TDD with ASP.Net MVC

Finn Vilsbæk

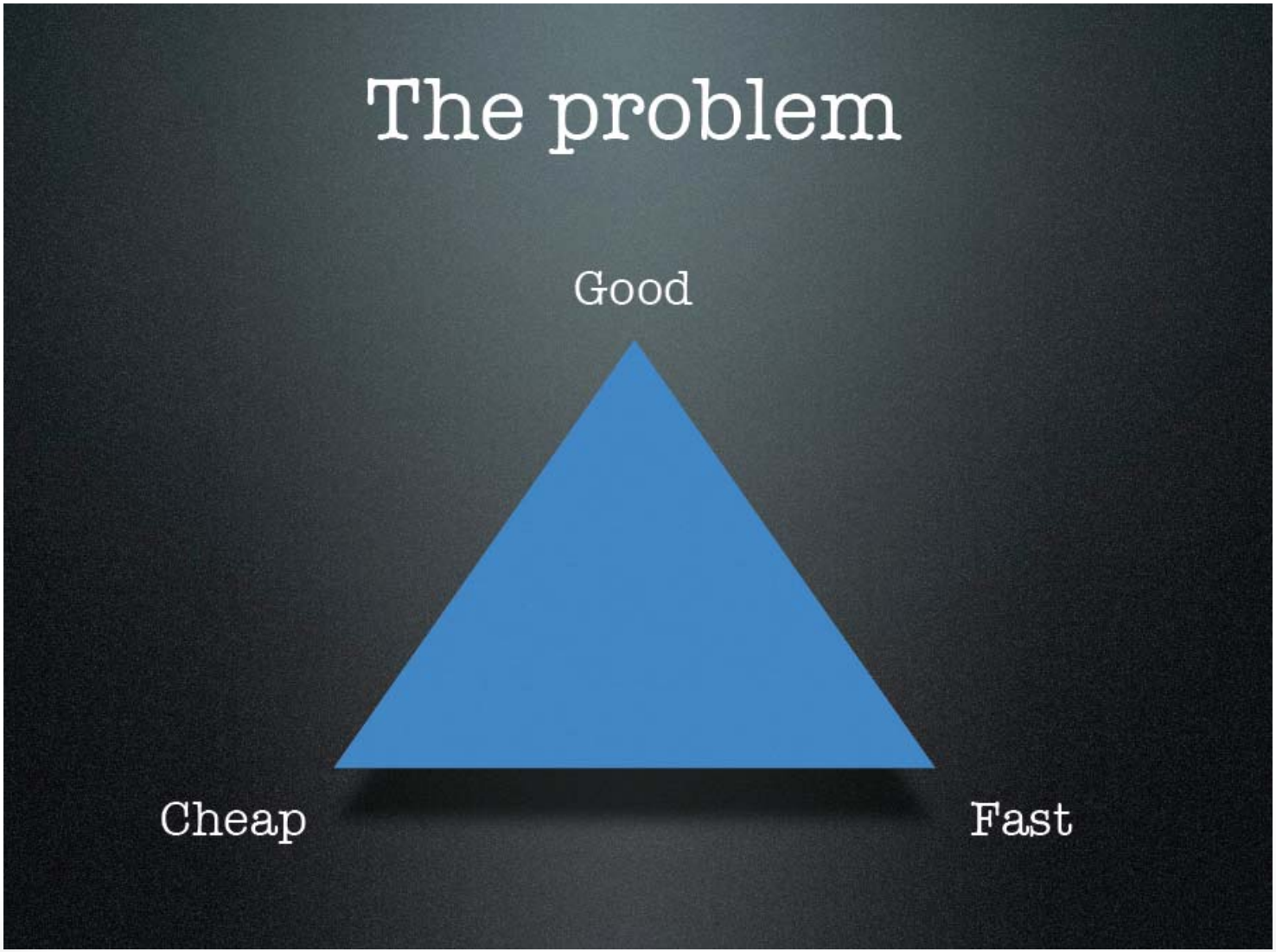
fvs@panmedia.dk

Subjects

- What is TDD?
- What is TDD in the context of ASP.Net MVC?
- How can it work for the business?
- How can it work for the programmer?
- How do we implement it in our company?

- **Literature: Sanderson and Freeman: *Pro ASP.Net MVC 3 Framework, unit tests in ch. 7, 8, 9 , various research papers and videos from TekPub.com***

What is TDD?



What is TDD?

