

# PHP alkalmazások minőségbiztosítása

- Kovács Ferenc (Tyrael)
- Fejlesztési vezető @ Escalion
- QA wannabe többek között

# PHP alkalmazások

- Webalkalmazások
- Cli alkalmazások
- Cronjobok
- Etc.

# Mi az a minőség?

- **Általánosságban:**
  - Az elvárásoknak való megfelelés foka.
- **Részekre bontva:**
  - A tervezés minősége (Quality of design)
  - A kivitelezés minősége (Quality of conformance)
- **Néhány szempont:**
  - Használhatóság
  - Teljesítmény
  - Biztonságosság

## Mit jelent nekünk a minőség?

- Projecttől függ.
- Fogalmazzuk meg, és dokumentáljuk le.
- A döntéseinknél vegyük figyelembe a minőségi elvárásainkat.
- Folyamatosan próbáljunk javítani a minőségen, ehhez pedig folyamatosan javítani kell a munkafolyamatainkon is.

## Mi az a minőségbiztosítás?

- Olyan eszközök és procedúrák, amik kidolgozásával, bevezetésével és betartásával biztosítani lehet az elvárt minőség elérését.
- Olyan buzzword, aminek keretében rengeteg pénzt ki lehet fizetni okos tanácsadó és auditor cégek számára azt gondolván, hogy csupán ettől, a hozzáállás, vagy a munkafolyamatok megváltoztatása nélkül javulni fog a kezünk közül kikerülő termékek minősége.

# Miért van szükség a minőségbiztosításra?




## We're sorry!

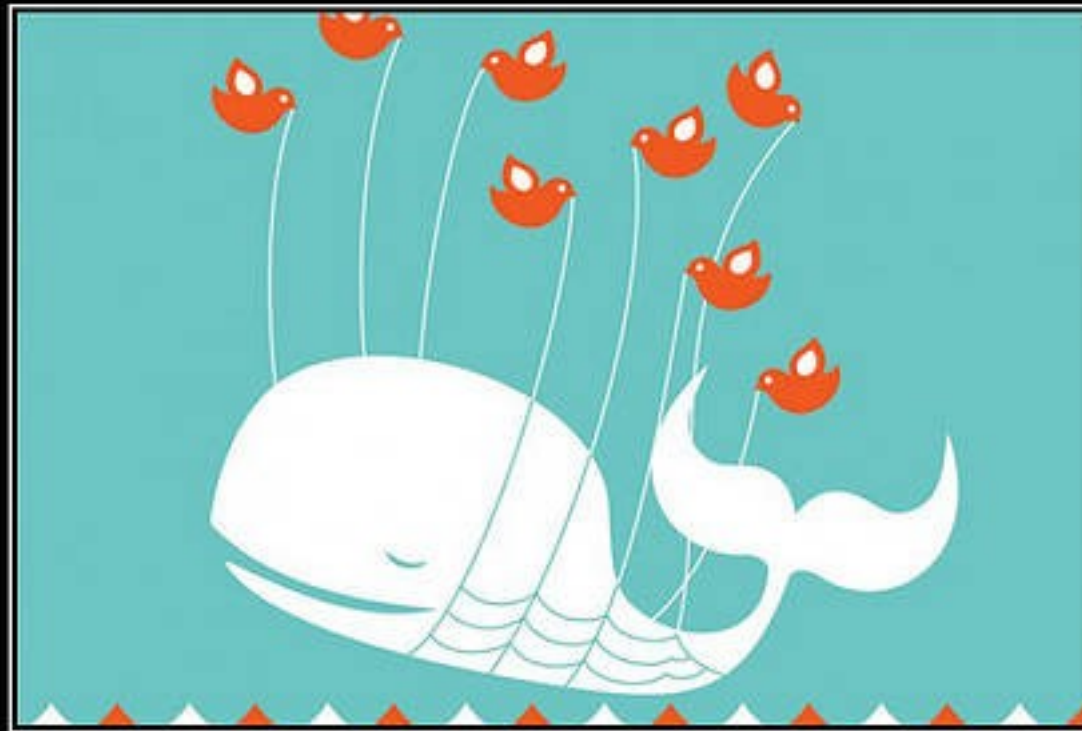
An error occurred when we tried to process your request. Rest assured, we're already working on the problem and expect to resolve it shortly.

If you were trying to make a purchase, please check [Your Account](#) to confirm that the order was placed.

