CLEAN CODE

Prato NV

Luk Weyens & Wouter Groeneveld

22/09/2016

INTRODUCTIE

GSM Uit aub

Verstand Aan aub

Vragen stellen toegelaten!

INHOUDSOPGAVE

1. Clean code: definities
2. Clean code: principes
3. Unit testing: definities
4. Unit testing: voorbeelden

CLEAN CODE DEEL 1: DEFINITIE

Wat is clean code?

Clean code is code that is easy to understand and easy to change.

Wat is **niet** clean code?

```
function process($obj, $continue = FALSE) {
        // echo "starting the process";
        if($obj->generate() == NULL) {
                return -1;
        }
        $result = $obj->generate();
        if($continue) {
                return $result->generate();
        }
        return $result;
}
```

Wat is **wel** clean code?

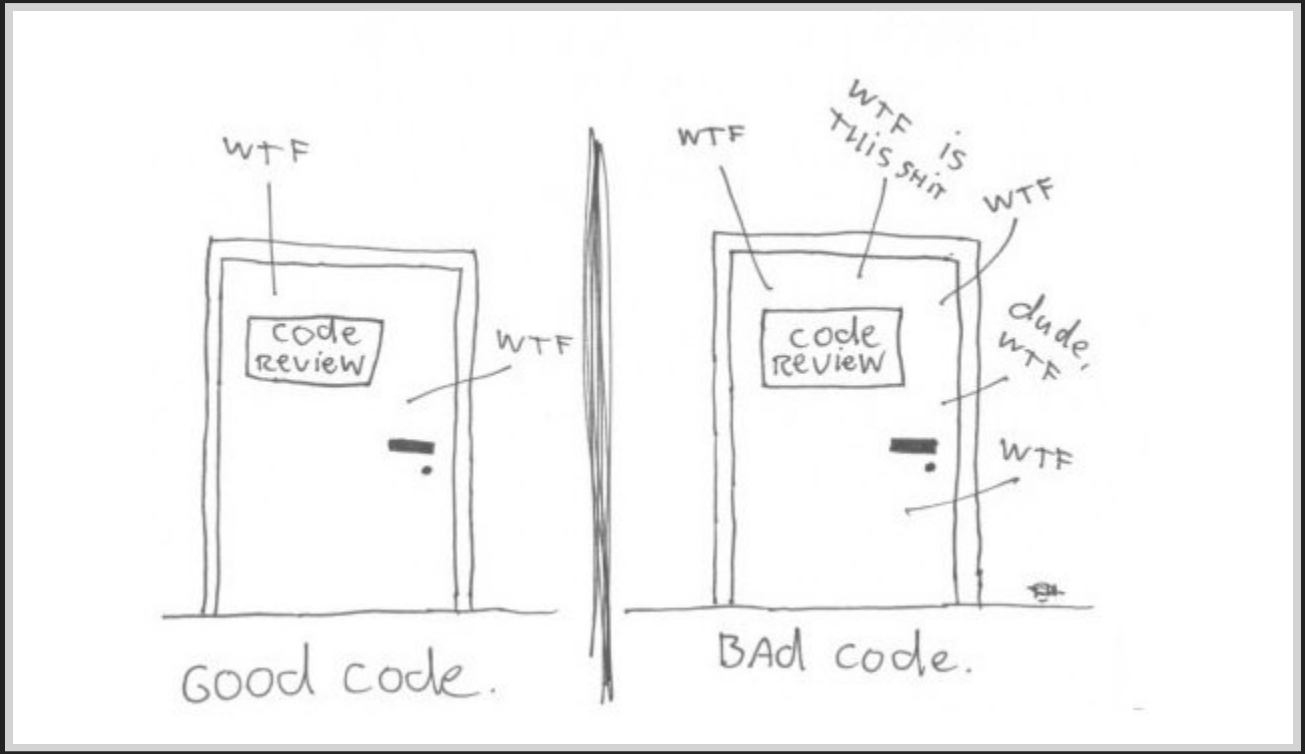
```
function getBookByISBN($isbn) {
        $book = array_search($isbn, $this->books);
        if($book == FALSE) {
                throw new BookNotFoundException("book was not found!
        }
        return $book;
}
```


CLEAN CODE?
WHO CARES?
quickmeme.com

LEESBAARHEID!

Code gets read a lot (at least whenever someone is writing more code), so any school of clean code should emphasize readability. Cleaning up a little wherever you go is required to keep code clean.

MEETBAARHEID:
"AANTAL WTFS PER MINUUT"



CLEAN CODE DEEL 2: PRINCIPES

1. NAAMGEVING:
WAT IS DE IDEALE NAAMGEVING?

- Het moet correct weerspiegelen **waar het voor dient**
- Het leunt aan bij de **taal** die gesproken wordt in het **domein**

