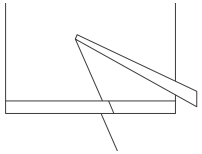
Bring ideas to life
VIA University College



Test Driven Development/Google Test

TDD

At the end of this session, you should

- Be able to use Google Test framework for testing C-programs
- Understand the Test Driven Development cycle
- Be able Unit Test C-programs
- Produce cleaner/simpler code of higher quality

What is good quality for Software

Designed Well

- High Cohesion
- Loose Coupling
  - Independent modules
- Clear Interfaces
- Little Redundancy
- Layered design
- Designed for Test
- Extensible
- Portable

What is good quality for Software?

Maintainable

"What makes the difference between working code and great code is maintainability" - David Rachamim

"You don't really know how good someone's code is until you try to change it" - Kristopher Johnson

- Well documented
- Code is readable – stick to coding and naming standards
- Code is simple – KISS Keep it simple stupid!
- Testable
  - Dependency injection
  - Encapsulation
- No "gold plating"
- Only optimised if needed
- Automated tests

What is good quality for Software?

It Works

"Beautiful readable , testable, maintainable code that fails to meet the customer needs is still a failure" - codingdave

- As specified by customer
- It solves the intended problems
- Users wants to use it
- It's stable

Whitebox Test/Unit Test

In Java we use jUnits to create unit tests

In C we can use GoogleTest and other frameworks to create unit tests

jUnit an example

`ProductionCode.java`

```
package edu.via.esw1;

public class ProductionCode {
    public int sum(int a, int b) {
        return a + b;
    }
}
```

jUnit an example

The Test Code

`ProductionCodeTest.java`

```
package edu.via.esw1;

import .....

class ProductionCodeTest {
   private ProductionCode sut;
   @BeforeEach
   void setup() {
      sut = new ProductionCode();
   }
   @AfterEach
   void teardown() {}

   @Test
   void test_sum() {
      // Arrange
      // Act
      int n = sut.sum(3,4);
      // Assert
      assertEquals(7, n);
   }
}
```