We apologize for the inconvenience.

[Continue shopping](#)  on the Amazon.com home page

# Miért van szükség a minőségbiztosításra?



## FAIL WHALE

Twitter: Failure is an option. At least once a day, or whenever you need it.



# Miért van szükség a minőségbiztosításra?

 **Error**

## Server Error

The server encountered a temporary error and could not complete your request.

Please try again in 30 seconds.

# Miért van szükség a minőségbiztosításra?



# Miért van szükség a minőségbiztosításra?

Általában inkább ezért:



Roszabb esetben pedig ezért:

```
Parse error: syntax error, unexpected $end, expecting ';' or '"' in /var/www/default/unexpected.php on line 3
```

# Mi az a hiba?

- A hiba az alkalmazás nem kívánt viselkedése.
- Vagy egy molylepke:

9/9

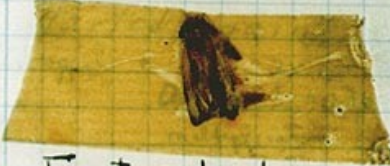
0800 Antam started  
 1000 " stopped - antam ✓

13<sup>00</sup> (032) MP-MC ~~1.982647000~~ { 1.2700 9.037 847 025  
 2.130476415 (3) 4.615925059 (-2) }  
 (033) PRO 2 2.130476415  
 connect 2.130676415

Relays 6-2 in 033 failed special speed test  
 in relay " 10.00 test.

Relays changed

1100 Started Cosine Tape (Sine check)  
 1525 Started Multi Adder Test.

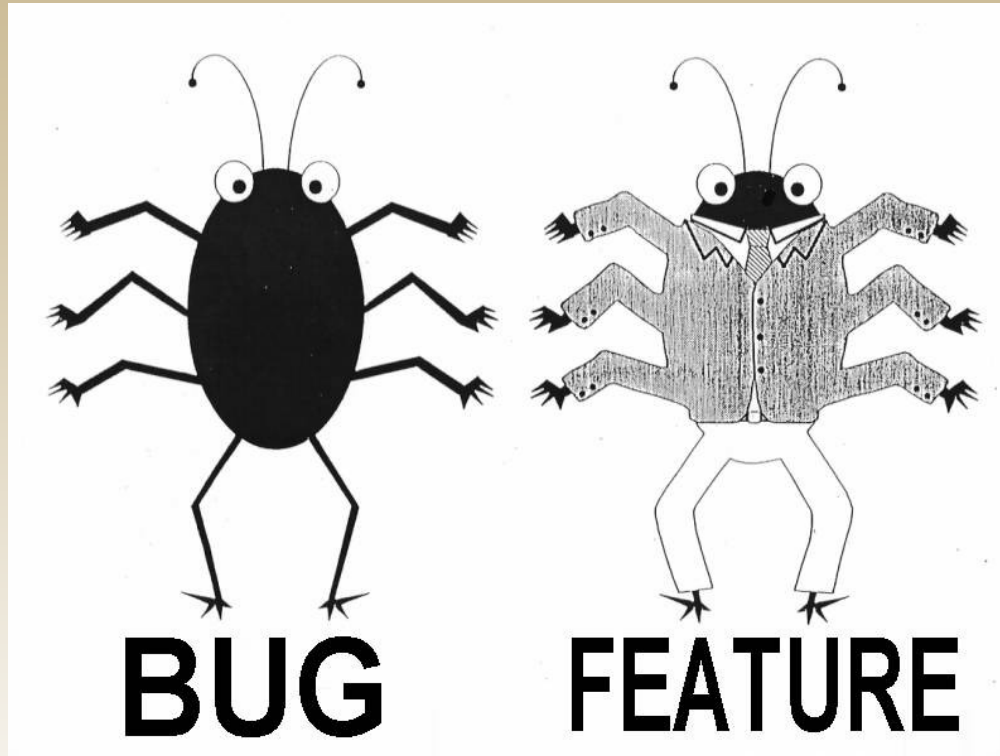
1545  Relay #70 Panel F  
 (moth) in relay.

First actual case of bug being found.