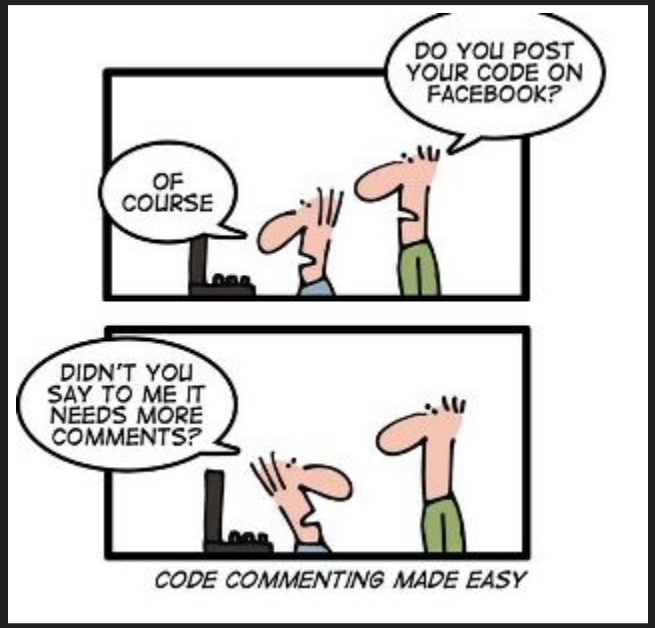
1. NAAMGEVING: METHODS/VARIABELEN

```
$result = $processor->process();
```

VS

```
$order = $library->checkOut();
```

1. NAAMGEVING: COMMENTS



1. NAAMGEVING: COMMENTS

```
/**
 *
 * Prolong the book
 *
 * @param    Book                $i The book to prolong
 * @param    DateTime            $j The date to prolong to
 * @param    Person              $k The person to prolong the book to
 * @return   nothing if OK
 *
 */
function prolong($i, $j, $k) {
        //Check if this book has been checked out by the same person
        if($i->person != $k)
                //BUSTED!
                return "This is not a book you checked out."
        else
                //Everything OK, prolong the book
                $book->checkOutDate = $date;
}
```

1. NAAMGEVING: COMMENTS

```
function prolong($book, $date, $person) {
        if(!$book->isCheckedOutBy($person))
                return "This is not a book you checked out."
        else
                $book->checkOutDate = $date;
}
```

2. FUNCTIES:
WAT IS EEN IDEALE FUNCTIE?

2. FUNCTIES: VERANTWOORDELIJKHEDEN

```
class Book {
        function prolong($date, $person) {
                // does only one thing: prolonging the book
        }
}
```

2. FUNCTIES: ARGUMENTEN

```
function checkoutBook($title, $isbn, $person, $enoughCredit = FALSE) {
        // ...
}
```

VS

```
class Person {
        function checkoutBook($book) {
                // ...
        }
}
```

2. FUNCTIES: SIDE EFFECTS

```
function prolong($date, $person) {
        if($this->isOverDue()){
                $person->credit -= 10;
        }
}
```


OEFENINGEN DEEL 1

OEFENINGEN DEEL 1: STORY 1

- Gegeven: **Empire**, **Sith**, **Jedi** classes vol funcs.
- Uw taak: verduidelijken! Wat doet wat? Naamgeving!
- Unit testen groen houden! (`cmd: > phpunit tests`)

PHPUNIT - RECAP

live demo

```php
    public function tearDown() {
        $this->bestellingen = null;
        Db::close();
    }

    public function testFindAllEmptyDb() {
        $this->assertEquals(0, count($this->bestellingen->findAll()));
    }

    /**
    * @expectedException InvalidArgumentException
    */
    public function testValideerBetaling_wrongAmount_throwsEx() {
        $id = $this->bestellingen->save([
            'naam' => 'jos',
            'bedrag' => '20'
            ]);
        $this->bestellingen->valideerBetaling($id, BetaalType::Bitcoin, 10);
    }

    public function testMarkAsDeleted_FlagsDeleted() {
        $id = $this->bestellingen->save([
            'naam' => 'jos'
            ]);
        $this->assertEquals(0, $this->bestellingen->get($id)['deleted']);
```

http://phpunit.de/manual/

OEFENINGEN DEEL 1: STORY 2

- maak een `function` op Empire, '**convert($jedi)**'
  Wat zou dit volgens u moeten doen?
- maak een `function` op Jedi, '**equip($saber)**'.
  Wat zou dit volgens u moeten doen?
- Vergeet niet de testen uit te breiden!

OEFENINGEN DEEL 1 - DISCUSSIE

- Zijn alle testen nog groen? Heb je er **bijgeschreven**?
- Naamgeving - ook doorgevoerd in testen?
- Onthou principes voor volgende oefeningen!

3. SCOPING - POGING 1

```
public class BookRepository {
        public $db;
        public $bookCache = [];
}

$bookRepositoryInstance->db->query('where isOverdue = TRUE');
```

3. SCOPING - POGING 2

```
public class BookRepository {
        private $db;
        public function getOverdueBooks() {
                return $db->query('where isOverdue = TRUE');
        }
}

$bookRepositoryInstance->getOverdueBooks();
```

3. SCOPING - GLOBAL #FAIL

```
$checkoutDate = NULL;
function isOverdue($book) {
        global $checkoutDate;
        // some domain logic here
        $checkoutDate = new Date();
}

$book->setCheckoutDate($checkoutDate);
```

Zie ook `static` e.a.

