
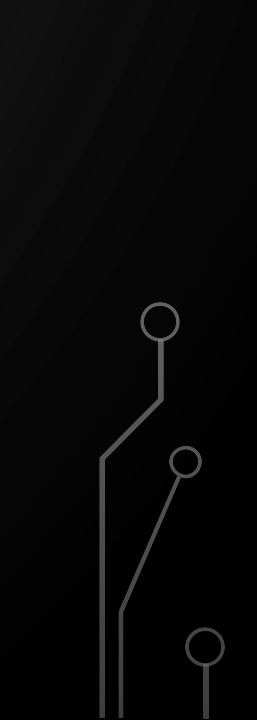


The image features a dark gray background with white decorative elements resembling circuit board traces. These traces are located in the four corners, forming abstract patterns of lines and circles. In the center, the text "DEVELOPMENT AND TESTING" is displayed in a clean, white, sans-serif font.

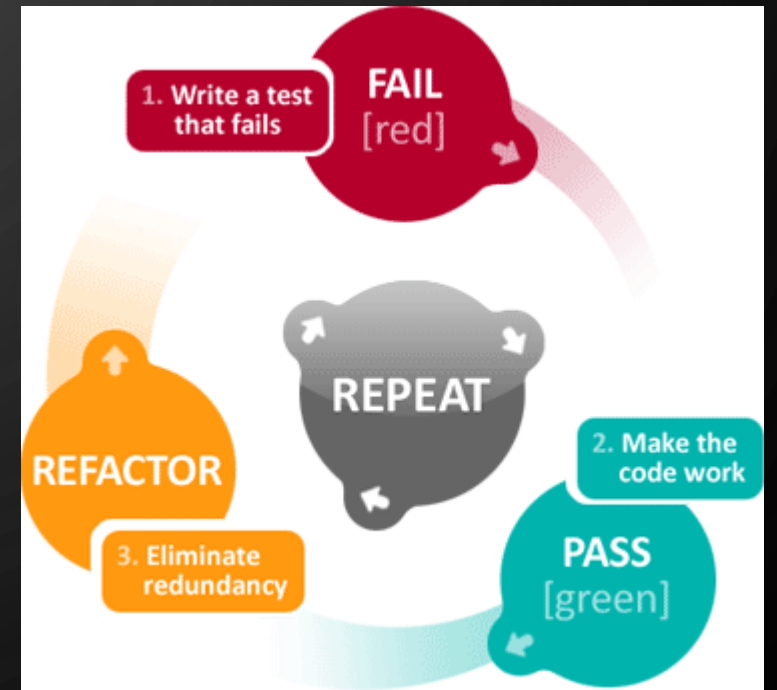
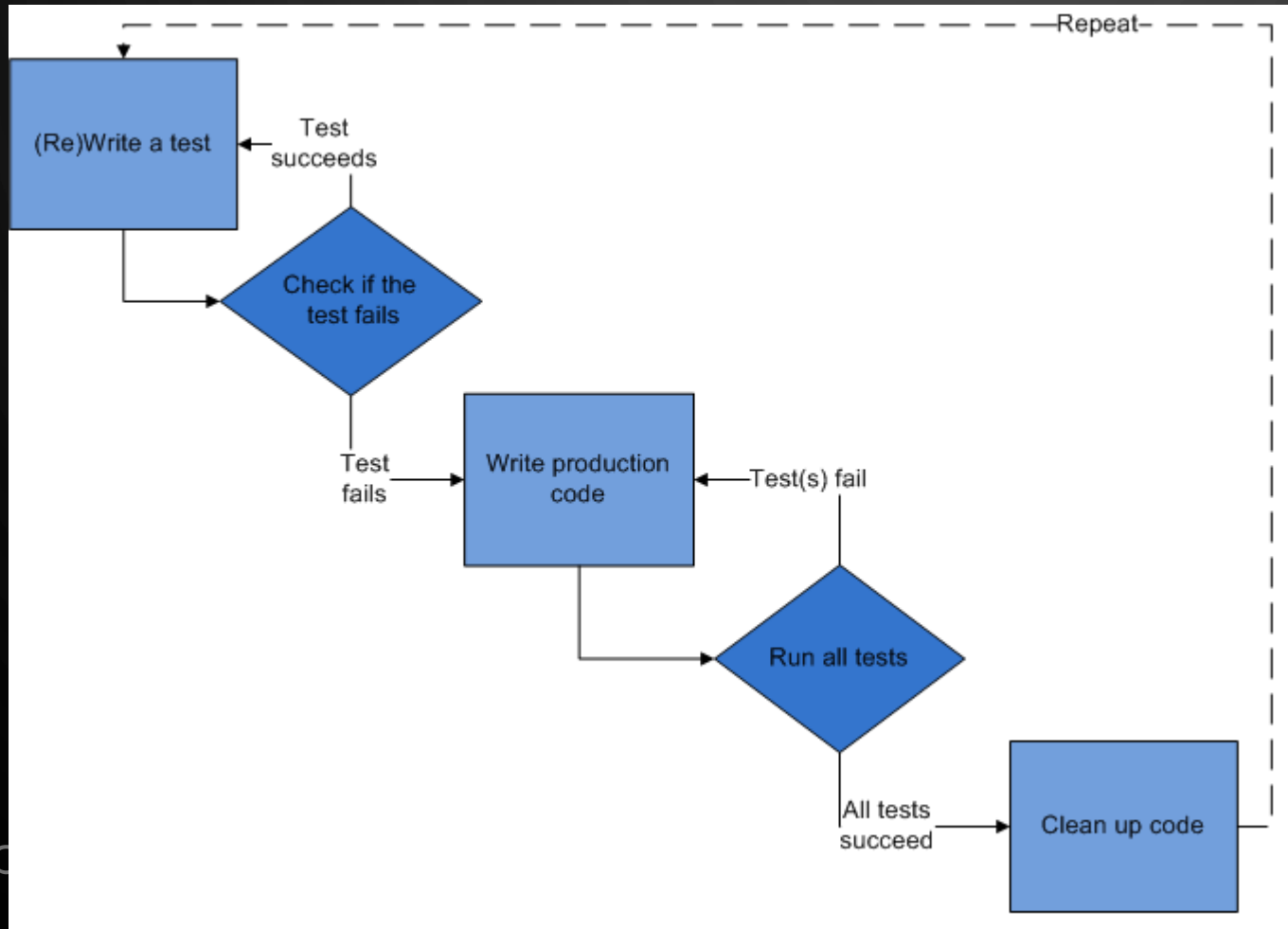
# DEVELOPMENT AND TESTING