- *" take small steps when writing software"*

What is TDD?

- *Red-Green-Refactor* workflow:

- 1 – Write a unit test that fails (no implementation)
- 2 – Implement just enough to make the test ‘go green’
- 3 - Refactor – remove duplicate code, etc.
- 4 – If the test fails – write code enough to make it pass
- 5 – Next test

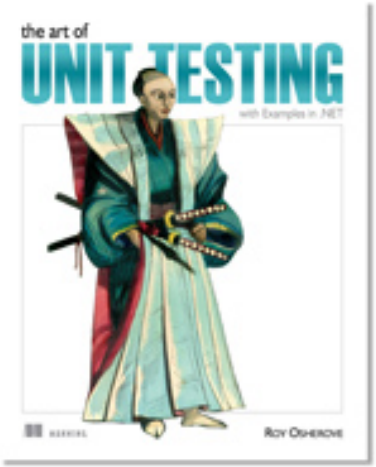
What is TDD?

- '*The classic definition*', Wikipedia:
- A **unit test** is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions turn out to be wrong, the unit test has failed.
- A "**unit**" is a **method** or **function**.

- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

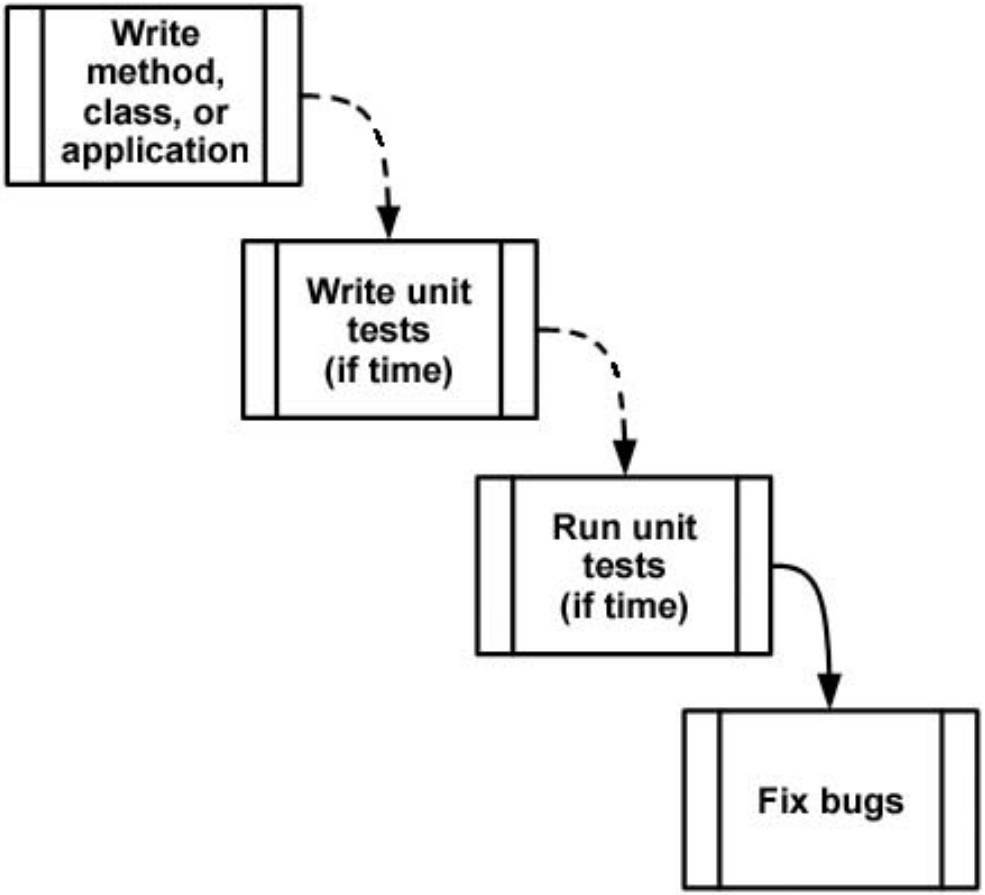
What is TDD?

What is TDD?



- Some quotes:
- *"Most people who try to unit-test their code either give up at some point or don't actually perform unit tests. Instead, they either rely on system and integration tests to be performed much later in the product lifecycle or they resort to manually testing the code via custom test applications or by using the end product they're developing to invoke their code."*
- *" Have you ever met a developer who has not tested his code before handing it over? Well, neither have I."*
- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

What is ~~TDD~~? © (..dotted lines optional..)



- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

What is TDD?

- A unit test should have the following properties:

- ❂ It should be automated and repeatable.
- ❂ It should be easy to implement.
- ❂ Once it's written, it should remain for future use.
- ❂ Anyone should be able to run it.
- ❂ It should run at the push of a button.
- ❂ It should run quickly.

- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

What is TDD?

- Ask yourself these questions about the tests you've written up to now:

- ❂ Can I run and get results from a unit test I wrote two weeks or months or years ago?
- ❂ Can any member of my team run and get the results from unit tests I wrote two months ago?
- ❂ Can I run all the unit tests I've written in no more than a few minutes?
- ❂ Can I run all the unit tests I've written at the push of a button?
- ❂ Can I write a basic unit test in no more than a few minutes?

- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

What is TDD?

- If you've answered "no" to any of these questions, there's a high probability that what you're implementing isn't a unit test. It's definitely some kind of test, and it's as important as a unit test, but it has drawbacks compared to tests that would let you answer "yes" to all of those questions.

- "What was I doing until now?" you might ask. You've done

  integration testing.

- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

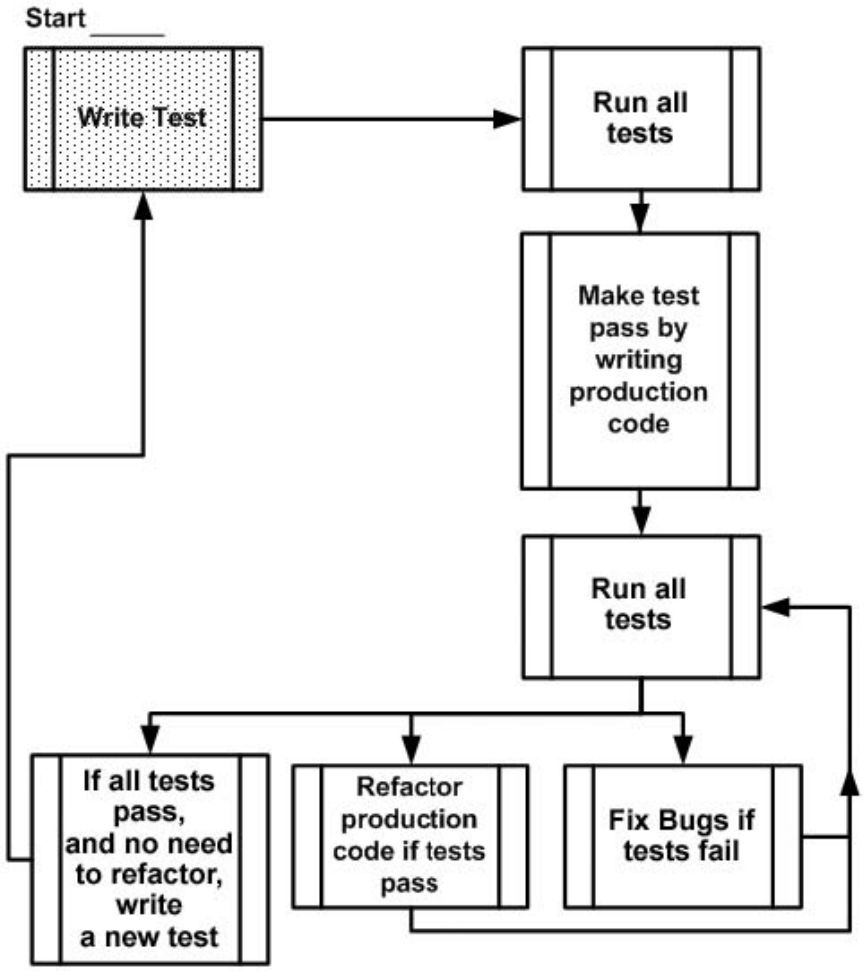
What is TDD?

- *'The extended definition':*

- "A **unit test** is an automated piece of code that invokes the method or class being tested and then checks some assumptions about the logical behavior of that method or class. A unit test is almost always written using a **unit-testing framework**. It can be written easily and runs quickly. It's **fully automated**, trustworthy, readable, and maintainable."

- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

What is TDD – workflow:



- (Source: The Art of Unit Testing http://www.manning.com/osherove/)

Why MVC for TDD?

