

# AVOID EXCEPTIONS

```
public void UpdateName(int id, string name)
{
    if (id == 0)
        throw new Exception("Invalid id");

    if (string.IsNullOrEmpty(name))
        throw new Exception("Name can not been empty");

    ...
}
```



You throw a lot of exceptions from your code even for non exceptional situations.

# XTREM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

?

YOAN  
THIRION

?



?

?

?


# COMMAND QUERY SEPARATION

```
public String printRental() { returns a string, this is a query...
    checkRentals();

    var result = new StringBuilder(); changes internal state...
                                    wait, what?

    for (var rental : rentals) {
        if (!calculated) {
            this.amount += rental.getAmount();
        }
        result.append(formatLine(rental));
    }
    result.append(format("Total amount | %f", this.amount));
    calculated = true;

    return result.toString();
}
```



A function should either :

- do something:
  - change the state of an object - a command
- return something:
  - return some information - a query.

# XTRM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

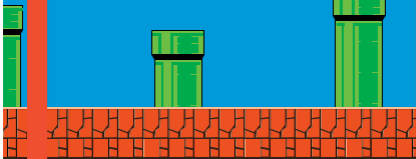
?

YOAN  
THIRION

?



?



# TELL DON'T ASK

Ask data on order to make decision  
Instead of telling what to do...

```
public void Run(OrderApprovalRequest request)
{
    var order = _orderRepository.GetById(request.OrderId);

    if (order.Status == OrderStatus.Shipped)
    {
        throw new ShippedOrdersCannotBeChangedException();
    }

    if (request.Approved && order.Status == OrderStatus.Rejected)
    {
        throw new RejectedOrderCannotBeApprovedException();
    }

    if (!request.Approved && order.Status == OrderStatus.Approved)
    {
        throw new ApprovedOrderCannotBeRejectedException();
    }

    order.Status = request.Approved ? OrderStatus.Approved : OrderStatus.Rejected;
    _orderRepository.Save(order);
}
```



Tell an object what to do rather than asking an object for  
its data and acting on it based on that data.

# XTRM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

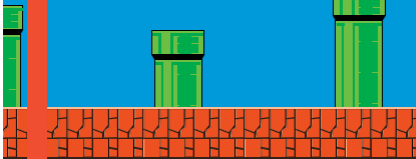
?

YOAN  
THIRION

?



?



# NO ENCAPSULATION

```
namespace Shipping.Domain
{
    public class Order
    {
        public decimal Total { get; set; }
        public string Currency { get; init; }
        public IList<OrderItem> Items { get; init; }
        public decimal Tax { get; set; }
        public OrderStatus Status { get; set; }
        public int Id { get; init; }
    }
}
```



no behavior (methods) in Order...  
with public getters and setters



An object with no domain purpose (just moving data from PointA to PointB) is called a Data Transfer Object.  
If you have this kind of objects in your domain or business layer, you are creating an anemic domain model...

# XTREM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

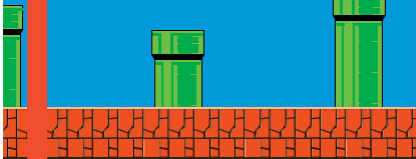
?

YOAN  
THIRION

?



?





# PRIMITIVE OBSESSION

```
public void MoveTo(  
    string address,  
    string postalCode,  
    string city,  
    string country)
```

```
{
```

```
    ...
```

how many parameters can you handle?

```
}
```



All these parameters represent something.  
What could it be?



Primitive obsession is a code smell in which primitive data types are used excessively to represent your data models.

# XTREM SMELLS

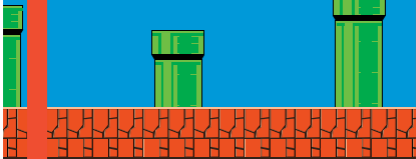
LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

?

YOAN  
THIRION

?



# KEEP DEPENDENCIES UP-TO-DATE

Package	Installed	Released	Latest	Released	Age (y)
AutoMapper	11.0.1	2022-02-04	12.0.1	2023-01-16	0.9
FluentAssertions	6.4.0	2022-01-22	6.11.0	2023-04-20	1.2
Microsoft.AspNetCore.Mvc.Testing	6.0.1	2021-12-14	7.0.5	2023-04-11	1.3
Microsoft.NET.Test.Sdk	16.11.0	2021-08-13	17.6.0	2023-05-16	1.8
Verify.Xunit	16.1.1	2022-02-08	19.14.1	2023-05-02	1.2
xunit	2.4.1	2018-10-29	2.4.2	2022-08-02	3.8
xunit.runner.visualstudio	2.4.3	2020-08-03	2.4.5	2022-05-05	1.8
lstudio					
coverlet.collector	3.1.0	2021-07-19	3.2.0	2022-10-29	1.3

Total is **16.8 libyears behind**



Say what?

17 years behind latest versions of our dependencies?

What would happen if we have to  
update them?



