iOS Testing Fundamentals

Max Humber
November 15, 2022

Why Test?

Why software testing is important

Few can argue against the need for quality control when developing software. Late delivery or software defects can damage a brand's reputation — leading to frustrated and lost customers. In extreme cases, a bug or defect can degrade interconnected systems or cause serious malfunctions.

Consider Nissan having to recall over 1 million cars due to a software defect in the airbag sensor detectors. Or a software bug that caused the failure of a USD 1.2 billion military satellite launch. [2] The numbers speak for themselves. Software failures in the US cost the economy USD 1.1 trillion in assets in 2016. What's more, they impacted 4.4 billion customers. [3]

Though testing itself costs money, companies can save millions per year in development and support if they have a good testing technique and QA processes in place. Early software testing uncovers problems before a product goes to market. The sooner development teams receive test feedback, the sooner they can address issues such as:

- Architectural flaws
- Poor design decisions
- Invalid or incorrect functionality
- Security vulnerabilities
- Scalability issues

When development leaves ample room for testing, it improves software reliability and high-quality applications are delivered with few errors. A system that meets or even exceeds customer expectations leads to potentially more sales and greater market share.

https://www.ibm.com/topics/software-testing

Why software testing is important

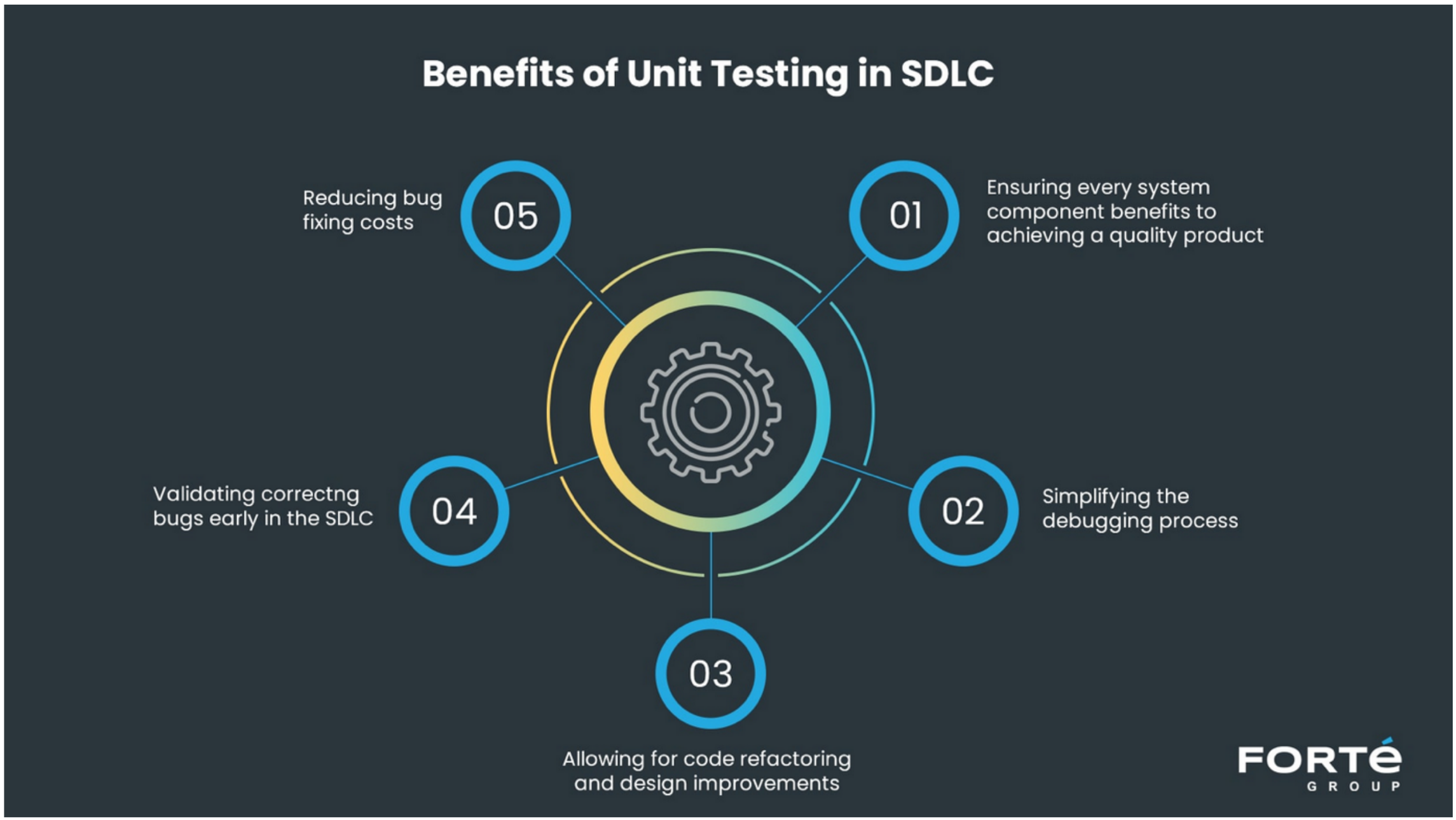
Few[illegible]e against the need for [illegible]when developing software. Late de[illegible]ware defects can[illegible]'s reputation — leading to frustrated and lost[illegible]extreme c[illegible]fect can degrade interconnected systems or cause[illegible]nctio[illegible]

