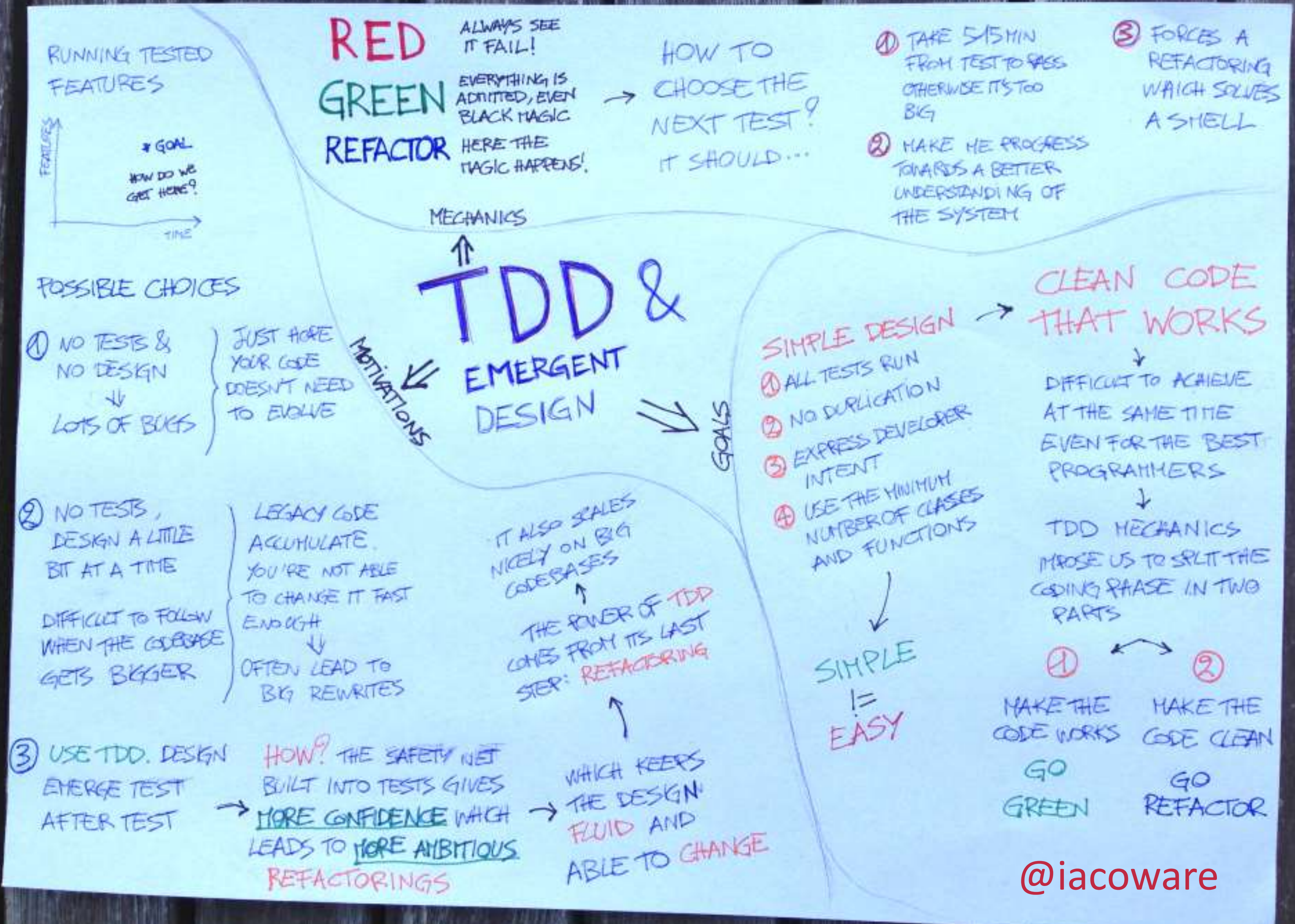
WRITING GOOD TESTS

TEST-DRIVEN DEVELOPMENT



TDD & EMERGENT DESIGN

RUNNING TESTED FEATURES

MECHANICS

RED — ALWAYS SEE IT FAIL!

GREEN — EVERYTHING IS ADMITTED, EVEN BLACK MAGIC

REFACTOR — HERE THE MAGIC HAPPENS!

