

And when your design is better, what are the consequences?

* Your code is easier to read
* Your code is easier to understand
* Your code is easier to test
* Your code is easier to reuse
* Your code **is easier to debug**
* Your code **has fewer bugs in the first place**

JUnit

http://www.onjava.com/pub/a/onjava/2003/04/02/javaxpckbk.html

http://blog.hubstaff.com/why-you-should-write-unit-tests/

TDD

**5 Common Misconceptions About TDD & Unit Tests**

https://medium.com/javascript-scene/5-common-misconceptions-about-tdd-unit-tests-863d5beb3ce9

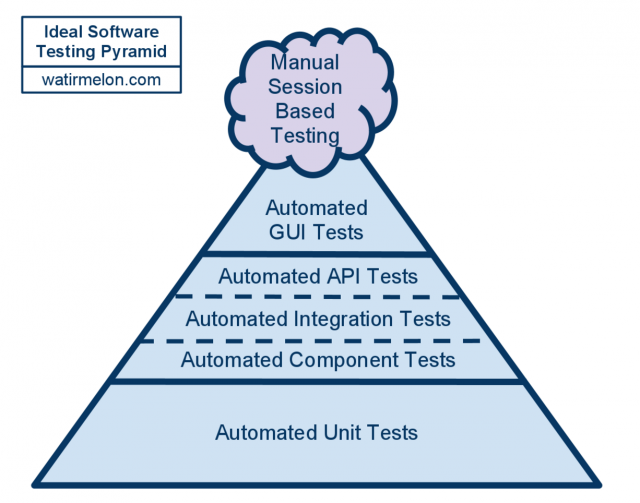
# The Outrageous Cost of Skipping TDD & Code Reviews

https://medium.com/javascript-scene/the-outrageous-cost-of-skipping-tdd-code-reviews-57887064c412

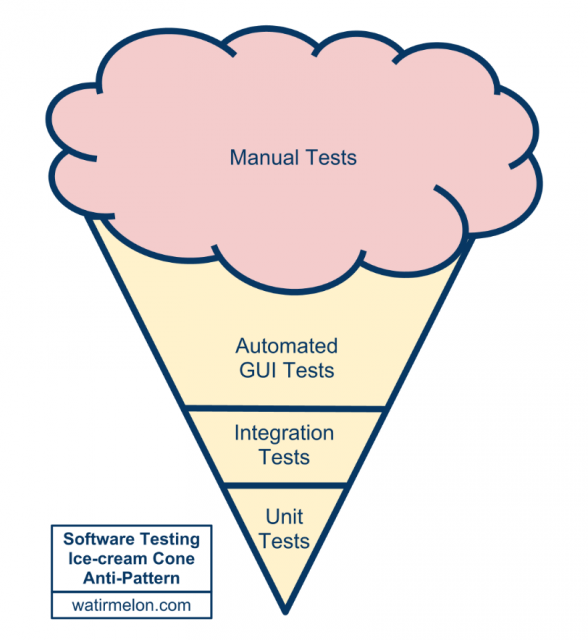
# Empirical Studies Show Test Driven Development Improves Quality

https://www.infoq.com/news/2009/03/TDD-Improves-Quality

The second problem of TDD is that test code grows linearly with production code. All lines of code require maintenance, which means cost. The cost is easiest to acknowledge when a change to existing functionality is made. The tests have to be modified as well as production code. This becomes a problem when multiple tests execute the same line of production code. This usually happens when there are too many high-level tests. The highest level test is an end-to-end test, which starts by invoking the GUI and goes all the way down to the data storage level. But it’s not only the highest-level test that can cause problems.



Tests that execute a whole component, for example a web service, are quite high-level too, compared to a unit test. The ice cream cone anti-pattern is a [**nice illustration of TDD gone wrong**](http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/). An ice cream cone emerges when you invert the ideal test automation pyramid. You can make ice cream by creating more integration tests than unit tests and more GUI tests than integration tests and even more manual regression tests than your automated GUI tests.



https://zeroturnaround.com/rebellabs/if-and-when-you-should-use-test-driven-development/