- MVC forces separation of concerns:
- Domain model and controller logic is *decoupled* from the UI
- HTML is separated from the rest of the application
- Test and maintenance of the code becomes simpler and easier

The MVC Pattern

- Interactions with an MVC application follow a natural cycle of user actions and view updates, where the view is assumed to be *stateless*.

- This fits nicely with the HTTP requests and responses that underpin a web application.

The MVC Pattern

- An MVC application will be split into at least three pieces:
- *Models*, which contain or represent the data that users work with. Ideal to aim for: **'Fat Model'**
- *Views,* which are used to render some part of the model as a UI. Ideal: **'Stupid View'**
- *Controllers*, which process incoming requests, perform operations on the model, and select a view to render to the user. Ideal: **'Thin Controller'**

The MVC Pattern

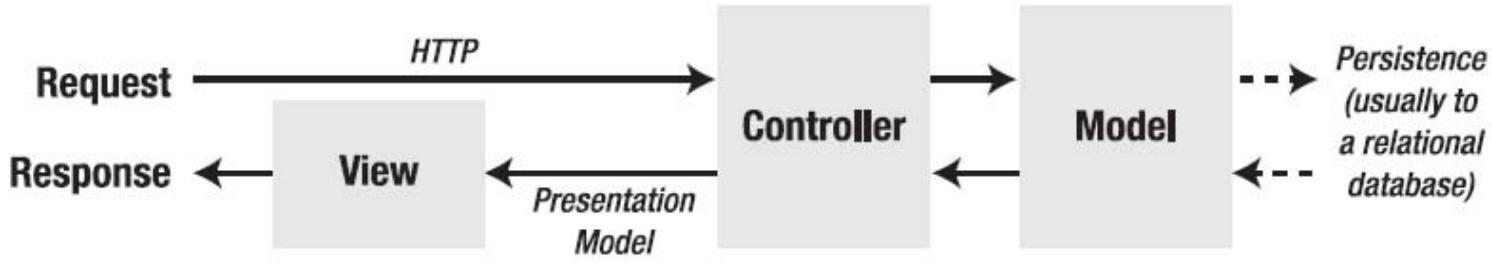


Figure 4-1. The interactions in an MVC application

Smart UI (anti)Pattern

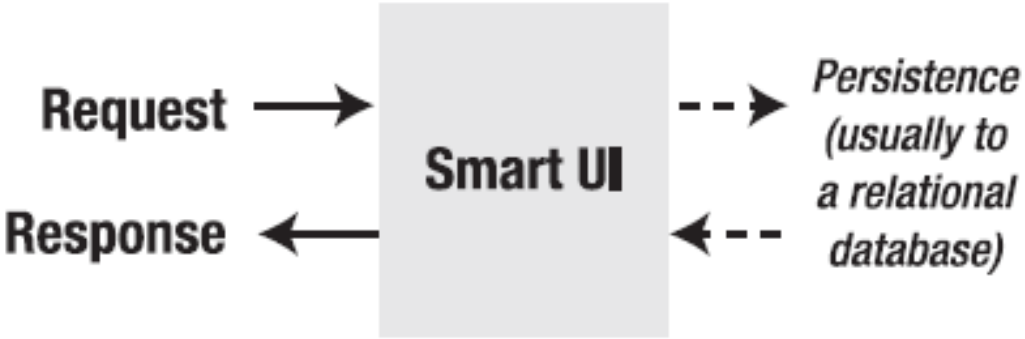


Figure 4-2. The smart UI pattern

Smart UI (anti)Pattern

The classic situation

Three Tier Architecture

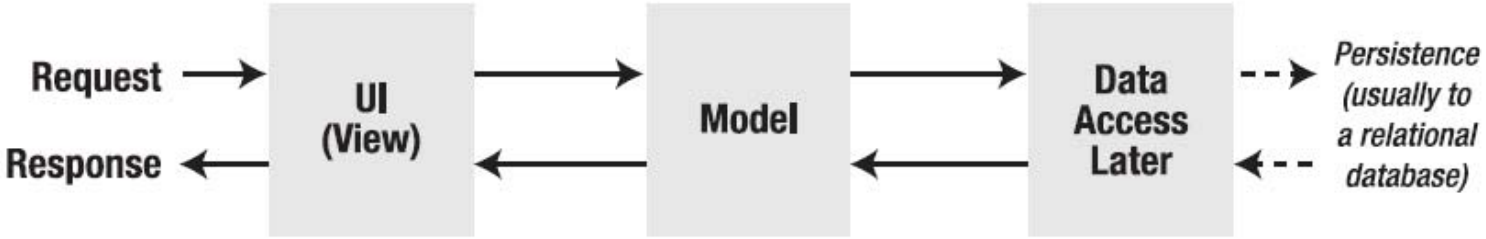


Figure 4-4. The three-tier pattern

Problem: when the UI layer is directly coupled to a click-and-event GUI framework (such as Windows Forms or ASP.NET Web Forms), it becomes almost impossible to perform automated unit tests.