GoogleTest an example

The production code we want to test

production.h

```
#pragma once
int production_sum(int a, int b);
```

production.c

```
int production_sum(int a, int b) {
   return a + b;
}
```

GoogleTest an example

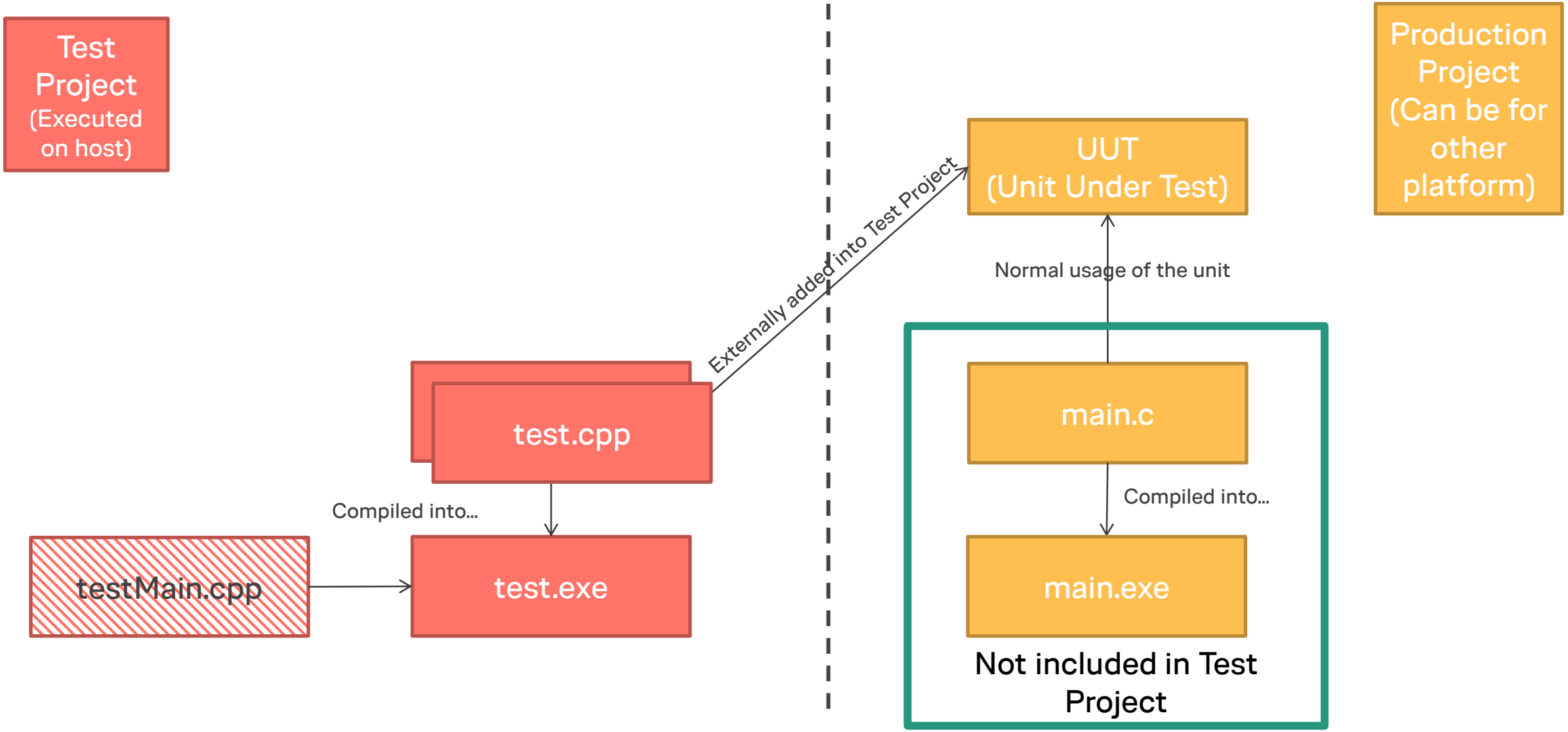
The Test Code

productionTest.cpp

NOTE: This is in GoogleTest project that are separated from the Production code project

```
#include "gtest/gtest.h"

extern "C" {
   #include <production.h>
}

class ProductionTest : public ::testing::Test {
protected:
   void SetUp() override {}
   void TearDown() override {}
};

TEST_F(ProductionTest, Test_sum) {
    // Arange
    // Act
    int n = production_sum(3, 4);

    // Assert/Expect
    EXPECT_EQ(n, 7);
}
```

Unit Test Program Structure



Basic assertions

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_TRUE(condition);	EXPECT_TRUE(condition);	condition is true
ASSERT_FALSE(condition);	EXPECT_FALSE(condition);	condition is false

https://github.com/google/googletest/blob/master/googletest/docs/primer.md

Binary comparison

https://github.com/google/googletest/blob/master/googletest/docs/primer.md

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_EQ(val1, val2);	EXPECT_EQ(val1, val2);	val1 == val2
ASSERT_NE(val1, val2);	EXPECT_NE(val1, val2);	val1 != val2
ASSERT_LT(val1, val2);	EXPECT_LT(val1, val2);	val1 < val2
ASSERT_LE(val1, val2);	EXPECT_LE(val1, val2);	val1 <= val2
ASSERT_GT(val1, val2);	EXPECT_GT(val1, val2);	val1 > val2
ASSERT_GE(val1, val2);	EXPECT_GE(val1, val2);	val1 >= val2

When doing pointer comparisons use *_EQ(ptr, nullptr) and *_NE(ptr, nullptr) instead of *_EQ(ptr, NULL) and *_NE(ptr, NULL). This is because nullptr is typed, while NULL is not. See the FAQ for more details.

If you're working with floating point numbers, you may want to use the floating point variations of some of these macros in order to avoid problems caused by rounding. See Advanced googletest Topics for details.

String comparison

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_STREQ(str1,str2);	EXPECT_STREQ(str1,str2);	the two C strings have the same content
ASSERT_STRNE(str1,str2);	EXPECT_STRNE(str1,str2);	the two C strings have different contents
ASSERT_STRCASEEQ(str1,str2);	EXPECT_STRCASEEQ(str1,str2);	the two C strings have the same content, ignoring case
ASSERT_STRCASENE(str1,str2);	EXPECT_STRCASENE(str1,str2);	the two C strings have different contents, ignoring case

https://github.com/google/googletest/blob/master/googletest/docs/primer.md

Floating-point comparison

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_FLOAT_EQ(val1, val2);	EXPECT_FLOAT_EQ(val1, val2);	the two float values are almost equal
ASSERT_DOUBLE_EQ(val1, val2);	EXPECT_DOUBLE_EQ(val1, val2);	the two double values are almost equal

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_NEAR(val1, val2, abs_error);	EXPECT_NEAR(val1, val2, abs_error);	the difference between val1 and val2 doesn't exceed the given absolute error

https://github.com/google/googletest/blob/master/googletest/docs/advanced.md

GoogleTest

Follow the guide: *Google Test and Mocking.pdf*

Stop reading when you come to Mocking!!!