4. OO DESIGN - POGING 1

```
$bookid = $_POST['bookid'];
function prolong($book, $date, $person) {
                $book->checkOutDate = $date;
                $person->bookList->push($book);
        }
}
prolong($db->getby($bookid), $_POST['date'], $person);

echo "thank you it has been prolonged"
```

4. OO DESIGN - POGING 2

```
class Book {
        function prolong($date, $person) {
                $person->addToBookList($this);
        }
}
$book = new Book();
$book->prolong($_POST['date'], $person);
$view->renderThankYouPage();
```

```
class BookTest extends PHPUnit_Framework_TestCase {
        public void testProlongAddsToPersonBookList() {
                // ...
        }
}
```

4. OO DESIGN: POLYMORPHISM - POGING 1

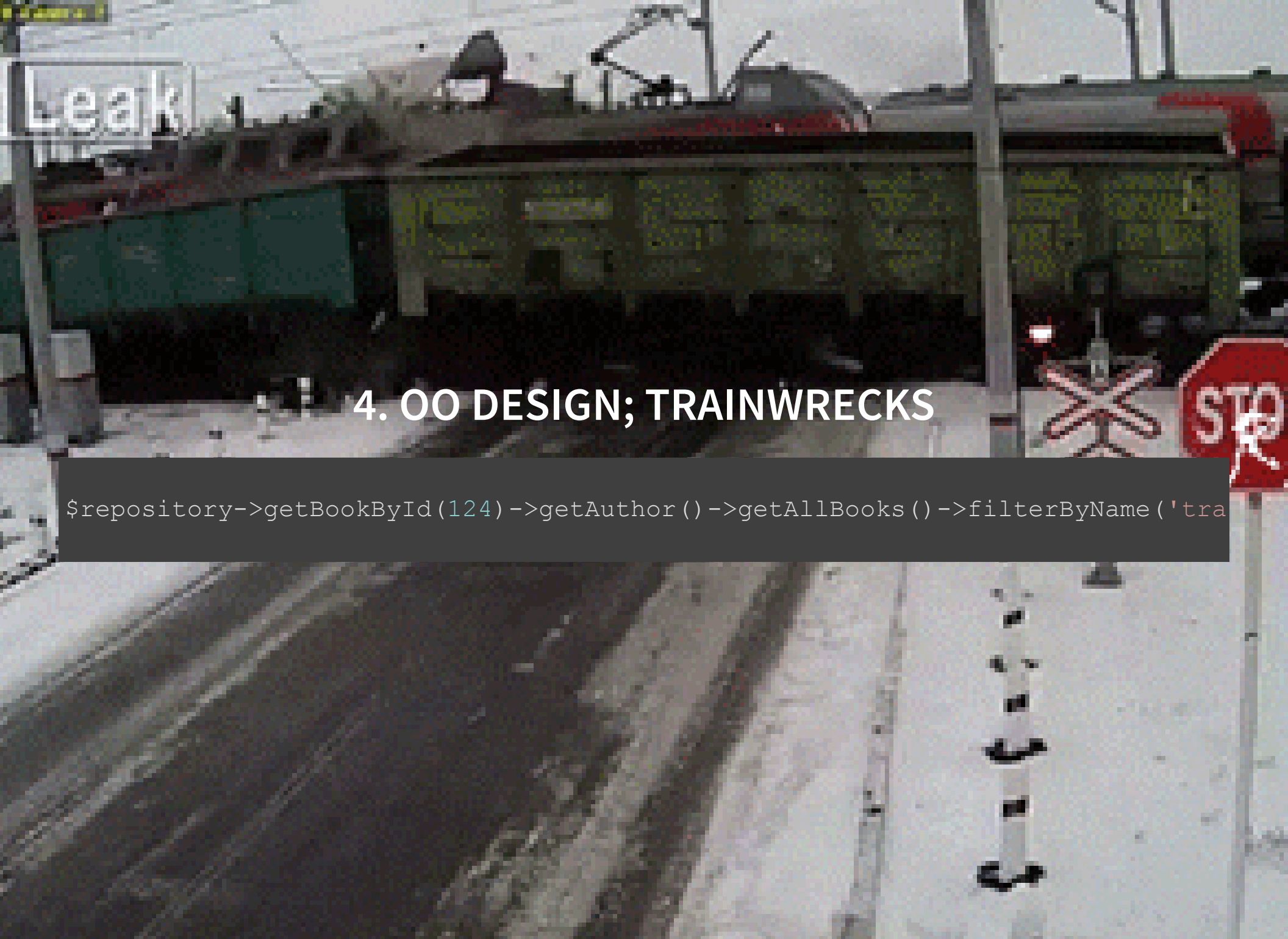
```
function getAisle($book) {
        switch($book->type) {
                case 'nonficition': return 400;
                case 'fiction': return 234;
        }
}

getAisle({ type: 'nonfiction' });
```

4. OO DESIGN: POLYMORPHISM - POGING 2

```
abstract class Book {
        abstract public function getAisle();
}
class FictionBook extends Book {
        function getAisle() {
                return 234;
        }
}
class NonFictionBook extends Book {
        function getAisle() {
                return 400;
        }
}

new NonFictionBook()->getAisle();
```



4. OO DESIGN; TRAINWRECKS

```
$repository->getBookById(124)->getAuthor()->getAllBooks()->filterByName('tra
```

4. OO DESIGN; ERROR HANDLING - POGING 1

```
function getBookById($id) {
        if($id == NULL) return -1;

        return $db->getById($id);
}
```

