Test Driven Development - By Example

Uttendorfer & Wriedt

Overview

- Motivation
- Theory behind TDD
- Example Live Coding
- Example & Hands-on

“Code that’s hard to test in isolation is poorly designed”

– Test Driven Evangelists

What makes good code ?

What makes good code ?

1. Easy to change

What makes good code ?

1. Easy to change
2. Easy to understand

What makes good code ?

1. Easy to change
2. Easy to understand
3. Enjoyable to use

The Rules

The Rules

1. Write new code only if an automated test has failed
2. Eliminate duplication
3. We must design organically, with running code providing feedback between decisions
4. We must write our OWN tests, because we can’t wait 20 times per day for someone else to write a test
5. Our Development environment must provide rapid response to small changes
6. Our design must consist of many highly cohesive, loosely coupled components, just to make testing easy

The Rules

1. Red - Write a little test that doesn’t work, and perhaps doesn’t even compile at first

2. Green - Make the test work quickly, committing whatever sins necessary in the process

3. Refactor - Eliminate all of the duplication created in merely getting the test to work

The Simple Rules

1. Start simply
2. Write automated tests
3. Refactor to add design decisions one at a time

Why TDD

1. If the defect density can be reduced enough, the quality assurance can shift from reactive work to proactive work.

2. If the number of nasty surprises can be reduced enough, then project manager can estimate accuracy enough to involve real customers in daily development

3. If the topics of technical conversation can be made clear enough, then software engineers can work in minute-by-minute collaboration instead of daily or weekly collaboration

4. Again, if the defect density can be reduced enough, then we can have shippable software with new functionality every day, leading to new business relationships with customers

Kent Beck, Test Driven Development - By Example