Setup a simple *Production code* project with a corresponding *GoogleTest* project

What to test for?

What tests are necessary?
What values should we test with?

Let the ZOMBIES guide us

ZOMBIES Guide to Test Cases

ZOMBIES Spelled out:

- Z – Zero
- O – One
- M – Many (or More complex)
- B – Boundary Behaviors
- I – Interface definition
- E – Exercise Exceptional behavior
- S – Simple Scenarios, Simple Solutions

Unpronounceable acronym:
DTSTTCPW. Spelling it out:

Do The Simplest Thing That Could Possible Work.

- Kent Beck

See James W. Grennings explanations:
http://blog.wingman-sw.com/archives/677#more-677

ZOMBIES Guide to Test Cases

Z – Zero

The first test **S**cenarios are for **S**imple post-conditions of a just created object/module.

These are the **Z**ero cases.

ZOMBIES Guide to Test Cases

O – One

The test **S**cenarios are for **S**imple tests dealing with single items

These are the **O**ne cases.

ZOMBIES Guide to Test Cases

M – More or More complex

The test **S**cenarios are for **S**imple tests dealing with more items or more complex scenarios

These are the **M**ore cases.

ZOMBIES Guide to Test Cases

B – Boundary Behaviors

The test **S**cenarios are for **S**imple tests dealing with the boundaries

These are the **B**oundary cases.

ZOMBIES Guide to Test Cases

I – Interface definition

These are not tests, but writing tests defines the needed interfaces for our modules

ZOMBIES Guide to Test Cases

E – Exercise Exceptional behavior

Test all odd situations and be sure that your system can handle them in a defined way

ZOMBIES Guide to Test Cases

S – Simple Scenarios, Simple Solutions

Test simple scenarios one by one, do not test many things in one test case!

Implement the simplest solutions to pass the tests!

TDD Guided by ZOMBIES

Zero

One

Many

Simple Scenarios

Simple Solutions

Boundaries

Behaviors

Interfaces

Exercise

Exceptions

Zombies photo thanks to Joel Friesen

What is TDD?

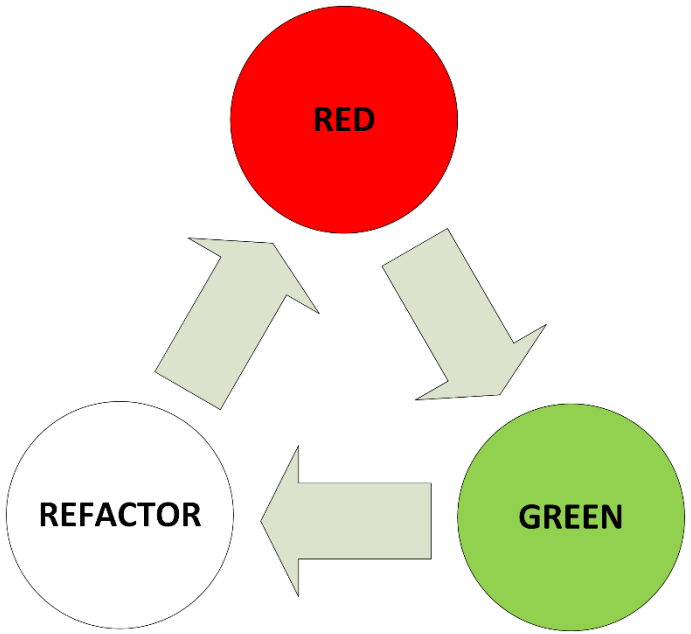
- Write a Test-list
  - What do we think we need to test for
- Writing **Test Code before Production Code**
- Follow Bob Martin's Three Laws of TDD[1]:

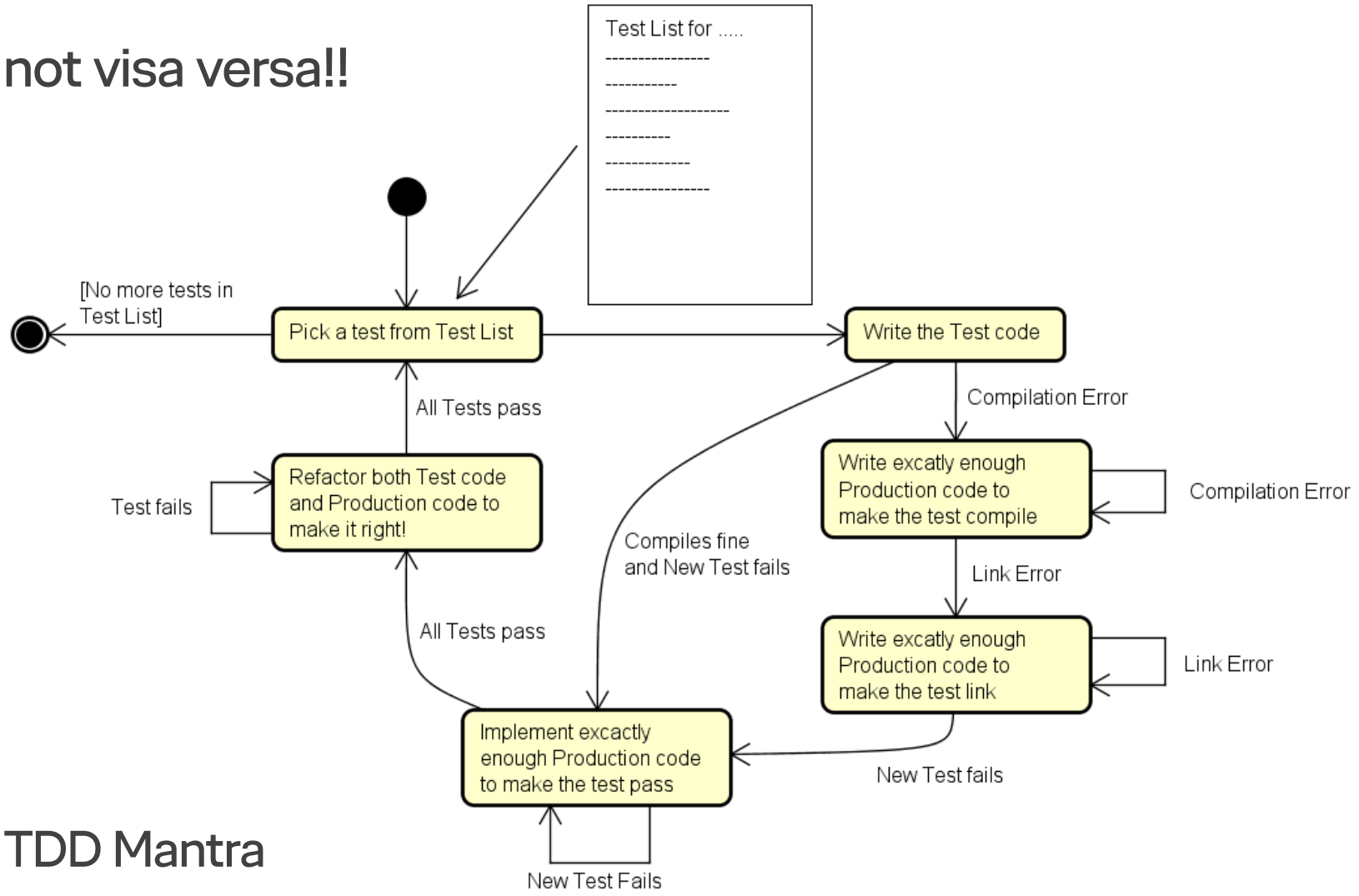
1. *You are **not** allowed to write any production code unless it is to make a failing unit test pass*
2. *You are **not** allowed to write any more of a unit test than is enough to fail; and compilation failures are failures*
3. *You are **not** allowed to write any more production code than is enough to pass the one failing unit test*

[1]http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

What is Test Driven Development (TDD)?

Let **code follow test** – not visa versa!!





RED-GREEN-REFACTOR – the TDD Mantra

Why Test Driven Development (TDD)?