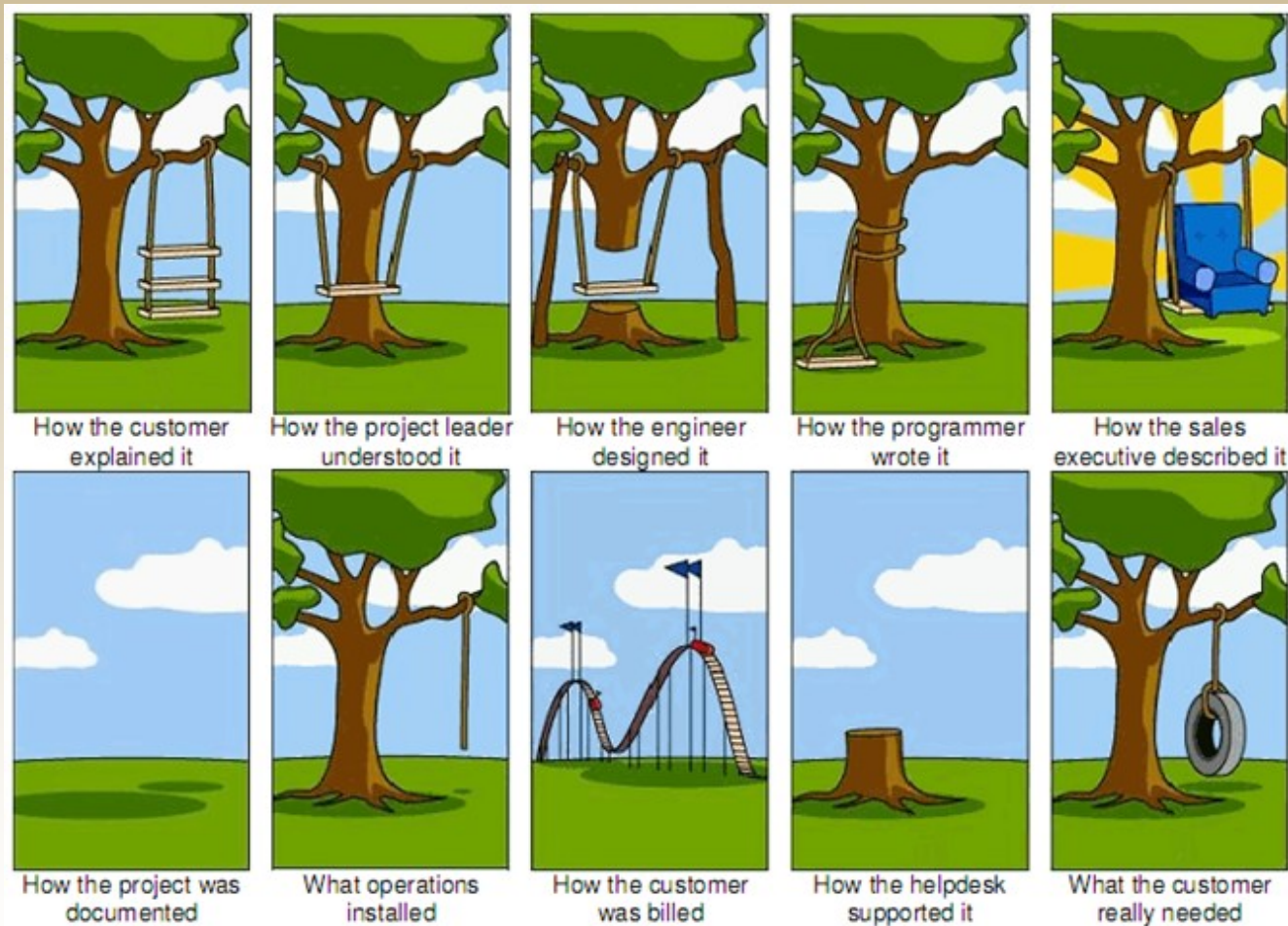
1630 Antam started.  
 1700 closed down.

Relay 3145  
 Relay 3370

# Mi a különbség a bug és a feature között?



## Kommunikációs problémák



# Honnan jönnek a hibák?

## A tesztelés hiánya



**TESTING**

I FIND YOUR LACK OF TESTS DISTURBING.