Conside[illegible]r 1 million cars due to a software defect in the airbag sensor de[illegible]ug that caused the failure of a USD 1.2 billion military satellite lau[illegible]s speak for themselves. Software failures in the US cost the economy [illegible]ts in 2016. What's more, they impacted 4.4 billion custom[illegible]

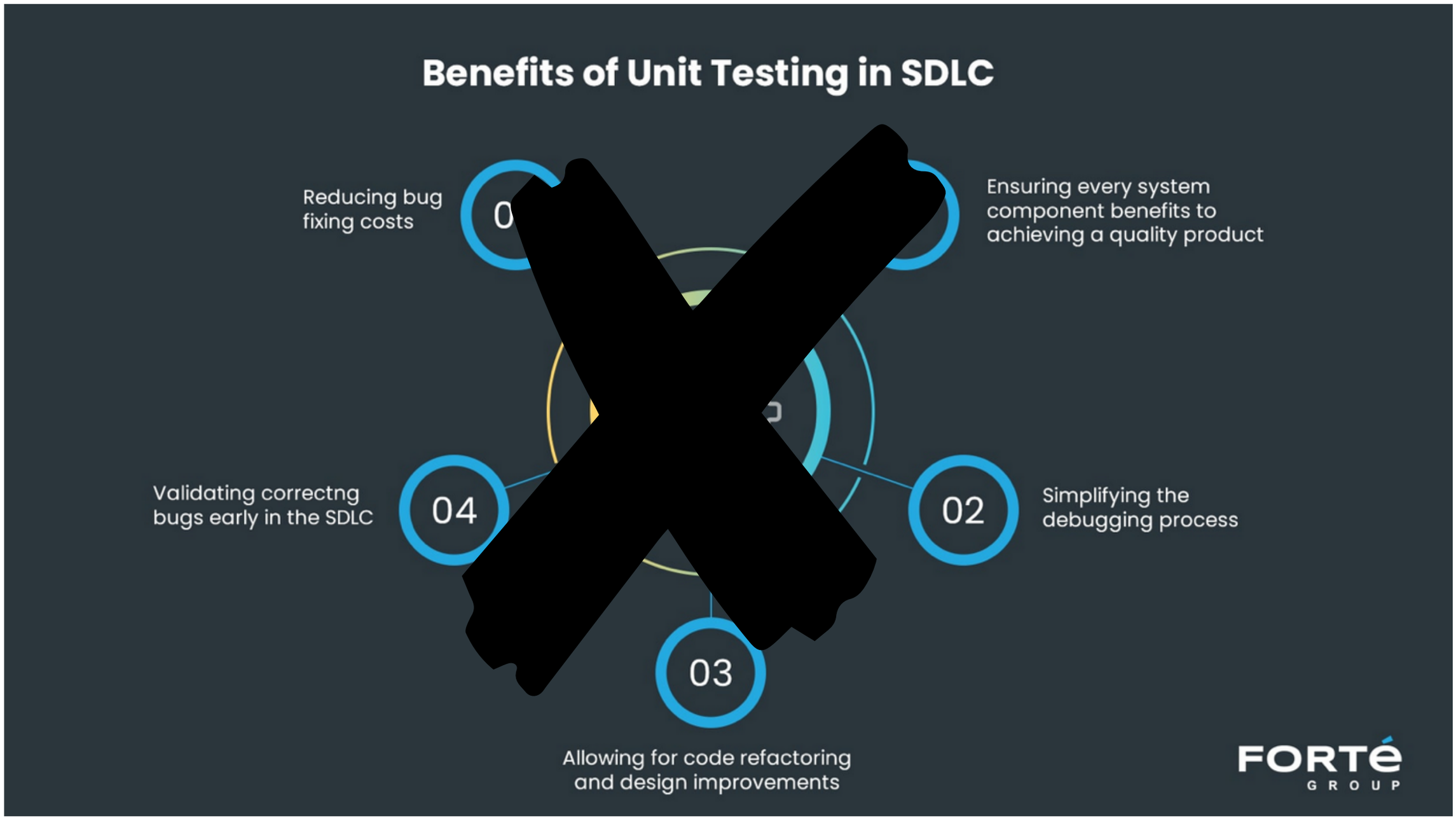
T[illegible]anies can save millions per year in development [illegible]ey ha[illegible]hnique and QA processes in place. Early [illegible]ng uncover[illegible]re a product goes to market. The sooner [illegible]nt teams receive[illegible]back, the sooner they can address issues such as:

