Test Driven Develop

Overview

TDD stands for "test-driven development".

Specifically, it means to write your tests before you write your code.

Pros

People commonly think of tests as being written in order to catch bugs. This is true.

However the *way* that tests catch bugs is a bit counter-intuitive. Many people imagine that the tests are run and *then* the bugs are caught.

What actually happens more often is the act of thinking through your code in order to create meaningful tests itself creates substantially higher-quality code that has fewer bugs to begin with. Designing your code to be testable has a similar positive effect.

TDD forces you to think through your code more thoroughly than just diving in and implementing. It encourages you to think in terms of the "contract" of the module that is about to be written.

Cons

TDD is counter-intuitive to the way that many developers naturally think and work. Software is more malleable than, say, constructing a house.

Many (very likely *most*, although I have no data on that) developers like to get a little something working, then write a test for it.

In a similar vein, sometimes the design is not clear early on, and some rapid prototyping is helpful to sketch out the outline of an approach. In such cases, writing a lot of tests very early on can feel wasteful, because there's a high likelihood of throwing away some or all of those tests as the design evolves.

That leads to the largest *perceived* con, which is that writing tests up front slows you down. This is arguable of course since proponents will point to the overall time savings from avoiding pitfalls, but nonetheless is a widely-held view.

Popularity

Because of the aforementioned cons, *pure* TDD is less common. However it is completely standard practice to write automated tests for your code, usually very close in time to when the code is written.

For example, solid development teams often have a fairly strict policy of refusing to merge pull requests that don't include tests, unless the change is "trivial" (only translation strings, say).

Should I use it?

You should try it! Pure TDD is a bit mind-bending (in a similar vein to trying a pure functional programming language like Haskell). Even if you fall back to the more-common "write tests at *about* the same time as writing the code" mode, imposing a hard requirement of careful and detailed forethought about the contract of the code you are about to write is an intriguing experience.