→ HOW TO CHOOSE THE NEXT TEST?

IT SHOULD...

① TAKE 5/15 MIN FROM TEST TO PASS OTHERWISE IT'S TOO BIG

② MAKE ME PROGRESS TOWARDS A BETTER UNDERSTANDING OF THE SYSTEM

③ FORCES A REFACTORING WHICH SOLVES A SMELL

MOTIVATIONS

POSSIBLE CHOICES

① NO TESTS & NO DESIGN ⇓ LOTS OF BUGS } JUST HOPE YOUR CODE DOESN'T NEED TO EVOLVE

② NO TESTS, DESIGN A LITTLE BIT AT A TIME

DIFFICULT TO FOLLOW WHEN THE CODEBASE GETS BIGGER } LEGACY CODE ACCUMULATE. YOU'RE NOT ABLE TO CHANGE IT FAST ENOUGH ⇓ OFTEN LEAD TO BIG REWRITES

③ USE TDD. DESIGN EMERGE TEST AFTER TEST → HOW? THE SAFETY NET BUILT INTO TESTS GIVES MORE CONFIDENCE WHICH LEADS TO MORE AMBITIOUS REFACTORINGS → WHICH KEEPS THE DESIGN FLUID AND ABLE TO CHANGE

↑

THE POWER OF TDD COMES FROM ITS LAST STEP: REFACTORING

↑

IT ALSO SCALES NICELY ON BIG CODEBASES

GOALS

SIMPLE DESIGN → CLEAN CODE THAT WORKS

① ALL TESTS RUN

② NO DUPLICATION

③ EXPRESS DEVELOPER INTENT

④ USE THE MINIMUM NUMBER OF CLASSES AND FUNCTIONS

↓

SIMPLE != EASY

CLEAN CODE THAT WORKS

↓

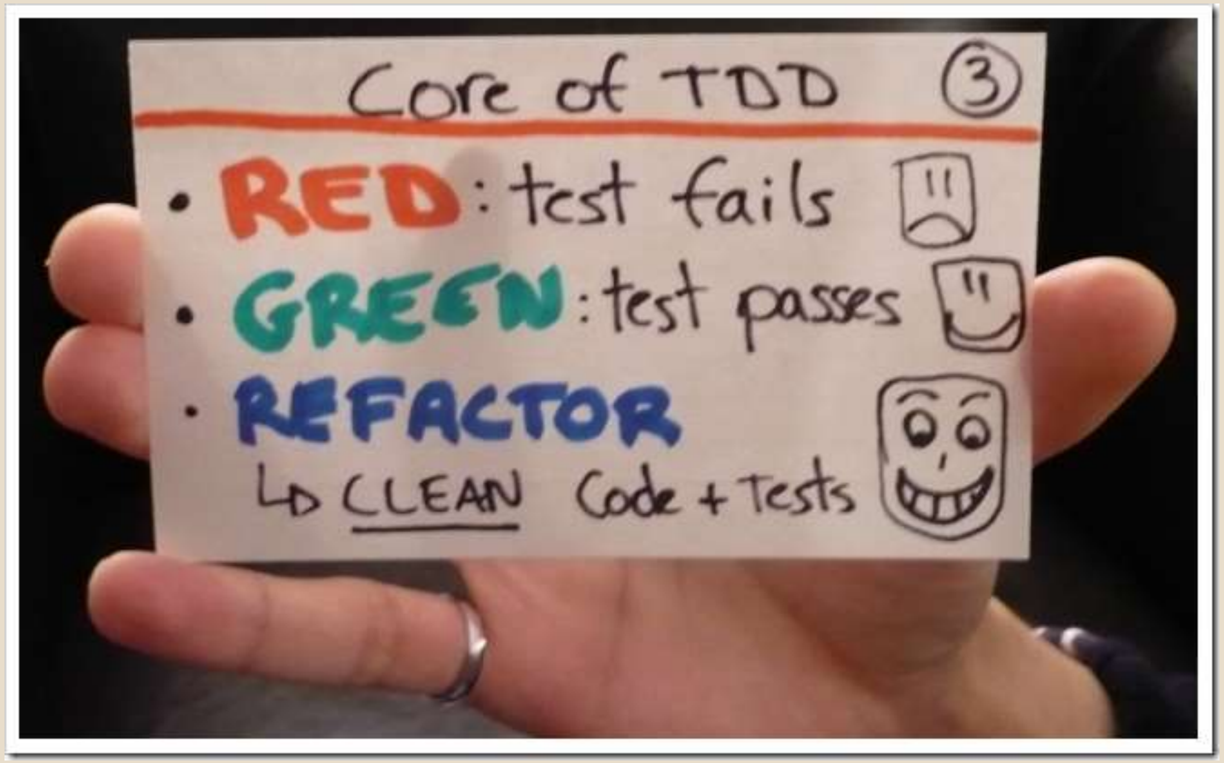
DIFFICULT TO ACHIEVE AT THE SAME TIME EVEN FOR THE BEST PROGRAMMERS

↓

TDD MECHANICS IMPOSE US TO SPLIT THE CODING PHASE IN TWO PARTS

① MAKE THE CODE WORKS — GO GREEN ⟷ ② MAKE THE CODE CLEAN — GO REFACTOR

@iacoware


Core of TDD ③

- RED: test fails
- GREEN: test passes
- REFACTOR
  ↳ CLEAN Code + Tests

TDD MECHANICS

TDD IS A MULTIFACETED SKILL

SKILL ON

CLEAN CODE

SKILL ON

REFACTORING SAFELY

SKILL ON

DISCOVER TEST CASES

SKILL ON

CHOOSE NEXT TESTS

SKILL ON WRITING TESTS