- [illegible]chitectural flaws
- Poor design decisions
- Invalid or incorrect functionality
- Security vulnerabilities
- Scalability issues

When development leaves ample room for testing, it improves software reliability and high-quality applications are delivered with few errors. A system that meets or even exceeds customer expectations leads to potentially more sales and greater market share.



https://fortegrp.com/the-importance-of-unit-testing

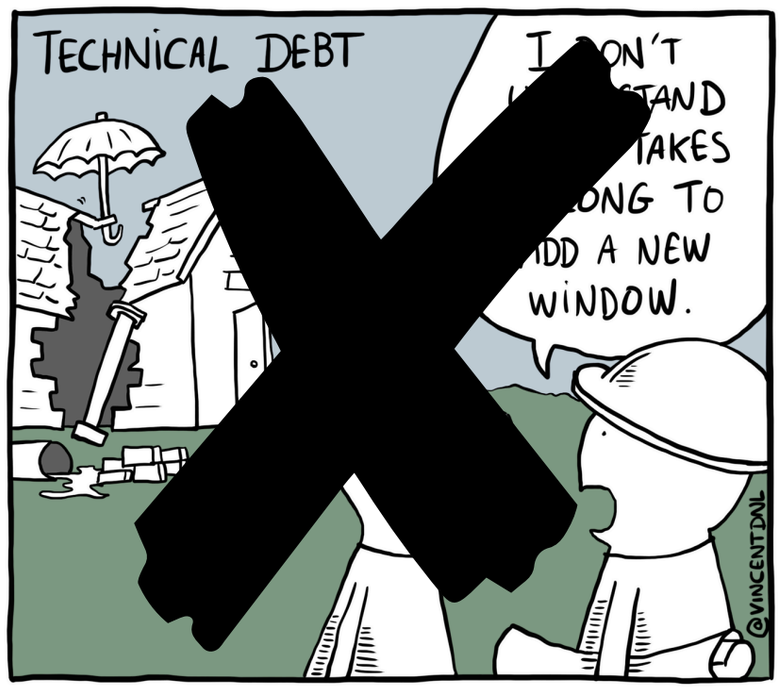


https://fortegrp.com/the-importance-of-unit-testing

Why Test?

Speed.