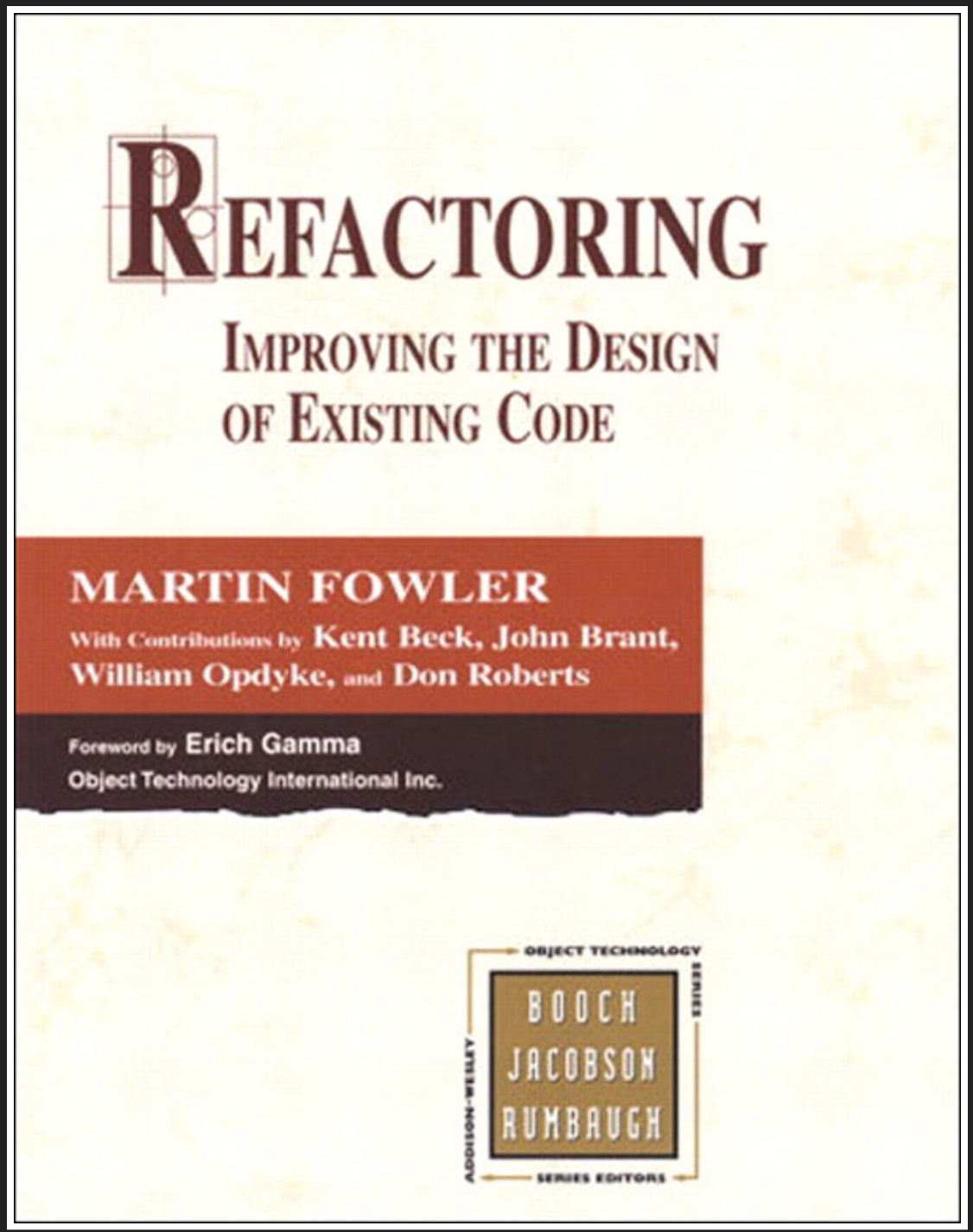
4. OO DESIGN; ERROR HANDLING - POGING 2

```
function getBookById($id) {
        if($id == NULL) throw new InvalidIdException("id cannot be null!");

        return $db->getById($id);
}
```

REFACTORING

Structuur veranderen, zonder inhoud te wijzigen!


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY


OEFENINGEN DEEL 2

DIERENTUIN - STORY 1

- maak een **Dierentuin** class
- kan verschillende **Dieren** (`class`) **ontvangen** (`function`)
- elk dier heeft een **grootte** en **naam**:
  Neushoorn (40), Giraf(25), Poema (10)
- elke dierentuin heeft x *beschikbareRuimte* afhankelijk van *grootte* van het dier
  wat doe je bij het ontvangen van een te groot dier?

```
class Dierentuin {
        public function bezoek() {
                // return [] of dieren
        }
        public function ontvangDier($dier) {
                // ??
        }
}
```

DIERENTUIN - STORY 2

- **voeder** functie op dierentuin
  return `TRUE` of `FALSE` indien genoeg voedsel voor elk ontvangen dier in de tuin
- voedsel heeft een **voedingswaarde**
  Elk dier eet even veel waarde als zijn grootte
- Verzin voedsel implementaties om alle edge cases te kunnen testen!

```
class Dierentuin {
        public function voeder($voedsel) {
                return TRUE;    // ??
        }
}
```

DIERENTUIN - DISCUSSIE

- Polymorfisme niet vergeten?
- Genoeg testen geschreven voor elk geval?
- Exceptions gebruikt ipv return code?