Fun way to work

Testing is boring

Working code is NOT a gamble

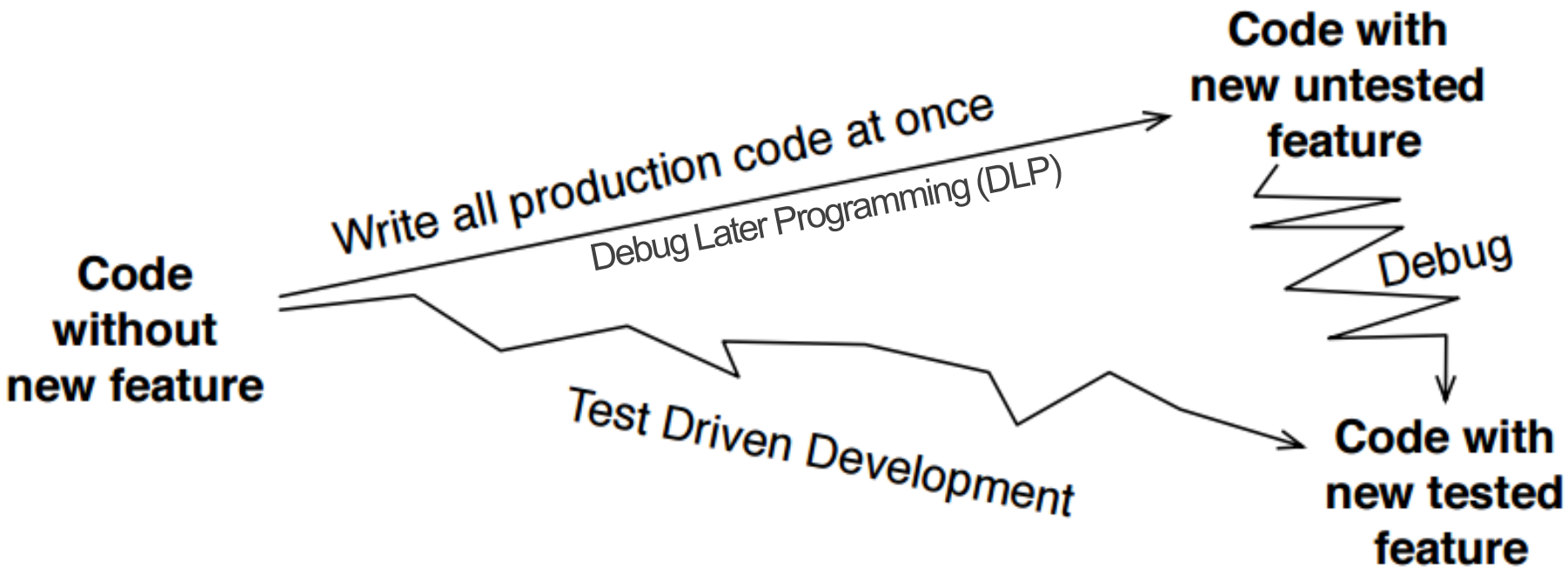
Combat Writer's blockage

Maintainable code

Predictable work pace

Debugging is difficult

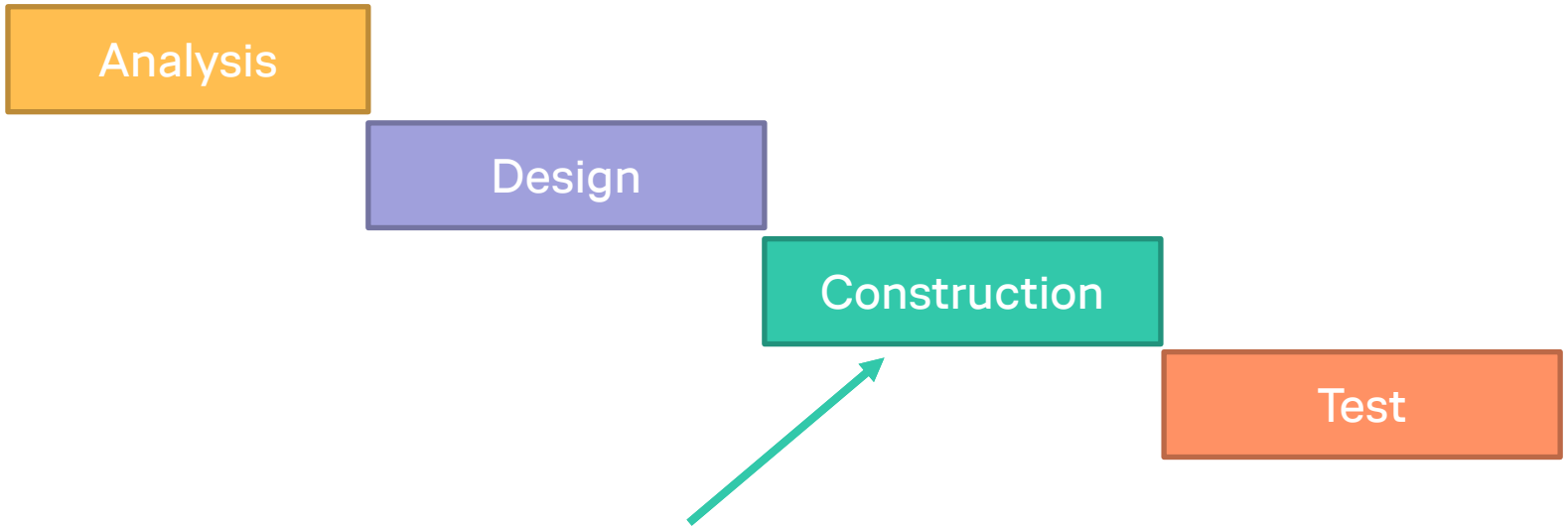
Different Approaches to Development



What is TDD?

What is TDD – nothing to do with testing, it has all to do with implementing!

