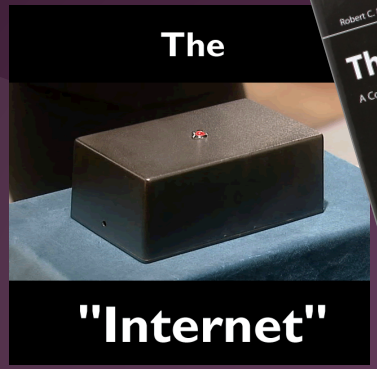
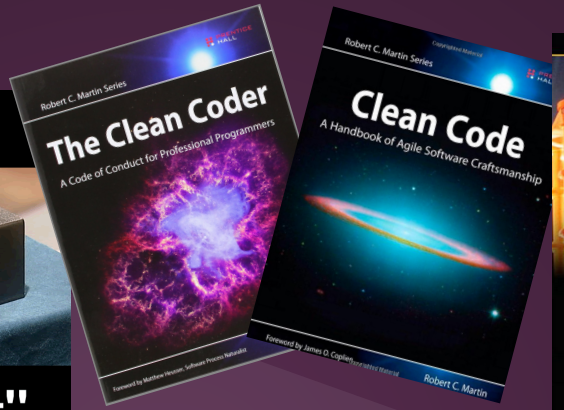
Addressing Technical Debt

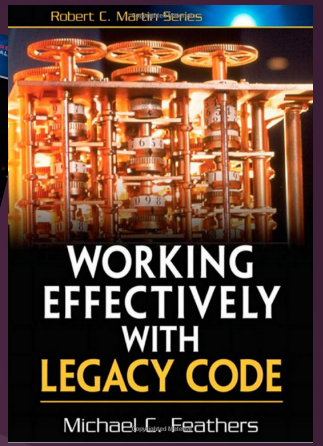
@kristinabalaam
medium.com/@kristinabalaam

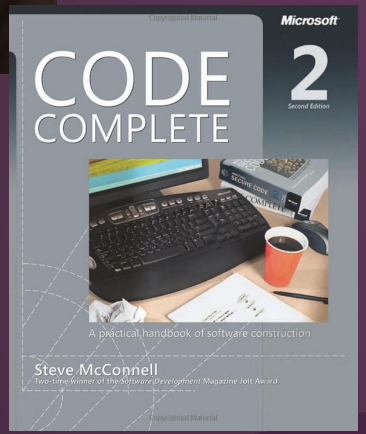
github.com/kristinaelise

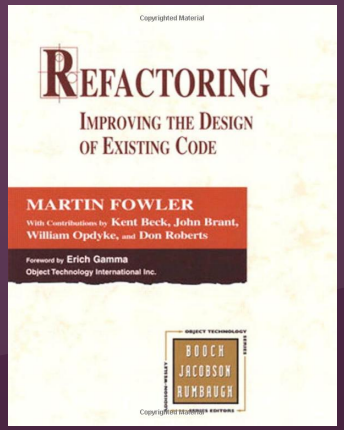
Steve McConnell











Martin Fowler

What is “technical debt?”

The additional work resulting from implementing code that is easy to implement in the short run, but that is poorly designed.

What is “technical debt”?

Ward Cunningham, 1992:

> *"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite...The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."*

(Techopedia.com, 2014)

What is “technical debt”?

Intentional vs Unintentional
vs Short Term vs Long Term

(Construx.com, 2014)

(Awesome blog post by Steve McConnell - http://www.construx.com/10x_Software_Development/Technical_Debt/ who wrote ‘Code Complete’.)

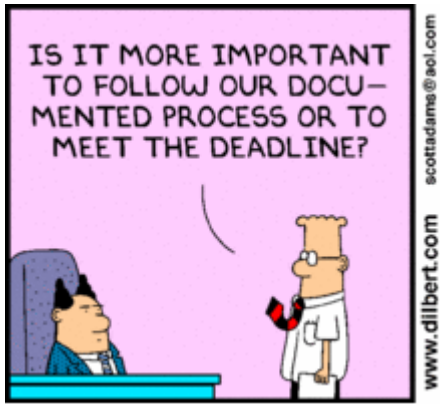
What is “technical debt”?