HET PARETO PRINCIPE

80% van het werk, met
20% van de inspanning

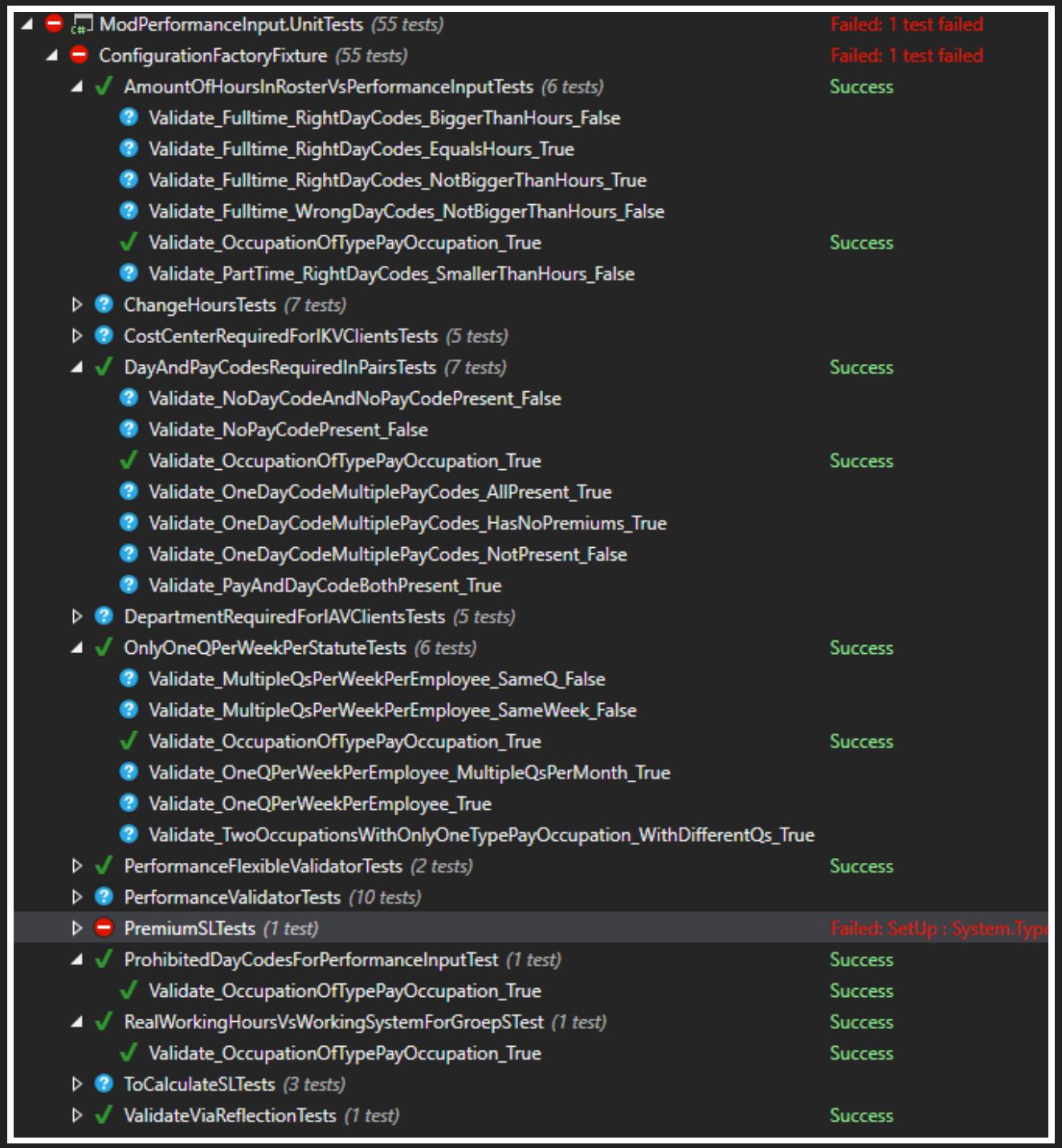
You cannot write perfect software.
Therefore, do not waste energy trying;
be pragmatic.

DRY: DON'T REPEAT YOURSELF

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

---- DAG 2 ----

CLEAN CODE DEEL 3: UNIT TESTING


ModPerformanceInput.UnitTests (55 tests) Failed: 1 test failed
ConfigurationFactoryFixture (55 tests) Failed: 1 test failed
AmountOfHoursInRosterVsPerformanceInputTests (6 tests) Success
Validate_Fulltime_RightDayCodes_BiggerThanHours_False
Validate_Fulltime_RightDayCodes_EqualsHours_True
Validate_Fulltime_RightDayCodes_NotBiggerThanHours_True
Validate_Fulltime_WrongDayCodes_NotBiggerThanHours_False
Validate_OccupationOfTypePayOccupation_True Success
Validate_PartTime_RightDayCodes_SmallerThanHours_False
ChangeHoursTests (7 tests)
CostCenterRequiredForIKVClientsTests (5 tests)
DayAndPayCodesRequiredInPairsTests (7 tests) Success
Validate_NoDayCodeAndNoPayCodePresent_False
Validate_NoPayCodePresent_False
Validate_OccupationOfTypePayOccupation_True Success
Validate_OneDayCodeMultiplePayCodes_AllPresent_True
Validate_OneDayCodeMultiplePayCodes_HasNoPremiums_True
Validate_OneDayCodeMultiplePayCodes_NotPresent_False
Validate_PayAndDayCodeBothPresent_True
DepartmentRequiredForIAVClientsTests (5 tests)
OnlyOneQPerWeekPerStatuteTests (6 tests) Success
Validate_MultipleQsPerWeekPerEmployee_SameQ_False
Validate_MultipleQsPerWeekPerEmployee_SameWeek_False
Validate_OccupationOfTypePayOccupation_True Success
Validate_OneQPerWeekPerEmployee_MultipleQsPerMonth_True
Validate_OneQPerWeekPerEmployee_True
Validate_TwoOccupationsWithOnlyOneTypePayOccupation_WithDifferentQs_True
PerformanceFlexibleValidatorTests (2 tests) Success
PerformanceValidatorTests (10 tests)
PremiumSLTests (1 test) Failed: SetUp : System.Type
ProhibitedDayCodesForPerformanceInputTest (1 test) Success
Validate_OccupationOfTypePayOccupation_True Success
RealWorkingHoursVsWorkingSystemForGroepSTest (1 test) Success
Validate_OccupationOfTypePayOccupation_True Success
ToCalculateSLTests (3 tests)
ValidateViaReflectionTests (1 test) Success

WAAROM UNIT TESTEN?

