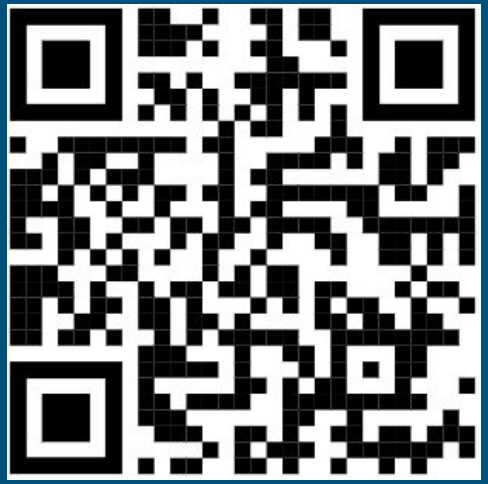
Test Driven Development

Mind in reverse?

Contents

Code quality

Does it matter?

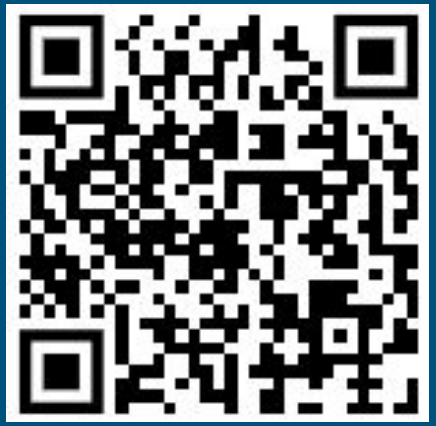


https://youtu.be/Iq_r7IcNmUk

Attributes of quality code

- Modularity
- Separation of concerns
- High cohesion
- Information hiding (abstraction)
- Appropriate coupling

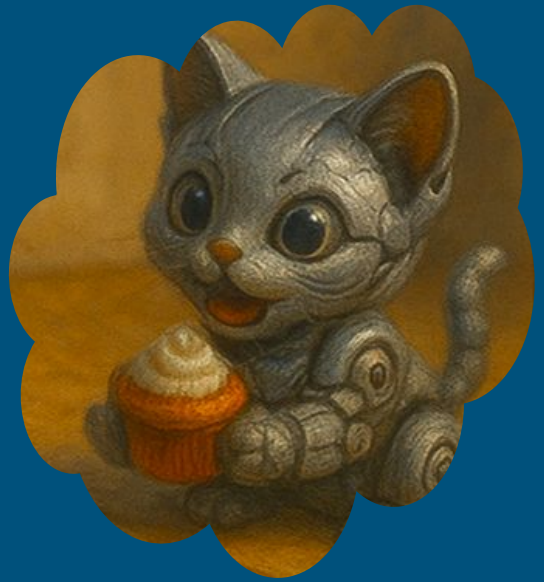
Dave Farley: Modern Software Engineering



You should scan the QR code and follow the YouTube channel by Dave Farley, Trisha Gee, Steve Smith, Emily Bache, Kevlin Henney & Kent Beck

Red, green, refactor

- Red: Write a test, run it, see it fail.
- Green: Write just enough code to make the test pass, run it, and see it pass
- Refactor: Modify the code and the test to make it clear, expressive, elegant, and more general. Run the test after every tiny change and see it pass.



Lets begin the madness!

Your task is to create an image of a coffee maker.

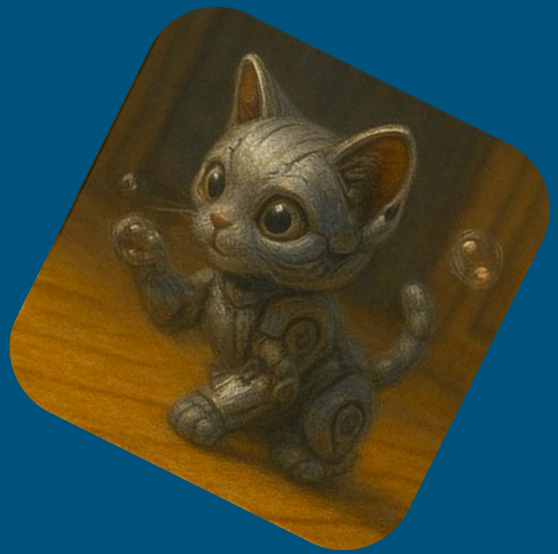
You can use paint, chatgpt, google image search, or pen and paper.

This is individual task, do not copy your neighbor

You have 15 seconds.

Ready?

GO!



The coffee maker problem

Now show your creations of a coffee maker with your peers.

How many different designs for coffee maker did we end up with?

The point of this experiment is to show to you how many different designs we can come up with for a coffee maker under pressure and vague requirements.

Now let’s set up a scene for us…

The scene

The High and Mighty Corporation you are working on has come up with a great way to save money and lift company profits!

