| Test-Driven Development: A Practical Guide, by Dave Astels	Book reviewed by Craig Murphy	craig murphy.com

developers magazine (Nov/Dec 2003)

Test-driven development (TDD) has become very popular in recent months. Until now there was only one textbook on the subject [Beck02]. "test-driven development: A Practical Guide" by David Astels, was published in July 2003 and complements Beck's original TDD work. This book is part of The COAD Series, named after Peter Coad, Borland's Senior Vice President and Chief Strategist. The book itself is a little over 550 pages long and offers up-to-the-minute advice on how TDD can change your development and testing philosophy.
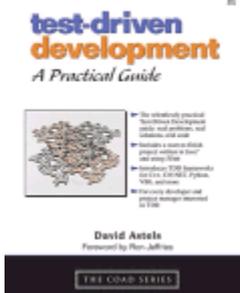
It's testing Jim, but not as we know it...

Astels' first sentence is "This is not a book about testing". True enough, TDD is not about testing in the traditional sense. TDD is about writing test code before we even write the implementation code. Sound odd? It is a paradigm shift, but it is something we should all get our heads around because it is, in my opinion, the one thing we can do that will allow us to spend less time in the debugger and more time writing good code and implementing the features our customer ask us for.

So how do we write a test before we write some code? I think it would be inappropriate for me to review this book without first explaining TDD, even if it is an elementary overview – I have provided one or two additional references in the resources section of this review.

Elementary TDD

I'm going to keep this short'n'simple: TDD is about test first development – a class containing a test is written before the class being tested is written. Why? Because TDD, and eXtreme Programming (XP), is all about doing the simplest thing possible: and if that means letting the compiler tell us that the class under test doesn't exist, so be it.

Listing 1 uses DUnit [DUnit], a TDD framework for Delphi. Clearly, compiling listing 1 is going to generate errors: TMagazineList does not exist: doing the simplest thing possible tells us we need to define a new class.

```
unit BUGEx;
interface
implementation
uses TestFramework;
type
TTestBUG = class(TTestCase)
  private MagazineList : TMagazineList;
  public
  published
end;
end.
```

Listing 1 – TMagazineList does not exist

Removing the compiler errors is easy: we simply define the TMagDevelopers class, as shown in listing 2.

```
type TMagazineList = class(TObject)
end;

TTestBUG = class(TTestCase)
  private MagazineList : TMagazineList;
  published
end;
```

Listing 2 - Problem solved, so far…

By default, when we create a new instance of TMagazineList, it should be fair to assume that the list size would be zero. That's something we can test, we can make sure that the property/function Size returns zero after object creation. Hang on, where did the Size property/function come from? For now, we're not interested in that, we're more interested in writing a test first. Listing 3 demonstrates how we might do this using DUnit.

```
type TMagazineList = class(TObject)
```

```
end;

TTestBUG = class(TTestCase)
  private MagazineList : TMagazineList;
  published procedure testEmptyList;
end;

procedure TTestBUG.testEmptyList;
begin
  MagazineList := TMagazineList.Create;
  CheckEquals(MagazineList.Size, 0, 'MagazineList should have a size of 0');
end;
```

Listing 3 – Property/Function Size does not exist

Once again, the compiler tells us that there is an error: Size is not defined. So let's define Size property/function.
Consider Listing 4: I've created a function Size that returns zero. I know that I'm repeating myself, this is part of
the TDD mantra: do the simplest thing possible. In this instance, please excuse the pun, returning zero allows the
testEmptyList to pass. As the class develops, obviously this will change, however for now it is enough.

```
type TMagazineList = class(TObject)
  private function Size : Integer;
end;

TTestBUG = class(TTestCase)
  private MagazineList : TMagazineList;
  public published procedure testEmptyList;
end;

procedure TTestBUG.testEmptyList;
begin
  MagazineList := TMagazineList.Create;
  CheckEquals(MagazineList.Size, 0, 'MagazineList should have a size of 0');
end;

function TMagazineList.Size: Integer;
begin
  Result := 0;
end;
```

Listing 4 – Defining the Size function allows the test to pass

What's so good about this?
Let's face it, as developers none of us really like testing our code repeatedly: largely due to the amount of time it
takes to test each 'grain' of functionality. It's very easy for us to forget to test a particular condition, path, of
combination of events.