SKILL ON WRITING GOOD TESTS

FOCUS ON

WRITING GOOD TESTS

PROPERTIES

TESTS MUST BE

ISOLATED

TESTS MUST BE

INDIPENDENT

TESTS MUST BE

DETERMINISTIC

AFFECT
EXECUTION

WHAT ABOUT TEST CODE?

BAD CODE SMELLS

BAD
TEST CODE
SMELLS

LEAD TO

FRAGILITY

ON REFACTORING

WHY SHOULD WE CARE?

WHAT'S AGILE PRACTICES' ULTIMATE GOAL?

REDUCE
THE COST
AND RISK
OF CHANGE

FRAGILE TESTS

INCREASE COSTS

FRAGILE TESTS

LOSS OF CONFIDENCE

FRAGILE TESTS REDUCE SUSTAINABILITY

PAY OFF YOUR TECHNICAL DEBT

WHERE IS COMPLEXITY?

```
[Fact]
public void ScenarioUnderTest()
{
    // Arrange
    // Act
    // Assert
}
```

START

FROM

ASSERT

01

COMPARING OBJECTS

```csharp
[Fact]
public void Subtract()
{
  var one = TimeRange.Parse("09:00-10:00");
  var two = TimeRange.Parse("09:15-09:45");

  var result = one.Subtract(two);

  Assert.Equal(2, result.Length);
  Assert.Equal(TimeSpan.Parse("09:00"), result[0].Begin);
  Assert.Equal(TimeSpan.Parse("09:15"), result[0].End);
  Assert.Equal(TimeSpan.Parse("09:45"), result[1].Begin);
  Assert.Equal(TimeSpan.Parse("10:00"), result[1].End);
}
```

```csharp
[Fact]
public void Subtract()
{
  var one = TimeRange.Parse("09:00-10:00");
  var two = TimeRange.Parse("09:15-09:45");

  var result = one.Subtract(two);

  Assert.Equal(2, result.Length);
  Assert.Equal(TimeSpan.Parse("09:00"), result[0].Begin);
  Assert.Equal(TimeSpan.Parse("09:15"), result[0].End);
  Assert.Equal(TimeSpan.Parse("09:45"), result[1].Begin);
  Assert.Equal(TimeSpan.Parse("10:00"), result[1].End);
}
```

WHICH KIND OF OBJECTS?