Def’n: Refactoring: *“…the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure.”*

(Fowler and Beck, 1999)

What is “technical debt”?

Technical debt is (usually) a necessary evil.



And it’s not **always** a bad thing.

What is “technical debt”?

Having technical debt is not the end of the world…as long as you manage it.

Don’t stress about feeling like you have to refactor all the time. We’ll see why.

Why you should care…as a manager

"If the [technical] debt grows large enough, eventually the company will spend more on servicing its debt than it invests in increasing the value of its other assets." (Construx.com, 2014)

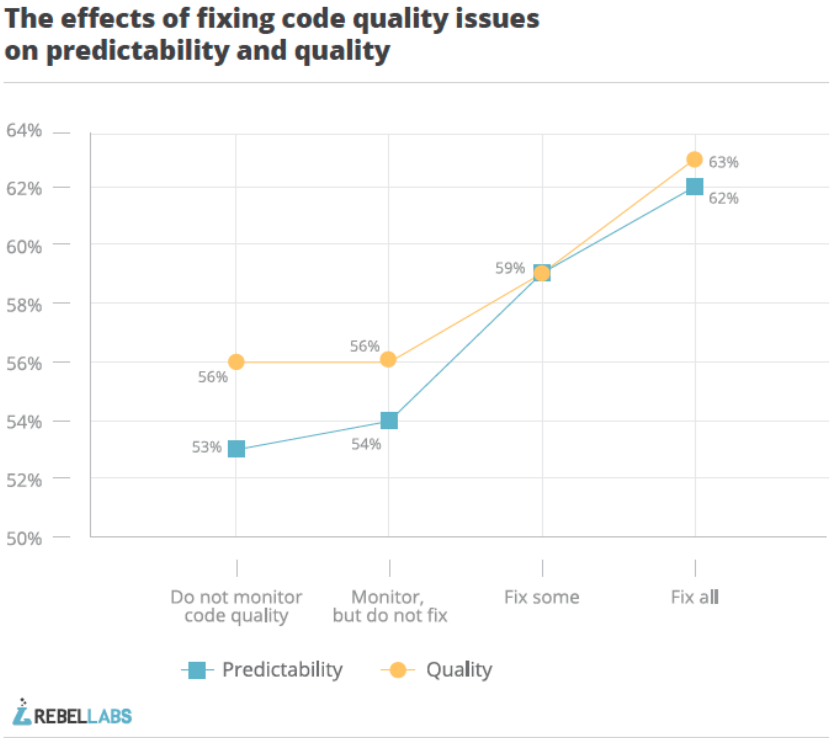
Why you should care…as a manager

“Developer Productivity Report” from Rebel Labs:

Two metrics:

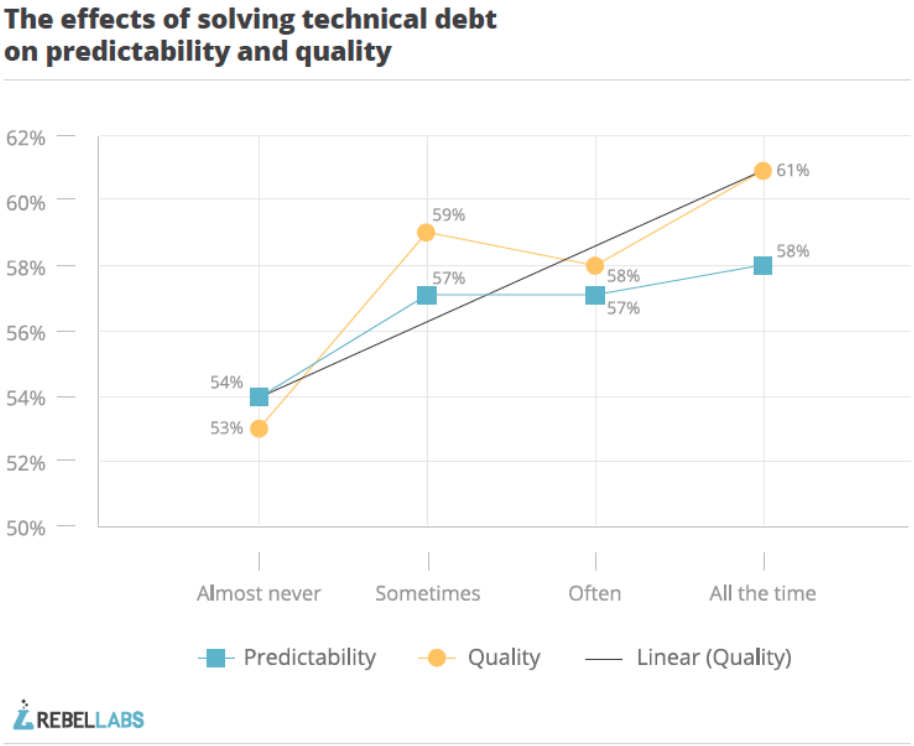
1. **Quality of Software:** frequency of critical/blocker bugs
2. **Predictability of Delivery:** delays in release, completing what was planned, scope creep.

Why you should care...as a manager



Fixing code quality issues has a pretty significant impact on both metrics.