TDD allows us to build up a comprehensive collection of test classes that we can use over and over again: Astels
calls these "Programmer Tests". Being able to re-run the tests at a later date is the key: don't you just hate having
to go back a re-test a class because you've improved or changed the implementation? For every single class that is
developed, there will be a corresponding collection of tests available. Astels drums this point home when he states,
"no code goes into production unless it has associated tests".

Book Structure
The book itself is split into five major sections: Background, Tools and Techniques, A Java Project: Test-Driven End
to End, xUnit Family members and Appendices. On the whole, this book is structured in such as way that it is
possible to treat the end to end project as a standalone section – which is good, because there's nothing worse
than having a book's content spread throughout a case study. The four remaining sections are easy reading and
logically follow on from each other. Let's take a look at each section in turn.

Background
This section contains three chapters: Test-Driven Development, Refactoring and Programming By Intention. The
chapter is a clean and simple examination of the ethos behind TDD. Astels presents a simple Java example, one
which I borrowed ideas from earlier in this review. Over the course of nine pages TDD is explained via the use of
this worked example – at each point Astels explains why he is doing certain things (such as the Size function
initially returning zero).

You might be wondering why there is a 30-page chapter about Refactoring in a TDD book? Sometimes 'doing the
simplest thing' isn't always the best and may result in code that works, but is poorly implemented. Hence we set
about refactoring the code: we change the original, known to be working code, without changing what the code
actually does. Refactoring ultimately improves program design, but necessitates that we re-test the affected

classes: since we have a collection of tests available, it is the simple matter of running those same tests against the refactored code. Astels spends much of the chapter discussing how to locate classes that are candidates for refactoring. Whilst his original example from chapter 1 is carried through into this chapter, Astels introduces a number of other examples in order to demonstrate refactoring. Further information about refactoring is available at http://www.martinfowler.com , http://www.refactoring.com or in the definitive text on the subject [Fowler99].

The third chapter discusses Programming by Intention – which is another technique that XP and TDD make use of. Put simply, this is a chapter about how best to name classes, functions and properties. The idea of using nouns or noun phrases to determine class names, adjectives and generic nouns for interfaces, verbs and verb phrases for method names is probably well known to us. But do we practice them? Programming by Intention (or Intent, if you will), is summed up by Astels: "It means making your intent clear when you write code". This chapter refers back to the previous two chapters, firming up on their salient points explaining how programming by intent goes hand-in-hand with those points (do the simplest thing and refactoring). Astels makes a number of good points about how code should be commented. By adopting the advice in the chapter, it soon becomes clear that there are only a few occasions that we really need to provide comments for, and when we do, they should emphasise "why" we're doing something, not "how" we're doing it (the code should tell us that).

Tools and Techniques
This section contains five chapters: JUnit, JUnit Extensions, JUnit-Related Tools, Mock Objects and Developing a GUI Test-First. It is fair to say that TDD's origins in the Java environment mean that many examples use the Java language – even the original definitive text [Beck02] uses Java for most of examples.