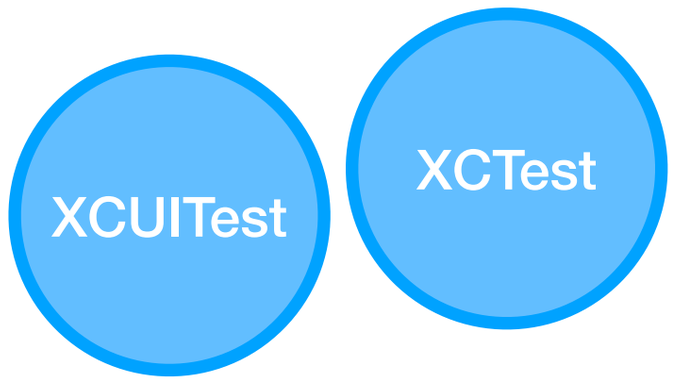

TECHNICAL DEBT
I DON'T UNDERSTAND WHY IT TAKES SO LONG TO ADD A NEW WINDOW.
@VINCENTDAL

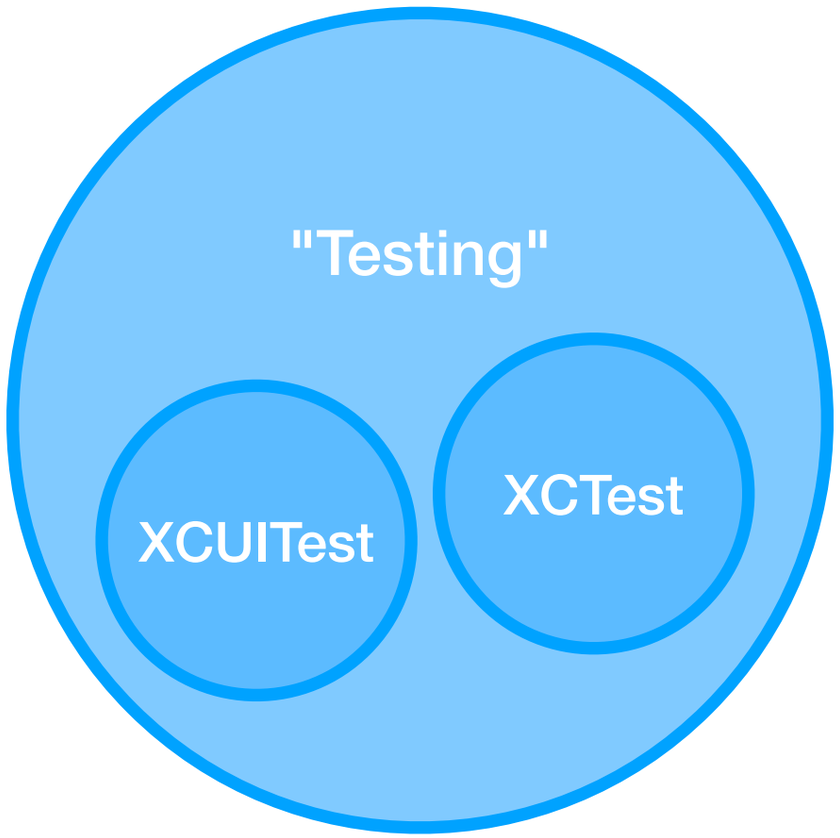

TECHNICAL DEBT
I DON'T
STAND
TAKES
ONG TO
DD A NEW
WINDOW.
@VINCENTDNL

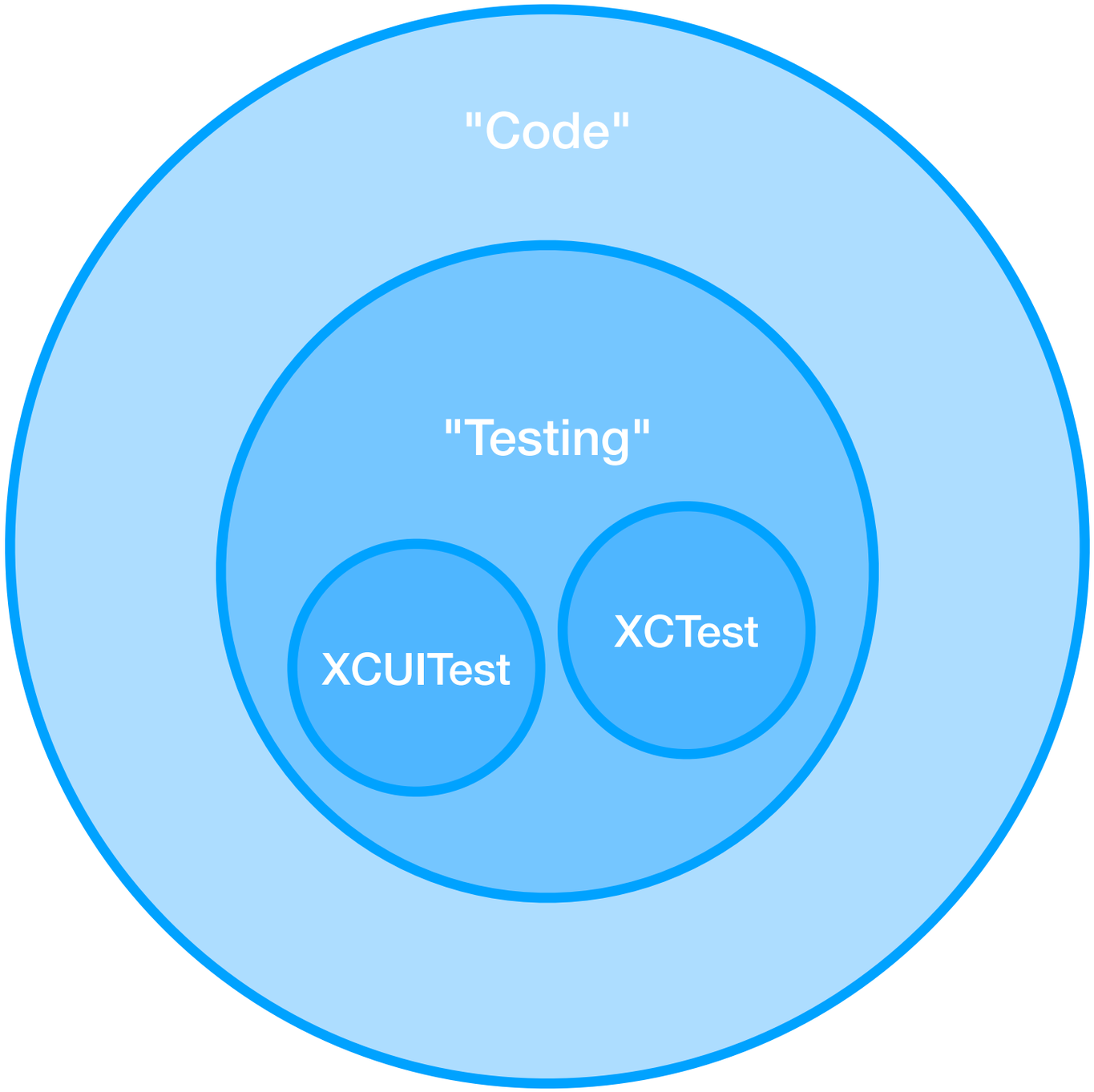

MOL
VOLVO
TŰZOLTÓSÁG

"Tech sludge"