VALUE OBJECTS

VALUE OBJECTS

MODEL FIXED QUANTITIES

VALUE OBJECTS

IMMUTABLE

VALUE OBJECTS

NO IDENTITY

VALUE OBJECTS

EQUAL IF THEY HAVE SAME STATE

```
[Fact]
public void Subtract()
{
  var one = TimeRange.Parse("09:00-10:00");
  var two = TimeRange.Parse("09:15-09:45");

  var result = one.Subtract(two);

  Assert.Equal(2, result.Length);
  Assert.Equal(TimeSpan.Parse("09:00"), result[0].Begin);
  Assert.Equal(TimeSpan.Parse("09:15"), result[0].End);
  Assert.Equal(TimeSpan.Parse("09:45"), result[1].Begin);
  Assert.Equal(TimeSpan.Parse("10:00"), result[1].End);
}
```

```csharp
[Fact]
public void Equality()
{
  var slot = TimeRange.Parse("09:00-10:00");
  var same = TimeRange.Parse("09:00-10:00");
  var differentBegin = TimeRange.Parse("08:00-10:00");
  var differentEnd = TimeRange.Parse("09:00-13:00");

  Assert.Equal(slot, same);
  Assert.NotEqual(slot, differentBegin);
  Assert.NotEqual(slot, differentEnd);
}
```

```csharp
[Fact]
public void Subtract()
{
  var one = TimeRange.Parse("09:00-10:00");
  var two = TimeRange.Parse("09:15-09:45");

  var result = one.Subtract(two);

  Assert.Equal(2, result.Length);
  Assert.Equal(TimeSpan.Parse("09:00"), result[0].Begin);
  Assert.Equal(TimeSpan.Parse("09:15"), result[0].End);
  Assert.Equal(TimeSpan.Parse("09:45"), result[1].Begin);
  Assert.Equal(TimeSpan.Parse("10:00"), result[1].End);
}
```

```
[Fact]
public void Subtract()
{
  var one = TimeRange.Parse("09:00-10:00");
  var two = TimeRange.Parse("09:15-09:45");

  var result = one.Subtract(two);

  Assert.Contains(TimeRange.Parse("09:00-09:15"), result);
  Assert.Contains(TimeRange.Parse("09:45-10:00"), result);
}
```

OBJECTS

OBJECTS

MODEL STATE AND PROCESSING OVER TIME

OBJECTS

MUTABLE

OBJECTS

HAS IDENTITY

OBJECTS

EQUAL IF THEY HAVE SAME IDENTITY

```
[Fact]
public void GetAllMessages()
{
  var messageId1 = Guid.NewGuid();
  var messageId2 = Guid.NewGuid();
  var store = new MessageStore();

  store.Add(DummyMessage(messageId1));
  store.Add(DummyMessage(messageId2));
  var result = store.GetAll();

  Assert.Equal(2, result.Count);
  Assert.NotNull(result.Single(x => x.Id == messageId1));
  Assert.NotNull(result.Single(x => x.Id == messageId2));
}
```

```
[Fact]
public void GetAllMessages()
{
  var messageId1 = Guid.NewGuid();
  var messageId2 = Guid.NewGuid();
  var store = new MessageStore();

  store.Add(DummyMessage(messageId1));
  store.Add(DummyMessage(messageId2));
  var result = store.GetAll();

  Assert.Equal(2, result.Count);
  Assert.NotNull(result.Single(x => x.Id == messageId1));
  Assert.NotNull(result.Single(x => x.Id == messageId2));
}
```

```
[Fact]
public void Equality()
{
  var message = DummyMessage(Guid.NewGuid());
  var same = DummyMessage(message.Id);
  var diffent = DummyMessage(Guid.NewGuid());

  Assert.Equal(message, same);
  Assert.NotEqual(message, diffent);
}
```

```
[Fact]
public void GetAllMessages()
{
  var messageId1 = Guid.NewGuid();
  var messageId2 = Guid.NewGuid();
  var store = new MessageStore();

  store.Add(DummyMessage(messageId1));
  store.Add(DummyMessage(messageId2));
  var result = store.GetAll();

  Assert.Equal(2, result.Count);
  Assert.NotNull(result.Single(x => x.Id == messageId1));
  Assert.NotNull(result.Single(x => x.Id == messageId2));
}
```

```
[Fact]
public void GetAllMessages()
{
  var messageId1 = Guid.NewGuid();
  var messageId2 = Guid.NewGuid();
  var store = new MessageStore();

  store.Add(DummyMessage(messageId1));
  store.Add(DummyMessage(messageId2));
  var result = store.GetAll();

  Assert.Contains(DummyMessage(messageId1), result);
  Assert.Contains(DummyMessage(messageId2), result);
}
```