The JUnit chapters follow the same concise style that the rest of the book adopts. They cover the JUnit class architecture, the assertions that are available and a graphical front-end. The graphical front-end, whilst built for use in a Java environment, is still worth reading about: after all, DUnit for Delphi is supplied with a very similar tool. Figure 1 presents a screenshot of the DUnit front-end testing tool: you can see that I have forced a test failure. The text of this chapter is still a worthy read: there are some real gems regarding "test first" strategy, organising test cases and test case construction.
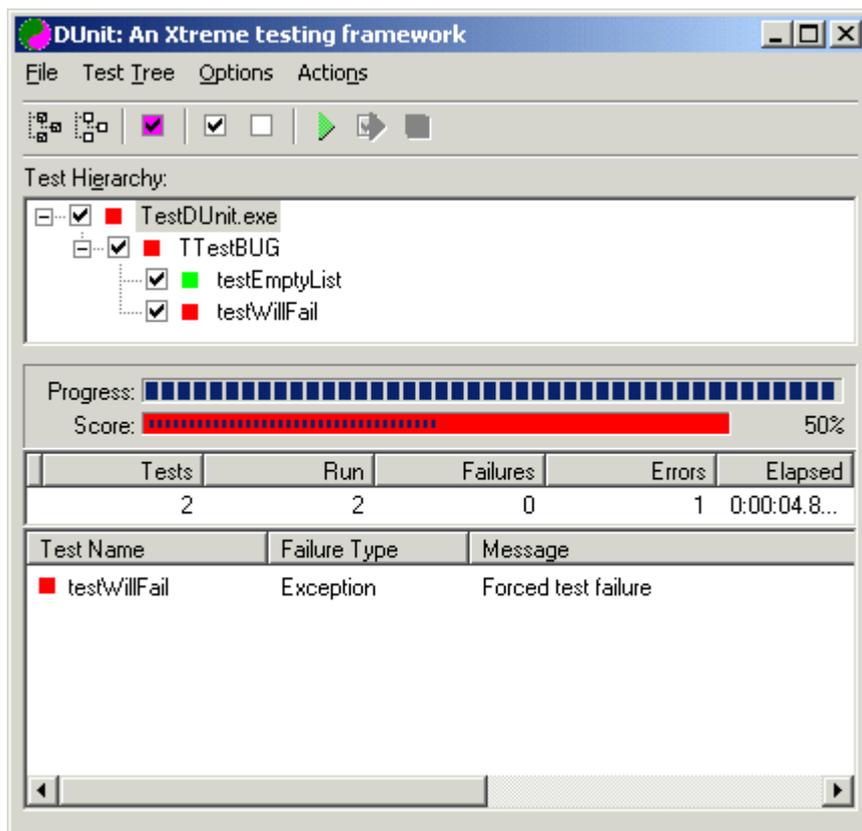

Figure 1 – DUnit in action

What's more exciting about this section is the coverage of mock objects and GUI testing. Astels dedicates 26 pages to the discussion of mock objects (colloquially known as mocks). Mock objects are used to simulate concrete objects: typically we might use a mock to simulate a database connection, for example. This chapter contains more new/fresh examples and includes UML sequence diagrams outlining the interaction between the various example objects, tests, and the mocks. There's also a good discussion about the 'uses for mock objects'. Not wishing to steal

too much of this book's thunder: the use of mocks promotes good class design and class interaction.

GUI testing is the subject of many newsgroup questions: Astels goes to the lengths of describing it as "tricky". TDD excels itself at class-level testing, where it becomes tricky is when class instantiation and use are controlled via a combination of check boxes, radio buttons, user input and buttons. Whilst most of the tools described in this chapter are Java-oriented, some of the techniques described could easily be applied in a Delphi environment.

A Java Project: Test-Driven End to End
Some 229 pages out of this book's 550 are dedicated to the development of a complete project. Oddly enough, the project itself is built using Java, however each of the development steps are equally applicable in a Delphi environment. We've already seen that this book touches on XP and this chapter is no different. The project itself is defined using a 'vision', a collection of 'user stories' and a set of 'tasks'. Astels then takes each of the 'stories' (elements of functionality) and describes how they could be implemented using a test-driven approach.

xUnit Family Members
What is xUnit? xUnit is a colloquial term that relates to specific language implementations of a unit testing framework. The most famous xUnit member is JUnit - a collection of classes that implement TDD in Java.

It's fair to say that is probably an xUnit implementation for whatever language you are using; if there isn't, it's not very difficult to write an xUnit implementation. Given that most xUnit implementations are supplied with source code, coupled with this book's detailed analysis of the JUnit architecture, writing your own xUnit isn't that arduous a task.

As we've already seen, there is a Delphi implementation of the xUnit framework: it's called DUnit (there's a URL in the Resources section of this review).

And yes, you know I couldn't write an article without mentioning it...there is an xmlUnit available! Nuff said.

Appendices
There are four appendices covering 31 pages. This is a welcome change; some of the books I have bought in the recent past have been made up of over one-third appendices.

The first appendix is an eight-page overview of eXtreme Programming (XP). Whilst (in my opinion) TDD can be used in a standalone fashion, it has origins in the lightweight software development methodology that calls itself XP. Without getting involved in discussing XP – because that's another article in itself – TDD is part of the XP modus operandi: TDD is considered good practice and so it becomes part of XP.