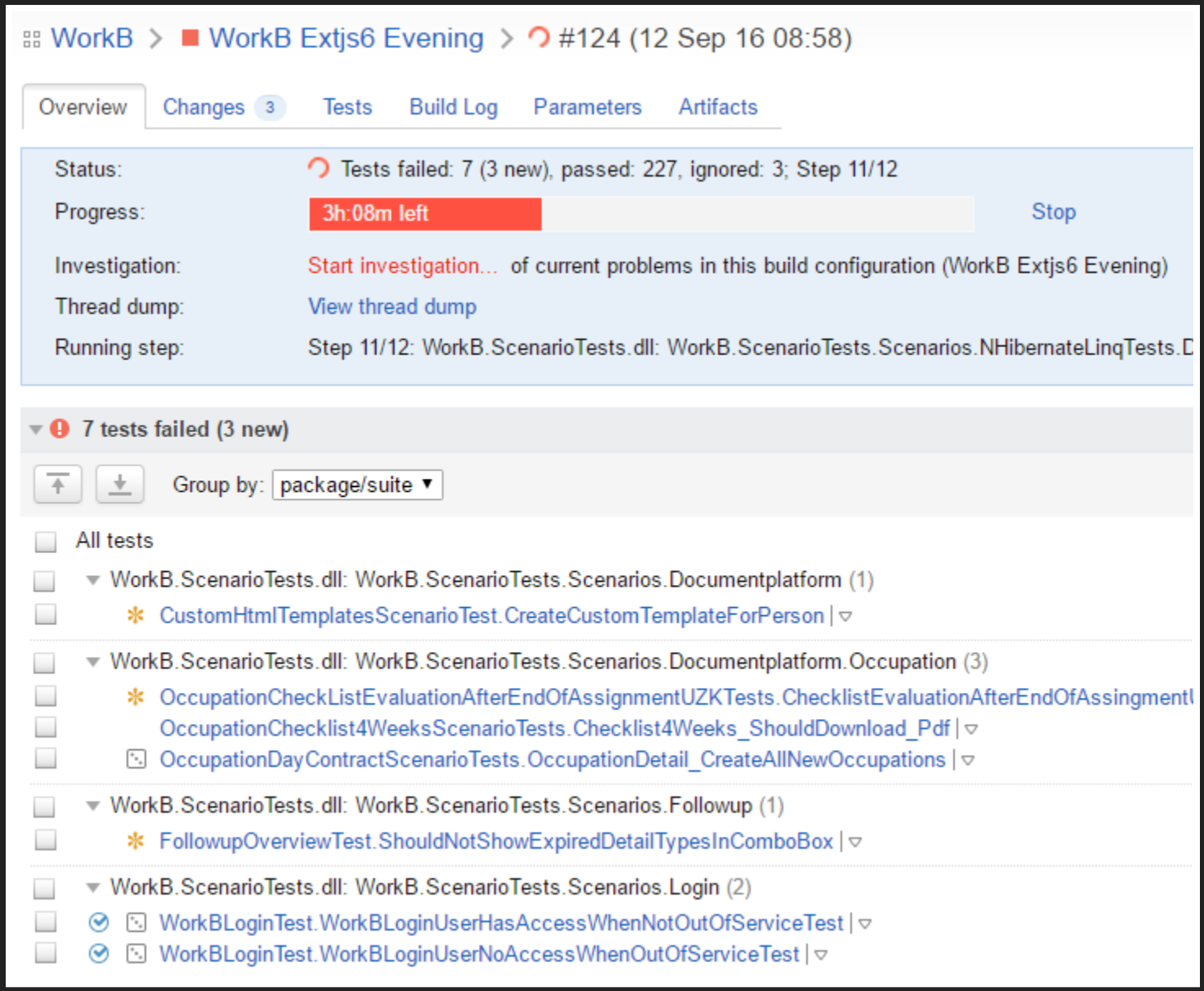
WAAROM UNIT TESTEN?

- **feedback!**
- Denk in code vanuit API standpunt
- Makkelijk voor pair om te volgen
- Alle mogelijke paden gedekt