## Honnan jönnek a hibák?

- A jó fejlesztő nem feltétlenül jó tesztelő:
  - Optimista fejlesztő: A pohár félig tele van.
  - Pesszimista tesztelő: A pohár kétszer nagyobb, mint a specifikációban.
- Pláne ha nem is jó fejlesztő
- Az sem segít, ha közelít a határidő
- Ismeretlen terepen könnyű hibázni
- Főleg ha van néhány clever megoldás:
  - `if (statusIsValid.compareTo( Boolean.FALSE ) != 0) skipValidation = false;`



## A fejlesztők írjanak hibátlan kódot!

- Minden nem-triviális szoftver tartalmaz legalább egy hibát.
- 20 hiba 1000 soronként átlagosnak tekinthető
- PhpMyAdmin: 125KLOC
- WordPress: 88KLOC
- Drupal: 146KLOC
- Ez csak a PHP kód mennyisége

# A fejlesztők írjanak hibátlan kódot!

- De az én kódom hibátlan!
  - És az általad használt PHP libek?
  - És a C libek?
  - A PHP core?
  - A C fordító?
  - A fájlrendszer?
  - A driverek?
  - A kernel?
  - A hardver?
  - A többi gép, szolgáltatás, amivel kommunikálsz?

## Mit tehetünk?

- Definiáljuk az elvárt viselkedést, és kezeljük a váratlan helyzeteket(fail-secure).
- A lehető legkevesebb hibát vezessük be.
- A hibákat a lehető legkorábbi fázisban találjuk meg és javítsuk ki.
- Kövessük nyomon a hibákat a bejelentéstől a javításig.
- Tegyük róla, hogy a már kijavított hibák ne jelenhessenek meg újra (regresszió).

# Definiáljuk a működést

- Követelményspecifikáció (SRS)
  - Funkcionális követelmények:
    - User story
    - Use case
    - Unit test
  - Nem-funkcionális követelmények
    - Rendelkezésreállítás (SLA)
    - Teljesítmény
    - Skálázhatóság
    - Karbantarthatóság
    - Stb.

## User story

- Scrum-ból már ismerős lehet
- Leírja az adott funkció/komponens/rendszer működését a felhasználó szemszögéből.
- Általában a megrendelőtől érkezik, ennél fogva nagyon High level.
- Pl: Bejelentkezett felhasználóként ki kell tudjam tölteni a profilomat, és megtekinthetem mások profiljait, de nem szerkeszthetem őket.
- A teljes rendszer nagyon sok US-ból áll össze.

## Use case

- Egy konkrét esemény leírása a rendszerben.
- Lehetőleg ne azt írja le, hogy hogyan, hanem hogy minek kell történnie.
- Ez már algoritmizálás.
- PI:
  - 1. A rendszer bekéri a belépési adatokat,
  - 2. A felhasználó megadja a nevét és a jelszavát,
  - 3. A rendszer leellenőrzi a megadott adatokat,
  - 4. A rendszer belépteti a felhasználót.

## Unit test

- Lehetővé teszi a kód legkisebb önálló egységeinek(függvény/metódus) a tesztelését.
- Definiálja a kód elvárt viselkedését.
- Megfelelő lefedettségénél segít észlelni a hibákat.
- PHPUnit – de facto standard.
- SimpleTest – érdeklődés hiányában elhunyt.

## Unit test

- Önmagában megérne egy külön előadást.
- A QA eszköztárunk egyik legfontosabb eleme.
- Meg kell tanulni tesztekét írni.
- Meg kell tanulni jó tesztekét írni.
- Ideális esetben a Unit test írás a fejlesztés szerves részét képezi.
- Ha előbb írod a tesztekét, mint a kódot, akkor beírhatod a CV-dbe a Test-driven Development kifejezést a tapasztalatok közé.