The second appendix consists of seven pages discussing Agile Modelling. Astels enlists the assistance of Scott Ambler – of http://www.ambysoft.com fame; Ambler is renowned in the modelling and persistence fields amongst others. This appendix is a succinct examination of the myths behind modelling – it's well written and hammers home the inaccuracies of the erroneous statements that occasionally raise their head when discussing XP, TDD and agile modelling.

The third appendix is nine pages long, and covers Online Resources: an appendix chock full of online newsgroups, xUnit software references, information about XP and agile modelling. Finally, the last appendix contains the answers to the book's 23 exercises. Whilst the answers are presented in Java, Astels does provide a few words explaining each answer.

Should I Buy This Book?
A book review that includes a reference to the definitive TDD text [Beck02] might seem a little strange. However, these books complement each other: both are worthy additions to your library. In fact, Astels' book promotes Beck's work on the inside back cover (and I'm reliably informed that future print-runs of Beck's book will be returning the favour). If you found Beck's book a worthy read and appreciate second opinions, this book is for you.

You should buy this book if you want to learn how to reduce the amount of time you spend testing your code. This book demonstrates how to build repeatable test cases that can be run at the touch of a mouse button.

You should by this book if you are fed up spending time in the debugger trying to figure out what's wrong with a piece of code. Traditional approaches where a programmer "writes code, tests code" often means that the programmer does not always re-test code after small changes have been made. This book will show you how to write test cases that can help prove that a piece of code works (or not, which is just as good, because you can then fix the problem before an end-user tells you about it!)

You should buy this book if you want to increase your productivity and increase your confidence. Writing tests before you write any code may sound as if it will take longer, this is not necessarily the case. This book drums home the fact that as programmers we like knowing that our code works: and there's no better way of proving that code works than by running a series of successful tests. The psychological effect of knowing that a series of tests run successfully has the knock effect of increasing the programmer's confidence and thus increases productivity (the craving for more successful tests means writing more tests, then more code!)

All in all, despite this book's Java orientation, it does document 106 tests in a real project. Determining what to test

can be difficult when starting out with TDD, having a collection of tests written for use in a real project is a good learning tool.

If your library of books that you keep close to hand already contains one or two of the books listed in the Resources section, this book is a worthy addition. Dave Astel's test-driven development: A Practical Guide, ISBN 0-13-101649-0, published 2003 by Prentice-Hall/Pearson Education. Expect to pay £20-24 depending upon your chosen bookseller.

Resources
Craig Murphy's "Test-Driven Development Using Delphi 8", The Delphi Magazine , Issue 104 (Mar 2004)
Kris Golko's "To err is human: automated testing of Delphi code with DUnit" UK-BUG Magazine Issue Nov/Dec 2002
Rob Bracken's "Testing: Quality Time with DUnit", The Delphi Magazine , Issue 76 (Dec01)
Sascha Frick's "An Introduction to Endo-Testing Using Mock Objects", The Delphi Magazine, Issue 96 (Aug03)
Dave Astel's web-site: http://www.adaptionsoft.com

More information about Mock Objects can be found here: www.mockobjects.com
Yahoo's popular testdrivendevelopment group: http://groups.yahoo.com/group/testdrivendevelopment
[DUnit] The DUnit group at SourceForge: http://dunit.sourceforge.net
[Beck02] Kent Beck, Test-Driven Development: By Example, Addison-Wesley, 2002, ISBN 0-321-14653-0
[Fowler99] Martin Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999, ISBN 0-201-48567-2

About the author
Craig is a Certified ScrumMaster (available for hire), an author, developer, speaker and Dilbert Evangelist – he specialises in all things XML, particularly SOAP and XSLT. Craig is also evangelical about C#, Test-Driven Development and Extreme Programming. He can be reached via his web site: http://www.craigmurphy.com (where you can also find the source code and PowerPoint files for all of Craig's UK-BUG articles, reviews and presentations).

If you would like Craig to write a review of your book or product, or if you would like him to write an article, please contact him using the e-mail address found at the bottom of http://www.craigmurphy.com.