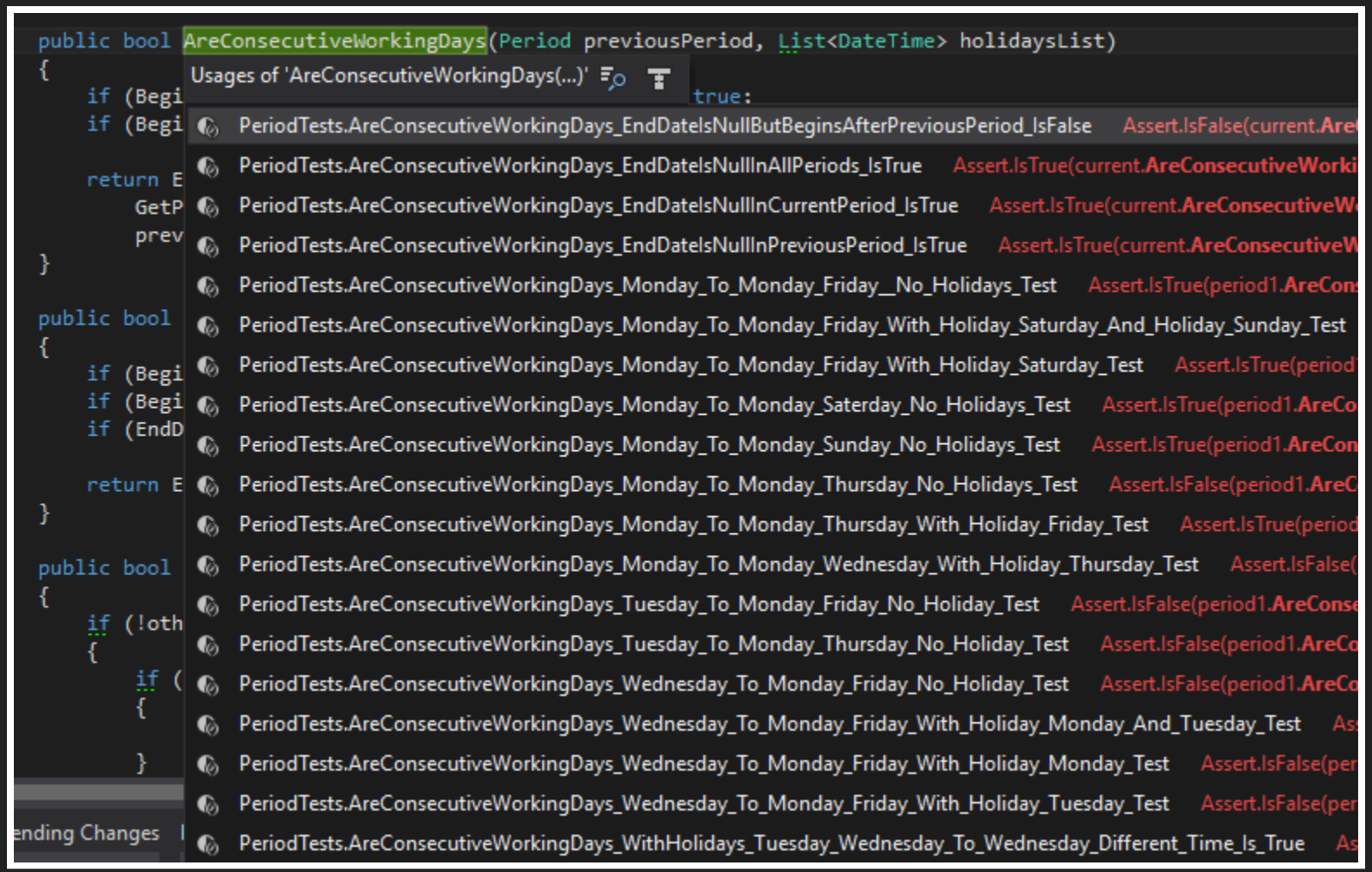
EIGENSCHAPPEN VAN EEN GOEDE TEST

- Ontdek sneller bugs
- Leesbaar
- Geautomatiseerd
- Snel & gefocuset
- Herhaalbaar
- Volgorde onafhankelijk
- Productie code makkelijker wijzigbaar

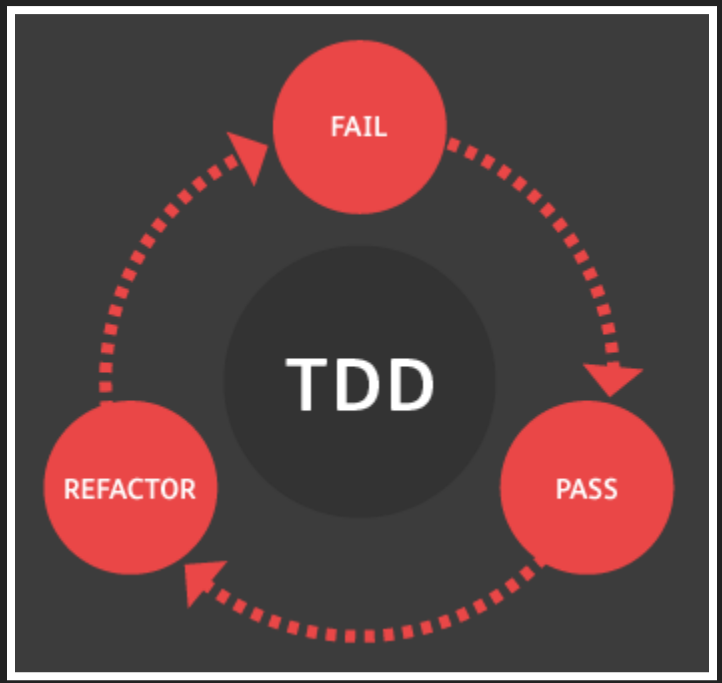
EIGENSCHAPPEN VAN EEN GOEDE TEST GEAUTOMATISEERD



EIGENSCHAPPEN VAN EEN GOEDE TEST
"LIVING DOCUMENTATION"