Topics


XCUITest
XCTest


"Testing"
XCUITest
XCTest


"Code"
"Testing"
XCUITest
XCTest

Code is read much more often than it is written, so plan accordingly

Raymond Chen

April 6th, 2007 | 0 | 0

Design for readability. Even if you don't intend anybody else to read your code, there's still a very good chance that somebody will have to stare at your code and figure out what it does: That person is probably going to be you, twelve months from now. When I advised against the use of BOOL function parameters, one commenter pointed out that Intellisense shows you what the parameters are. Sure, that's a great help when you're writing the code, but you write the code only once. You read it a lot. An anonymous commenter pointed out that hovering over the offending function in the IDE shows the function declaration. While that may be true, it's still not a great solution. First, it means that you have to read code with your hand on the mouse. Second, it means that you can't skim code because you're going to hit a `CreateEvent` and then have to wait a little while for the tooltip to appear and then read and parse the tooltip and match the parameters up to what you have on the screen. This throws off your rhythm. Imagine if you had to read code that had gone through a ROT-13 filter. Sure, the IDE might help you by decoding the text under the cursor if you hover over it, but even instant help isn't fast enough.

And finally, another commenter pointed out that this doesn't help you if you're reading code anywhere outside your IDE. It might be in a magazine, in a printout, in a bug report, on an overhead projector, on a web page, in a Usenet message, or in an email message. Good luck getting Intellisense to help you out there.

https://devblogs.microsoft.com/oldnewthing/20070406-00/?p=27343

Software As Liability

Code is a liability.

What the code does for you is an asset.

Producing more code is not always a gain. Code is expensive to test and maintain, so if the same work can be done with less code that is a plus. Don't comment out dead code, just delete it. Commented out code goes stale and useless very fast, so you may as well delete it sooner rather than later, to loose the clutter. Keep good backups to make it easier.

Perhaps only LegacyCode is a liability. Once the code has been factored so that it can be easily maintained and reused, then it becomes an asset.

https://wiki.c2.com/?SoftwareAsLiability

Goals

1 Legible

1 Legible

2 Maintainable

1 Legible
2 Maintainable
3 Replaceable

Testing will help!

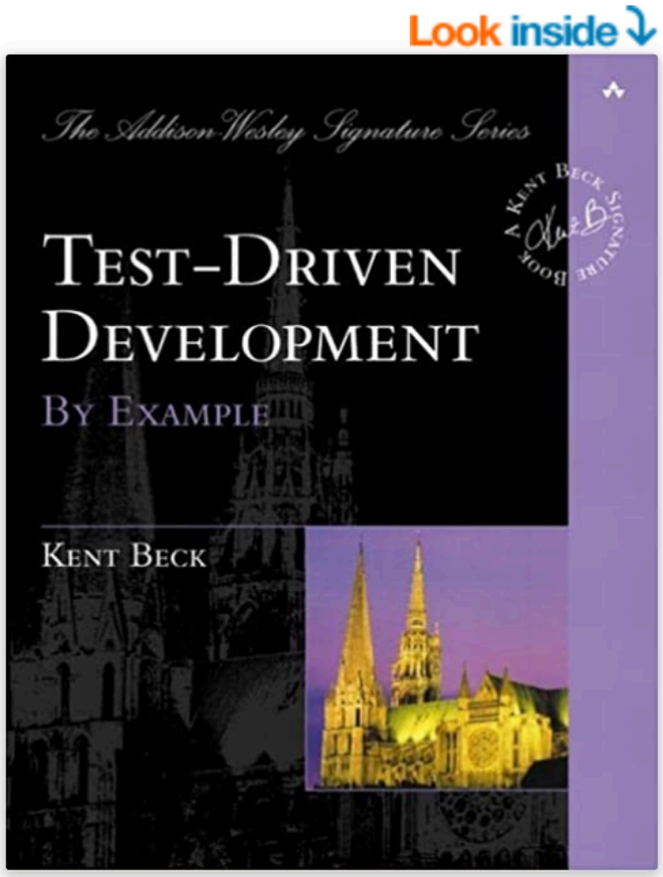
1 Legible
2 Maintainable
3 Replaceable

Testing will help!

1 Legible
2 Maintainable
3 Replaceable

So that you can go **faster**

"TDD"



See this image

Follow the Author

Kent Beck

Follow

Test Driven Development: By Example 1st Edition

by Kent Beck (Author)

4.5 out of 5 stars 458 ratings

See all formats and editions

Kindle	**Paperback**
$24.99	**$36.00 - $44.99**
Read with Our **Free App**	17 Used from $24.94 10 New from $37.03

Quite simply, test-driven development is meant to eliminate fear in application development. While some fear is healthy (often viewed as a conscience that tells programmers to "be careful!"), the author believes that byproducts of fear include tentative, grumpy, and uncommunicative programmers who are unable to absorb constructive criticism. When programming teams buy into TDD, they immediately see positive results. They eliminate the fear involved in their jobs, and are better equipped to tackle the difficult challenges that face them. TDD eliminates tentative traits, it teaches programmers to communicate, and it encourages team members to seek out criticism However, even the author admits that grumpiness must be worked out individually! In short, the premise behind TDD is that code should be continually tested and refactored. Kent Beck teaches programmers by example, so they can painlessly and dramatically increase the quality of their work.