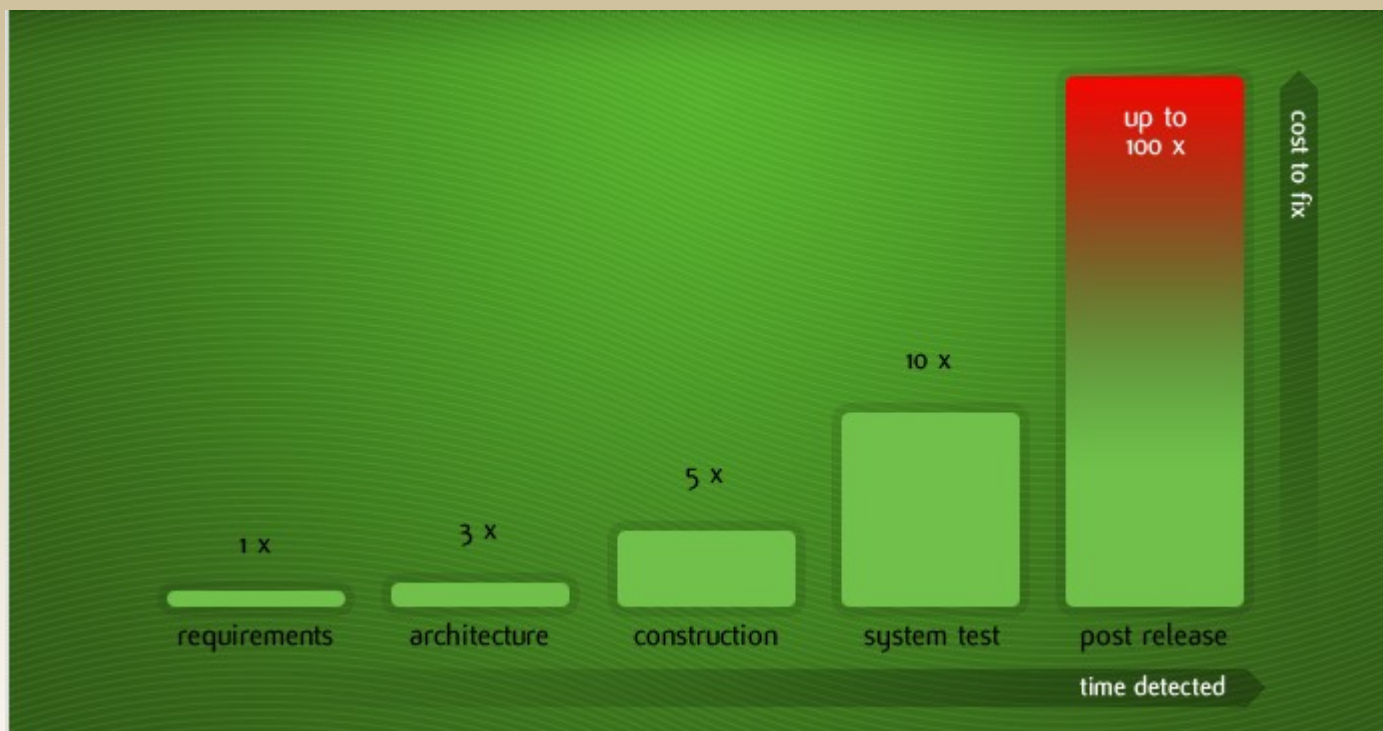
## Nem funkcionális követelmények

- Rögzíteni kell a követelményeket.
- A tervezésnél figyelembe kell venni a követelményeket.
- A rendszer annyira robusztus, mint a leggyengébb összetevője:
  - Nem tudsz 99.9999% rendelkezésre állást garantálni, ha az áramszolgáltatás csak 99.99%-ot tud.
- Folyamatosan monitorozni kell ezen feltételek teljesülését, és ha szükséges, módosítani a munkafolyamatokat.

## Kezeljük a váratlan helyzeteket

- Hibakezelés, minden hibát logoljunk, elemezzünk.
- A felhasználók fele ne jelenítsünk meg hibákat.
- ErrorDocument(apache), error\_page(nginx), fallback(lidirectord), etc.
- A php engine 500as hibát küld ha:
  - szabálytalanul fejeződik be a script futása
  - és nincs bekapcsolva a hibakijelzés
  - és nem lett kiküldve már kimenet
  - **ÉS** nincs bekapcsolva az xdebug

A hibákat a lehető legkorábbi fázisban találjuk meg és javítsuk ki.



## Issue tracking

- Minden feladat kerüljön bele.
- Megfelelően legyenek priorizálva a feladatok.
- Mindig a prioritás szerint foglalkozzunk a feladatokkal.
- Ne legyenek gazdátlan issuek.
- Tartsunk bugfix sprinteket.
- Nagyon fontos a jó, reprodukálható hibajegy, néha ez több munka, mint a javítás.
- Ne zárjuk le ész nélkül a hibákat invalid vagy worksforme jelzővel.

# Regresszió

- A regresszió a hibák zombija, hiába öljük meg, mégis időről időre visszajön.
- Minden javított hiba után írjunk egy tesztet kifejezetten arra a hibára, és innentől kezdve nyugodtan alhatunk.
- Ha visszatér egy bug, vizsgáljuk meg, hogy mi okozhatta, és vegyük elejét az ilyen eseteknek.
- Nincs idegesítőbb, mint amikor sokadszorra jön elő ugyanaz a probléma.