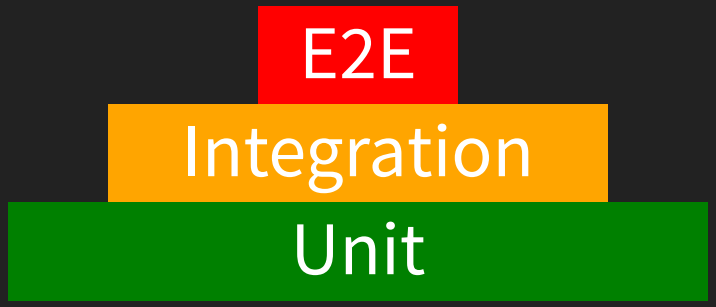


TEST DRIVEN DEVELOPMENT (TDD)




TDD: "BUGFIX" EDGE CASE
live demo

SOORTEN TESTEN



UNIT TESTING

- Onafhankelijk van externen (db, webservice, ...)
- Snel!
- Véél testen
- Test normaal pad & limieten
- "actieve vijand van de code"

INTEGRATION TESTING

- Test geïntegreerd met externen (db, webservice, ...)
- Test integratie **twee verschillende lagen**
- Trager dan unit tests
- Minder test cases

END TO END TESTING

- Test hele applicatie!
- niet alle limieten
- traag, moelijker onderhoudbaar
- Test integratie **alle** lagen

-- oefeningen dag 2 (groter) --

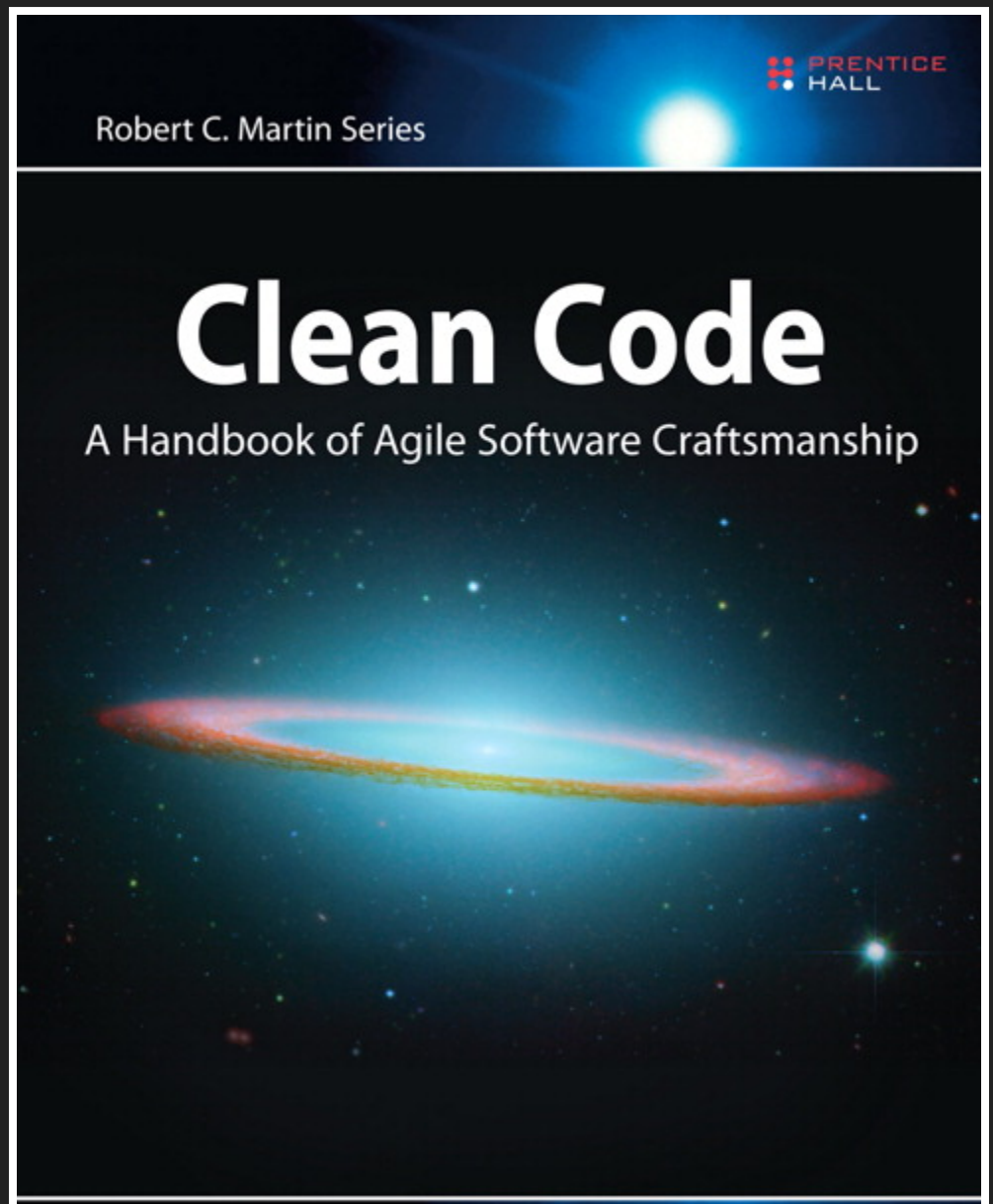
-- oefening uitleg

-- oefening oplossing; gezamelijke discussie


prato
http://www.prato.be
vacatures@prato.be

RESOURCES

- https://github.com/wgroeneveld/cleancode-course
- The Essence of clean code
- The Essence of pragmatic programmer
- A pragmatic programmer Quick reference card




Foreword by James O. Coplien
Robert C. Martin