WHEN YOU CAN'T CHANGE THE OBJECT?

```csharp
[Fact]
public void GetContactsDetails()
{
  var controller = new ContactsController();

  var result = controller.Details(Guid.NewGuid());

  var viewResult = Assert.IsType<ViewResult>(result);
  Assert.Equal(String.Empty, viewResult.ViewName);
  Assert.IsAssignableFrom<ContactModel>(viewResult.Model);
}
```

```
[Fact]
public void GetContactsDetails()
{
  var controller = new ContactsController();

  var result = controller.Details(Guid.NewGuid());

  var viewResult = Assert.IsType<ViewResult>(result);
  Assert.Equal(String.Empty, viewResult.ViewName);
  Assert.IsAssignableFrom<ContactModel>(viewResult.Model);
}
```

```
[Fact]
public void GetContactsDetails()
{
  var controller = new ContactsController();

  var result = controller.Details(Guid.NewGuid());

  MvcAssert.IsViewResult<ContactModel>(result);
}
```

02

TESTING ONLY ONE CONCERN

```
[Fact]
public void SingleElement()
{
  var input = "<element />";
  var tag = Tag.Parse(input);

  Assert.True(tag.IsEmpty());
  Assert.Equal("element", tag.Name());
  Assert.Empty(tag.ChildNodes());
}
```

```
[Fact]
public void SingleElement()
{
  var input = "<element />";
  var tag = Tag.Parse(input);

  Assert.True(tag.IsEmpty());
  Assert.Equal("element", tag.Name());
  Assert.Empty(tag.ChildNodes());
}
```

TEST CAN
FAIL FOR
TOO MANY
REASONS

```
[Fact]
public void SingleElement()
{
  var input = "<element />";
  var tag = Tag.Parse(input);

  Assert.True(tag.IsEmpty());
  Assert.Equal("element", tag.Name());
  Assert.Empty(tag.ChildNodes());
}
```

```
[Fact]
public void SingleElementNoContent()
{
  var input = "<element />";
  var tag = Tag.Parse(input);

  Assert.True(tag.IsEmpty());
}
```

```
[Fact]
public void SingleElementName()
{
  var input = "<element />";
  var tag = Tag.Parse(input);

  Assert.Equal("element", tag.Name());
}
```

```
[Fact]
public void SingleElementNoChildren()
{
  var input = "<element />";
  var tag = Tag.Parse(input);

  Assert.Empty(tag.ChildNodes());
}
```

03
TESTING BEHAVIOUR

STATE

VS

BEHAVIOUR

FOCUS ON
TESTING STATE

```
[Fact]
public void NewStackIsEmpty()
{
  var stack = new Stack();

  Assert.True(stack.IsEmpty());
}
```

```
[Fact]
public void StackIsNotEmptyAfterAddItem()
{
  var stack = new Stack();

  stack.Push(5);

  Assert.False(stack.IsEmpty());
}
```

```
[Fact]
public void StackCountIncreaseAfterAddItems()
{
  var items = new[] { 1, 2, 3 };
  var stack = new Stack();

  stack.Push(items[0]);
  stack.Push(items[1]);
  stack.Push(items[2]);

  Assert.Equal(items.Length, stack.Count());
}
```

FOCUS ON

TESTING BEHAVIOUR

```csharp
[Fact]
public void PopOneItem()
{
  var item = 5;
  var stack = new Stack();

  stack.Push(item);

  Assert.Equal(item, stack.Pop());
}
```

```
[Fact]
public void PopManyItems()
{
  var items = new[] { 1, 2, 3 };
  var stack = new Stack();

  stack.Push(items[0]);
  stack.Push(items[1]);
  stack.Push(items[2]);

  Assert.Equal(items.Reverse(),
      new[] { stack.Pop(), stack.Pop(), stack.Pop() });
}
```

```
[Fact]
public void PopEmptyStack()
{
  var stack = new Stack();
  Assert
    .Throws<InvalidOperationException>(() => stack.Pop());
}
```

04

AVOID TESTING PRIVATE METHODS

```csharp
public class Invoice
{
  // ...

  public Eur Total()
  {
    var gross = ComputeGrossAmount(/* args */);
    var tax = ComputeTaxAmount(/* args */);
    return gross.Add(tax);
  }
}
```

```
public class Invoice
{
  // ...

  Eur ComputeGrossAmount(/* args */)
  {
    /* many lines of complex
     * and commented code */
  }

}
```

```
public class Invoice
{
  // ...

  Eur ComputeTaxAmount(/* args */)
  {
    /* many lines of complex
     * and commented code */
  }

}
```

```
[Fact]
public void GrossAmountCalculation()
{
  // We want invoke private ComputeGrossAmount method
}

[Fact]
public void TaxAmountCalculation()
{
  // We want invoke private ComputeTaxAmount method
}
```

DON'T DO IT!