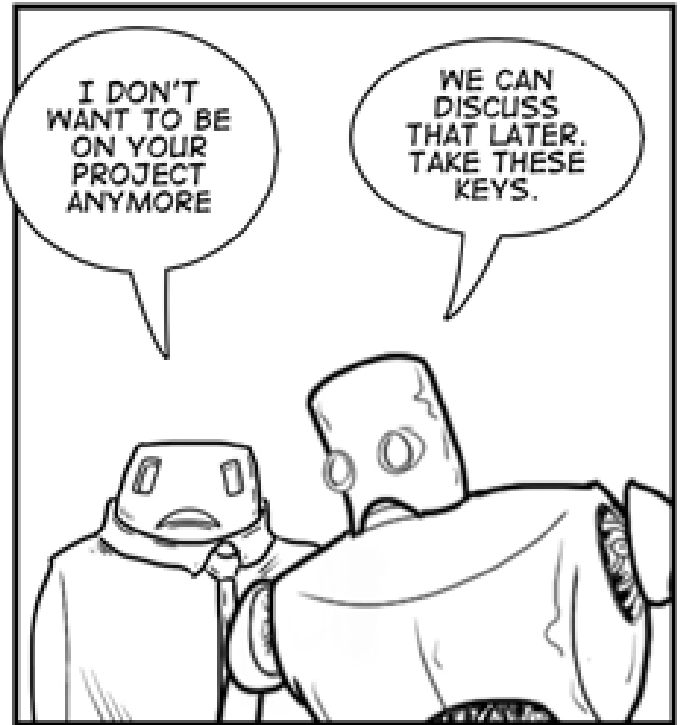
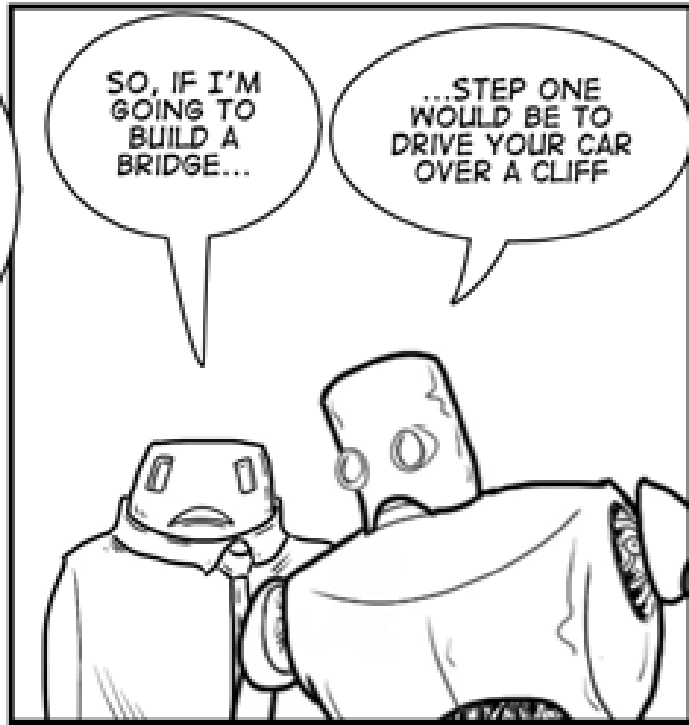
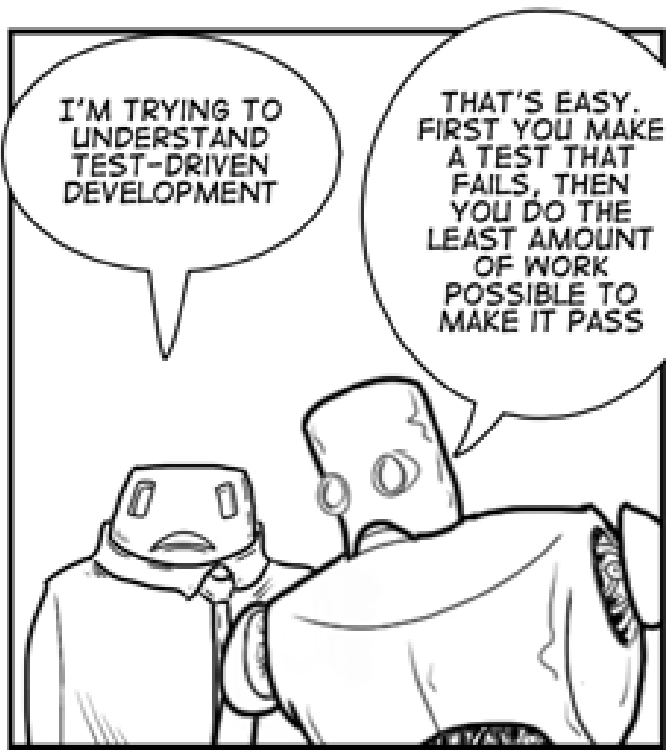