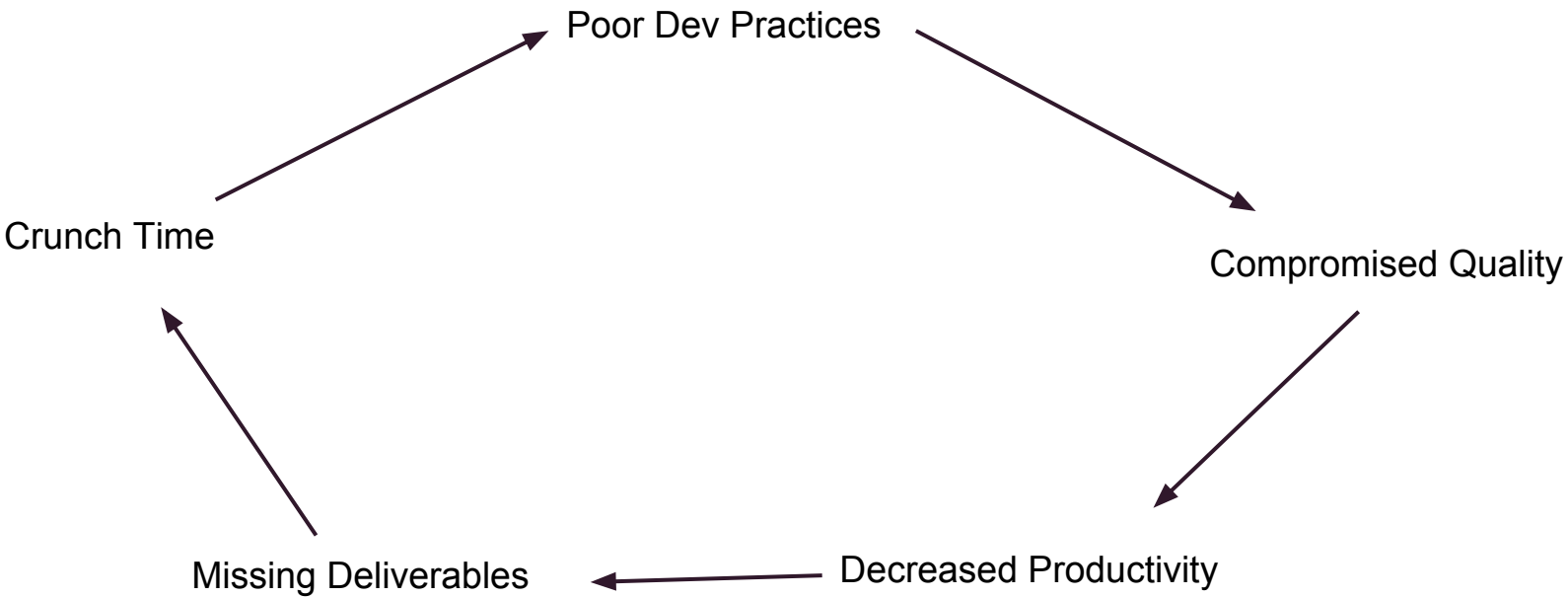
Why you should care…as a manager



"Repaying" SOME technical debt is almost as effective as tackling it all.

Just…don't do nothing.

Why you should care...as a manager



Why you should care…as a dev

Refactoring improves the design of your software.

Why you should care…as a dev

Addressing technical debt makes your software easier to understand.

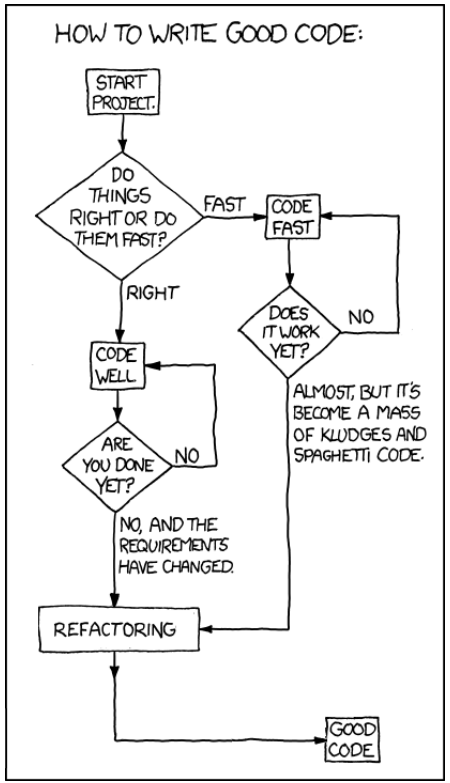
Why you should care…as a dev

Refactoring helps you find bugs.

Why you should care...as a dev

Practicing consistent refactoring helps you code faster in the long run.

Why you should care...as a dev



It makes you a better developer.

Recognizing Technical Debt

Outside the code

- simple features take far longer to complete than expected
- the project is riddled with bugs
- your team is consistently failing to meet deadlines - and it's NOT because of scope creep