# Gyakorlati tanácsok

- Fejlesztői környezetek, konfiguráció kezelés
- Verziókezelés
- Adatbázis migráció
- Deployment
- Folyamatos Integráció
- Dokumentáció
- Statikus kódanalízis
- Tesztelés
- Refactorálás

# Fejlesztői környezetek

- Devel:
  - minden fejlesztőnek saját.
  - ha linuxra fejlesztünk, akkor legyen linux.
  - töredék méretű teszt adatokkal.
- Staging/Test:
  - a produkciós környezethez a lehető legnagyobb mértékben hasonlító rendszer
  - az adatok ideális esetben az élesről vannak tükrözve/replikálva.

# Fejlesztői környezetek

- Live/production:
  - Kód csak a deploy folyamaton keresztül kerülhet ki.
  - A hibák nem kerülhetnek megjelenítésre a felhasználók számára (max karbantartás üzenet).
  - A hibák logolásra kerülnek(központi logolás, mondjuk syslog-ng), és a fejlesztés/üzemeltetés értesül róla.
  - Folyamatosan monitorozni kell a helyes működést, és követve az előre elkészített forgatókönyveket beavatkozni a hibás működés esetén.



# Configuráció kezelés

- Az eltérő environmentekhez tartozó configokat lehetőleg kívülről injectáljuk be:
  - környezeti változók
  - auto\_prepend\_file
  - hidedf
- Mindig a produkciós környezet legyen az alapértelmezett.
- Figyeljünk oda a cli/cronjobokra, ha környezeti változó kell a futáshoz.

## Configuráció kezelés

- A különböző konfigurációs állományok is legyenek verziókezelés alatt, ezáltal könnyebb azonos beállításokra hozni több szervert, valamint könnyebb követni a változások tényét és okait is (commit logok).
- Ne bízunk a default konfigurációs beállításokban, általában a könnyű használhatóságra vannak optimalizálva, nem pedig a nagy teljesítményre, vagy biztonságos működésre.

- Commit szabályok
  - Minden commit előtt update-elj.
  - Minden commithoz adj meg commentet.
  - Ne fogj össze egy commitba több feature-t, bugfixet.
  - Commitolj sűrűn, de ne küldj be hibás kódot.
  - Lehetőleg igyekezz a nap végére mindig commitolható állapotot kialakítani.

- Branchelés, mergelés
  - Hosszabb, vagy elhúzódó refactornak csinálj feature branchet, így nem akadályozod a többiek munkáját.
  - Mergelésnél amikor lehet igyekezz kézi feloldás nélkül megoldani a conflictokat (talán csak nem lett mergeelve egy korábbi commit).

- Fejlesztési ágak
  - Trunk/Head
  - Testing/Staging
  - Live/Production
- Feature branchek
  - Hozd létre ha szükség van rá, mergeld vissza ha kész, majd nyugodtan töröld le.
- Tagelés
  - Tageld az RC-ket, a release-eket, sose mozgass taget.

# Commit hookok

- Pre-commit hook:
  - Megkövetelhetjük, hogy legyen megadva commit comment, vagy hogy a módosított kód szintaktikailag helyes legyen (php -l).
- Post-commit hook:
  - Küldhetsz emailt a commitról.
  - Elindíthatod a CI buildet (lásd később).
  - Elindíthatod az élesítést.

# Adatbázis migráció

- Tartsd a schema-t a verziókezelőben.
- Figyelj oda, hogy a schema módosítás lockolja az adatbázist.
- Ha replikációt használsz, akkor figyelj oda a slave-lagra.
- Ha leállítás nélkül akarsz migrálni, akkor jól jöhet a Master-Master replikáció, vagy valami schemaless nosql megoldás.
- Phing/dbdeploy.

# Adatbázis migráció

- Kétfázisú migráció: először hozd létre az új oszlopot(prepare), aztán vidd ki a kódot(activate), ha nem jó rollback, ha megy, akkor a következő deploy eltávolíthatja a feleslegessé vált oszlopokat.
- Mindig legyen rollback/downgrade terved.
- Ha nem tudsz online migrálni, akkor még mindig üzemelhetsz read-only üzemmódban csökkentett funkcionalitással, amíg tart a migrálás.



