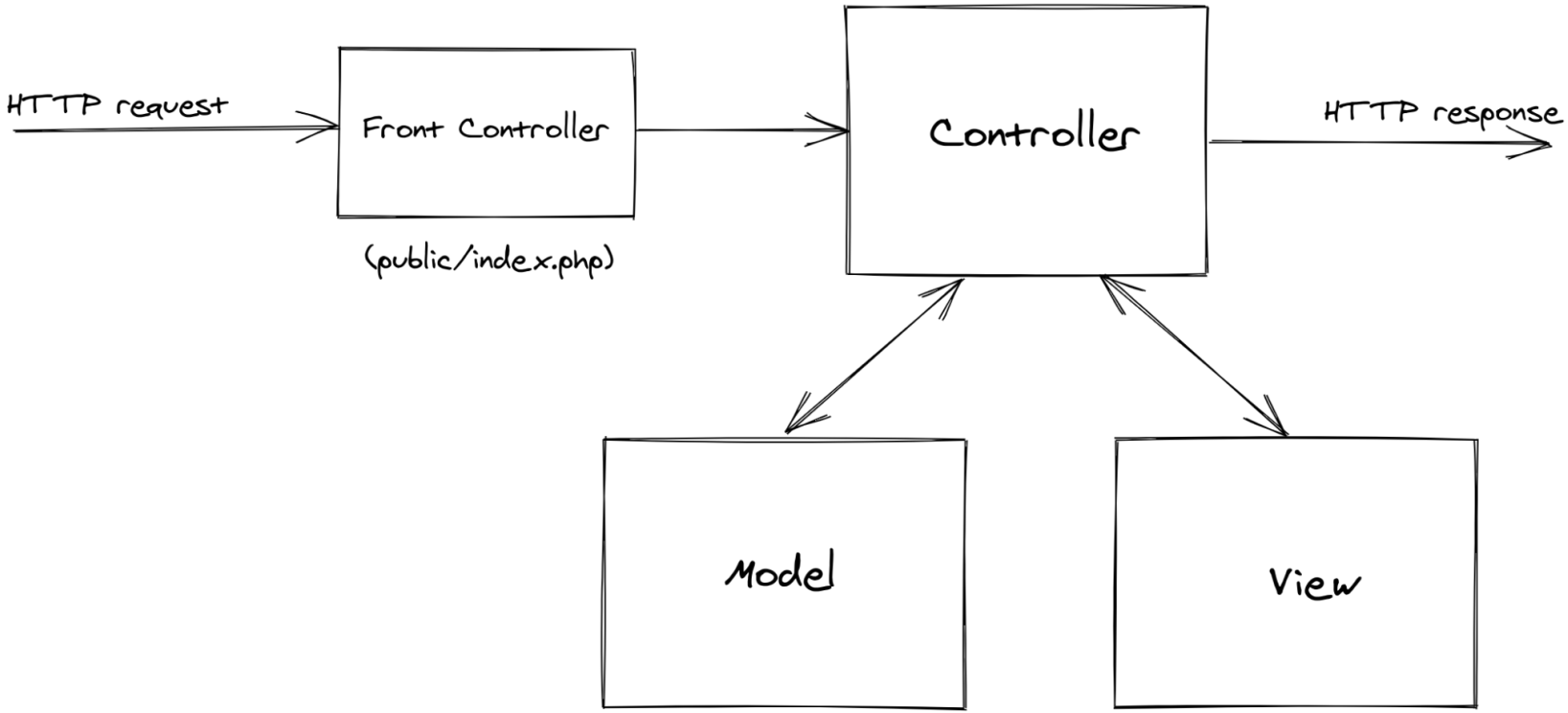
```php
class Cart
{
    public function __construct(PDO $pdo)
    {
        $this->pdo = $pdo;
    }
}
$pdo = new PDO(/* dsn */);
$cart = new Cart($pdo);
```

MVC

- **Model-View-Controller** (MVC) è un design pattern di tipo architetturale che suddivide un'applicazione in tre insiemi:
  - Model, business logic (es. accesso al db)
  - View, l'output di un'applicazione (es. HTML)
  - Controller, logica di routing



Front Controller



Riferimenti

- Arnon Axelrod, *Complete Guide to Test Automation*, Apress, 2018
- Gerald D. Everett, *The Value of Software Testing to Business: The Dead Moose on the Table,* The Magazine for Professional Tester, June 2008
- Bob Hunt, Tony Abolfotouh, *Software Test Costs and Return on Investment (ROI) Issues*, Presentation, 2014
- Jussi Koskinen, *Software Maintenance Costs*, University of Eastern Finland, 2015
- Divya Kumar, K. K. Mishra, *The Impacts of Test Automation on Software's Cost, Quality and Time to Market*, Proceeding of International Conference on Communication, Computing and Virtualization, 2016
- P. Laplante, F. Belli, J. Gao, G. Kapfhammer, K. Miller, W.E. Wong, D. Xu, *Software Test Automation*, Advances in Software Engineering, 2010
- Chris Vander Mey, *Shipping Greatness: Practical lessons on building and launching outstanding software, learned on the job at Google and Amazon*, O'Reilly, 2012
- Planning Report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, NIST, May 2002
- Rudolf Ramler, Klaus Wolfmaier, *Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost*, Proceeding of International Workshop on Automation of Software Test, Shanghai, China, 2006

Riferimenti (2)

- Himanshu Sheth, How To Generate PHPUnit Coverage Report In HTML and XML?, LambdaTest 2021
- Martin Fowler, L'arte del Refactoring , Apogeo 2019, ISBN 9788850334834
- Matt LeMay, Agile per tutti , Apogeo 2019, ISBN 9788850334858
- Robert C. Martin, Clean Agile , Apogeo 2019, ISBN 9788850335046
- Robert C. Martin, Clean Architecture , Apogeo 2018, ISBN 9788850334391
- Robert C. Martin, Clean Code , Apogeo 2018, ISBN 9788850334384
- Fabio Mora, DevOps , Apogeo 2019, ISBN 9788850334506
- Michael Nygard, L'arte del Rilascio , Apogeo 2018, ISBN 9788850334674
- Jacopo Romei, Extreme Contracts: Il knowledge work dalla negoziazione alla collaborazione , 2017,
- ISBN 978-1973312529
- D.Thomas, A.Hunt, Pragmatic Programmer edizione 20° anniversario, Apogeo 2020, ISBN
- 9788850335121
- E. Zimuel, Sviluppare in PHP 7, Tecniche Nuove, pp. 432, 2019

Grazie!

Contatti: enrico@zimuel.it

Linkedin: https://www.linkedin.com/in/ezimuel/

Twitter: https://twitter.com/ezimuel