# DEVELOPMENT (ONE OUT OF MANY PERSPECTIVES)

1. Solve
  2. Implement
    1. Write test
    2. Write code
    3. Repeat
  3. Integrate
  4. Release
- 
- 

# TEST DRIVEN DEVELOPMENT (TDD)



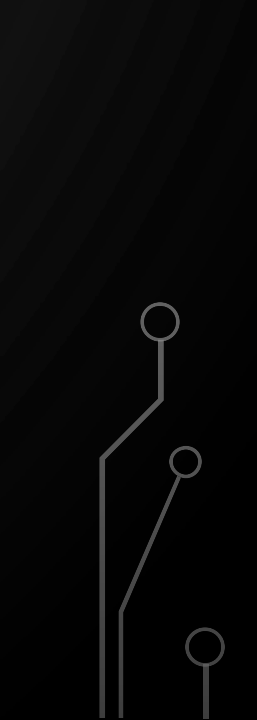
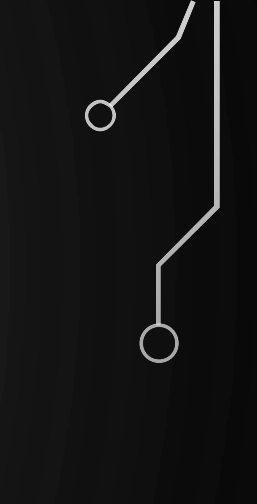





# PRACTICAL EXAMPLE

- Lets practice some TDD on the following example

***Your project manager at BusyBody Inc says he needs a feature implemented which determines the total amount of time a worker at the company spends at their desk. He says the number of hours each day is already being measured and is stored in an internal array in the code base.***



The slide features a dark gray background with white decorative elements resembling circuit board traces and nodes. These elements are located in the four corners: top-left, top-right, bottom-left, and bottom-right. The central text is in a clean, white, sans-serif font.