# Deployment

- Ne menjen eseményszámba, legyen rutinszerű.
- A Phing kényelmes, de nem nagy szám sajátot sem írni.
- Minél kevesebb dolgot módosítasz egy élesítéssel, annál kisebb az esélye, hogy eltörsz valamit, szóval élesíts gyakran kis módosításkészletet.

# Deployment

- Figyelj oda arra, hogy a deploy folyamat atomi és konzisztens legyen a clusterben:
  - Először vidd ki a lebuildelt kódot (prepare).
  - Utána tereld rá a forgalmat (activate).
  - Ezt meg lehet oldani többféleképpen is, jó esetben csak a loadbalancerrel kell újraolvasztatni a configot.
  - Ne felülírd a kint lévő kódot, hanem mindig a régi verziók mellé tedd le, ezáltal a rollback gyorsabb és nincs probléma a cache-ekkel.

# Folyamatos Integráció (CI)

- „Buildbot”
- Megadott feltételek(x időnként, ha változik a kód a repóban, etc.) teljesülése esetén végrehajtja újrabuildeli a projectet.
- Általában az alap build taskok:
  - Húzd ki a kódot a repóból.
  - Futtasd le rajta a Unit teszteket.
  - Készíts kimutatásokat az eredményről.
  - Hiba esetén értesítsd a fejlesztőt.

# Folyamatos Integráció (CI)

- Javaban íródott:
  - Hudson (jelenleg talán a legjobb választás)
  - PHPUnderControl(CruiseControl)
  - Bamboo (Atlassian)
- PHPban íródott:
  - Arbit (CI+issue tracking+wiki)
  - Xinc (halódik)

## Folyamatos Integráció (CI)

- PHPUnit
- pdepend : nice metrics (CylCo, NPath, etc.)
- phpcs : PHP Code Sniffer
- phpcpd : copy paste detector
- phpdcad : dead code detector
- PHP\_CodeCoverage : code coverage :)
- phpdoc : documentation builder
- PHPMD : PHP Mess Detector (Cyclomatic Complexity, NPath Complexity)

- Dokumentációból is lehet túl sok, de ez ritka. :)
- Válasszuk szét aszerint, hogy kinek szól:
- Fejlesztői dokumentáció:
  - Inline kommentek
  - PHPDoc
  - High-level rendszerterv
- Üzemeltetői
- Felhasználói dokumentáció

# Statikus kódanalízis

- Bytekit/bytekit-cli: bytekód analízis, ki lehet vele pl. szűrni zend tokenekre (EVAL, PRINT).
- Phploc: kódméretet(KLOC, osztályok, függvények száma, tesztek) lehet vele becsülni
- Phantm: típusellenorzes
- PATAWAN: PHP antipattern detector
- RIPS sebezhetőség detektor

- **Funkcionális:**
  - Unit tesztek
  - Integrációs, rendszer tesztek
  - Regressziós tesztek
  - (User)Acceptance tesztek
  - Selenium tesztek
- **Nem funkcionális:**
  - Security tesztek(whitebox, blackbox)
  - Teljesítmény tesztek



## Unit Tesztelés

- Reflection: futásidőben manipulálhatóak az osztályok.
- Runkit: futásidőben manipulálhatóak a beépített függvények, osztályok, állandók.
- VfsStream: mockolható fájlrendszer
- mysqlnd\_uh : userland mysql handler

- Óvatosan!
- Ha van elegendő Unit test, akkor bátrabban lehet módosítani, hiszen jó eséllyel kiszúrod, ha valamit eltörsz a változtatással.
- Ha nincs elegendő, akkor írd!
- Ne írd újra egy teljes rendszert nulláról, ha nem muszáj.
- Szinte sosem muszáj.
- Egyszerre egy dolgot javíts ki, keresd a könnyű prédát és folytasd iteratívan.

# Összefoglalás

- Legyen tervünk!
- Ismerje a tervet mindenki, akinek szükséges.
- Ragaszkodjunk a tervhez.
- De folyamatosan keressük a javítási lehetőségeket a tervben.
- Minden nagyobb probléma után vizsgáljuk meg, hogy mi okozta a problémát, és javítsuk a folyamatokat ennek megfelelően.

Vég(r)e!

Kérdések?

Köszönöm a figyelmet!

[info@tyrael.hu](mailto:info@tyrael.hu)

<http://tyrael.hu>

<http://twitter.com/Tyr431>