DON'T DO IT!

DON’T DO IT!

CONSIDER APPLYING

METHOD

OBJECT

MAKE AN OBJECT OUT OF THE METHOD

```
public class Invoice
{
  // ...

  public Eur Total()
  {
    var gross = new GrossAmount(/* args */).Compute();
    var tax = new TaxAmount(/* args */).Compute();
    return gross.Add(tax);
  }
}
```

NOW TEST EXTRACTED OBJECTS

05

SAME LEVEL OF ABSTRACTION

```
[Fact]
public void SetCommand()
{
  var cache = new Cache();
  var dispatcher = new CommandDispatcher(cache);

  dispatcher.Process("SET foo bar");
  var actual = cache.Get("foo");

  Assert.Equal("bar", actual);
}
```

```
[Fact]
public void SetCommand()
{
  var cache = new Cache();
  var dispatcher = new CommandDispatcher(cache);

  dispatcher.Process("SET foo bar");
  var actual = cache.Get("foo");

  Assert.Equal("bar", actual);
}
```

ACT AT HIGHER

ASSERT AT LOWER

REMEMBER DIP?

DON'T CROSS THE STREAM

```csharp
[Fact]
public void SetCommand()
{
  var cache = new Cache();
  var dispatcher = new CommandDispatcher(cache);

  dispatcher.Process("SET foo bar");
  var actual = cache.Get("foo");

  Assert.Equal("bar", actual);
}
```

```csharp
[Fact]
public void SetCommand()
{
  var cache = new Cache();
  var dispatcher = new CommandDispatcher(cache);

  dispatcher.Process("SET foo bar");
  var actual = dispatcher.Process("GET foo");

  Assert.Equal("bar", actual);
}
```

WHEN IT ISN'T YOUR RESPONSIBILITY?

```
[Fact]
public void SetCommand()
{
  var log = new TransactionLog();
  var cache = new PersistentCache(log);

  cache.Set("foo", "bar");
  var written = log.History().Take(1).Single();

  Assert.Equal("SET foo bar\r\n", written);
}
```

```
[Fact]
public void SetCommand()
{
  var log = new TransactionLog();
  var cache = new PersistentCache(log);

  cache.Set("foo", "bar");
  var written = log.History().Take(1).Single();

  Assert.Equal("SET foo bar\r\n", written);
}
```

PURE FABRICATION

```csharp
[Fact]
public void SetCommand()
{
  var log = new TransactionLog();
  var cache = new PersistentCache(log);

  cache.Set("foo", "bar");
  var written = log.History().Take(1).Single();

  Assert.Equal("SET foo bar\r\n", written);
}
```

```
[Fact]
public void SetCommand()
{
  var log = new TransactionLog();
  var cache = new PersistentCache(log);



  cache.Set("foo", "bar");
  var written = log.History().Take(1).Single();

  Assert.Equal("SET foo bar\r\n", written);
}
```

```
[Fact]
public void SetCommand()
{
  var log = new TransactionLog();
  var cache = new PersistentCache(log);
  var monitor = new Monitor(log);

  cache.Set("foo", "bar");
  var written = log.History().Take(1).Single();

  Assert.Equal("SET foo bar\r\n", written);
}
```

```
[Fact]
public void SetCommand()
{
  var log = new TransactionLog();
  var cache = new PersistentCache(log);
  var monitor = new Monitor(log);

  cache.Set("foo", "bar");
  var written = monitor.LastLog();

  Assert.Equal("SET foo bar\r\n", written);
}
```

WHEN SIDE EFFECTS AREN'T DIRECTLY RELATED?

```csharp
[Fact]
public void CommandStatistics()
{
  var channel = new StatisticsChannel();
  var cache = new Cache(channel);
  var dashboard = new RealTimeDashboard(channel);

  cache.Set("foo", "bar");
  var received = dashboard.LastReceived();

  Assert.Equal("NAME: SET; ARGS: foo bar", received);
}
```

```
[Fact]
public void CommandStatistics()
{
  var channel = new StatisticsChannel();
  var cache = new Cache(channel);
  var dashboard = new RealTimeDashboard(channel);

  cache.Set("foo", "bar");
  var received = dashboard.LastReceived();

  Assert.Equal("NAME: SET; ARGS: foo bar", received);
}
```