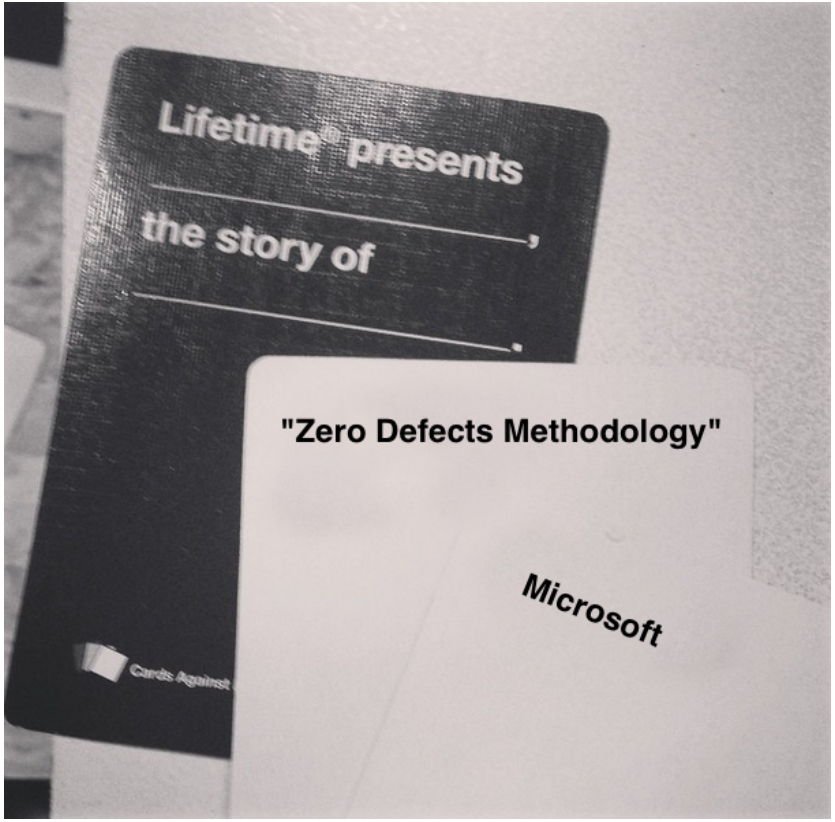
Recognizing Technical Debt

Inside the code

- duplicated code (duplicated code = duplicated bugs)
- super long methods, large classes
- long parameter lists
- hacked things (shotgun surgery)

http://xkcd.com/292/

Preventing (extensive) Technical Debt



Fix bugs before writing new code

"A best practice or principle of a successful team, it means the project team commits to do work at the highest quality possible at the time it is being done, and each team member is individually responsible for helping achieve the desired level of quality. The zero-defect mindset does not mean that the deployed solution must be "perfect" with literally no defects; rather, it establishes perfection as a consistent goal for the team to strive for." [Microsoft Solution Accelerator glossary]

Preventing (extensive) Technical Debt

Practice looking at small pieces of your code & self-critiquing.

Preventing (extensive) Technical Debt

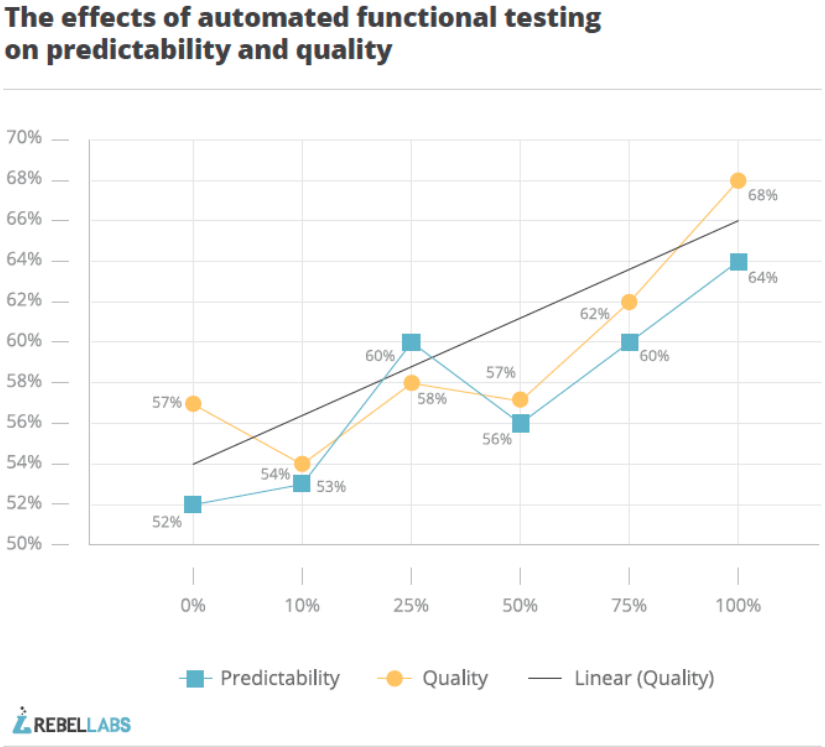
~~Don't~~ Try to not be lazy.

"Your code is your house. You have to live in it."