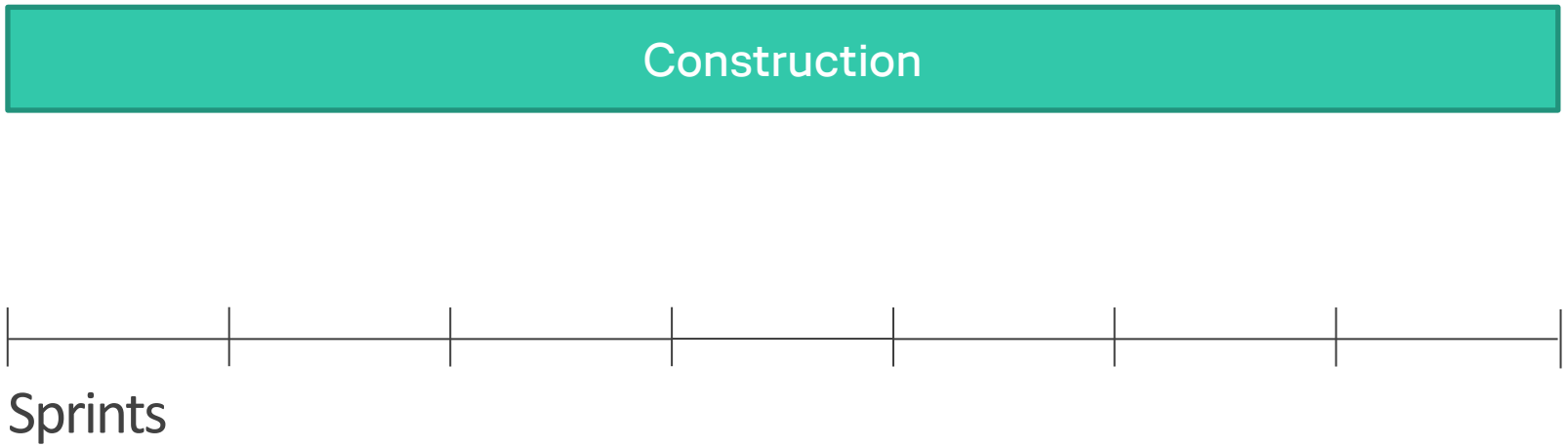
Some of the UP disciplines



Here you can do Test Driven Development

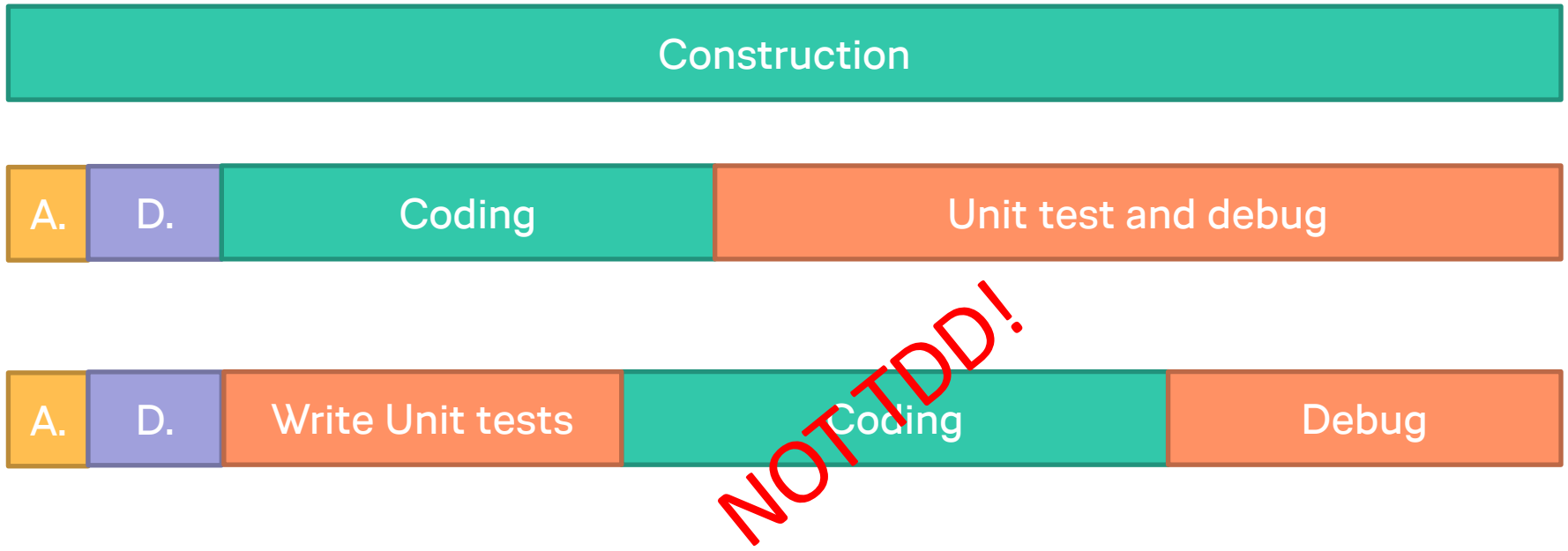
What is TDD