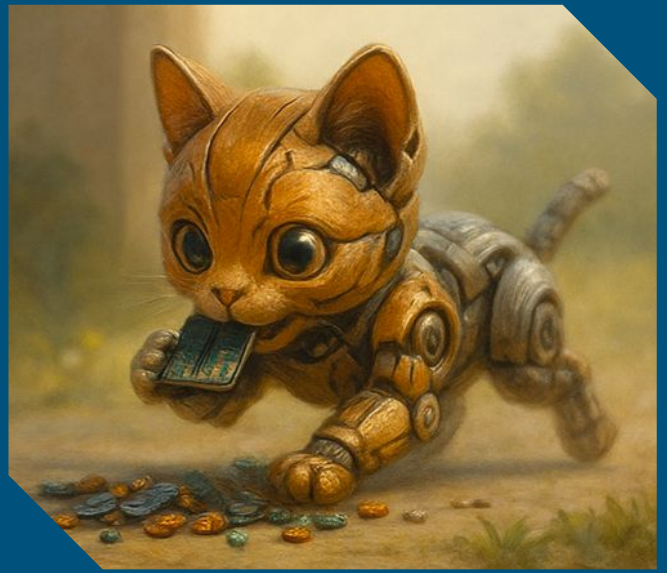
“We have decided to replace our coffee makers with a virtual one! Since most of you work from home most of the time, we have decided to provide you with a virtual coffee maker so you can enjoy your virtual coffee in your virtual office together!

Now create one for us!”

Lets see what LLM is gonna do

What is wrong with the implementation provided by the LLM?

The code itself compiles and works, so it’s not that.



Key points wrong with LLM implementation

None of these functionalities were actually requested by our High and Mighty Corporate executives.

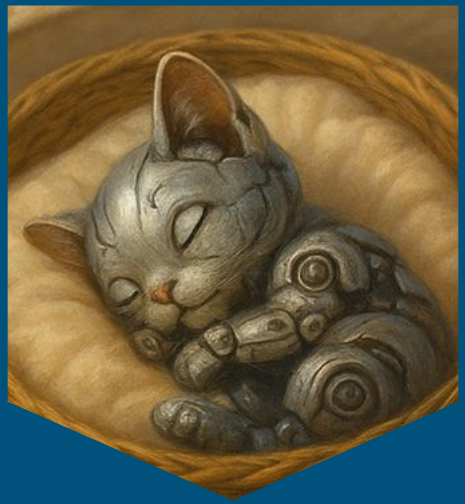
So we wasted precious time and effort… or well in this case, we wasted tokens on the LLM.

Furthermore, if you look at the implementation and test code, you can see that they are actually tightly coupled on to each other. Add a new recipe? You need to add them to the function and modify the tests. So in a sense, we are not actually testing if we provided what we were asked to do, we are testing if the code written is the code written.

So lets build the minimum viable product.

We have no requirements whatsoever, so let’s go along those lines.

What does our CoffeeMaker class need to be a minimum viable product?



Getting feedback from the users

So now we have actually provided our High and Mighty Corporation with a virtual coffee maker and it took no time at all, we even managed to ship it under budget! Fabulous, right?

And it’s good, but now the issues start to arise… some employees have raised concerns about the coffee maker always being plugged into the virtual power outlet causing it to consume electricity during off-hours.

Our executives have requested us to have the coffee maker switch off during off hours.

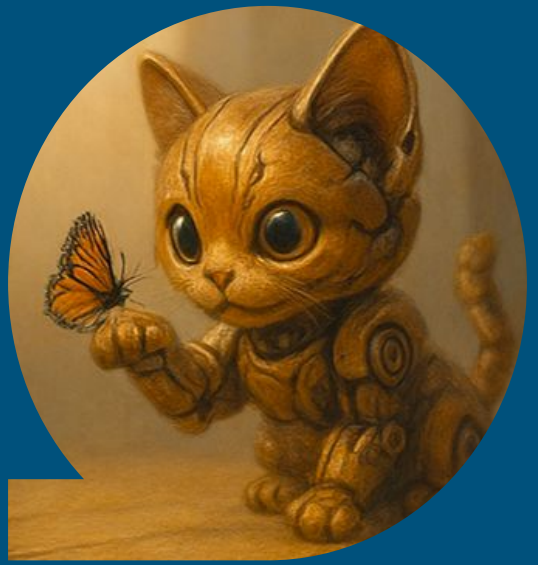
Red, green, refactor

So let’s not go our usual route, instead let’s focus on building the test cases first!

What are we actually testing?

Let’s do the simple case first.

We know that office hours are 0900-1700.



Let’s stop for a moment before we rush forwards.

So we know that the office hours are 0900-1700.

Except if it’s a holiday, then the office is closed.

Oh and what if it’s a crunch time? Then everyone is working extra hours.

