Delphi Unit Testing
Why you want it – How you do it

Jeroen Wiert Pluimers | jeroen@BeSharp.net

20121025 | ITDevCon.it | Verona, Italy

BeSharp.net ™

Intro

- Conference
  - Company
    - Person
      - Love to teach

BeSharp.net ™

Welcome

- Learn by
  - Demos
  - Discussion
- Your input is important!

BeSharp.net ™

Agenda

- Introduction to Unit Testing
  - About the process
  - Short demos from the Delphi side
- Other session which is more advanced:
  - Tecniche avanzate per Unit Test
    by Marco Breveglieri
    at 10:20
    in room TComponent (room “self”)

BeSharp.net ™

Unit testing

- Two questions for you
  - who does unit testing?
  - why not?

BeSharp.net ™

You should!

- Unit testing is about quality:
  - it helps you to get a better feel of how good your code is
- No need to have 100% code coverage
  - Any unit testing makes it easier to change code:
    - Refactor to improve the design
    - Adapt code for new business requirements
    - Etc..
- Remember:
  - a bug is an opportunity for writing a test

BeSharp.net ™

Am I doing it?

› Yes, but still not enough.

From one extreme to the other

› No testing at all

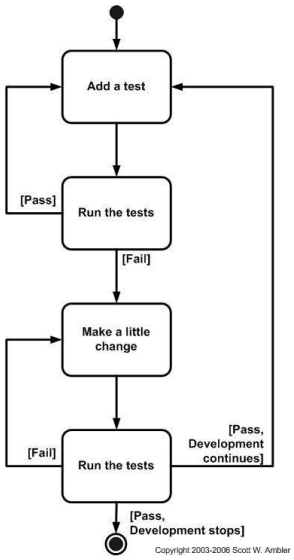
› It compiles: ship it!

› Common Sense

› Ad Hoc Testing
(aka Monkey Testing)

› Sanity Testing
(aka Smoke Testing)

› TDD aka Test First Development ----->



It is part of your test suite

Test suite part	What it tests
› Unit testing	your code
› Regression testing	things you broke
› Integration testing	externals
› Acceptance testing	your product

Testing paradox

› Every means you use to prevent or find bugs leaves a residue of subtler bugs

› Then why test at all?

› You end up with less bugs!

Why are people afraid of unit testing?

- Unit testing is complex
  - it is not: that's why this session is here
- It needs to cope with dependencies
  - that what the Spring framework session was for
- You need to simulate the outside world
  - that's what Mock objects are for
- You need to repeat tests
  - Use continuous integration
  - and a version control system

BeSharp.net ™

No need for TDD to do unit testing

- You can write unit tests after your code was written
- A bug report is a good opportunity to write a unit test
  - Now you know how to measure the fix:
    - If the unit test succeeds, the fix is OK
- Writing unit tests
  - Will reveal existing bugs
- Having unit tests
  - Is a bug repellent: prevents many new bugs

BeSharp.net ™

Basic unit test strategies

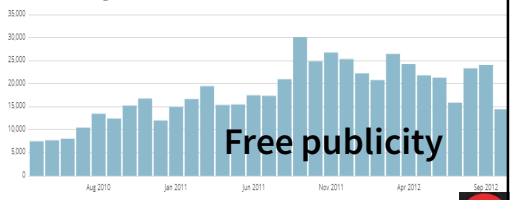
- Test the extremes of your code
  - As boundary conditions often are bug prone
- Test for effects that should occur
  - Positive testing: test functionality
- Test for effects that should not occur
  - Negative testing: prevent bad things from happening

Unit testing saves time

- This sounds like a paradox:
  - Writing a unit test costs time, how can it save time?
  - Because
    - you don't loose time testing using manual steps over and over again
    - Unit tests can be repeated reliably, manual steps not
- Remember: Unit tests can be run automated

Similar to why I write a blog

- wiert.me is a on-line binary-log of things I have done
- Allows for searching
  - things I did before, but forgot how I did it or where I found the information
- Cost
  - Initial cost: on average 30 minutes per blog entry
  - Total cost: 3 years: 1100 entries == 550 hours
  - Average cost: < 10% of a working day
- Gain
  - 100's of hours by
    - Finding back stuff
    - Becoming a better writer
    - Easier event preparation