BREAK
THE FLOW

OBSERVER PLUS SELF SHUNT

```
public interface ICacheSubscriber
{
  void NotifyCommandExecuted(string name, string args);
}
```

```csharp
public class CacheTests : ICacheSubscriber
{
  string ExecutedCommandName;
  string ExecutedCommandArgs;

  public void NotifyCommandExecuted(string name,
                                    string args)
  {
    ExecutedCommandName = name;
    ExecutedCommandArgs = args;
  }

  // ...
}
```

```csharp
[Fact]
public void CommandStatistics()
{
  var channel = new StatisticsChannel();
  var cache = new Cache(channel);
  var dashboard = new RealTimeDashboard(channel);

  cache.Set("foo", "bar");
  var received = dashboard.LastReceived();

  Assert.Equal("NAME: SET; ARGS: foo bar", received);
}
```

```
[Fact]
public void CommandStatistics()
{

  var cache = new Cache(channel);



  cache.Set("foo", "bar");



  Assert.Equal("NAME: SET; ARGS: foo bar", received);
}
```

```
[Fact]
public void CommandStatistics()
{

  var cache = new Cache(channel);

  cache.Set("foo", "bar");


  Assert.Equal("NAME: SET; ARGS: foo bar", received);
}
```

```
[Fact]
public void CommandStatistics()
{
  ICacheSubscriber subscriber = this;
  var cache = new Cache(subscriber);

  cache.Set("foo", "bar");



  Assert.Equal("NAME: SET; ARGS: foo bar", received);
}
```

```csharp
[Fact]
public void NotifyCommandExecution()
{
  ICacheSubscriber subscriber = this;
  var cache = new Cache(subscriber);

  cache.Set("foo", "bar");

  Assert.Equal("SET", this.ExecutedCommandName);
  Assert.Equal("foo bar", this.ExecutedCommandArgs);
}
```

06
RESPECT ABSTRACTION

```
[Fact]
public void Overcrowding()
{
  var moreThanThreeNeighbors = 6;
  var cell = new Cell(alive: true);

  var alive = cell.StayAlive(moreThanThreeNeighbors);

  Assert.False(alive);
}
```

```
[Fact]
public void Overcrowding()
{
  var moreThanThreeNeighbors = 6;
  var cell = new Cell(alive: true);

  var alive = cell.StayAlive(moreThanThreeNeighbors);

  Assert.False(alive);
}
```

SEGREGATE DECISIONS

```
[Fact]
public void Overcrowding()
{
  var moreThanThreeNeighbors = 6;
  var cell = new Cell(alive: true);

  var alive = cell.StayAlive(moreThanThreeNeighbors);

  Assert.False(alive);
}
```

```
[Fact]
public void Overcrowding()
{
  var moreThanThreeNeighbors = 6;
  var cell = Cell.Live();

  var alive = cell.StayAlive(moreThanThreeNeighbors);

  Assert.False(alive);
}
```

INFORMATION HIDING

07

COMPLEX OBJECT CREATION

```
[Fact]
public void OrderCost()
{
  var order = new Order(Location.TakeAway,
                 new OrderLines(
                   new OrderLine("Latte",
                           new Quantity(1),
                           Size.Small,
                           new Pounds(1, 50)),
                   new OrderLine("Cappuccino",
                           new Quantity(2),
                           Size.Large,
                           new Pounds(2, 50))));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

```
[Fact]
public void OrderCost()
{
  var order = new Order(Location.TakeAway,
                new OrderLines(
                  new OrderLine("Latte",
                          new Quantity(1),
                          Size.Small,
                          new Pounds(1, 50)),
                  new OrderLine("Cappuccino",
                          new Quantity(2),
                          Size.Large,
                          new Pounds(2, 50))));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

EXTRACT
FACTORY METHOD

```
[Fact]
public void OrderCost()
{
  var order = CreateOrderWithLatteAndCappuccino();
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

OBJECT MOTHER

