IPC 2019: TDD Workshop (21.10.2019)

@spriebsch
@s_bergmann

TDD erstmals in 50er bei Gemini/Apollo NASA Mission
dann wiederentdeckt in den 2000er für Softwareentwicklung

testen in anderer Abteilung ist schwierig, diese wissen gar nicht was "richtig" ist, ob es funktioniert.
Testplan? Dokumentation veraltet.

BDD ist eher ein Systemtest (Dokumentation im Code)

Tests dokumentieren die Erwartungen

Test grün = stoppen mit entwickeln = Feature ist gut genug
red-green-refactor Zyklus (geil für Farbenblinde, kann aber umkonfiguriert werden)
Tests nicht anfassen während Refactorings (z.B. Methoden umbenennen)

Wie will man Code schreiben wenn man nicht weiß wie das Ergebnis sein soll. Macht man sich drüber Gedanken und schreib Tests dafür. -> Akzeptanzkriterien

Neue Entwickler brauchen klare Vorgaben -> Akzeptanzkriterien -> Tests

TDD vereinfacht die kognitive Last -> unterbrechungsvolles Arbeiten ist möglich

manchmal weiß man noch nicht was das Ergebnis ist - dann ist TDD hart bzw. nicht zielführend

TDD hat uns nicht vor Software ohne Businesswert geschützt -> macht aber TDD nicht nutzlos

Test motiviert Code zu schreiben. hab ich eine Codestruktur im Kopf und muss mir dazu erstmal einen Test ausdenken, funktioniert das nicht

starte mit Happy Path, Fehlerfälle können später dazukommen
Tests - positive Feedback-Schleife - eigene Drogen produzieren
Edge-Cases primär austesten bringt keinen Businesswert

tcr Script -- test-commit-revert -- Commit wenn Tests grün durchlaufen, Auto-revert wenn Tests fehlschlagen - schon krass

Tests führen auch gleich Syntaxcheck aus.

arbeiten in Baby-steps führt in sehr kleine Feedback-Schleifen

Tests des aktuellen Projektbereichs sollten **innerhalb 1 Sekunde durchlaufen**. wenn es länger dauert, kann man nicht TDD machen. weil Gehirn wird abgelenkt (Kaffee, Emails, Kollegenschwatz, Facebook)

niemals PHPUnit global installieren
man arbeitet an verschiedenen Projekten
mit verschiedenen Code-Ständen

Composer kann PHPUnit installieren (z.B. als Dev-Dependency)
Aber Tools sollten nicht per Composer installiert werden
weil IDE löst diese Dependencies auf
und es gibt Lib-Versions-Konflikte
die gibt es nicht in PHARs, weil diese im PHAR alle in einen Random-Namespace stehen
(in Composer Roadmap steht Random Namespaces)
und PHARs können mit PGP signed werden - verhindert gehackte Pakete ("npm leftpad")

phive install --copy phpunit
ist nicht der perfekte Default-Fall
PHAR mit einchecken! warum? Repo ist unabhängig - kann man auch in 20 Jahren auschecken und alles ist noch da

PHP kann keine PHARs erkennnen wenn diese nicht ".phar" enden
deswegen PHAR hinlegen und Symlink mit ohne ".phar"

Strict Mode in PHPUnit wird auch "German Mode" genannt übern Teich :)

assertSame mein "same type", nicht nur "==" vergleichend

man kann auch Business-Logik innerhalb des Tests schreiben, dann als Methode darin extrahieren, dann in eigene Klasse auslagern

PHPUnit 8.5 (Dezember) kommt evtl. mit Folding für Datasets/Dataprovider-getriebene Tests

Obelus ÷ (Division-Zeichen) https://en.wikipedia.org/wiki/Obelus

Tic-Tac-Toe -- "the only winning move is not to play"

Business-Analyse Methode one-to-four
erst mach jeder für sich Gedanken
dann tauscht man sich zu zweit aus, bringt seine Ergebnisse zusammen
dann bringt man die Ergebnisse von zwei 2er Gruppen zusammen
diese 4er Gruppen präsentieren jeweils das Ergebnis
(günstigerweise schreibt der der ansagt, sonst ggf. lost-in-translation)

https://github.com/thePHPcc/ipc19-test-driven-development

TDD Chicago Style (Kent Beck) Outside-In / Top-Down (Mockist Style)
TDD London Style (Fowler) Inside-Out / Bottom-Up

---

kleinste Einheit? ein Feld

Outside-In -> erstmal eine API schreiben

PHPStan vs Psalm
Warum lieber Psalm?
PHPStan `include` Dateien, führt ihn quasi aus. Problematisch mit Legacy-Code.
Psalm parst PHP-Dateien via PHP-Parser, führt sie nicht aus, analysiert sie tatsächlich nur statisch.
Psalm hat Vimeo dahinter, bezahlt Entwickler in Vollzeit daran zu entwickeln (und erneuern ihre alte Codebasis sukzessive darüber)

private Methods und Propertys sind private und sollen nicht zu testen
es wird ausschließlich über public Methoden getestet

"Fake it until you make it"
klein starten..
TicTacToe mit einem Feld statt einem Array starten
nur eine placeX und eine isFinished
"what is the simplest implementation which could possibly work"

--
https://laravel-news.com/tips-to-speed-up-phpunit-tests

Nur fehlgeschlagene Tests ausführen
phpunit --order-by=defects --stop-on-defect

PHP Unit sollte trocken ohne Tests in ~50ms durchlaufen.
Aufm Mac mit PHP7.2 und Xdebug dauerts 140ms

an TDD gewöhnt man sich, bis es sich wie Fahrrad fahren mit Helm anfühlt.
Ohne ist irgendwie unsicher, macht man nicht.

Wie mit ungetesteten Legacy Code umgehen?
"stopping to feed the black hole"
neuen Code greenfield daneben stellen, testgetrieben entwickeln, dann per Glue-Code anbinden

TDD funktioniert sehr gut wenn man frisch startet