HiRez.io - Test Effective TDD Guide

1. What Is The Subject Under Test?

```
describe('SubjectName', ()=>{ ... });
```

2. How Do I Get a Reference To That Subject?

```
Given(() => { ... }); // <-- Root level describe()
```

3. What Is The *Action*?

If the unit is an object, add this wrapper:

```
describe('INIT / METHOD / EVENT: MethodOrEventName', ...
```

Trigger the action inside of:

```
When(() => { ... });
```

4. What Is The *Output*?

```
Then('should...', () => { ... });
```

3 possible *output* types:

Direct *(return value)*, **Indirect** *(side effect)*, **Collaboration** *(Test double's method)*

5. What Is The *Input*? (that will cause this output)

```
Given(() => { ... });
```

3 possible *input* types:

Direct *(func parameter)*, **Indirect** *(outer scope var)*, **Collaboration** *(Test double's method)*

6. What's The Quickest Way To Make The Test Pass?

Write just enough code to make the test runner go green

7. Did I Cover All Possible Use Cases?

Use case = Input + Output pair.

Did you check:
Unhappy paths? Conditions ("else"), errors (sync, async)

Wrap each use case with a describe()
Extract the "When()" to the outer scope.

```
When(() => { });

   describe('GIVEN...', () => {
     Given(() => { });

     Then('should...', () => { });

   });
```

8. If The Test Passes, Could The Code Be Cleaner?

Refactor the production code to clean it up,
while keeping the test runner green

9. Did I TDD Every New Test Double Method?

For each new spy, create a new spec file and go back to step 1.

TIP: PRINT THE FIRST 2 PAGES 👆

(And Keep it nearby)

Welcome to HiRez.io's Test Effective TDD Guide

The purpose of this guide is to save you valuable time by giving you the next natural question to ask yourself while TDDing your code.

This guide is part of HiRez.io's Test Effective Courses which can be found here.

It's a result of my years of testing experience condensed into simple easy to follow steps.

Now let's elaborate on each of the steps in the guide, to give you some more context on each one.

This can be a good reminder for what you've learned in the courses.

TDD Steps - The Longer Version

Let's go over the steps in more detail...

---

1. What Is The "Subject Under Test"?

WHAT?

Write the subject name (it could be the class name or the function for example) in the "Root Describe" (the first one in the test file).

Write only the name, to keep it simple and concise.

WHY?

To make sure it'll be super easy to spot where's the "Root Describe" and what this test is all about.

EXAMPLE:

```
describe('SubjectName', ()=>{ ... });
```

---

2. How Do I Get a Reference To That Subject?

WHAT?

“Getting a reference” means - to have the actual final subject under test.

If it’s a class, it needs to be instantiated.

If it’s an Angular service, it needs to be injected.

You always want to get a reference in the root “Given” of your test.

WHY?

To make sure there’s only one place you need to look for if you need to make changes to the way you’re creating your subject.

Plus, you want to ALWAYS “reset” the testing environment before each test. (and this includes your subject under test)

EXAMPLE:

```
describe('GIVEN...', () => { // <-- Root level describe()

     Given(() => {
         // This is where you call “new MyClass()” if needed
         // or inject dependencies etc.
     });

});
```

3. What Is The “Action”?

WHAT?

Action is the thing that is triggering some kind of a change in the state of your app.

For functions - it is easy, they have only 1 action type - the function call itself.

For objects - wrap all the test cases of the action type in a `describe()` and give it the action type prefix.

There are only 3 action types for objects -

1. Initialization of the subject *(i.e what happens when you call `new MyClass()`)*

```
describe('INIT', ...
```

2. Method Call *(i.e what happens when you call the method)*

```
describe('METHOD: methodName', ...
```

3. Event *(i.e what happens when you trigger an event)*

```
describe('EVENT: eventName', ...
```

Always trigger the action inside of When():

```
When(() => { ... });
```

WHY?

To make it easier to group all action related tests together.

EXAMPLE:

```
describe('METHOD: doSomething', () => {
     When(() => {
        subjectUnderTest.doSomething();
     });
});
```

4. What Is The “Output”?

WHAT?

“Output” is what happens after we trigger the action.

There can be only 3 types of outputs:

1. **Direct** - The return value of a method / function.

2. **Indirect** - Side effect outside of the method / function

3. **Collaboration** - call to a test double’s method
   (“test double” is a general name for a fake, mock, spy, etc)

The output should be checked in the “`Then()`” function which should be written with a descriptive title.

WHY?

To make it easy to look at all the outputs in one place.

EXAMPLE:

```
Then('should...', () => {
   expect(actualResult).toEqual(expectedResult)
});
```

5. What Is The “Input”?

WHAT?

“*Input*” is the state that causes the *output* (after triggering the *action*)

There can be only 3 types of inputs: (same three by the way)

4. **Direct** - The method / function’s parameters
5. **Indirect** - changing an outer scope variable.
6. **Collaboration** - value returned from a test double’s method.

The input should be set in the “`Given()`” function.

WHY?

To make it easy to look at all the inputs in one place.

EXAMPLE:

```
Given(() => {
   subjectUnderTest.name = 'Bonnie';
});
```

6. What's The Quickest Way To Make The Test Pass?

WHAT?

The goal here is to make the test you've just written to pass (go green) in the shortest way possible.

Do not add untested logic while trying to make the test pass.

WHY?

To make sure you're not forgetting to test the code you're adding.

This could prevent sneaky bugs.

7. Did I Cover All The Possible Use Cases?

WHAT?

A “use case” is a pair of *input* and *output*.

Usually, for every different input there will be a different output.

In libraries from the “`given`” family (like `@hirez_io/jasmine-given`) you can move the `When()` function outside of the use cases.

So whenever you have more the one “use case”, consider extracting the `When()` outside of the describe and wrap the use case in its own `describe()` with the “GIVEN …” description.

After each use case ask yourself:

Do I still have more **unhappy paths** to cover? **conditions** (“else”)? or **errors** (sync, async)? If so, add them as more use cases.

WHY?

To make each “use case” shorter and more concise.

Some people prefer to leave the “When()” inside each use case to remove some “voodoo” from the code.

Do what you feel makes more sense to you.

EXAMPLE:

```
When(() => { });

describe('GIVEN...', () => {
  Given(() => { });

  Then('should...', () => { });

});
```

8. If The Test Passes, Could The Code Be Cleaner?

WHAT?

Once your test passes and is considered “complete” (no need to add more lines), now is the time to refactor your code if it requires it.

As long as your tests are green you can continue to modify and refactor the code covered by those tests.

WHY?

When we rush to write a minimal amount of code just to make the test pass, we might not stop to “design the code” thus ending up with more messy code.

The idea here is to use the tests as a “safety net” and to design your code (i.e changing names, breaking into smaller functions) to make it more readable and to make sure each part has only one responsibility (Single responsibility principle).

9. Did I TDD Every New Test Double's Method?

WHAT?

For each test double's method, we need to make sure there is a corresponding test of the "real collaborator" it represents.

So once you're done with the current test, go over all the test doubles you created as part of your test driven development and make sure you have symmetric tests for their methods.

WHY?

It's easy to forget to test test doubles, so it's a good habit to always ask yourself this question when you're done writing all the tests for a particular subject.

That's IT! 😎

I'm sure you'll find this guide useful in your day to day TDD.

Make sure to print the first 2 pages and keep it near you computer - this will make you a TDD master much faster.

If you have any suggestions, questions or anything else you want to share, please feel free to contact us at bonnie@hirez.io

Happy Testing!
Shai Reznik