```
[Fact]
public void OrderCost()
{
  var order = TestOrders
                 .CreateOrderWithLatteAndCappuccino();
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

```
Order CreateEmptyOrder() { /* ... */ }
Order CreateEmptyOrder() { /* ... */ }

Order CreateOrderWithLatteAndCappuccino() { /* ... */ }
Order CreateEmptyOrder() { /* ... */ }

Order CreateOrderWithLatteAndCappuccino() { /* ... */ }

Order CreateOrderWithLatteMultipleQuantities() { /* ... */ }
Order CreateEmptyOrder() { /* ... */ }

Order CreateOrderWithLatteAndCappuccino() { /* ... */ }

Order CreateOrderWithLatteMultipleQuantities() { /* ... */ }

Order CreateOrderWithCaffeSpecialDiscount() { /* ... */ }
```

POOR NAME

LOSS OF CONTEXT

DOESN'T SCALE

COMMON
SHARED FIXTURE

```
[Fact]
public void OrderCost()
{
  var order = CreateCommonOrder();
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

```csharp
[Fact]
public void DiscountedOrderCost()
{
  var order = CreateCommonOrder();
  order.Add(Discounts.ByQuantity(2));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

```
[Fact]
public void DiscountedOrderCost()
{
  var order = CreateCommonOrder();
  order.Add(Discounts.ByQuantity(2));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

SAME FAILURE

```
[Fact]
public void OrderCost()
{
  var order = new Order(Location.TakeAway,
                 new OrderLines(
                   new OrderLine("Latte",
                           new Quantity(1),
                           Size.Small,
                           new Pounds(1, 50)),
                   new OrderLine("Cappuccino",
                           new Quantity(2),
                           Size.Large,
                           new Pounds(2, 50))));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

DECOUPLE WHAT IS NEEDED

FROM HOW
IT IS STRUCTURED

BUILDER PATTERN

```
[Fact]
public void OrderCost()
{
  var order = new Order(Location.TakeAway,
                new OrderLines(
                  new OrderLine("Latte",
                          new Quantity(1),
                          Size.Small,
                          new Pounds(1, 50)),
                  new OrderLine("Cappuccino",
                          new Quantity(2),
                          Size.Large,
                          new Pounds(2, 50))));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

```csharp
[Fact]
public void OrderCost()
{
  var order = new OrderBuilder()
          .WithLocation(Location.TakeAway)
          .WithLine(new OrderLineBuilder()
                  .WithName("Latte")
                  .WithPrice(new Pounds(1, 50)))
          .WithLine(new OrderLineBuilder()
                  .WithName("Cappuccino")
                  .WithQuantity(new Quantity(2))
                  .WithSize(Size.Large)
                  .WithPrice(new Pounds(1, 50)));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

STILL

EXPOSE

STRUCTURE

INTRODUCE HIGHER LEVEL METHODS

```
[Fact]
public void OrderCost()
{
  var order = new OrderBuilder()
          .WithLocation(Location.TakeAway)
          .WithLine(new OrderLineBuilder()
                  .WithName("Latte")
                  .WithPrice(new Pounds(1, 50)))
          .WithLine(new OrderLineBuilder()
                  .WithName("Cappuccino")
                  .WithQuantity(new Quantity(2))
                  .WithSize(Size.Large)
                  .WithPrice(new Pounds(1, 50)));
  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

```
[Fact]
public void OrderCost()
{
  var order = new OrderBuilder()
            .TakeAway()
            .WithOne("Latte", "1,50")
            .WithTwoLarge("Cappuccino", "1,50");

  var result = order.Cost();
  Assert.Equal(new Pounds(6, 50), result);
}
```

DOMAIN SPECIFIC LANGUAGE

RECAP

STOP THE MADNESS

TEST CODE

NEED LOVE

LISTEN TO TEST CODE

TOO MUCH INTIMACY

EXPRESS INTENT

FOCUS ON
WHAT

NOT ON

HOW

REMOVE DUPLICATION

INTRODUCE ABSTRACTIONS

RESPECT ABSTRACTION LEVELS

EVOLVE TEST INFRASTRUCTURE

BABY STEPS

CONTINUOUS REFACTORING

IMPROVE YOUR SKILLS

MATTEO BAGLINI
FREELANCE
SOFTWARE DEVELOPER
TECHNICAL COACH
@matteobaglini

Thank You