Read less

ISBN-10	ISBN-13	Edition	Publisher
9780321146533	**978-0321146533**	**1st**	**Addison-Wesley Professional**

▲ I have complicated feelings about TDD (buttondown.email/hillelwayne)

323 points by jwdunne 74 days ago | hide | past | favorite | 441 comments

▲ sedachv 73 days ago | prev | next [–]

TDD use would be a lot different if people actually bothered to read the entirety of Kent Beck's _Test Driven Development: By Example_. It's a lot to ask, because it is such a terribly written book, but there is one particular sentence where Beck gives it away:

> This has happened to me several times while writing this book. I would get the code a bit twisted. “But I have to finish the book. The children are starving, and the bill collectors are pounding on the door.”

Instead of realizing that Kent Beck stretched out an article-sized idea into an entire book, because he makes his money writing vague books on vague "methodology" that are really advertising brochures for his corporate training seminars, people actually took the thing seriously and legitimately believed that you (yes, you) should write all code that way.

So a technique that is sometimes useful for refactoring and sometimes useful for writing new code got cargo-culted into a no-exceptions-this-is-how-you-must-do-all-your-work Law by people that don't really understand what they are doing anymore or why. Don't let the TDD zealots ruin TDD.

▲ evouga 73 days ago | parent | next [–]

This seems to be the case with a lot of "methodologies" like TDD, Agile, XP, etc. as well as "XXX considered harmful"-style proscriptions.

A simple idea ("hey, I was facing a tricky problem and this new way of approaching it worked for me. Maybe it will help you too?") mutates into a blanket law ("this is the *only* way to solve *all* the problems") and then pointy-haired folks notice the trend and enshrine it into corporate policy.

But Fred Brooks was right: there are no silver bullets. Do what works best for you/your team.

▲ I have complicated feelings about TDD (buttondown.email/hillelwayne)

323 points by jwdunne 74 days ago | hide | past | favorite | 441 comments

▲ sedachv 73 days ago | prev | next [–]

TDD use would be a lot different if people actually bothered to read the entirety of Kent Beck's _Test Driven Development: By Example_. It's a lot to ask, because it is such a terribly written book, but there is one particular sentence where Beck gives it away:

> This has happened to me several times while writing this book. I would get the code a bit twisted. "But I have to finish the book. The children are starving, and the bill collectors are pounding on the door."

Instead of realizing that Kent Beck stretched out an article-sized idea into an entire book, because he makes his money writing vague books on vague "methodology" that are really advertising brochures for his corporate training seminars, people actually took the thing seriously and legitimately believed that you (yes, you) should write all code that way.

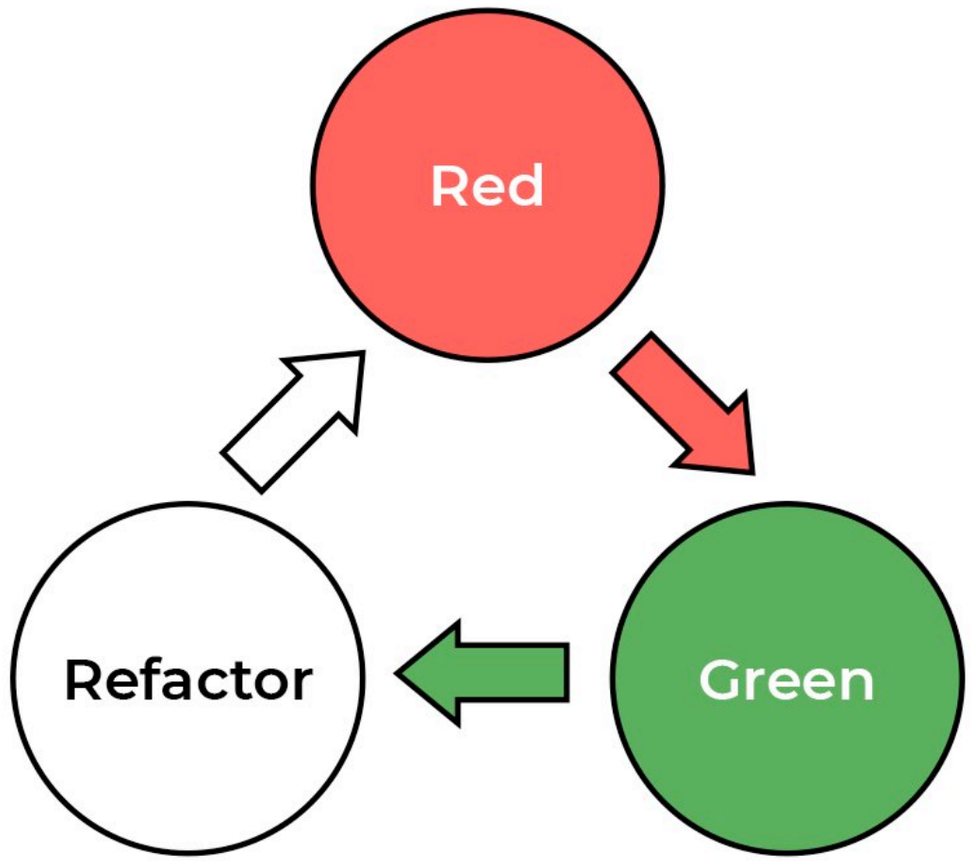
So a technique that is sometimes useful for refactoring and sometimes useful for writing new code got cargo-culted into a no-exceptions-this-is-how-you-must-do-all-your-work Law by people that don't really understand what they are doing anymore or why. Don't let the TDD zealots ruin TDD.

