


A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. They are set against a dark blue background with subtle diagonal lines.

# Rethinking TDD

Martin Cartledge



"I get paid for code that works, not for tests, so my philosophy is to test as little as possible to reach a given level of confidence."

- Kent Beck, Test Driven Development: By Example



# Another presentation on TDD?

First of all, I know!

I suck at tests! I acknowledge how important testing is, and I want to get better at it.

I have heard about all of the benefits, and not only want to understand them at a deeper level, but how to apply them in my day-to-day job.

I have a feeling many developers dread the “testing phase” of a bug fix, feature, whatever.



# Objective: what is it all for?

Reapproach the TDD methodology in a modern, *pragmatic* way.

- What is TDD?
- Benefits of TDD
- How can we use this today?



## Guiding principles: TL;DR

Automation and honesty.

*Automate* your tests before you write your code.

This keeps you (and your code) *honest* from the start.



## The process: where the rubber meets the road

- Write a failing test (red)
- Write the minimum code required to make that test pass (green)
- Refactor the code for design and maintainability (refactor)



## Red: make it fail

We need to create a way to *reliably* filter a dataset by an event type ....

Let's start with the tests shall we?



# Green: make it work

Okay, to no one's surprise, our test *did not work*.

Let's *change that*.






# Refactor: make it better

Take a moment and consider the following:

- What happens if a parameter is null, empty, or undefined?
- Can this be written in a way that is easier to understand?
- Is there a need for a default case and/or value to be returned? e.g. in the case of an unsupported eventType

Let's improve our codes maintainability, defensiveness, and design.



“Can you trust code you yeeted over from Stack Overflow? NO!

Can you trust code you copied from somewhere else in your code base? NO!

Can you trust code you just now wrote carefully by hand, yourself?  
NOOOO!

All you crazy MFs are completely overlooking the fact that software engineering exists as a discipline because you cannot EVER under any circumstances TRUST CODE.” - Steve Yegge