The best way to keep dependencies up-to-date is to  
dedicate time regularly for it.  
Use libyear: simple measure of software dependency  
freshness.

# XTREM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

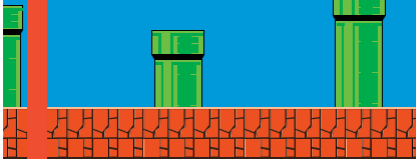
?

YOAN  
THIRION

?



?



# TEST DATA BUILDERS

```
[Fact]
public void Converts_total_amount_to_usd()
{
    // Arrange
    var reportGenerator = new ReportGenerator();
    var novel = new Novel(
        "Le Nœud",
        12.99,
        new Author(
            "Guy de Maupassant",
            new Country("France", Currency.Euro, Language.French)
        ),
        Language.French,
        new List<Genre>
        {
            Genre.Romance
        });

    var educationalBook = new EducationalBook(
        "71848_Code",
        29.87,
        new Author(
            "Uncle Bob",
            new Country("USA", Currency.UsDollar, Language.English)
        ),
        Language.English,
        Category.Computer);

    var novelPurchasedBook = new PurchasedBook(novel, 2);
    var educationalPurchasedBook = new PurchasedBook(educationalBook, 7);

    var invoice = new Invoice(
        "John Doe",
        new Country(
            "France",
            Currency.Euro,
            Language.French
        ));

    // Act
    invoice.AddPurchasedBook(novelPurchasedBook);
    invoice.AddPurchasedBook(educationalPurchasedBook);

    _inMemoryRepository.AddInvoice(invoice);

    // Assert
    reportGenerator.GetTotalAmount().Should().Be(334.97);
    reportGenerator.GetNumberOfIssuedInvoices().Should().Be(1);
    reportGenerator.GetTotalSoldBooks().Should().Be(9);
}
```

Hard to decipher

Long setup,  
so high cognitive load



Irrelevant data  
everywhere

Which data influence the behavior?  
hard to say...



"Test Data Builder eliminates the irrelevant, and amplifies the essentials of the test." - Mark Seemann

# XTRM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

?

YOAN  
THIRION

?



?

?

?

# FACTORY PATTERN

```
public class OrderItem
{
    public Product Product { get; }
    public int Quantity { get; }
    public decimal TaxedAmount { get; }
    public decimal Tax { get; }

    public OrderItem(IProductAPI productApi, CreateOrderItem itemRequest, int quantity)
    {
        var product = productApi.GetByName(itemRequest.Name);
        Product = product ?? throw new UnknownProductException();
        Quantity = quantity;
        Tax = (product.UnitaryTax() * quantity).Round();
        TaxedAmount = (product.UnitaryTaxedAmount() * quantity).Round();
    }
}
```

calls an API in a constructor



what?

this constructor makes a check and can throw an Exception...

it is definitely not the kind of stuff you can expect  
when instantiating an object...

constructors should not contain any logic...  
used only to initialize objects state / fields



Avoiding direct object construction allows us to abstract  
the decision-making process from the calling class.

# XTRM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

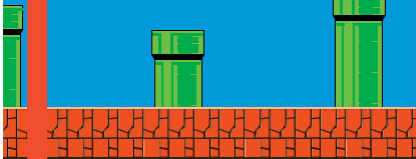
?

YOAN  
THIRION

?



?





# MUTATION TESTING

```
[Fact]
public void Assert_Portfolio_Is_Not_Null()
{
    var portfolio = new Portfolio();
    var result = portfolio.Evaluate(bank, Currency.KRW);

    Assert.NotNull(portfolio);
}
```



WTF? how portfolio could be null?  
what is the behavior we want to test here?

```
@Test
@DisplayName("5 USD + 10 USD = 15 USD")
void shouldAddMoneyInTheSameCurrency() {
    var portfolio = portfolioWith(
        dollars(5),
        dollars(10)
    );

    var result = portfolio.evaluate(bank, USD);
}
```



wait...  
where is the assertion part in this test?



Mutation Testing is a technique which enables us to evaluate the quality of a test suite. It can help reveals the kind of low test quality demonstrated above.

# XTREM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

?

YOAN  
THIRION

?



?

?

?

# PROPERTY-BASED TESTING

```
private static Stream<Arguments> nonPassingExamples() {  
    return Stream.of(  
        of(0),  
        of(4000)  
    );  
}  
  
@ParameterizedTest()  
@MethodSource("nonPassingExamples")  
void returns_empty_for_decimal_out_of_range(int number) {  
    assertThat(convertToRoman(number))  
        .isEmpty();  
}
```



are we confident enough in our production code  
with the examples here?

we can delegate the generation of test cases to  
our computer by using Property-Based Testing.



Property-Based Testing verifies that a function, program  
or any system under test abides by a property.  
We identify and test invariants.

# XTREM SMELLS

LES TONTONS  
CRAFTERS

GUILLAUME  
FAAS

?

YOAN  
THIRION

?



?