Preventing (extensive) Technical Debt

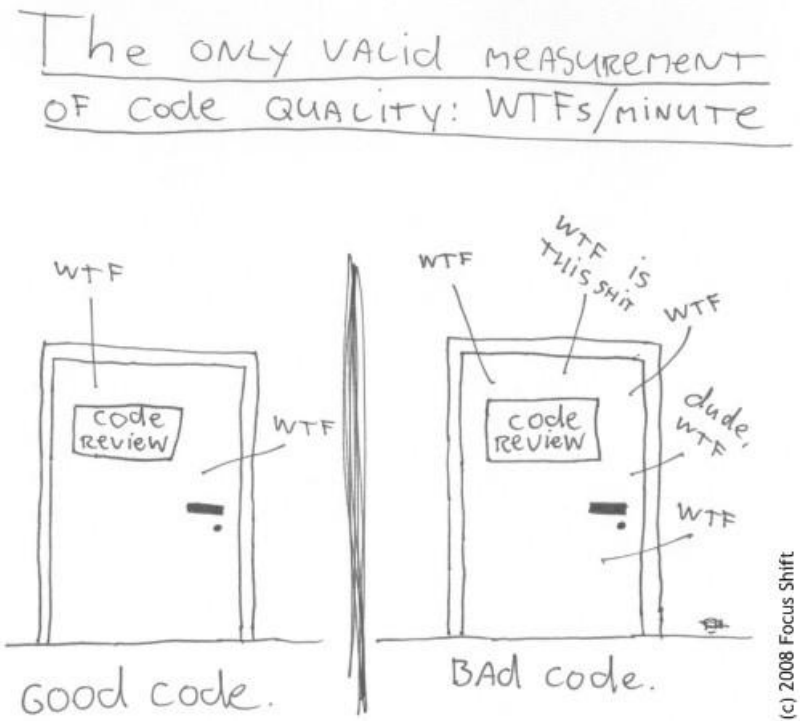
Test Your Sh!t!

Preventing (extensive) Technical Debt



Automated. Unit Testing.
Regression Testing.
Just test it.

Preventing (extensive) Technical Debt



EMPHASIZE CODE QUALITY!

Quality. Quality. Quality.
It does matter.

Handling Technical Debt

Don’t Refactor ALL The Things!

http://hyperboleandahalf.blogspot.ca/2010/06/this-is-why-ill-never-be-adult.html

Handling Technical Debt

I. Refactor in small stages

II. Refactor & optimize when the time is available to do this WITHOUT impacting the time you'll take to reach a deadline in the future

Handling Technical Debt

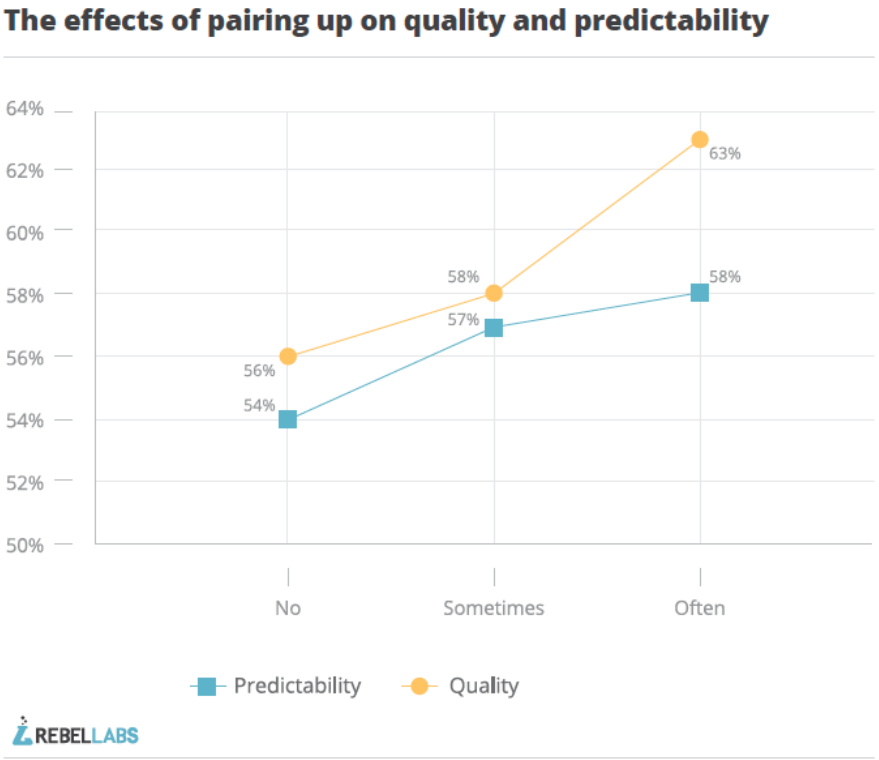
III. Refactor when you fix bugs.