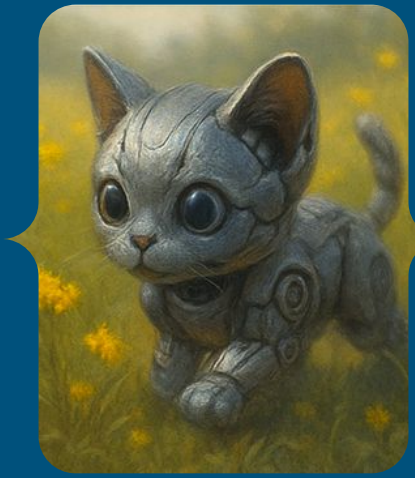
And what if our operations expand to different time zones?

How do we design our code to enable these different behaviors required from the coffee maker? It gets murky really fast.

Dependency injection

Let’s refactor the timer into its own service and use dependency injection to provide our CoffeeMaker with the service.

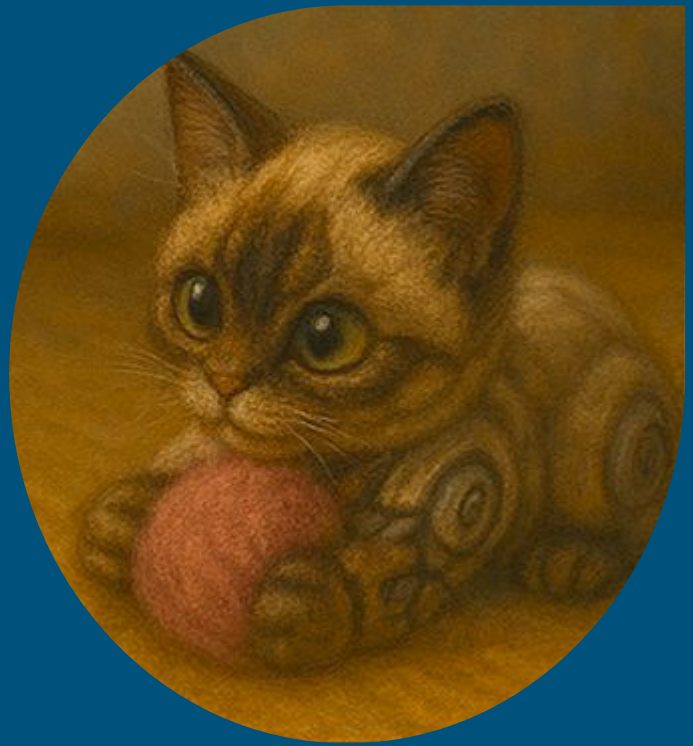
Now we can actually test the timer and coffeemaker on their own… we have separated the concerns and decoupled them from each other allowing us to make changes to them separately.



More requests!

Our users have asked if it would be possible to have more recipes for the coffee! Not everyone likes their coffee pitch black.

This request was the most upvoted within the virtual office survey and our executives have asked us to provide two new options for a recipe. Coffee with milk and coffee with oat drink as a vegan option.



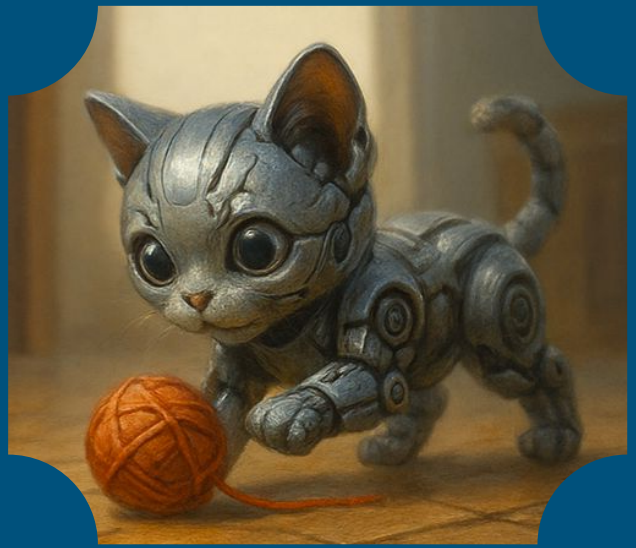
Red, green, refactor

Now what tests should we have for a recipe?

Should we place the recipes within our CoffeeMaker?

Can we make use of dependency injection?

What name should we use for our recipes provider?



Not a new thing

“A software system can best be designed if the testing is interlaced with the designing instead of being used after the design. [...]”

–Excerpt from Report of The Nato Software Engineering Conference by Alan Perlis (1968)

“The original description of TDD was in an ancient book about programming. It said you take the input tape, manually type in the output tape you expect, then program until the actual output tape matches the expected output. [...]”

–Kent Beck, author of Extreme Programming (XP)

Recap

Test Driven Development is not only about writing tests, it is about designing your code to exhibit the attributes of high quality code: modularity, separation of concerns, high cohesion, information hiding (abstraction), appropriate coupling and testability.

You can introduce TDD into an existing project, but trying to retrofit it to everything can be painful experience without refactoring the whole codebase. Use it for anything new you are building.

The side-effects of TDD is that you generate the tests, which in themselves are the documentation of the specification itself.

Using TDD to create your programs will force you to think about your design, as opposed to the classical way of banging your head against the compiler until things work.

Me & kittens thank you!