▲ evouga 73 days ago | parent | next [–]

This seems to be the case with a lot of "methodologies" like TDD, Agile, XP, etc. as well as "XXX considered harmful"-style proscriptions.

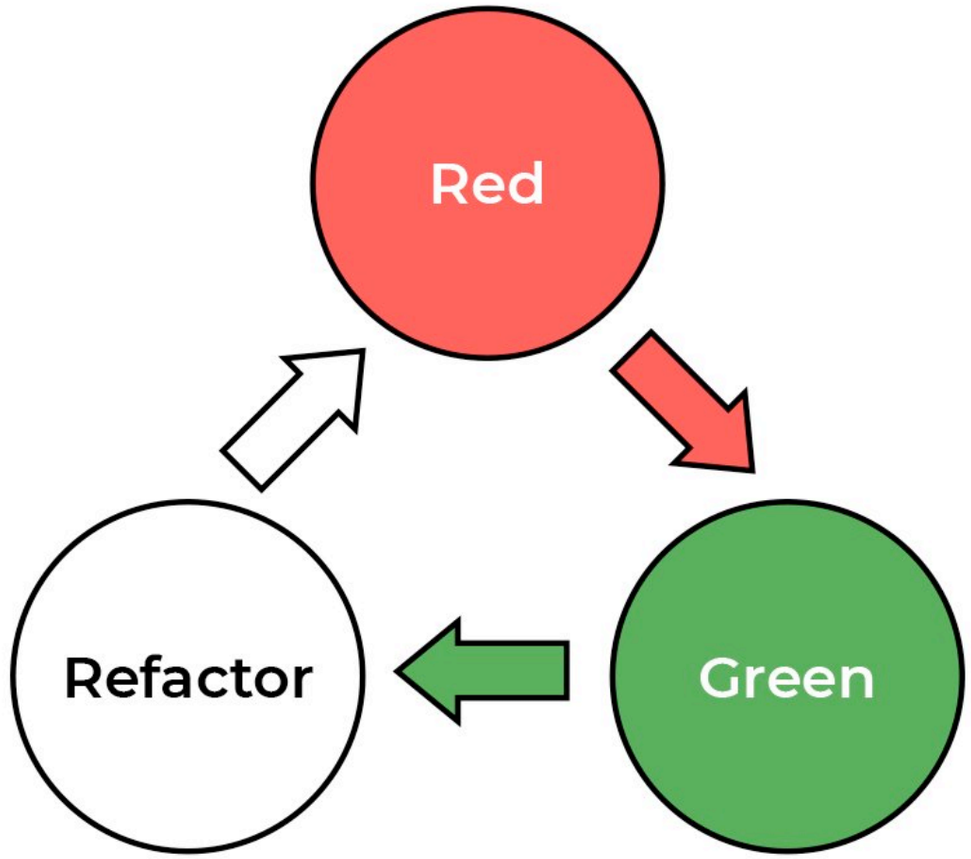
A simple idea ("hey, I was facing a tricky problem and this new way of approaching it worked for me. Maybe it will help you too?") mutates into a blanket law ("this is the *only* way to solve *all* the problems") and then pointy-haired folks notice the trend and enshrine it into corporate policy.

But Fred Brooks was right: there are no silver bullets. Do what works best for you/your team.



Red	Write a test that fails.
Green	Write code to pass the test.
Red	Clean up your code and the test.

Think about:
What you **want** to develop



Think about:
How to **improve**

Think about:
How to make tests **pass**

Red	Write a test that fails.
Green	Write code to pass the test.
Red	Clean up your code and the test.