# PRACTICAL EXAMPLE

- How do we solve this?
  - Compute the sum!

# PRACTICAL EXAMPLE

- First we write a test
  - in other words, set up the scaffolding of the code instead of a function which you don't know if it works or not – and continue to struggle finding bugs

```
public static double sum(double[] arr) {  
    return Double.POSITIVE_INFINITY; //note this clearly does not work and is thus failing  
}
```

```
public static void main() {  
    double[] arr = {0, 1, 1, 2, 3, 5, 8};  
    if(sum(arr) != 20)  
        System.out.println("Test failed?!?!?! I suck!"); //you don't, its supposed to fail!  
}
```



TESTING

I FIND YOUR LACK OF TESTS DISTURBING.

# PRACTICAL EXAMPLE

- Before we continue, lets review
  - Positives
    - Scaffolding, function interface, and test all implemented
    - We know it is good design
    - Tests to tell if the code is correct, before we struggle with debugging many lines of code
  - Negatives
    - Code isn't written until later.....but is that really that bad? NO
- In fact, with TDD you code **FASTER** and more **EFFECTIVELY** than without it



# PRACTICAL EXAMPLE

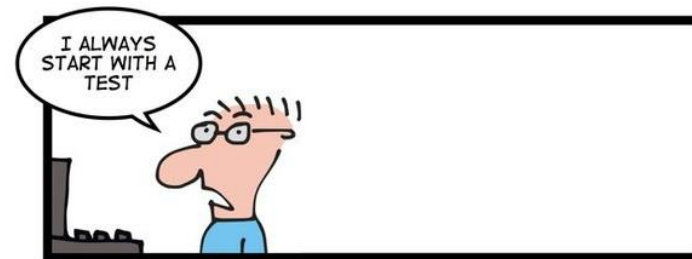
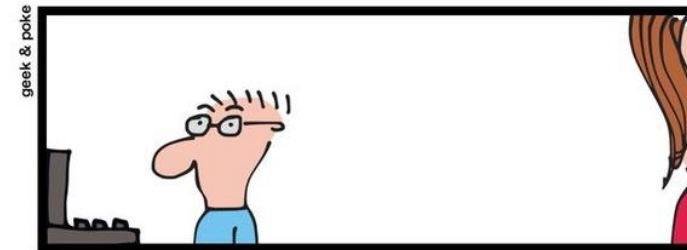
- Now the code – and then run the test!

```
public static double sum(double[] arr) {  
    double s = 0;  
    for(double x : arr)  
        s += x;  
    return s;  
}
```

# THINGS TO REMEMBER

- Always have code that compiles
- Test writing is an art that takes practice (and more learning!)
- Compile and test often!

## SIMPLY EXPLAINED



TDD

# TESTING FRAMEWORKS

- Many frameworks exist CppUnit, JUnit, etc.
- We will be using a much more simple unit testing framework developed by me
  - A unit test is a check of one behavior of one “unit” (e.g., function) of your code
  - If you have downloaded the lab zip for today open it and look there
  - Follows **SETT** – unit testing paradigm
    - **Setup** – create data for input and predetermine the output
    - **Execute** – call the function in question
    - **Test** – analyze correctness and determine true/false for test
    - **Teardown** – cleanup any data, close buffers, etc

# UNIT TEST EXAMPLE

```
public static boolean testSum() {  
    //setup  
    double[] arr = {0, 1, 1, 2, 3, 5, 8};  
    double ans = 20;  
  
    //execute  
    double s = sum(arr);  
  
    //test  
    return s == ans;  
  
    //teardown - here is empty  
}
```

# TDD - EXERCISE

- Write a Java function to find the median of an array of integers
  - Do test driven development, starting with a good unit test(s)
  - After test is created and checked, code the function
- Pair program
- Once completed, you should use the extra time to work on Programming Assignment 5