Consequence: In the worst scenario, the three-tier pattern's lack of enforced discipline in the UI tier means that many such applications end up as thinly disguised smart UI applications, with no real separation of concerns. This gives the worst possible outcome: an untestable, unmaintainable application that is excessively complex.

Modelling an auction

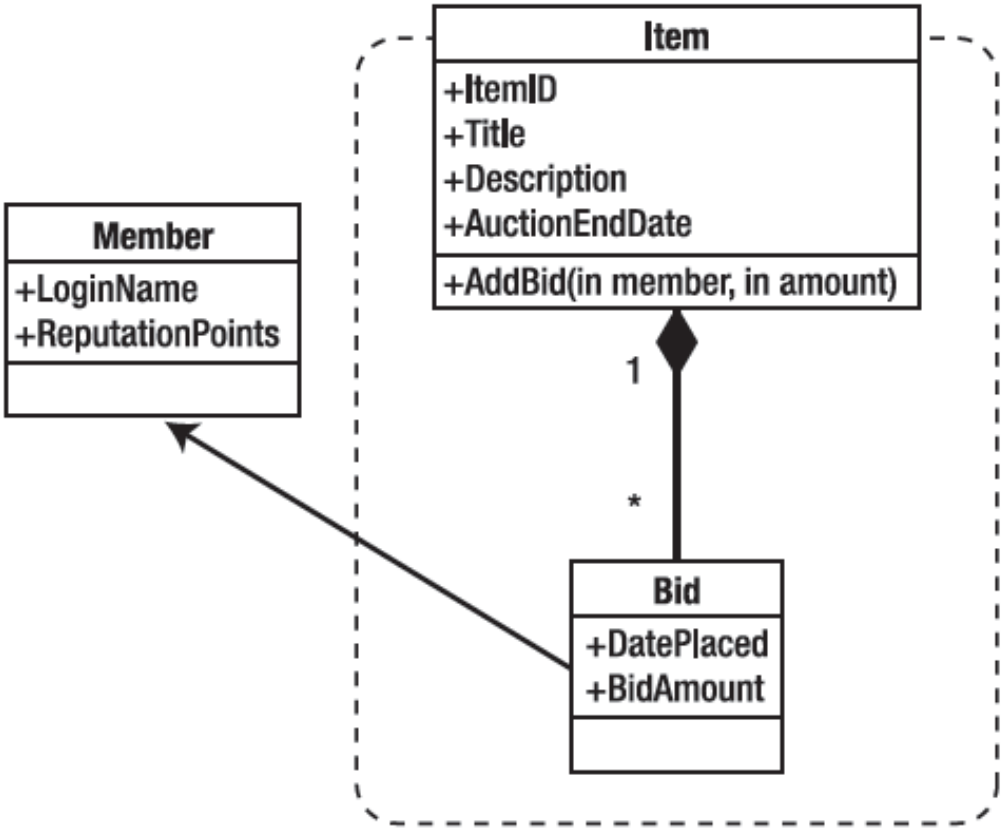


Figure 4-6. The auction domain model with aggregates

The Auction in C#

Listing 4-1. *The C# Auction Domain Model*

```
public class Member {
    public string LoginName { get; set; } // The unique key
    public int ReputationPoints { get; set; }
}

public class Item {
    public int ItemID { get; private set; } // The unique key
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTime AuctionEndDate { get; set; }
    public IList<Bid> Bids { get; set; }
}

public class Bid {
    public Member Member { get; set; }
    public DateTime DatePlaced { get; set; }
    public decimal BidAmount { get; set; }
}
```

Unit Testing

- Arrange (set up a scenario)
- Act (run some operations)
- Assert (verify results of said operations)

Unit Testing with MS Test

- "..the behavior we want is the ability to add a bid to an item, but only if the bid is higher than all previous bids for that item." ...this translates to the following unit tests:

- 1) If no bid is placed, all positive values for a bid are accepted.
- 2) If a bid is already placed, only a bid with a higher value can be accepted as a new bid.
- 3) If one or more bids are already placed, a bid with a lower value cannot be accepted.