IV. Refactor when you add a function.

V. Refactor as you code review

(Fowler and Beck, 1999)

Handling Technical Debt



Try Pair Refactoring!

Handling Technical Debt

Set Goals.

Figure out what you want to/need to fix before diving in.

Don’t be afraid to backtrack if you’ve lost focus. Go back to the original goal.

TOOLS!

Basically: software analytics.

Pros? See where bugs are occurring, application performance. Even customer experience.

Con? $$$

TOOLS!

PHPUnit A(n awesome) testing framework

Code

src/Money.php

```php
<?php
class Money
{
    private $amount;

    public function __construct($amount)
    {
        $this->amount = $amount;
    }

    public function getAmount()
    {
        return $this->amount;
    }

    public function negate()
    {
        return new Money(-1 * $this->amount);
    }

    // ...
}
```

Test Code

tests/MoneyTest.php

```php
<?php
class MoneyTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testCanBeNegated()
    {
        // Arrange
        $a = new Money(1);

        // Act
        $b = $a->negate();

        // Assert
        $this->assertEquals(-1, $b->getAmount());
    }

    // ...
}
```

Loads of resources, documentation including:

https://phpunit.de/presentations.html (“Be Nice to Future You”)

TOOLS!

PHP_Codesniffer

- detects violations in set coding standards for PHP/Javascript/CSS
- can also prevent some semantic errors

```
composer global require
'squizlabs/php_codesniffer=*'
```

OR

http://www.webcodesniffer.net/
Browser/Server-based.

TOOLS!

PHPDCD - “PHP Dead Code Detector”

Finds all the methods, functions in a project that haven’t been called at least once

https://github.com/sebastianbergmann/phpdcd

PHPCPD - “PHP Copy-Paste Detector”

https://github.com/sebastianbergmann/phpcpd

PHPMD - “PHP Mess Detector”

Scans source code for “unused parameters/properties/methods, suboptimal code, overcomplicated expressions, possible bugs”. A PMD (Java source code analyzer) for PHP.

http://phpmd.org/

TOOLS!

DOCUMENT!

Your future self will thank you.

(http://phpdox.de/ is a handy & quick little option that generates documentation for PHP Projects)

Thank you!
Questions?

Works Cited

Techopedia.com, (2014). *What is Technical Debt? - Definition from Techopedia*. [online] Available at: http://www.techopedia.com/definition/27913/technical-debt [Accessed 20 Nov. 2014].

Fowler, M. and Beck, K. (1999). *Refactoring*. Reading, MA: Addison-Wesley.

Rebel Labs, (2013). *Developer Productivity Report*. ZeroTurnAround.com. http://zeroturnaround.com/rebellabs/download/?token=80eb432c1d5edfc886f91ad2169c139196a99127

Construx.com, (2014). *Technical Debt-10x Software Development | Construx*. [online] Available at: http://www.construx.com/10x_Software_Development/Technical_Debt/ [Accessed 20 Nov. 2014].