Used in the Construction discipline

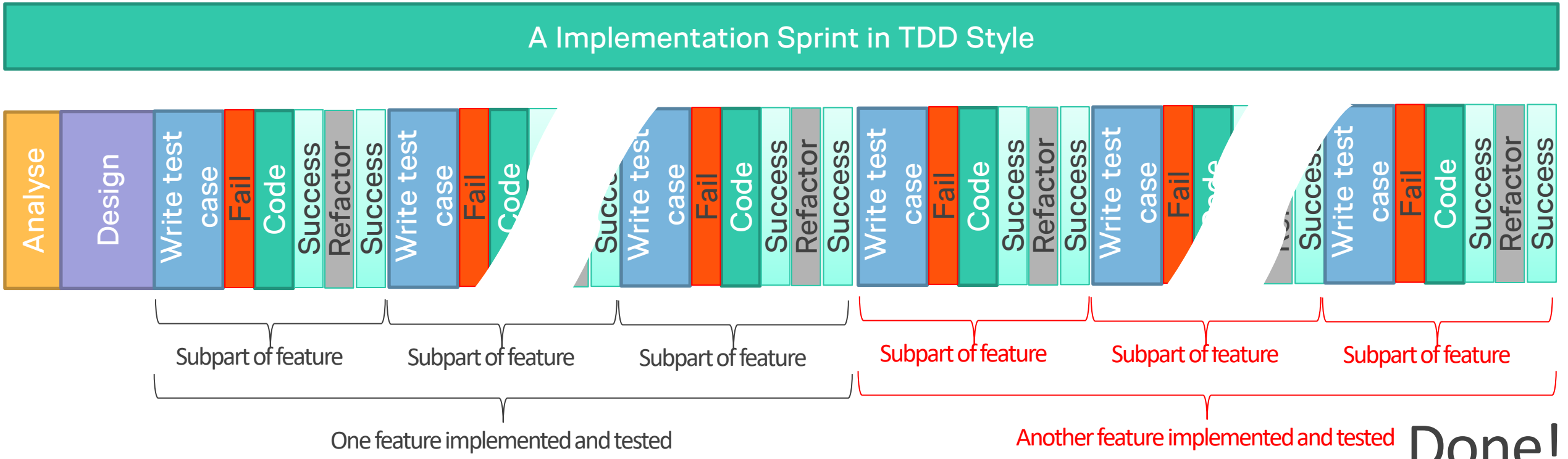


What TDD is not!

One implementation sprint in **non**-TDD style



What is TDD?