>> linie 95, AdminControllerTest .cs

Unit Testing

Making the example code fail and pass: AddBid method

```
public void AddBid(Member memberParam, decimal amountParam) {

    // failing the three bid tests deliberately
    throw new NotImplementedException();

    // method implementation, allowing the three bid tests to pass
    //if (Bids.Count() == 0 || amountParam > Bids.Max(e => e.BidAmount)) {
    //    Bids.Add(new Bid() {
    //        BidAmount = amountParam,
    //        DatePlaced = DateTime.Now,
    //        Member = memberParam
    //    });
    //} else {
    //    throw new InvalidOperationException("Bid amount too low");
    //}
}
```

Unit Testing

- Example: unit test of the login functionality in a web app (*AccountController*)
- Example: unit test of a shopping cart (*CartTest*, *CartControllerTest*, BookStore_Ch9)

Unit Testing and DI

- Video materials:
- 1) Dependency Injection and IoC– Rob Conery
- 2) Unit Testing, Red-Green-Refactor / ReSharper / TDD workflow – Rob Conery
- (Source: http://tekpub.com)

Why is MVC suitable for TDD?

- > Your TDD workflow starts with Dependency Injection. Without DI, no ‘atomised’ unit tests, and no real possibility of using mock tools like MOQ, Rhino Mocks and other utilities in the correct manner.

- > A good continuous integration setup where the company both manages source control and maintains a build server running daily automated builds will support the TDD process well.

Why is MVC suitable for TDD?

- Q: Can't I do TDD without Dependency Injection?
- A: To some degree, you can do an approximation – but DI allows you to painlessly mock complex objects like the HttpContext, which again is crucial to unit testing large web applications effectively.
- Conclusion: we direly need a work environment where testing is baked in as a primary concern from the outset – and ASP.Net MVC actually delivers on this aspect, making it easy to unit test the produced code.

When is TDD a bad idea?

- Examples: winforms code that calls many COM objects, applications where many different software solutions need to interchange data - here, *integration testing* is the reasonable way forward.

- Practical example: Outlook add-in and small console application, which runs a script that exports XML for later rendering on a web page

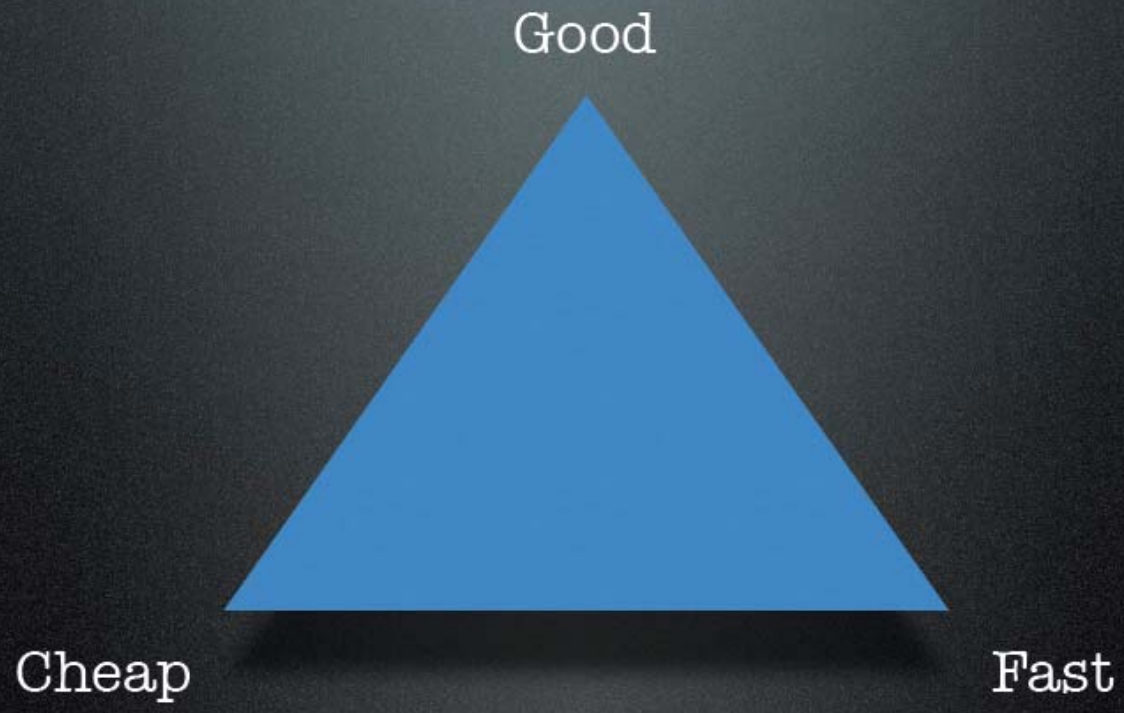
Summary

- A research study, and the most relevant findings
- Video: MS study, **Nachi Nagappan**
- Conclusions, some highlights from the report

