Your Requirements Specification as an Executable Test Suite

Brad Richardson
Archaeologic, Inc.

Website: https://everythingfunctional.com
Email: everythingfunctional@protonmail.com
Twitter: @everythingfunct
Everywhere else: @everythingfunctional

What are software tests?

What are automated tests?

What makes a good test?

Organizing Behavior Tests (What)

```
function run_test(input) result(example_result)
    class(input_t), intent(in) :: input
    type(transformed_t) :: example_result

    type(test_item_t) :: example

    select type (input)
    type is (test_item_input_t)
        example = input%input()
        example_result = transformed_t(test_result_item_input_t( &
                example%run()))
    class default
        example_result = transformed_t(transformation_failure_t( &
                fail("Expected to get a test_item_input_t")))
    end select
end function
```

```
function test_passing_case_behaviors() result(test)
    type(test_item_t) :: test

    test = given( &
            "a passing test case", &
            test_item_input_t(example_passing_test_case()), &
            [ when( &
                    "it is run", &
                    run_test, &
                    [ ... &
                    , then__("it knows how many asserts there were", check_num_asserts) &
                    , ... &
                    ]) &
            ])
end function

...

function check_num_asserts(input) result(result_)
    class(input_t), intent(in) :: input
    type(result_t) :: result_

    type(test_result_item_t) :: example_result

    select type (input)
    type is (test_result_item_input_t)
        example_result = input%input()
        result_ = assert_equals(NUM_ASSERTS_IN_PASSING, example_result%num_asserts())
    class default
        result_ = fail("Expected to get a test_result_item_input_t")
    end select
end function
```

In what scenario

I do what thing

And expect what outcome

Organizing Behavior Tests (How)

```
function run_test(input) result(example_result)
    class(input_t), intent(in) :: input
    type(transformed_t) :: example_result

    type(test_item_t) :: example

    select type (input)
    type is (test_item_input_t)
        example = input%input()
        example_result = transformed_t(test_result_item_input_t( &
                example%run()))
    class default
        example_result = transformed_t(transformation_failure_t( &
                fail("Expected to get a test_item_input_t")))
    end select
end function
```

What sequence of calls

```
function test_passing_case_behaviors() result(test)
    type(test_item_t) :: test

    test = given( &
            "a passing test case", &
            test_item_input_t(example_passing_test_case()), &
            [ when( &
                    "it is run", &
                    run_test, &
                    [ ... &
                    , then__("it knows how many asserts there were", check_num_asserts) &
                    , ... &
                    ]) &
            ])
end function

...

function check_num_asserts(input) result(result_)
    class(input_t), intent(in) :: input
    type(result_t) :: result_

    type(test_result_item_t) :: example_result

    select type (input)
    type is (test_result_item_input_t)
        example_result = input%input()
        result_ = assert_equals(NUM_ASSERTS_IN_PASSING, example_result%num_asserts())
    class default
        result_ = fail("Expected to get a test_result_item_input_t")
    end select
end function
```

What are the inputs

What are the outputs

Organizing Property Tests

```
function test_collection_properties() result(test)
    type(test_item_t) :: test

    test = describe( &
            "A test collection", &
            test_item_input_t(example_passing_collection()), &
            [ it_("can tell how many tests it has", check_num_cases) &
            , ... &
            ])
end function

function check_num_cases(input) result(result_)
    class(input_t), intent(in) :: input
    type(result_t) :: result_

    type(test_item_t) :: example_collection

    select type (input)
    class is (test_item_input_t)
        example_collection = input%input()
        result_ = assert_equals(NUM_CASES_IN_PASSING, example_collection%num_cases())
    class default
        result_ = fail("Expected to get a test_item_input_t")
    end select
end function
```

Any input or state

What is always true

What can be performed

An Example: What is a leap year?

```
pure function is_leap_year(year)
    integer, intent(in) :: year
    logical :: is_leap_year

    is_leap_year = mod(year, 4) == 0
end function
```

The Ugly

```
module ugly_test
    use is_leap_year_m, only: is_leap_year
    use vegetables, only: &
            result_t, test_item_t, assert_not, assert_that, describe, it

    implicit none
    private
    public :: test_is_leap_year
contains
    function test_is_leap_year() result(tests)
        type(test_item_t) :: tests

        tests = describe("is_leap_year", [it("works", check_is_leap_year)])
    end function

    function check_is_leap_year() result(result_)
        type(result_t) :: result_

        result_ = &
                assert_not(is_leap_year(1)) &
                .and.assert_that(is_leap_year(4))
    end function
end module
```

```
$ fpm test -- -v
Running Tests

Test that
    is_leap_year
        works

A total of 1 test cases

All Passed
Took 1.1328e-5 seconds

Test that
    is_leap_year
        works
            Was not true
            Was true

A total of 1 test cases containing a total of 2 assertions
```