Done!

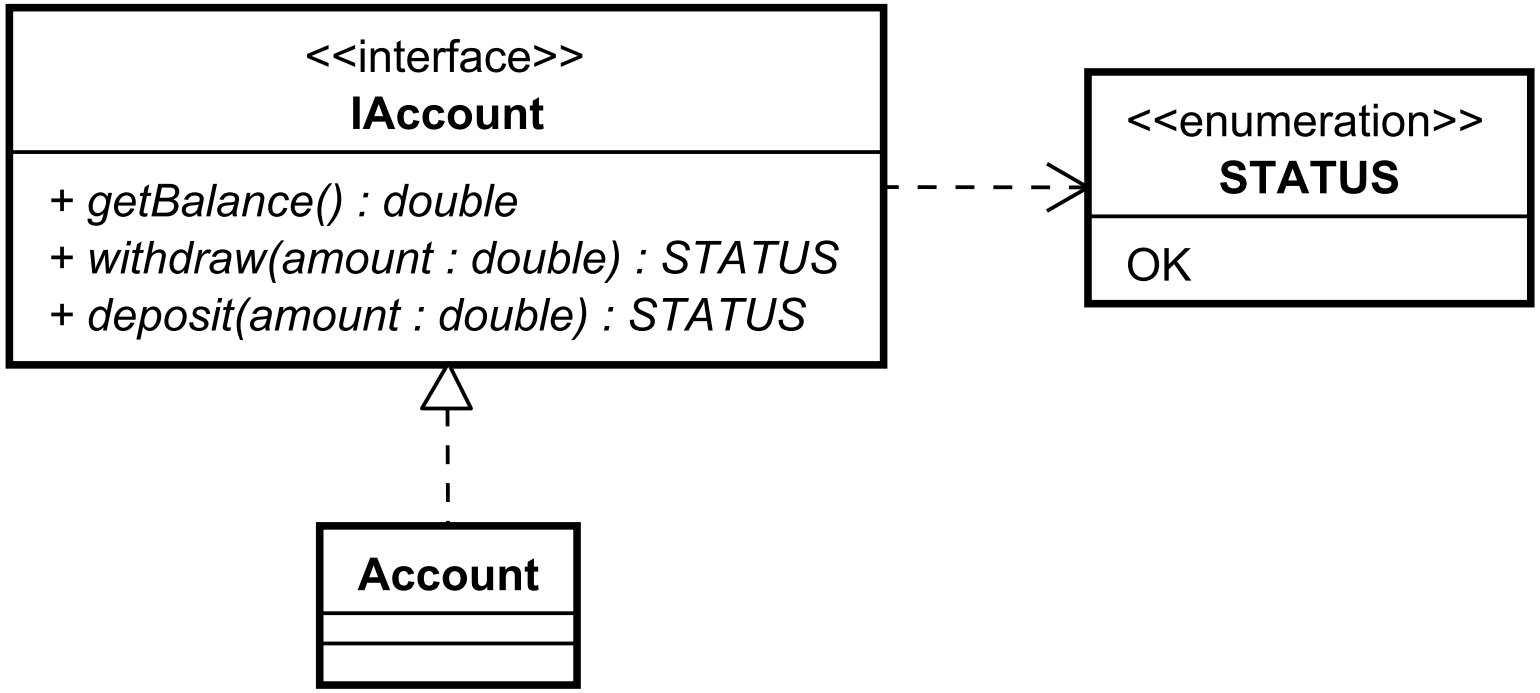
It works!

You can prove it

Are you too busy to improve?

A worked example/TDD Demo – an Account

The draft design



Account Test List

1. Account balance must be zero when the account is created
2. It is not possible to deposit a negative amount
3. Positive deposit is OK
4. It is not possible to withdraw a negative amount
5. Money can be withdrawn from the account
6. The account must never be negative
7. Interest can be added to the account
8. … more …

Exercise – Going from Design to C-code Part II

Do Exercise 6.1 "Linked List" and implement it as an Abstract Data Type (ADT)

USE TEST DRIVEN DEVELOPMENT (TDD)!!

Note:

A *void* * in C is comparable with a reference to *Object* in Java

It is a pointer that can point to anything like *Object* can refer to anything in Java

In both cases we **must** cast it to make it point/refer to what we are interested in

Exercise – Going from Design to C-code Part II

Do Exercise 6.2 “Use of your linked list”

This is the Home Work for next Week!!