Summary

- **How can this help the company?**
- Depending on how business critical the system is, the 15- 35% spent in extra time can really cut down on the bug fixing post-release – and they are a boon if new programmers join the team: '..unit tests *almost* don't lie..'
- Kirrily Robert: The following 16 borrowed slides present a persuasive business case for TDD.

The problem



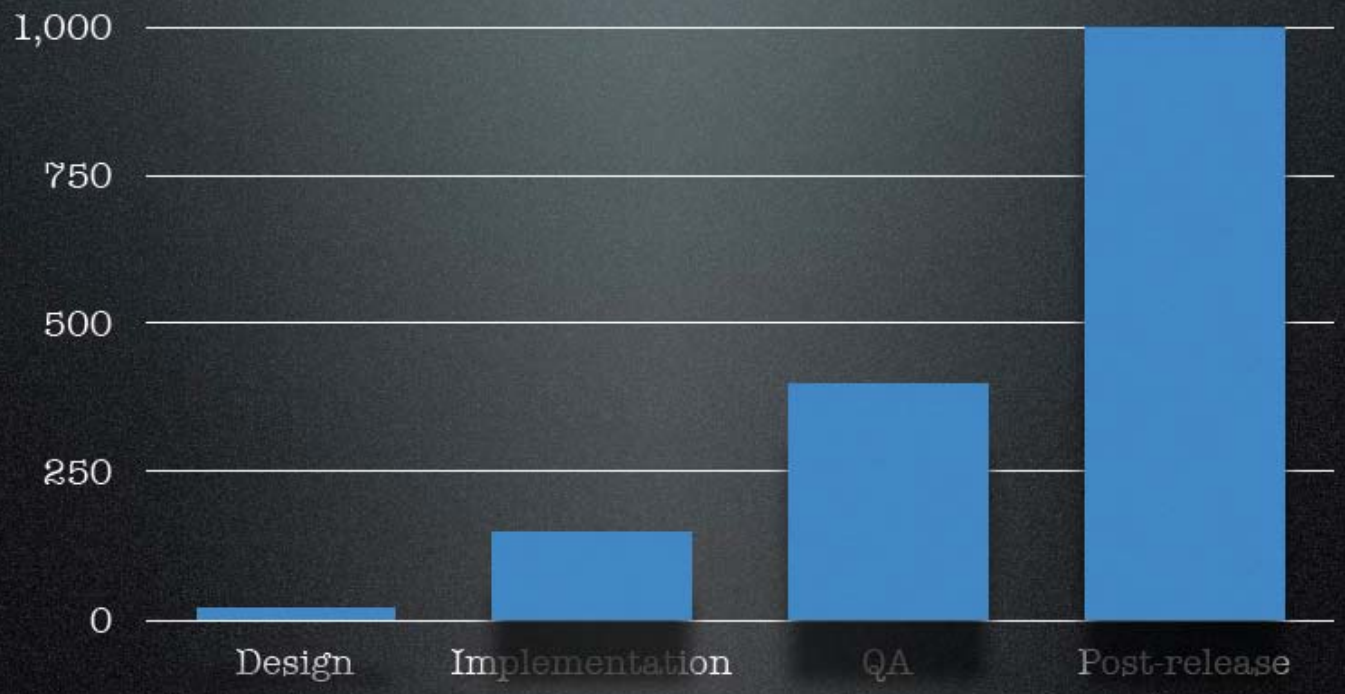
No silver bullet

However, with testing...