The Bad

```
function test_is_leap_year() result(tests)
    type(test_item_t) :: tests

    tests = describe(&
            "is_leap_year", &
            [ it("is true for leap years", check_leap_year) &
            , it("is false for non leap years", check_non_leap_year) &
            ])
end function

function check_leap_year() result(result_)
    type(result_t) :: result_

    result_ = &
            assert_that(is_leap_year(2016), "2016") &
            .and.assert_that(is_leap_year(2000), "2000")
end function

function check_non_leap_year() result(result_)
    type(result_t) :: result_

    result_ = &
            assert_not(is_leap_year(1999), "1999") &
            .and.assert_not(is_leap_year(1900), "1900")
end function
```

```
$ fpm test -- -v
Running Tests

Test that
    is_leap_year
        is true for leap years
        is false for non leap years

A total of 2 test cases

Failed
Took 4.6693e-5 seconds

Test that
    is_leap_year
        is true for leap years
            Was true
                User Message:
                    |2016|
            Was true
                User Message:
                    |2000|
        is false for non leap years
            Was not true
                User Message:
                    |1999|
            Expected to not be true
                User Message:
                    |1900|

1 of 2 cases failed
1 of 4 assertions failed
```

The Good

```
function test_is_leap_year() result(tests)
    type(test_item_t) :: tests

    tests = describe(&
            "is_leap_year", &
            [ it( &
                    "returns false for years that are not divisible by 4", &
                    check_not_divisible_by_4) &
            , it( &
                    "returns true for years that are divisible by 4 but not by 100", &
                    check_divisible_by_4_but_not_100) &
            , it( &
                    "returns false for years that are divisible by 100 but not by 400",
                    check_divisible_by_100_but_not_400) &
            , it( &
                    "returns true for years that are divisible by 400", &
                    check_divisible_by_400) &
            ])
end function
```

```
$ fpm test -- -q -v
Running Tests

A total of 4 test cases

Failed
Took 1.27397e-4 seconds

Test that
    is_leap_year
        returns false for years that are not divisible by 4
            Was not true
                User Message:
                    |2002|
            Was not true
                User Message:
                    |2003|
        returns true for years that are divisible by 4 but not by 100
            Was true
                User Message:
                    |2004|
            Was true
                User Message:
                    |2008|
        returns false for years that are divisible by 100 but not by 400
            Expected to not be true
                User Message:
                    |1900|
            Expected to not be true
                User Message:
                    |2100|
        returns true for years that are divisible by 400
            Was true
                User Message:
                    |2000|
            Was true
                User Message:
                    |2400|

1 of 4 cases failed
2 of 8 assertions failed
```

The Fancy

```
function test_is_leap_year() result(tests)
    type(test_item_t) :: tests

    tests = describe(&
            "is_leap_year", &
            [ it( &
                    "returns false for years that are not divisible by 4", &
                    [ example_t(integer_input_t(2002)) &
                    , example_t(integer_input_t(2003)) &
                    ], &
                    check_not_leap_year) &
            , it( &
                    "returns true for years that are divisible by 4 but not by 100", &
                    [ example_t(integer_input_t(2004)) &
                    , example_t(integer_input_t(2008)) &
                    ], &
                    check_leap_year) &
            , it( &
                    "returns false for years that are divisible by 100 but not by 400", &
                    [ example_t(integer_input_t(1900)) &
                    , example_t(integer_input_t(2100)) &
                    ], &
                    check_not_leap_year) &
            , it( &
                    "returns true for years that are divisible by 400", &
                    [ example_t(integer_input_t(2000)) &
                    , example_t(integer_input_t(2400)) &
                    ], &
                    check_leap_year) &
            ])
end function
```

```
$ fpm test -- -q -v
Running Tests

A total of 4 test cases

Failed
Took 1.27397e-4 seconds

Test that
    is_leap_year
        returns false for years that are not divisible by 4
            Was not true
                User Message:
                    |2002|
            Was not true
                User Message:
                    |2003|
        returns true for years that are divisible by 4 but not by 100
            Was true
                User Message:
                    |2004|
            Was true
                User Message:
                    |2008|
        returns false for years that are divisible by 100 but not by 400
            Expected to not be true
                User Message:
                    |1900|
            Expected to not be true
                User Message:
                    |2100|
        returns true for years that are divisible by 400
            Was true
                User Message:
                    |2000|
            Was true
                User Message:
                    |2400|

1 of 4 cases failed
2 of 8 assertions failed
```

Additional Resources

- Much inspiration for the examples was taken from Kevlin Henney's talk here: https://www.youtube.com/watch?v=tWn8RA_DEic
- Many of my views on testing techniques were taken from The Art of Unit Testing by Roy Osherove: https://www.manning.com/books/the-art-of-unit-testing-second-edition
- You can find the Vegetables source code at: https://gitlab.com/everythingfunctional/vegetables
- Feel free to reach out with questions and comments at: everythingfunctional@protonmail.com
- Slides and code at https://github.com/fortran-lang/talks