BeSharp.net ™

Preparing the classes you want to test

- Class Under Test should be as independent as possible
- Dependencies:
  - Friendly class
    - Can be safely used without side effects
  - Stub class
    - Fakes a dependency by returning canned responses
  - Mock class
    - Simulates an object for testing purposes
- You should test at the dependency boundaries.
- And make the dependencies flexible and testable (Spring Framework, Mocks, Interfaces)

BeSharp.net ™

Testing on new development:

```
1. write a test
2. repeat
3.      watch it fail
4.      write code
5.      test it
6. until passed
```

› If writing unit tests for your code is too hard, then your code should be simplified to make the tests on the parts easier to write.

BeSharp.net ™

DUnit

› uses old style RTTI
  – all Test methods need to be published
› runs as GUI or Console app

› Demos:
  – CdsXsdTest
    › tests XSD code from Delphi
  – XMLEOSErrorTest
    › proper XML character set encoding
  – Number2StringProjectTests
    › Dutch/English/German (need Italian help!) for writing numbers
      – AddMatch(1316, 'one thousand three hundred sixteen');

BeSharp.net ™

Examples

- Sample message
  - TestNumber2String: ETestFailure
  - at $005730B3
  - Key=1444, expected: <
  - eintausendvierhundertvierundvierzig> but was: <
  - einstausendvierhundertvierundvierzig>

BeSharp.net ™

Testing and Exceptions

- Un-expected exceptions
  - Catched by DUnit
  - Fail your test
- Expected exceptions
  - Always expect (demo)
    - procedure **StartExpectingException**(e: ExceptionClass);
    - procedure **StopExpectingException**(msg :string);
    - procedure **CheckException**(AMethod: TTestMethod;
      AExceptionClass: TClass;
      msg :string);
  - Expect sometimes, but not allways
    - Use this pattern:
      - try
      - except
        - on EMyExpectedException do
        - begin
        - // this exception is harmles in this specific situation
        - end;
      - end;

BeSharp.net ™

When you have lots of similar test

- Store your tests
- Write some wrapper code to simplify testing

- Examples …

Delphi-Mocks

- from VSoft (Vincent Parret)
- Based on TVirtualInterface
  - http://docwiki.embarcadero.com/VCL/XE2/en/RTTI.TVirtualInterface
    - TVirtualInterface creates an implementation of an interface at run time.
    - All interface methods raise an OnInvoke event of type TVirtualInterfaceInvokeEvent

Final Advice (thanks Nick Hodges!)

- Only test the code that you want to work properly
  - focus your test on the most important code
- Don't test code that you don't care if it is buggy
    - Experimental code
    - One off code
  - make sure that this potential buggy code is flagged somehow in your version control system

References

- Overview of tools (2009, so some old)
  - http://blog.vi-kan.net/2009/tdd-unittesting-and-delphi/
- Nick Hodges (Gateway Ticketing, ex Borland)
  - http://www.nickhodges.com/category/Unit-Testing.aspx
    - Unit testing
    - Spring
    - Mocks
    - …

Delphi Mock libraries

- http://stackoverflow.com/questions/293755/what-is-your-favorite-delphi-mocking-library
- Best: Delphi Mocks
  - git checkout
    - https://github.com/VSoftTechnologies/Delphi-Mocks.git
  - site:
    - https://github.com/VSoftTechnologies/Delphi-Mocks
  - Introduction:
    - http://twitter.com/#!/FinalBuilder/status/115414466677579776
    - http://www.finalbuilder.com/Resources/Blogs/tabid/458/EntryId/287/Introducing-Delphi-Mocks.aspx

References

- Code Coverage
  - http://code.google.com/p/delphi-code-coverage/source/checkout
- Integrates with Hudson
  - http://stackoverflow.com/questions/531546/measuring-code-coverage-in-delphi/3115701#3115701

Q & A | Discussion

Jeroen Wiert Pluimers

If you have questions after the session,
please contact me

jeroen@BeSharp.net @jpluimers wiert.me

BeSharp.net ™