Arrange-Act-Assert (AAA)

ARRANGE

Setup the class you want to test

ACT

Perform the action you want to test

ASSERT

Check if the result matches your expectation

Fast

Isolated

Repeatable

Self-validating

Timely

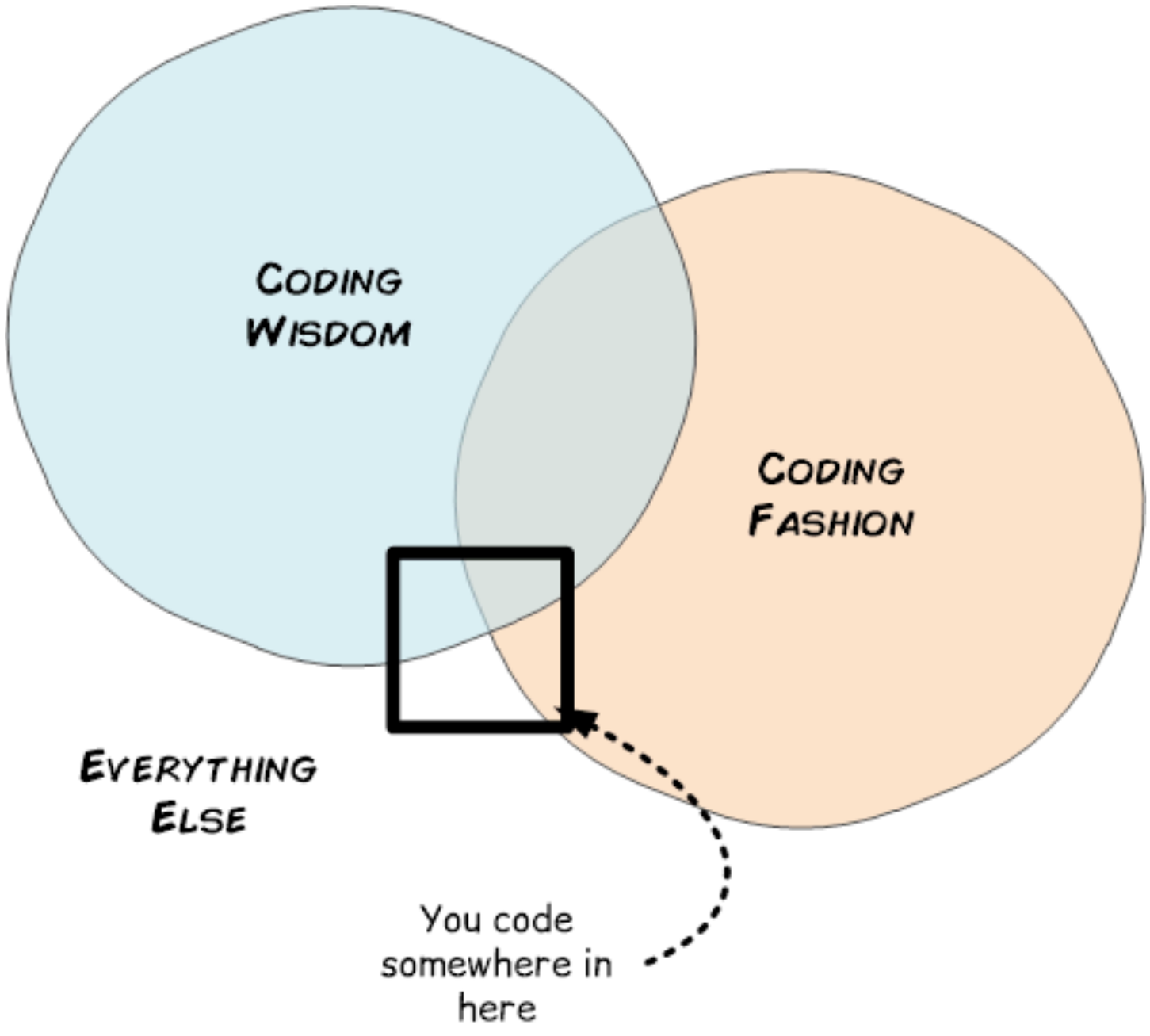
Fast (Duh!)

Isolated (Independent and AAA)

Repeatable (Deterministic)

Self-validating (Nothing manual)

Timely (New code == New Tests)

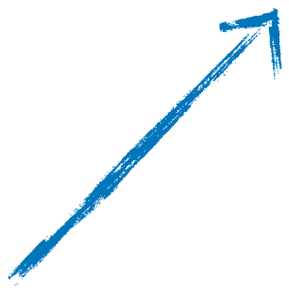

Coding
Wisdom
Coding
Fashion
Everything
Else
You code
somewhere in
here

01 XCTest Basics

02 Asynchronous Code

03 Protocols

04 Networking!



Lots to cover!

05 Test Doubles / Mocking

06 SwiftUI Basics

07 XCUITest Basics

08 Code Organization

Let's code!

Demo

Q&A

Break

That's all Folks!