- A bit faster
- A bit cheaper
- A bit better

Faster

Time taken to fix bugs



Cheaper

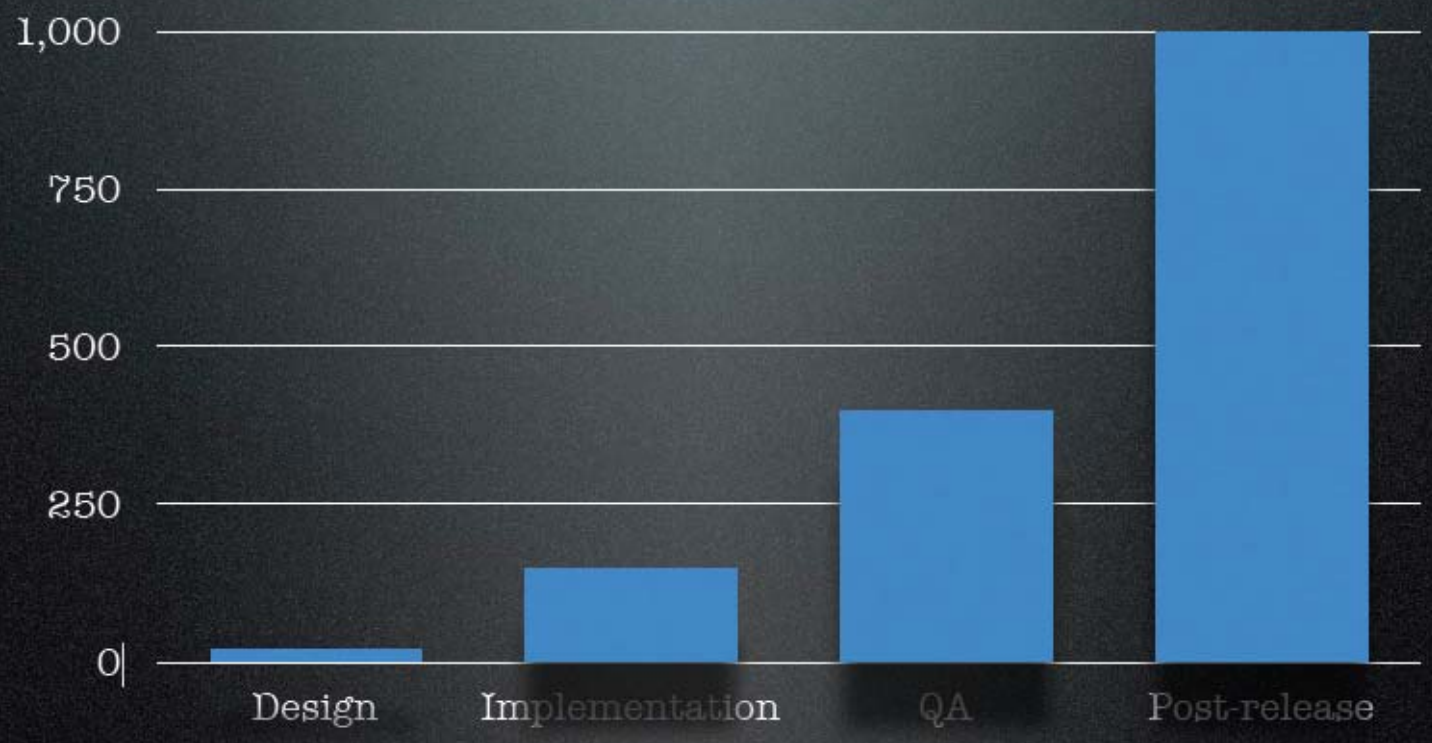
Technical debt

- “We’ll leave it for now”
- Delayed payment plan
- Compound interest

The most powerful force in the universe is compound interest.

Albert Einstein

Time taken to fix bugs



Easy payment plan

- Don't go into debt
- Make regular payments
- Pay down the principal

Cheap programmers

- Best programmers 10x as effective
- Testing can close the gap (somewhat)

Better

Software quality

- “Instinctive”
- Hard to measure

Software Kwalitee

- Indicative
- Measurable
- Testable

The solution

- Testing
- Test Driven Development

Summary

- **How can TDD help the programmer?**

- TDD represents a very structured approach to unit testing – instead of everyone having his or her individual approach to testing, the company can choose to implement the practice as daily routines to be followed, with a consistent end result.

  Inspiration: courses in DK, free online videos featuring Roy Osherove.

Summary

- **How do we implement?**

- A lot of work.. 'people management' is necessary, and having a rock solid technical infrastructure in the form of a customised build server / continuous integration setup is also a necessity.

- Small shops and freelancers often use a 'free tools approach' – ie. Mercurial, TeamCity for ASP. Net MVC based open source projects.

Thanks!

panmedia

Finn Vilsbæk

fvs@panmedia.dk

26200661